BASIC RESEARCH NEEDS IN

The Science of Scientific Software Development and Use

Investment in Software is Investment in Science

Basic Research Needs in
The Science of Scientific Software Development and Use
Investment in Software is Investment in Science

Report of the Department of Energy Advanced Scientific Computing Research Workshop
13-15 December 2021

Organizing Committee	Michael A. Heroux	Sandia National Laboratories
	David E. Bernholdt	Oak Ridge National Laboratory
	Lois Curfman McInnes	Argonne National Laboratory
	John R. Cary	University of Colorado
Report Co-Authors	Daniel S. Katz	University of Illinois at Urbana-Champaign
	Elaine M. Raybourn	Sandia National Laboratories
	Damian Rouson	Lawrence Berkeley National Laboratory
Additional Contributors	Hartwig Anzt, Michelle Barker, Aaron S. Brewster, Joshua S. Brown, Leslie Carr, Jeff Carver, Sou Cheng, Bin Dong, Ben Dudson, Kjiersten Fagnan, Stephen M. Fiore, Sandra Gesing, Simon Hettrick, Fred J. Hickernell, Ignacio Laguna, Spencer Smith, Reed Milewicz, Todd Munson, Erdal Mutlu Pacific, Manish Parashar, Slaven Peles, David M. Rogers, Philip C. Roth, Benjamin Sims, Matthew Sottile, Gregory R. Watson, Nic Weber, Jason Wiese, Justin M. Wozniak, Chen Zhang *(for affiliations see Appendix D)*	
DOE Point of Contact	Hal Finkel	Advanced Scientific Computing Research
Workshop Support	Jody Crisp	Oak Ridge Institute for Science and Education
	Deneise Terry	Oak Ridge Institute for Science and Education
	Paul Hudson	Oak Ridge Institute for Science and Education
	Rose Lynch	Argonne National Laboratory

CITATION: Michael A. Heroux, David E. Bernholdt, Lois Curfman McInnes, John R. Cary, Daniel S. Katz, Elaine M. Raybourn, Damian Rouson. Basic Research Needs in The Science of Scientific Software Development and Use. United States Department of Energy, Advanced Scientific Computing Research, 2023. doi:10.2172/1846009.

Contents

Executive Summary **iv**

1 Introduction **1**
1.1 Priority Research Directions and Crosscutting Themes 1

2 Priority Research Directions **3**
2.1 PRD 1: Tools 3
2.1.1 Introduction 3
2.1.2 Key Questions 3
2.1.3 Scientific Challenges and Opportunities 4
2.1.4 Research Thrusts 5
2.2 PRD 2: Software Teams 7
2.2.1 Introduction 7
2.2.2 Key Questions 8
2.2.3 Scientific Challenges and Opportunities 8
2.2.4 Research Thrusts 9
2.3 PRD 3: Trustworthy Science 11
2.3.1 Introduction 11
2.3.2 Key Questions 11
2.3.3 Scientific Challenges and Opportunities 12
2.3.4 Research Thrusts 12
2.4 Scientific and Technology Impacts 14

3 Crosscutting Themes **16**
3.1 Theme 1: The Human Element 16
3.1.1 Introduction 16
3.1.2 Opportunities 16
3.1.3 Connections to PRDs 18
3.2 Theme 2: Workforce 19
3.2.1 Introduction 19
3.2.2 Opportunities 19
3.2.3 Connections to PRDs 21
3.3 Theme 3: Centrality of Software 22
3.3.1 Introduction 22
3.3.2 Opportunities 23
3.3.3 Connections to PRDs 23
3.4 Connections among Crosscutting Themes 24

4 Conclusions **25**

Bibliography **30**

Appendices

A Workshop Description and Call for Position Papers **31**
A.1 Description 31
A.2 Position Paper Submission 31
A.2.1 Motivation 31
A.2.2 Invitation 32
A.2.3 Submission Guidelines 32

B Workshop Agenda **35**

C Accepted Position Papers **37**

D Workshop Participants **45**

E Pre-Workshop Report **53**

List of Tables

1.1 Summary of Priority Research Directions and their relative scope. Each PRD represents numerous important questions that merit R&D investments in pursuit of the goal of improving the development and use of scientific software toward improved scientific impact. 2
1.2 Summary of crosscutting themes and their primary goals. Each theme represents an overarching approach to conducting our work in scientific software development and use, guiding how we plan and execute our efforts. 2

C.1 Accepted position papers. 37

D.1 Summary of registered participants. Company types are self-reported and have not been verified. 45
D.2 Registered Workshop Participants. Type denotes invited participants (I), observers (O), or support staff (S). Roles include: organizer (O), U.S. DOE Point of Contact (POC), keynote speaker (KS), lightning speaker (LS), breakout leader (BL), and contributor to the final report (R). 45

Executive Summary

Investment in Software is Investment in Science

Increasingly powerful and affordable computing has revolutionized scientific and scholarly discovery across a broad range of fields. Computing relies on software, which has been rapidly growing in scope, diversity, and complexity. At the same time, the methods, processes, and tools used to produce and utilize this essential software are often *ad hoc*, and the study and improvement of them are often done without the benefit of direct funding or prioritization. Consequently, concerns are growing about the productivity of the developers and users of scientific software, its sustainability, and the trustworthiness of the results that it produces.

Increased investment, especially in the characterization and improvement of how scientific software is developed and used, is important for sustaining and improving the impact of software as the scope and complexity of scientific efforts expand. Without this investment, we face the risk of diminishing returns on our software investments because the demands for increased functionality, usability, reliability, and more will not be sufficiently met.

The US Department of Energy Office of Science (DOE/SC) is at the forefront of modern software-enabled scientific discovery across numerous areas of computational, experimental, and observational science, including major investments in national user facilities that support these activities. For many years, DOE/SC software investments have provided tremendous value to the scientific community. We want to continue and further improve the value of DOE/SC software efforts by using a scientific approach to understanding and improving how scientific software is developed and used.

In December 2021, the DOE/SC Office of Advanced Scientific Computing Research (ASCR) convened a workshop on basic research needs for the Science of Scientific-Software Development and Use (SSSDU). Through keynote presentations, lightning talks, and breakout groups, which built on insights from 124 pre-workshop position papers, participants discussed the current practice of software development, maintenance, evolution, and use, and considered how the scientific method could be used to examine these practices and develop more evidence-based approaches to enhance the impact of software and computing on all areas of science.

Workshop participants identified three priority research directions (PRDs) and three important cross-cutting themes that center on the following overarching insight: *Software has become an essential part of modern science, impacting discoveries, policy, and technological development. To maintain and improve confidence in science delivered via software, we must improve the processes and tools that help us create and use software, and this enhancement requires a deep understanding of the diverse array of teams and individuals doing the work.*

Priority Research Directions

PRD1: Develop next-generation tools to enhance developer productivity and software sustainability

Key questions: *How can we leverage and build upon emerging tools that incorporate artificial intelligence/machine learning (AI/ML) to improve the productivity of scientific software developers? How can we create and adapt tools to improve developer effectiveness and efficiency, software sustainability, and support for the continuous evolution of software? How can we support and encourage the effective adoption of new tools by developers?*

Numerous tools assist developers with their activities. Many tools that are well-established in the larger software world might not be widely known or used in scientific software development. We anticipate the rise of future software-generating environments that translate scientific programmer intent into source code fragments that can then be tuned and refined. We also expect advances in tools that assist in generating software tests, documentation, clean and readable source code, and more. The challenges faced by the scientific software community include ensuring that future tools account for scientific software requirements that might not be high priorities in other fields and addressing opportunities that may be unique to scientific software but are still critical. Work is also needed to facilitate the adoption of new tools, including effective training, along with help in incorporating these tools into already-complex workflows.

PRD2: Develop methodologies and tools to comprehensively improve team-based scientific software development and use

Key questions: *What are the roles of software in scientific processes? What are the key roles on scientific software teams? What factors contribute to the effectiveness of scientific software teams? What practices, processes, and tools can help improve the development, sustainment, evolution, and use of scientific software by teams?*

As the fundamental understanding of scientific software improves, we foresee that the methodologies and tools we need will also change to better match and support how developers, users, and other scientific software stakeholders work toward the goal of accelerating scientific discovery. Although many scientists have extensive intuition about the principles and dynamics of how their community develops, uses, and sustains its software products, research is needed to establish a deeper and broader understanding of software's role in scientific processes. We believe that developing common mental models around fundamentals can help the scientific community as a whole observe, understand, and improve the effectiveness and efficiency of scientific discovery through better methodologies and tools for the development, use, and sustainment of scientific software, particularly in the context of diverse collaborative teams.

PRD3: Develop methodologies, tools, and infrastructure for trustworthy software-intensive science

Key questions: *How can we facilitate and encourage effective and efficient reuse of data and software from third parties while ensuring the integrity of our software and the resulting science? How can we provide flexible environments that "bake in" the tracking of software, provenance, and experiment management required to support peer review and reproducibility?*

Scientific results are trustworthy only if all aspects of the scientific process can be trusted to produce correct, transparent, reproducible, and replicable results. Although every scientific software developer intends to produce trustworthy results, the current state of the practice varies tremendously from team to team. There are many concerns to consider in ensuring trustworthy computational results, and all of them must be addressed to some degree if we are to make quantitative and qualitative progress. Our data and software must be managed, archived, and retrievable, and our computational steps must be recorded and available for future use. We must be able to easily detect and correct perturbations in state and execution, and we must be able to preserve provenance down to the finest granularity for assessment and audit.

Crosscutting Themes

Theme 1: We need to consider both human and technical elements to better understand how to improve the development and use of scientific software.

It is common to assume that challenges encountered in software-intensive research can be overcome simply by some new technical innovation. However, people—individuals and teams—play enormous roles in the development and use of software, often in both the challenges and in the potential solutions. Therefore, we consider this human element essential in our efforts to understand and improve scientific software development and use, including engagement with social, cognitive, and information scientists and others who have not historically been part of the DOE computational research community.

Theme 2: We need to address urgent challenges in workforce recruitment and retention in the computing sciences with growth through expanded diversity, stable career paths, and the creation of a community and culture that attract and retain new generations of scientists.

Developing high-performance scientific software requires the combined contributions of many people with a wide range of skills and backgrounds, most of which are also in high demand in the broader marketplace. Our ability to acquire staff for current and future teams requires strong and sustained efforts to educate, recruit, and retain a diverse workforce by cultivating a supportive and inclusive culture within the computing sciences [1]. We urgently need broad and sustained community collaboration to change the culture and demographic profile of scientific computing with multipronged approaches to expand the pipeline and workforce.

Theme 3: Scientific software has become essential to all areas of science and technology, creating opportunities for expanded partnerships, collaboration, and impact.

Powerful and affordable computing has revolutionized the scientific enterprise. Computing also provides the basis of many technological and engineering products and increasingly informs policy and other consequential decisions. Owing to the broader importance of scientific software, efforts to better understand and improve its development and use should find allies beyond DOE, including practitioners in academia and industry, as well as other government agencies that fund computationally and software-intensive research.

Impacts and Opportunities

We foresee a variety of positive impacts from pursuing the Priority Research Directions, as described in this report. First and foremost, perhaps, are improvements in the effectiveness and efficiency of scientific software development activities. Since so much scientific software development and use is now carried out in team settings, we expect that a better understanding of how teams work in this context, the interplay between teams, and more loosely connected software communities will likewise impact effectiveness and efficiency, but also inclusivity, equity, and other aspects of the human side of both science and software. Finally improving the trustworthiness of software-intensive science not only addresses one of the fundamental tenants of the scientific method, but will also engender greater trust from those who rely on our answers to inform their own work, decisions, and policies.

The Crosscutting Themes identified in the report provide opportunities that we can leverage and in some cases obligations that we must address, to carry out the R & D based on the Priority Research Directions. We need to pay attention to the human aspects of software development and use, as well as the technical, and engage with social, cognitive, and information scientists to achieve better understanding. We *must* build a workforce to support our software development needs that are not only well-trained, but diverse, inclusive, and equitable. And in all of this, we can leverage the fact that software has become central to the entire

scientific enterprise to look for allies beyond DOE in our efforts to improve our understanding of the science of scientific software.

Ultimately, it comes back to the tagline which arose from this workshop:

Investment in software is investment in science.

Chapter 1

Introduction

Increasingly powerful and affordable computing has revolutionized scientific and scholarly discovery across a broad range of fields. Computing relies on software, which has been rapidly growing in scope, diversity, and complexity. At the same time, the methods, processes, and tools we use to produce and utilize this essential software are often *ad hoc*, and the study and improvement of them are often done without the benefit of direct funding or prioritization. Consequently, concerns are growing about the productivity of the developers and users of scientific software, its sustainability, and our ability to trust the results that it produces.

The Department of Energy Office of Science (DOE/SC) is at the forefront of modern software-enabled scientific discovery across numerous areas of computational, experimental, and observational science, including major investments in national user facilities supporting these activities. In December 2021, the DOE/SC Office of Advanced Scientific Computing Research (ASCR) convened a workshop on basic research needs for the Science of Scientific Software Development and Use (SSSDU, Appendix A). Through keynote presentations, lightning talks, and breakout groups (Appendix B), which built on insights from 124 pre-workshop position papers (Appendix C), 264 registered participants (Appendix D) discussed the current practice of software development, maintenance, evolution, and use, and considered how the scientific method could be used to examine these practices and develop more evidence-based approaches that would enable us not merely to continue, but also enhance the impact of software and computing on all areas of science.

Workshop participants identified three priority research directions and three important crosscutting themes that center on the following overarching insight: *Software has become an essential part of modern science, impacting not only discovery but also policy and technological development; to maintain and improve confidence in science delivered via software, we must improve the processes and tools that help us create and use it, requiring a deep understanding of the diverse teams and individuals doing the work.*

1.1 Priority Research Directions and Crosscutting Themes

The SSSDU workshop brought together a diverse collection of people with backgrounds in mathematics, software engineering, and computational, computer, social, and cognitive sciences. This mix of experiences enabled discussion that blended technical and human elements, resulting in many ideas for future directions. From these ideas, we have distilled three Priority Research Directions (PRDs) that focus on the fundamental impacts that the many ideas, when aggregated, could have in the future.

In addition to PRDs, we observed three crosscutting themes that appear as patterns across many of the ideas from the workshop. Each of these themes also cohesively involves technical and human elements. While the PRDs describe work that we think is important to accomplish, the complementary themes describe

how to conduct our work. That is, the PRDs are the *what*, and the themes are the *how*.

Table 1.1: Summary of Priority Research Directions and their relative scope. Each PRD represents numerous important questions that merit R&D investments in pursuit of the goal of improving the development and use of scientific software toward improved scientific impact.

Priority Research Directions	**Primary Scope**
PRD1: Develop next-generation tools to enhance developer productivity and software sustainability. Ideas under PRD1 tend to focus on impacting developers by improving their effectiveness and efficiency, reducing the cost and time of development and resulting in products that have higher quality and are easier to maintain.	Individual
PRD2: Develop methodologies and tools to comprehensively improve team-based scientific software development and use. Ideas under PRD2 tend to focus on achieving impact by improving team constitution and interactions of team members.	Team
PRD3: Develop methodologies, tools, and infrastructure for trustworthy software-intensive science. The expected impact of ideas under PRD3 is primarily societal, helping to build confidence and trust in the results that the scientific community produces.	Community

Table 1.2: Summary of crosscutting themes and their primary goals. Each theme represents an overarching approach to conducting our work in scientific software development and use, guiding how we plan and execute our efforts.

Crosscutting Themes	**Primary Goal**
Theme 1: We need to consider both human and technical elements to better understand how to improve the development and use of scientific software. As we explore our PRDs, we benefit from considering both technical and human elements holistically.	Leverage diverse scientific domains
Theme 2: We need to address urgent challenges in workforce recruitment and retention in the computing sciences with growth through expanded diversity, stable career paths, and the creation of a community and culture that attract and retain new generations of scientists. Exploration of our PRDs must keep in mind the continuous need to bring in new community members, retain and advance current members, and support complete career lifecycles.	Promote inclusive culture and establish rewarding and stable career paths
Theme 3: Scientific software has become essential to all areas of science and technology, creating opportunities for expanded partnerships, collaboration, and impact. Because with proper planning, coordination, and collaboration the costs and benefits of scientific software development and use can be shared across communities, we must seek and establish partnerships with other organizations that care about scientific software.	Establish partnerships across communities

Chapter 2

Priority Research Directions

2.1 PRD1 Develop next-generation tools to enhance developer productivity and software sustainability

2.1.1 Introduction

Numerous tools assist developers with their activities, spanning the entire lifecycle of a software system. Many well-established tools in the larger software world are not widely known or used in scientific software development. Furthermore, it can be challenging in some cases to apply available tools to the *scientific* software context. Moreover, there may be opportunities in which tailored tools could be developed that would particularly benefit scientific software development and use.

Leveraging advances in machine learning/artificial intelligence (ML/AI) as well as other approaches, we expect improvements in tools that assist in generating software elements (such as tests, documentation, and clean and readable source code), suggest refactorings, assist with porting, and more. In the foreseeable future, tools may assist developers by generating code based on simple expressions of the scientific programmer's intent, which can then be tuned and refined to create the desired codebase.

The challenges for the scientific software community are to assure that future tools take into account the requirements of scientific software that may not be high priorities in other fields and to address opportunities that may be unique, but critical, to scientific software. Work is also needed to facilitate the adoption of new tools, including effective training and help in incorporating tools into already-complex workflows where near-term needs often reduce the priority of activities that may have a modest immediate impact yet substantial longer-term benefit.

The process of creating code, documentation, tests, and other tangible artifacts is an individual activity. However, scientific software, particularly in the DOE community and similar R&D communities, is typically developed by a team with diverse skills, for multiple users, under the funding and interest of multiple stakeholders, with oversight and review of contributions. There are needs and opportunities for tooling to better support both individual and team aspects of scientific software development and use.

2.1.2 Key Questions

- How can we leverage and build upon emerging AI/ML tools to improve the productivity of scientific software developers?
- How can we create and adapt tools to improve developer effectiveness and efficiency, software sustainability, and support for the continuous evolution of software?
- How can we support and encourage the effective adoption of new tools by developers?

2.1.3 Scientific Challenges and Opportunities

1. **Emerging AI/ML tools and environments will impact how software artifacts are generated and managed, requiring adaptation to the changing economy of how software is developed, used, and maintained.** There is a wide range of opportunities to expand and improve the tools available to scientific software developers through the application of emerging methodologies, such as AI/ML, to provide novel and enhanced capabilities. Tools leveraging expanded generational capabilities can translate high-level expressions of programmer intent into code (fragments) that can be further tuned and refined as needed, or facilitate porting to new platforms. Techniques capable of generating working scientific code might also be adapted to generate related artifacts, such as tests, and certain kinds of documentation. Tools might exploit enhanced capabilities to recognize and categorize patterns to identify code likely to benefit from new computer hardware or find portions of a code base that might be at greater risk in terms of correctness, trustworthiness, security, or maintainability.

2. **Tooling choices require an improved understanding of how scientific software developers make their decisions.** Many types of tools in widespread use in the broader software community are little used by scientific software developers or have been adopted much later and often more slowly. Examples include the regular use of static and dynamic analysis tools, and the slow adoption of "DevOps" approaches such as continuous integration. A better understanding of how scientific software developers make decisions about tooling, along with approaches to accelerate the identification and dissemination of potentially useful tools, would provide benefits for existing tools and help inform the design and dissemination of new tools.

3. **Understanding what the scientific and HPC software communities have in common with other software development settings is required to improve the leverage and impact of relevant software tools.** Tools will be more useful and more likely to be used if they can help address the distinctive aspects of scientific and high-performance computing and also work in this context. We need to better understand the commonalities of mainstream software development contexts with scientific and HPC software communities; where they differ, we need to devise strategies that can facilitate the development of tools that can serve broad audiences yet can be meaningfully tailored to specialty needs at a reasonable level of effort and cost (perhaps based on software product line concepts, for example). Distinctive features of scientific software development include the need to support programming languages and programming models common in this community, the importance of floating-point computation, the rise of multi-precision techniques, and the use of new and emerging computing environments. Distinctive aspects of software teams and the scientific context may also influence how tools are designed and used. Such aspects include the diversity of technical backgrounds and contributions present in many teams, the exploratory and highly iterative nature of scientific research, and the need to ensure the trustworthiness of the scientific results obtained.

4. **Effective and efficient use of novel architectures requires understanding and addressing the unique needs of HPC scientific software tooling.** Among the distinctions of HPC software, the advanced scientific computing community has historically been a leader in the adoption and exploitation of novel computer architectures. New leading-edge machines are deployed on short cycles compared to the lifetimes of many of the software packages used on them. Scientific software developers who want to be able to access the new resources must port and tune their codes, often while the development of new features for the science continues in parallel. Differences from one generation of HPC platforms to the next may be significant, and sometimes multiple very different architectural paths need to be pursued concurrently. And yet, historically, scientific software developers have had little by way of tooling to facilitate the sometimes enormous work entailed in these frequent transitions.

For example, of clear benefit would be tools to identify portions of a codebase amenable (or not) to porting to a new platform, as well as tools to assist systematically with the necessary refactorings of data structures. Tools to aid the assessment of the impacts of different approaches to concurrency, floating-point numerics, and other characteristics of the code for different platforms or algorithms would support exploration, as well as testing and verification of new implementations. More broadly, porting to new platforms is far from the only reason that scientific software changes. As noted, many software products produced in this community are long-lived, meaning that active development of features to support new scientific goals is typically carried out alongside efforts to maintain existing features, reduce the backlog of technical debt [2], and address bugs that might be discovered. Once again, the community would benefit from better availability and use of tools to assist, accelerate, and automate such activities.

5. **The collaborative and team nature of scientific software development and use can benefit from identifying and developing new tooling.** Modern computational science and engineering teams are often composed of specialists in numerous different technical areas, bringing their expertise together to implement the code and carry out research using it. Tooling that maximizes the ability of each to contribute their expertise and experience—supporting separation of concerns while simultaneously facilitating necessary interactions across disciplinary and other boundaries and integration of contributions—can help make such teams more effective as a whole.

2.1.4 Research Thrusts

1. **Strategies for tool selection and adoption in the scientific software community and how to facilitate dissemination**

   (a) **Strategies and processes for leveraging emerging AI/ML capabilities for content development:** Generative AI tools such as ChatGPT [3] from OpenAI, CoPilot [4] from GitHub, and Bard [5] from Google are enabling new content development workflows that are qualitatively more productive than in the past. Numerous early reports cite improvements in code development, programmer satisfaction, and code review [6].

   (b) **Strategies and processes for training and adoption**: Scientific software developers will need training and support to integrate new tools into their development environments.

   (c) **Strategies for incentivizing investment in continual improvement:** The need to focus on near-term deliverables should be balanced with investments to improve future productivity and sustainability.

2. **Approaches to design or adapt tools to address particular needs of the scientific software community, while maintaining sufficient generality that much of the long-term development, support, and evolution can be driven by a broader community**

   (a) **Strategies and tools to adopt software product line approaches in scientific software, including in the tools themselves:** Adopting a product line approach means that likely changes (variabilities) will be incorporated into the process and tools so that when the inevitable need for solution space exploration occurs, the code, documentation, and test cases can be kept up to date [7, 8].

   (b) **Strategies to assure scientific software requirements are considered in community tools:** To keep general-purpose tools relevant for scientific software, we want to ensure our requirements are considered in the design and implementation of these tools.

3. **Novel capabilities and tools to support scientific software development and use for individual activities**

   (a) **Creation and adaptation of new tools for improving developer effectiveness and efficiency, and software sustainability:** The broader software community will provide a powerful collection of new programming tools. The scientific software community must participate in the creation process to ensure our requirements are addressed. We also need to customize new tool platforms to address our particular needs.

   (b) **Support for the continuous evolution of software:** Many scientific software products are continuously developed. Developer tools need to be designed to support progressive refinement and augmentation of source code, documentation, and related artifacts. Refinement may take many forms, including moving from an initial "quick and dirty" experiment or prototype toward more robust, maintainable "production" implementations; the addition of new capabilities, performance improvements, and many other goals. Different parts of a large code base may be undergoing different types of evolution simultaneously.

   (c) **Facilitating porting scientific software to new computing environments:** A hallmark of advanced scientific computing, particularly in the DOE community, is the exploration and production use of computational platforms that are at the leading edge of what's available, sometimes including prototypical or experimental (i.e., not commercially available) systems. Tools that can assist in the process of porting code to new platforms have the potential to accelerate the ability to exploit new platforms for both research and production science. Relevant capabilities include identifying blocks of code that are likely to be able to benefit from distinctive features of the platform, as well as advising on or even carrying out transformations of the code to the target platform. To be most useful, it should be relatively quick to generate tooling to support new hardware platforms—if months or years are needed to create a tool in the first place, many developers will proceed with their porting efforts without it.

4. **Tools to help improve quality and robustness and to help ensure the correctness of scientific software**

   (a) **Tools to generate tests and advise on testing strategies:** Emerging tools and integrated development environments support developers in creating and maintaining testing capabilities. Adapting and adopting these tools for scientific projects is important, especially for large, complex code bases.

   (b) **Tools for use during software development:** Emerging tools and integrated development environments provide increasing support for static testing and code generation. Adapting and adopting these tools to generate correct code and identify likely software defects and unintended limitations is important, especially at the earliest development stages.

   (c) **Advanced tools for testing floating-point numerics:** Floating-point data and algorithms are prevalent in scientific software, and errors can be hard to detect. We need tools that support developers in creating correct code and finding errors. Also important are features that can be quickly modified to support new and emerging hardware platforms.

5. **Novel capabilities and tools to support scientific software development and use "at scale" and in the large (teams, composite software, etc.)**

   (a) **Tools to predict risk and complexity and evaluate code quality as code evolves:** Automated scores can be produced to estimate the complexity of code changes, including traditional metrics

like code points, cyclomatic complexity, and the number of library dependency interactions. More advanced tools from natural language processing could be applied to produce high-level structural analysis that simply scores the depth of code changes. Such scoring methods, while ultimately opaque, could be used internally by large scientific teams to monitor the health of a rapidly evolving codebase with a large developer community, and could become an automated part of the CI process [9].

(b) **Tools to facilitate the development, implementation, and maintenance of high-level domain-specific abstractions:** While libraries and custom languages have a long history as tools for implementing domain-specific abstractions, they are largely one-off efforts that are often not easily (re)used outside of the project or group initially commissioning them. The developers, maintainers, and users of such tools would benefit from strategies for making domain-specific abstractions available in ways that are more easily extended to neighboring domains, composed, and otherwise more broadly reused [10].

(c) **High-level programming languages and domain-specific languages and libraries (DSLs):** DSLs can help with rapid development for various domains [11, 12, 13]. DSLs can also help to separate the concerns of different members of multi-disciplinary project teams, particularly domain specialists (who use the DSLs to express their computations), computer scientists (who design and implement the tools that map the DSLs onto the target computer hardware), and performance engineers (who work between the two to ensure that the results are as performant as possible across the various target platforms).

(d) **Tools to support the use of approaches such as model-based systems engineering (MBSE) and domain-specific modeling (DSM) in scientific software:** These approaches have been recognized in other communities as providing benefits, particularly when dealing with large complex software systems produced by large teams through the development and use of conceptual domain models to facilitate capturing and use of domain knowledge throughout the software lifecycle [14, 15]. Identifying, adapting, and creating tools that meet the needs of scientific software teams to support the creation and maintenance of models and generation of artifacts, such as code, test cases, documentation, etc. from them, would allow our community to also take advantage of such approaches.

2.2 PRD2: Develop methodologies and tools to comprehensively improve team-based scientific software development and use

2.2.1 Introduction

As fundamental understanding of scientific software improves, we foresee that the methodologies and tools we need will also change, better matching and supporting how developers, users, and other stakeholders of scientific software work toward the goal of accelerating scientific discovery. While many scientists have extensive intuition, derived from experience, about the principles and dynamics of how their particular community develops, uses, and sustains its software products, research is needed to develop a deeper and broader understanding of software's role in scientific processes, as well as factors that contribute to successful software teams and user communities. We believe that developing common mental models around fundamentals can help the scientific community as a whole observe, understand, and improve the effectiveness and efficiency of scientific discovery through improved stakeholder expectations, leading to better methodologies and tools for the development, use, and sustainment of scientific software, particularly in the context of diverse, collaborative teams.

2.2.2 Key Questions

- What are the roles of software in scientific processes?
- What are key roles on software teams?
- What factors contribute to the effectiveness of scientific software teams?
- What practices, processes, and tools can help improve the development, sustainment, evolution, and use of scientific software by teams?

2.2.3 Scientific Challenges and Opportunities

1. **The roles of software in scientific processes require deeper understanding and characterization.** While many scientists have extensive intuition, derived from experience, about the principles and dynamics of how their particular community develops, uses, and sustains its software products, research is needed to more clearly understand software's role in scientific processes. We believe that developing common mental models around fundamentals can help the scientific community as a whole observe, understand, and improve the effectiveness and efficiency of scientific discovery through better methodologies and tools for the development, use, and sustainment of scientific software, particularly in the context of diverse, collaborative teams.

2. **Critical roles in scientific software teams require identification and characterization.** Scientific software teams are composed of more than just developers and users. Depending on the size of the team and user community, there are functions such as integration, user support, and regression test monitoring that may need distinct staffing beyond the team members who are designing and developing features. For larger teams, there may even be roles for team building, raising community awareness of the software, and making sure that stakeholders are aware of the latest impacts and plans. For products that push the boundaries of high performance, roles focused on architecture trends and exploration of algorithms and programming environments may be essential. Awareness of potential team roles is important for staffing and establishing realistic expectations from stakeholders.

3. **The attributes of sustainable scientific software require deeper characterization.** A key concern about software is its sustainability. Many definitions exist [16, 17, 18]. However, there is no broadly recognized taxonomy. Existing definitions tend to be specific to a particular subset of the scientific software community, represent a subset of the broad elements needed by the community, and are not widely adopted by the scientific software community. 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 processes for development and support, and other factors that influence the sustainability of software products [19]?

4. **Factors that contribute to the success of scientific software development teams and user communities need to be identified.** A prerequisite for motivating interventions (methodologies, policies, and tools) and assessing their impact on scientific teams and software is knowing the factors that contribute to success. Understanding the dynamics of successful scientific software teams—and collaborations among multiple software teams—can help the community as we progress toward ever more challenging endeavors in computational science [20, 21, 22, 23, 24, 25]. What processes, methods, and practices contribute to the productivity of individual team members and teams overall? What factors impact how well a scientific software team functions and the quality and sustainability of its tools and products? What are mechanisms for collaboration among multiple software teams (whose products

need to be used in combination for next-generation science), and what factors impact how effectively software teams collaborate? Are there anti-patterns and other factors that contribute to failure?

2.2.4 Research Thrusts

1. **Roles of software in scientific processes**

   (a) **Refine our software definitions:** We often speak simply of "software" or "code", but these terms have different meanings for different members of the teams that develop and use software and code. For some, a software product may be a tool used to carry out their scientific research objectives, but for others, the same software may be a product of their research that they are sharing with others in the hope it will be useful.

   (b) **Expand our understanding of external dependencies and their value and risks:** For some, external dependencies are a useful way of leveraging other people's expertise and capabilities, whereas others consider dependencies a necessary evil, and still others strongly prefer to produce their own implementations, even when knowing that these duplicate the functionality of existing software.

   (c) **Understand the user-dependent roles of software:** Having a better understanding of the roles that software plays in the scientific processes of scientific user community members will help inform how we engage with community members. Moreover, this understanding will help to identify the need for advanced functionality in software tools, including AI, to help software more effectively serve these critical roles.

2. **Critical team member roles in team-based scientific software development**

   (a) **Clearly define and characterize the fundamental roles of members of scientific software teams:** Scientific software teams require the combined contributions of people with a variety of diverse backgrounds and skills, often including domain scientists, computer scientists, applied mathematicians, and Research Software Engineers (RSEs), whose expertise in both software engineering and research help to bridge across disciplines to produce high-quality software. We need to characterize the fundamental roles of team members across the spectrum of work needed for scientific software. Beyond classical software team roles, the role of RSEs has received increased attention in recent years [26, 27, 28] as has the potential for including social and cognitive scientists [29]. Even so, characterizing the nature and roles of scientific software developers and teams is still incomplete and worth further study.

   (b) **Understand and anticipate emerging and future roles:** Next-generation scientific challenges will require the contributions of a wide range of developers and users. We need to identify and characterize the various roles on scientific software teams that will help teams expand to function in software ecosystems that facilitate cross-project collaboration. We should explore roles such as project coordinators, who help software teams to plan work and handle the logistics of coordination, and we should investigate how experts in social and cognitive science can help facilitate collaborations, especially across aggregate teams. Assuming that different categories of developers and users likely have different needs, the various categories of developers and users should also be defined, including the distinctions between small groups and large teams (and teams of teams). Awareness of potential team roles is important for staffing and establishing realistic expectations from stakeholders. This understanding can help to identify opportunities attuned to the needs of various team roles for software tools, including AI-based functionality.

3. **Attributes of successful and sustainable scientific software**

   (a) **Explore and characterize the role of non-functional requirements:** The software engineering community often uses the term “non-functional requirements” or “-ilities” to characterize software. The many "-ilities" (e.g., extensibility, interoperability, and portability, to name but a few) provide many different dimensions in which software can be assessed, helping to think more rigorously about it.

   (b) **Explore how to leverage non-functional requirements to improve software:** Applying this kind of analysis to scientific software can help to identify the relative importance of different non-functional requirements ("-ilities") in different situations, and how these attributes can contribute to a software package being successful in meeting its (scientific) goals and being sustainable over time. Such analysis can help to identify opportunities for further research in software methodologies and tools, including AI, to help improve software quality and sustainability.

4. **Team-based factors that contribute to the success of scientific software development teams and user communities, as well as inefficiencies and gaps that would benefit from new research on methodologies and tools, including AI-based functionality**

   (a) **Explore and characterize the fundamentals of team communication and coordination across development, testing, and deployment activities:** Collaboration on scientific software development brings the need to communicate among team members about software design and feature plans, orchestrate development activities, as well as onboard and offboard team members. A variety of general-purpose collaborative software development tools have evolved, ranging from communication channels and issue trackers over software repositories to frameworks that support Continuous Integration (CI), testing, and Continuous Deployment (CD).

   (b) **Understand and improve requirements, analysis, and design capabilities in scientific software product development:** Aside from tools for the technical realization of the software development process, software development methodologies have emerged that try to orchestrate the software design and development cycle in a standardized process. Also, identifying where the need for improvements in both developing and using scientific software will motivate process improvements, the creation of new approaches, and the development of new tools, including AI-based functionality.

   (c) **Understand the inefficiencies in scientific software processes and identify gaps where research is needed on methodologies, practices, and tools:** While virtually all larger scientific software projects follow a software development methodology and make use of tools for collaborative software development, it can be difficult for a project to assess how the development process could be improved beyond the current approach. Such understanding is difficult not only because the internal view of project participants typically inhibits a broad view of the development process, but also because there is insufficient research on how to assess the effectiveness of a software development process and how to start an improvement process, which depends on the specific software project, development methodology, and scientific community, among others [30, 31, 32].

   (d) **Understand the roles and impact of policies in team-based scientific software:** An important area focuses on community policies regarding style, design, and behavior (for a particular software development group and collections of complementary software products). Likewise, a code of conduct for software teams documents expectations and standards of ethical behavior and explicitly states how developers should treat one another.

5. **Scientific developer and user community expansion**

   (a) **Expand the user base:** The effective use of HPC software technologies, even for people whose primary focus is domain-specific science and engineering, tends to require a 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.

   (b) **Purposefully identify and engage with the broader community:** Often, a scientific software product is developed by a small team of developers and used by a small group of users. However, the software may be used by a much larger community, and the community may have a variety of needs that are not being met by the software. We need to identify and characterize the various communities that use scientific software, and explore how to engage with them to better understand their needs and how to meet them.

2.3 PRD3 Develop methodologies, tools, and infrastructure for trustworthy software-intensive science

2.3.1 Introduction

The trustworthiness of results is the foundation of scientific progress. Trust in scientific results most readily comes from transparency in how results were obtained, repeatability and reproducibility of obtaining the results by the author and an independent party, respectively, using the same experimental setup, and replicability of results by obtaining a consistent answer using a different experimental approach [33, 34].

Most scientific results incorporate computational tools and results as part of the overall scientific process. Since overall trustworthiness inherently depends on the trustworthiness of each step in the scientific process, it is apparent that our computational results must be trustworthy. Key concepts that must be addressed are transparency, reproducibility, replicability, and uncertainty quantification.

While many challenges inhibit trustworthiness, some of the most common are software errors, unmanaged changes in software and data environments, stochasticity in the computational approach, execution on boutique systems that are not widely accessible, such as leadership and prototype systems, and inaccuracies in the computational models.

These challenges are further exacerbated by an incomplete understanding of the roles and responsibilities of trusted scientific results in the community, especially as we work to improve the trustworthiness of our results. Broad community engagement is required as we move toward rewarding trustworthy results more than producing a high volume of papers, increasing publisher expectations of transparency and reproducibility in published results, and increasing funding agency activities that assess the quality of results from their funded projects. Understanding the roles and responsibilities of trustworthy scientific results is a prerequisite to developing a holistic strategy for improvement [35].

2.3.2 Key Questions

- How can we build a community-wide understanding of the roles and responsibilities for trustworthy scientific results?
- How can we make trustworthy computational results the norm for scientific research?

- How can we create holistic computational systems to support trustworthy computational results where we can assure the software we develop and the software we use from other sources can be trusted and upgraded to address new risks?

2.3.3 Scientific Challenges and Opportunities

1. **Achieving the common goal of trustworthy scientific results requires the scientific research community—from scientists to administrators to publishers and funding agencies—to understand and act upon their roles and responsibilities in pursuit of this goal.** The economy around funding, producing, and evaluating scientific results involves many people. A particular scientific community will generally value a similar level of quality and investment in generating results. Low quality or underinvestment will generally lead to results that are unacceptable to stakeholders. Similarly, investing a lot of time and effort, relative to peers, could disadvantage a team by reducing the volume of scientific output even though results are more easily reproduced and more trustworthy. If a community's reward system (rank, tenure, promotion, etc.) does not acknowledge the value of these increased efforts, investments in improved trustworthiness will be deemed too risky, or even undesirable. To make community-wide advances, the roles and responsibilities of its members must be defined, understood, and realized so that all members can coordinate activities toward improved trustworthiness. Further, stakeholders must agree on expectations within the community as to what constitutes trustworthiness.

2. **Better and more widely available practices, processes, and tools are required to support trustworthy computations.** We believe the most practical approach to improving the trustworthiness of computational results is to make provenance capture and replay as automatic and portable as possible. Most computational science teams are not in a position to easily realize this approach. Some impediments can be addressed by better practices using existing platforms, e.g., Jupyter notebooks. Others can be addressed by designing and adopting new infrastructure tools and platforms that enable the capture and replay of results computation. The remaining impediments appear to be more difficult to overcome, such as the capture of high-volume data and replay on boutique systems like supercomputers and experimental hardware/software environments.

3. **Scientific computing environments are needed to seamlessly support the capture and retention of sufficient information for transparent, repeatable, and reproducible results.** Building on top of practices, processes, and tools in the previous item, we need to provide environments that enable end-to-end capture and versioning of the data, software, and computational steps that produce each computational result. In recent years, the emergence of Jupyter notebooks, containers (e.g., Docker), and domain-specific end-to-end environments such as Weights & Biases have qualitatively improved the ability of some scientific communities to achieve practical and portable transparency, repeatability, and reproducibility. At the same time, many other scientific computing environments do not provide the same assurances without significant investment from each computational science team. Teams who develop their own computational software are generally left to their own *ad hoc* approaches to assure the software they write and use from other sources is trustworthy. Finally, advanced computational science experiments are becoming even more complicated as workflows spread across multiple facilities in a kind of internet of workflows.

2.3.4 Research Thrusts

1. **Improved understanding of trustworthy computational results and the ability to effectively elevate the trustworthiness of our results within the computational science community**

(a) **Develop a community-wide understanding of trustworthiness concepts in the context of computational science:** To advance our abilities to deliver trustworthy scientific results, we need to establish common concepts, terms, processes, and goals for improving trustworthiness [36, 37, 38].

(b) **Establish an understanding of the roles and responsibilities for trustworthy computational results:** Improving the trustworthiness of computational results requires improvements and investments across the scientific enterprise. Realizing improvement will, at least initially, increase the cost of producing results and slow down the rate of generating them. The increase in trustworthiness may not be appreciated immediately since the impact of incorrect results is felt strongest when experienced downstream by future activities that relied on correct results. These initial investments will require commitment from the community to help build momentum.

(c) **Establish training materials and opportunities to learn about the fundamentals of trustworthy science:** Concepts, terminology, and objectives for trustworthy science are not uniformly understood or described across scientific domains. Creating flexible training content that could be inserted into existing computational courses and establishing training opportunities and expectations, especially for new community members, are essential to advancing trustworthiness and are in fact prerequisites for many other activities.

2. **New concepts, practices, processes, and tools to support improved trustworthiness, including leverage of advances from other software domains where trustworthiness approaches are more advanced**

   In addition to understanding the basics of trustworthy computations and establishing architectures and platforms for trustworthy computations, we need to cultivate new ideas and draw upon ideas from other communities. Some application domains such as real-time safety-critical systems have already cultivated advanced approaches to understanding and better assuring trustworthy results [39, 40]. Ideas that could be translated include:

   (a) **Develop assurance case templates for scientific software, where an assurance case provides an organized and explicit argument for trustworthiness:** Assurance cases are already effectively used for real-time safety-critical systems [41].

   (b) **Create requirements documentation templates and tools:** Trustworthiness can truly be judged only against an unambiguous statement of the assumptions, goals, scope, and functional and non-functional requirements [42].

   (c) **Establish methodologies and tools for metamorphic testing:** In scientific software, metamorphic relations come from physics or known mathematical properties. Examples include the relationship that the output flow rate of an incompressible fluid increases as the input flow rate increases and properties of symmetry that can be statistically tested with high confidence.

   (d) **Explore how to best capture the rationale for requirements and design:** Current documentation often focuses on *what* and *how* but less on *why*. When change occurs in the future, the rationale is vital information for judging the appropriateness and feasibility of the change.

3. **Computational environments and platforms to support trustworthy computations and workflows**

   One of the easiest ways to advance transparency and reproducibility is to establish computational environments where provenance capture and replay are built into the system. Jupyter notebooks enable this approach to some extent, but more work is needed within and across computational environments.

Significant success requires automatic provenance capture, where modifications are automatically integrated into software and data repositories for the user (think of the history that the undo feature keeps in a word processor) and replay is readily available (think of the redo functionality in a word processor). Some specific examples include:

(a) Develop widely deployable provenance capture methods that allow computational results to be mapped back to the exact code versions and inputs that were used to produce them.

(b) Establish methods for reducing the ambiguity present in developed code, e.g., use of units of measure and type annotations to make the intention of the code explicit, especially in dynamically typed languages [43].

(c) Embed diagnostic capabilities into computational environments to support improved detection and correction of errors.

2.4 Scientific and Technology Impacts

As we pursue these PRDs, we foresee the following impacts.

1. **More effective and efficient software efforts:** New tools and methodologies will enable scientists to more effectively and efficiently develop and use scientific software. These tools and methodologies will be designed to improve the productivity of software developers and to improve the sustainability of the software they develop. Of particular importance will be the integration of emerging AI/ML tools into our software tools and workflows.

2. **Improved software team interactions:** Software teams will realize important improvements in their interactions with each other and with their users. These improvements will be enabled by new tools and methodologies that will help teams to improve team performance and better understand and address the needs of their users.

3. **Improved interactions across teams:** New tools and methodologies will enable teams to better understand and address the needs of other teams and realize important improvements in their interactions with these teams.

4. **Improved community interactions:** New tools and methodologies will enable broad engagement with the scientific community, leading to a more effective and efficient impact of our efforts to advance scientific discovery.

5. **A robust ecosystem and community that will improve software development and use**: Better environments for developers and users will accelerate software development, reduce cost, and lead to better products, including applications and software technologies. Furthermore, we expect that the entry of individuals into the scientific software community will be easier so that teams will have improved experiences and morale, while also expanding the breadth of people who can contribute.

6. **Increased reuse of results and computational environments**: Scientific progress will accelerate as the community is better able to leverage and build upon their own results and the results of others. Transparent generation of repeatable and reproducible results will enable a team to more easily build on its own results in the future and enable other teams to leverage their work. Reproducibility will also enable better replicability, extensibility, and other leveraged activities that will further accelerate scientific progress.

7. **Increased trust in scientific results**: The first and most important impact of elevating the priority of trustworthy scientific results will be increased trust in scientific results. We should expect fewer errors in results, greater confidence in our analyses, and better certainty in our conclusions. We should anticipate fewer retractions in our publications and greater trust from those who rely on our answers to inform their own work, decisions, and policies.

8. **Accelerated scientific discovery**: Establishing computational environments and workflows that have increased trustworthiness will initially increase the cost of generating results and lower the apparent output of scientific teams. However, over time costs should decrease and output should increase, ultimately leading to faster time to results at a reduced level of effort. This will lead to faster scientific discovery and more rapid progress in our understanding of the world.

Chapter 3

Crosscutting Themes

3.1 Theme 1: We need to consider both human and technical elements to better understand how to improve the development and use of scientific software.

3.1.1 Introduction

Concern for people involved with a project is always part of how we develop and use software for scientific research. At the same time, we seldom address this concern objectively or explicitly inform our decisions with scientific knowledge from domains outside of the specific developer and user communities creating and using the software. For example, we seldom leverage theory, methods, or deep knowledge about our concerns from cognitive, social, and economic sciences, or subdisciplines in software engineering such as user experience. An important opportunity to accelerate progress in improving scientific software development and use is to routinely incorporate knowledge from these domains and others into our activities going forward, especially through collaboration with domain experts who focus on human elements as part of their research.

3.1.2 Opportunities

Expressly considering human elements in scientific software development and use will lead to better and more sustainable products. Leveraging and adapting knowledge from cognitive, social, economic, and related science fields will inform and improve *how* we conduct our work on the way to accomplishing our primary goals of producing high-quality scientific software products and using those products toward scientific discovery [44, 45, 46].

Individual Scope Opportunities. Awareness of how a person conducts work and makes decisions can be informed by knowledge from human factors, user experience, and cognitive sciences. In scientific software, we can benefit from a focus on both users and developers, taking into account that many people are themselves both users and developers. Some particular opportunities for scientific software include the following.

- **User experience:** User Experience (UX) activities often focus on the use of a specific product and improving its usability in isolation. Scientific software is often used within a larger context of a scientific workflow. Taking into account this broader context is important for the overall effectiveness of development and use. Other techniques such as using a formal approach to defining user personas (major types of users) and journey stories (how each type of user will engage with the software environment)

as part of a larger workflow can have a positive impact on how scientific software is developed and used.

- **Developer experience:** The software development landscape has changed dramatically in the past decade. The availability of collaborative platforms such as GitHub and GitLab is one example. We foresee that software development approaches will continue to change in qualitative ways. For example, the advent of code generation tools such as GitHub CoPilot will change the role of the human developer from one who types in all the programming statements to being the overseer of how the code is generated and tested for correctness. Helping scientific software developers learn about and incorporate these new tools and workflows into their activities will be essential and can be informed by techniques cultivated in the cognitive and social sciences.

Team Scope Opportunities: Social sciences can inform a variety of important challenges and opportunities in team-based scientific software development and use. Scientific software is developed and used within a highly competitive environment where success ultimately hinges on the novelty and impact of scientific results. Competition and collaboration are often intertwined in complex ways. Improving communication and interactions within and across teams can account for specific challenges in scientific software environments.

- **Interaction within a team:** Communication science is at the heart of improved interaction within a team—dynamics can be articulated with fine granularity to identify sources of interpersonal conflict and misperceptions, as well as best practices for small and large software teams that can be shared among teams within a project or program.

- **Team of teams:** A team of teams implies a degree of coordination and hierarchy that enables multiple teams to be aware of, respond to, and inform other teams performing complementary or similar work [47]. Team science is interdisciplinary, touching on many topics in social and behavioral science, including cognition, organizational communication, sociology, information management systems, and others [48]. Team of teams concerns primarily large organizations of aggregate teams and scaling the agility and best practices of small teams as they grow.

- **Cross-team communication:** Scientific progress for a given scientific team is often informed by results from independent teams working across multiple domains and other teams working within the same domain. Intercultural communication is among the social sciences specializing in the enhancement of interactions across teams, especially those who perceive themselves to be culturally different from others. The science of intercultural communication can be particularly useful in the articulation of strategic messaging, motivating incentives, and unifying constituents with competing goals and objectives.

Community Scope Opportunities. The potential impact of scientific results and the roles that software development and use play in producing results mean that the overt engagement of a software team with its users, sponsors, and the broader community can and should be part of team activities. Cognitive and social sciences can make these engagements more effective and efficient.

- **User community engagement:** Cognitive and social sciences can help identify the needs and scientific software requirements of specific user communities.

- **Sponsor community engagement:** Economic, cognitive, and social sciences can assist informed decision-making for scientific software projects and programs unique to sponsor communities. Additional collaborators include data scientists, modeling, and data mining experts.

- **Societal and international community engagement:** Communication science in particular is uniquely positioned to assist with strategic messaging, culture change, diffusion of innovations, and propagating lessons learned and best practices of scientific software to all collaborators in open science.

Broad Scope Opportunities. Among many concerns that can be informed by cognitive and social sciences, improving incentives, productivity, and adoption of new ideas is very attractive to consider using theory, methods, and lessons learned from cognitive, social, economic, and other scientific communities:

- **Incentives systems:** Individuals and teams define and prioritize activities and outcomes based in part on perceived incentives, explicitly stated or not. Explicitly considering people's varying incentives and how these impact the overall prioritization of the project's activities and goals over time can have many positive impacts. One common challenge on teams comes from students and postdoctoral staff who may be incentivized to produce demonstration software implementations to achieve results and move on in their careers, with minimal investment in sustainability or usability. In contrast, a better approach for the project overall might be a more methodical, robust design and implementation. Social and economic sciences have amassed a large body of studies exploring the science of incentives.
- **Productivity:** Intentional development workflows and processes are required to accelerate discovery, and sustainability and reproducibility must not be compromised or ignored. User experience practitioners commonly use a variety of methods drawing on heuristics and lessons learned from human-computer interaction, cognitive science, and human factors.
- **Culture:** Many of the important changes that enable qualitative improvement for any community require changing community and team culture, as well as the priorities of individuals. Methodologies and tools that have been developed to understand and improve user experience, the organization of software teams and the business models for software have generally been created for and applied to domains that are not focused on scientific software. Translating, adapting, and adopting existing approaches, as well as creating new approaches, will be required. Communication science and information systems management have successfully applied models of technology acceptance and the diffusion of innovations through communities and other social systems.

3.1.3 Connections to PRDs

PRD1 - Tools for productivity and sustainability

- Individual productivity improvements are strongly connected to a deep understanding of how to improve the effectiveness and efficiency of contributors.
- The overall satisfaction of scientific software developers and users benefits from an informed perspective of what makes individuals happy about work.
- Decisions about tool evaluation, adoption, and use are strongly dependent on the human element. Social science research techniques will be invaluable in improving our understanding.

PRD2 - Team-based scientific software

- Team-based activities for developers and users of scientific software must be informed by an understanding of how teams interact, what social and cognitive factors are most important for successful teams, and how to cultivate improvement in team-based activities.
- Tool design and adoption to support software team activities must be informed by human factors that represent the biggest challenges and opportunities for improvement.

PRD3 - Trustworthy science

- Trustworthy science requires building community awareness of the roles, responsibilities, incentive structures, and more, that lead to increased prioritization of trust-building activities toward obtaining computational results.
- Improving the trustworthiness of computational results requires understanding and changing the culture of the scientific enterprise as a whole—including funding agencies, publishers, and academic, lab, and industry leadership—toward a greater expectation of trustworthy results.

3.2 Theme 2: We need to address urgent challenges in workforce recruitment and retention in the computing sciences with growth through expanded diversity, stable career paths, and the creation of a community and culture that attract and retain new generations of scientists.

3.2.1 Introduction

The DOE national laboratories, like many other scientific research organizations, face growing needs and challenges in recruiting and retaining a skilled workforce. Government and academic sectors face fierce competition for talent attracted to lucrative industrial workplace benefits. In addition, numerous studies have shown that diverse organizations, teams, and communities perform more creatively and effectively—and thus are demonstrably more innovative and productive [49, 50, 51]. However, many existing computing sciences workplaces contain fairly small percentages of women and even smaller minority populations. The 2014 DOE Advanced Scientific Computing Advisory Committee (ASCAC) Workforce Subcommittee Letter states: "All large DOE national laboratories face workforce recruitment and retention challenges in the fields within Computing Sciences that are relevant to their mission. ... Future projections indicate an increasing workforce gap and a continued underrepresentation of minorities and females in the workforce unless there is an intervention." [52]

3.2.2 Opportunities

We thus must tackle urgent workforce challenges to be able to address next-generation scientific opportunities. Key workforce challenges and opportunities focus on understanding and disaggregating demographic data, devising effective inclusivity interventions, expanding HPC training and outreach, improving equitability in access and outreach, expanding inclusivity of technical approaches, and ensuring sustainable career paths.

Demographic Opportunities. Understanding what is needed and what we can reasonably achieve in terms of workforce diversity requires data on current workforce demographics and the pipeline from which we can attract new people to computing sciences. The National Laboratories Directors' Council provides data that can help in understanding the current workforce at a level of granularity that is coarse but useful. [53]

- **Improve retention of junior staff:** Demographic data indicate that the overall DOE lab graduate student population includes 32% women and 19% under-represented minorities (URM), where the latter category includes African American/Black, Hispanic/Latino, and American Indian/Alaskan Native people. The DOE laboratory postdoctoral population (term starting positions) includes approximately 26% women and 7.7% under-represented minorities, while the DOE laboratory technical research

staff includes approximately 20% women and 13% under-represented minorities. Given the importance of postdoctoral and graduate student populations as feeders to the technical research staff talent pool, we need to retain these junior community members as they progress in their careers and further increase representation going forward.

- **Improve understanding through disaggregating data:** We need to understand how similar or different the numbers for *computing sciences* are relative to the overall technical research staff. That is, we need to determine whether data aggregation is hiding information that might be better understood through disaggregation (the separation of information into smaller units to elucidate underlying trends and patterns). In April 2022, the Biden-Harris administration released recommendations for advancing the use of equitable data by disaggregating survey data to better characterize the experiences of historically underserved groups. [54] This suggests that in addition to disaggregating the aforementioned demographics by technical discipline so that we can focus on computing, there would also be a benefit in disaggregating groups such as URMs by race and ethnicity.

Workforce Recruitment. Many potential contributors to the scientific community never join because they do not have sufficient engagement with other scientists or enough awareness of their potential. Reaching out to future scientists is essential as we go forward.

- **Devise effective inclusivity interventions:** Many successful recruitment models exist; however, they often rely on existing social networks and, to date, have largely resulted in a workforce with an underrepresentation of various minorities. The challenge is not only to develop new approaches to broaden the reach, but also to change longstanding recruitment, onboarding, and retention practices. We need approaches that foster a sense of belonging to members of underrepresented groups while also respecting cultural frames of reference. [1, 55]

- **Improve equitability in access and outreach:** Diversifying the workforce that develops scientific software also requires ensuring accessible formats for documents and inclusive conduct of scientific meetings. Textbooks, articles, and programming language standards need to be provided in a variety of formats that are accessible to those with vision impairments or learning differences. Meetings should be conducted in accessible and inclusive formats. Likewise, connecting with new and future community members, and sustaining those connections, requires diverse approaches to assure we reach people in a way that is effective for them, including first-generation scholars (students who are the first in their families to attend college). A person whose family or broader community has provided examples of scientific careers may naturally see a pathway to a scientific career. On the other hand, another person without those examples may need different approaches or adaptations for the message to be received and seen as intended for them.

- **Expand inclusivity of technical approaches:** Any thoughtful effort to foster an inclusive work community must address the diversity of thought. A diverse workforce does work differently and does different work, finding interest in a more diverse collection of scientific problems. Allocating resources for a diversity of technical approaches helps to achieve equity. A more inclusive workforce will lead to different problems addressed in addition to different solutions discovered.

- **Expand HPC training and outreach:** Essential tools to help broaden participation of underrepresented groups in the computing sciences are a rich array of outreach and training materials that convey computational science impact and opportunities, targeting especially undergraduate and graduate students yet also those in middle and high school. Challenges include framing educational materials in a way that engages people from a wide range of needed backgrounds (including not only traditional lower-level computer science perspectives but also application-oriented perspectives). We need

to re-examine curricular materials in relation to what is needed for the high-performance scientific computing workforce, identify gaps, and pursue opportunities for expanding course offerings and providing training resources for people new to scientific computing.

Workforce Retention. Many opportunities exist to attract the next generation of scientists, especially among underrepresented communities. Once part of the community, opportunities to continue investing in staff are equally important.

- **Create sustainable career paths:** Due to traditional metrics for career advancement focusing on publications rather than software contributions, people who focus on the important work of software development often face challenges in professional recognition and career advancement. Community organizations [56] are working for change—advancing understanding of the importance of high-quality software in multidisciplinary collaboration and the integrity of computational research, and articulating key issues and needs to stakeholders, agencies, and the broader research community to effect changes in policies, funding, metrics, and reward structure. For example, Research Software Engineering (RSE, https://society-rse.org and https://us-rse.org) has emerged as an increasingly recognizable career track, helping to build community, mentoring, and professional recognition for scientific software specialists. Work is needed on a variety of fronts.

- **Understand and improve diversity:** We need to understand how diversity in roles connects with diversity, equity, and inclusion issues in hiring and retention. Are there particular roles where the workforce is more or less diverse, and if so, what are the underlying causes? Can we leverage the emergence of new or more defined roles to create additional recruiting pipelines to reach out to groups or institutions that are currently underrepresented in the field?

- **Understand and support important scientific community roles:** We need to better understand how institutions, research groups, funding agencies, etc. support (or fail to support) the career needs of scientific software professionals in different roles. Are reward structures and career paths aligned with their needs and professional goals? We need to better understand recruitment and retention challenges across the full range of scientific software roles. Are certain roles particularly challenging to fill due to recruitment and retention problems? Are there needs for new recruiting pipelines, training programs, recruitment strategies, salary structures, etc.?

- **Expand definitions of research impact for more accurate assessment:** We need to address practices of research assessment. Widely used assessment practices (such as the number of publications and the amount of funding generated) do not adequately represent the work of the majority of RSEs and similar roles. We need to understand how to measure the work of these roles if we are to demonstrate their value, and the measurements need to persuade the research community that these roles make significant contributions to their research.

3.2.3 Connections to PRDs

PRD1 - Tools for productivity and sustainability

- Building effective tools for productivity and sustainability depends on people who have diverse skills and experience encompassing the entire software lifecycle and an ability to look at scientific software practices from holistic perspectives. This includes computer scientists, research software engineers, applied mathematicians, and domain scientists, as well as contributors with other backgrounds, including those with social/behavioral science training and human-computer interaction expertise.

PRD2 - Team-based scientific software

- Building effective scientific software development teams requires a diverse and well-qualified workforce who can fill the many different roles required for team success.

PRD3 - Trustworthy science

- Trustworthy science requires a highly qualified and diverse scientific software community.

3.3 Theme 3: Scientific software has become essential to all areas of science and technology, creating opportunities for expanded partnerships, collaboration, and impact.

3.3.1 Introduction

Powerful and affordable computing has revolutionized the scientific enterprise. Furthermore, computing provides the basis of many technological and engineering products and increasingly informs policy and other consequential decisions. Due to the broad importance of scientific software, efforts to better understand and improve its development, role, and use should find allies beyond DOE, including practitioners in academia and industry, as well as other government and nonprofit agencies that fund software- and computationally-intensive research. This includes studying a software product itself, the projects that support and depend on it, and the people who do the work to develop and use it, in the context of their projects, organizations, and careers. These types of studies involve several different disciplines, including computer science, management, organizational studies, business, sociology, anthropology, and economics, which are outside the typical experience of most computational and data scientists.

In the early days of computing, scientific software was the major portion of all software developed. In recent decades, software products have become omnipresent in our lives, well beyond scientific pursuits, with scientific software now being a small fraction of overall software. On the other hand, the intersection of interests between mainstream and scientific software has greatly increased, bringing both common and conflicting interests. Where there is commonality, it increases the ability of scientific software to adopt and easily adapt practices and tools from mainstream software development. Among the most salient conflicting interests at present is the competition for skilled software people.

Broadly speaking, modern scientific software includes a focus on simulation and modeling, often called computational science and engineering, which has become a recognized and respected component of modern science and technology, driving not only leading-edge scientific discovery, but also increasingly informing policy in many areas, and providing the basis of many technological and engineering products. Scientific software also includes data processing, including data analysis and machine learning, which have likewise been enabled by the rapid growth of computational power and software support. Relying on the combination of modeling and simulation capabilities and data processing, experimental and observational science have also, in many cases, become data-, compute-, and software-intensive. These factors place the ability to understand and improve our approach to the development and use of scientific software at the center of our ability to effectively continue the advances that we rely on in science and technology.

This understanding has multiple components, including understanding the software itself (computer science and software engineering), understanding the applications of the software (a wide variety of science and engineering disciplines), and understanding the software in context, such as how it is supported, developed, and used by projects and organizations and how the careers of the people who develop and use it are impacted (management, organizational studies, business, sociology, anthropology, and economics). Each decision to develop new software, maintain existing software, or use existing software involves a series of

tradeoffs related to multiple stakeholders, again including developers, users, funders, hiring organizations, and disciplinary communities. These in turn involve human and organizational incentives, benefits, and costs, most of which are not well-understood at the time the decisions need to be made, and which can also be difficult to understand and analyze in hindsight. However, improvements in understanding the factors that go into these decisions will lead to improved results, including better, more sustainable software, better scientific understanding, and better, more sustainable careers for the people involved.

3.3.2 Opportunities

Partnering with users and producers of scientific software outside the DOE. The pervasiveness of scientific software means that software developers and users at many organizations outside of the traditional DOE community will have similar experiences and common interests in obtaining a better understanding of those experiences. This includes academia, industry, and other national laboratories. Indeed, a significant portion of scientific software of interest to the DOE community is developed in collaboration with or entirely by people primarily in other communities. Opportunities to examine a broader base of software and users, as well as the potential to share costs of such R&D will benefit both DOE and the broader scientific software ecosystem.

Partnering with researchers from outside of computing. Expertise and experience from numerous other disciplines can benefit the study of scientific software development and use and the teams that do this work. For example, management, organizational studies, business, sociology, anthropology, and economics can bring useful and important insights to scientific software, as they have brought to many other endeavors. Creating such collaborations is a challenge, requiring careful design of incentives (including career rewards) that encourage them, methods for creating them (e.g., workshops to bring different types of researchers together to share differing experiences), methods for sustaining them (e.g., funding opportunities for collaborative projects), and methods for sharing and recognizing outputs (e.g., journals, conferences, prizes).

3.3.3 Connections to PRDs

PRD1 - Tools for productivity and sustainability

- Software developers inside and outside of the DOE community will have many considerations in common regarding choices about and adoption of tooling in their work. This expands both the potential pool for study and the possible sponsors who would be interested in and benefit from such studies.
- Studies of tool choice and tool adoption are well suited to the background and interests of researchers in the social sciences.
- Similarly, tools that enhance software development and use within the DOE community are likely also to benefit many in the broader scientific software community. This expands the potential user base for such tools, as well as the number of organizations that could benefit from sponsoring their development.

PRD2 - Team-based scientific software

- Gaining a better understanding of how teams develop and use scientific software would benefit greatly from the involvement of social scientists of various stripes, who study people and teams in other contexts.

PRD3 - Trustworthy science

- The delivery of trustworthy science is of high importance throughout the scientific enterprise. Indeed, there may be other organizations that are even more dependent than the DOE on software to guide consequential decisions. This may motivate partnerships in R&D into making software-driven science more trustworthy.

3.4 Connections among Crosscutting Themes

Just as each Crosscutting Theme identified in the workshop is connected to each Priority Research Direction (Secs. 3.1.3, 3.2.3, and 3.3.3), so too are the themes connected to each other.

Theme 1 - The human element *and* Theme 2 - Workforce recruitment and retention

- A scientific software community that pays attention to human elements will generally be more welcoming to new people and better able to tap into the potential of new workforce members.
- Reducing the barriers to effective and efficient development and use of scientific software will increase the number of people with diverse backgrounds who can participate in scientific software activities.

Theme 1 - The human element *and* Theme 3 - Centrality of software

- The need to gain a better understanding of the people involved in scientific software development and use naturally connects with the need to engage with researchers in the social and information sciences who traditionally study people and organizations.
- A better understanding of the human elements of scientific software development and use will facilitate more effective and efficient partnering with collaborators outside of DOE and outside of computing, at the level of individual projects as well as across different types of institutions and research sponsors.
- The ubiquity of software provides opportunities to examine the human element of scientific software development and use in comparison with the larger software world, helping to identify both common and distinctive aspects of the scientific software community.

Theme 2 - Workforce recruitment and retention *and* Theme 3 - Centrality of software

- The challenges and opportunities associated with workforce recruitment and retention have much in common across all scientific software communities.
- Different scientific software communities may find it easier to offer certain kinds of support and recognition to scientific software developers, offering opportunities for natural experiments* to better understand how such strategies may influence recruitment and retention.
- The ubiquity of software provides opportunities to examine approaches to workforce recruitment and retention in the larger software world and compare them with those in scientific software.

*Observational study in which an event or a situation that allows for the random or seemingly random assignment of study subjects to different groups is exploited to answer a particular question. [57]

Chapter 4

Conclusions

In this report, we have identified three priority research directions (PRDs) targeting individual, team and community concerns (Table 1.1), along with three crosscutting themes (Table 1.2) that provide guidance and goals for how we can pursue these directions. The PRDs and themes provide a foundation and framework for future research and development to improve scientific software development and use. We anticipate that the outcomes of these R&D activities will have strong positive impacts on the development and use of software for science. Some specific outcomes we expect to see are:

- Improved software development practices and tools that will reduce the cost and time of development and result in products that have higher quality and are easier to maintain.
- Improved team constitution and interaction of team members, resulting in more effective and efficient teams.
- Improved confidence and trust in the results that the scientific community produces.

As we pursue these research directions, we must keep in mind the three crosscutting themes that will guide our efforts:

- Consider both technical and human elements holistically.
- Bring in new community members, retain and advance current members, and support complete career lifecycles.
- Establish partnerships across communities.

These themes provide a framework for how we can conduct our work and are essential for our overall success.

The PRDs and themes described in this report represent a new kind of R&D scope for the DOE Office of Advanced Scientific Computing Research (ASCR). ASCR's research mission focuses on the creation of new scientific computing algorithms and software. The R&D identified in this workshop report will enable ASCR R&D communities to invest in improving the way they conduct their software efforts, bringing a *scientific* approach that should provide sustained rigor and value to ASCR-funded software projects.

Our emphasis on the science of scientific-software development and use is a new direction for ASCR that is consistent with its scientific mission. We expect that this new direction will have a positive impact on the DOE mission, and we look forward to working with the DOE community to make this vision a reality.

The expected outcome from investing in the PRDs and themes described in this report is better scientific impact. *Investment in software is investment in science.*

Bibliography

[1] Amanda Lee and Jeffrey C. Carver, FLOSS Participants' Perceptions About Gender and Inclusiveness: A Survey, in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 677–687, 2019, doi:10.1109/ICSE.2019.00077.

[2] Philippe Kruchten, Robert L Nord, and Ipek Ozkaya, Technical debt: From metaphor to theory and practice, *IEEE Software* **29**, 18 (2012), doi:10.1109/MS.2012.167.

[3] OpenAI, Inc., ChatGPT, online, `https://chat.openai.com`, 2023.

[4] GitHub, Inc., GitHub Copilot: Your AI pair programmer, online, `https://github.com/features/copilot`, 2023.

[5] Google, Inc., Bard, online, `https://bard.google.com`, 2023.

[6] Chris Reddington, How companies are boosting productivity with generative AI, online, `https://github.blog/2023-05-09-how-companies-are-boosting-productivity-with-generative-ai/`, 2023.

[7] Krzysztof Czarnecki, Overview of generative software development, in *In Proceedings of Unconventional Programming Paradigms (UPP) 2004, 15-17 September, Mont Saint-Michel, France, Revised Papers*, pages 313–328, Springer-Verlag, 2004, doi:10.1007/11527800_25.

[8] W. Spencer Smith, John McCutchan, and Fang Cao, Program Families in Scientific Computing, in Jonathan Sprinkle, Jeff Gray, Matti Rossi, and Juha-Pekka Tolvanen, editors, *7th OOPSLA Workshop on Domain Specific Modelling (DSM'07)*, pages 39–47, Montréal, Québec, 2007.

[9] N. U. Eisty, G. K. Thiruvathukal, and J. C. Carver, A Survey of Software Metric Use in Research Software Development, in *2018 IEEE 14th International Conference on e-Science (e-Science)*, pages 212–222, 2018, doi:10.1109/eScience.2018.00036.

[10] Ildevana Poltronieri Rodrigues, Márcia de Borba Campos, and Avelino F. Zorzo, Usability Evaluation of Domain-Specific Languages: A Systematic Literature Review, in Masaaki Kurosu, editor, *Human-Computer Interaction. User Interface Design, Development and Multimodality*, pages 522–534, Cham, 2017, Springer International Publishing, doi:10.1007/978-3-319-58071-5_39.

[11] Tomaz Kosar, Marjan Mernik, and Jeffrey C. Carver, Program Comprehension of Domain-Specific and General-Purpose Languages: Comparison Using a Family of Experiments, *Empirical Software Engineering* **17**, 276 (2012), doi:10.1007/s10664-011-9172-x.

[12] Tomaz Kosar, Saso Gaberc, Jeffrey C. Carver, and Marjan Mernik, Program comprehension of domain-specific and general-purpose languages: replication of a family of experiments using integrated development environments, *Empirical Software Engineering* **23**, 2734 (2018), doi:10.1007/s10664-017-9593-2.

[13] Arne N. Johanson and Wilhelm Hasselbring, Software Engineering for Computational Science: Past, Present, Future, *Computing in Science & Engineering* **20**, 90 (2018), doi:10.1109/MCSE.2018.021651343.

[14] Jeff Estefan, Survey of Model-Based Systems Engineering (MBSE) Methodologies, *INCOSE MBSE Focus Group* **25**, 1 (2008).

[15] Ana Luísa Ramos, José Vasconcelos Ferreira, and Jaume Barceló, Model-Based Systems Engineering: An Emerging Approach for Modern Systems, *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* **42**, 101 (2012), doi:10.1109/TSMCC.2011.2106495.

[16] Will Schroeder and Michael Grauer and Matt McCormick and Marcus D. Hanwell and Jean-Christophe Fillion-Robin, How Sustainable is Your Software?, online, `https://www.kitware.com/how-sustainable-is-your-software/`, 2020.

[17] Daniel S. Katz, Defining Software Sustainability, online, `https://danielskatzblog.wordpress.com/2016/09/13/defining-software-sustainability/`, 2016.

[18] The Software Sustainability Institute, Online sustainability evaluation, online, `https://www.software.ac.uk/resources/online-sustainability-evaluation`, 2020.

[19] Nasir U. Eisty, George K. Thiruvathukal, and Jeffrey C. Carver, Use of Software Process in Research Software Development: A Survey, in *Proceedings of the Evaluation and Assessment on Software Engineering*, EASE '19, page 276–282, New York, NY, USA, 2019, Association for Computing Machinery, doi:10.1145/3319008.3319351.

[20] Raquel Asencio, Dorothy R Carter, Leslie A DeChurch, Stephen J Zaccaro, and Stephen M Fiore, Charting a course for collaboration: a multiteam perspective, *Translational Behavioral Medicine* **2**, 487 (2012), doi:10.1007/s13142-012-0170-3.

[21] Nicholas Berente and James Howison, Strategies for Success in Virtual Collaboration: Structures and Norms for Meetings, Workflow, and Technological Platforms, in Kara L. Hall, Amanda L. Vogel, and Robert T. Croyle, editors, *Strategies for Team Science Success: Handbook of Evidence-Based Principles for Cross-Disciplinary Science and Practical Lessons Learned from Health Researchers*, pages 563–574, Springer International Publishing, Cham, 2019, doi:10.1007/978-3-030-20992-6_43.

[22] Olivia B. Newton, Samaneh Saadat, Jihye Song, Stephen M. Fiore, and Gita Sukthankar, EveryBOTy Counts: Examining Human–Machine Teams in Open Source Software Development, *Topics in Cognitive Science* (2022), doi:10.1111/tops.12613.

[23] Stephen M. Fiore, Interdisciplinarity as Teamwork: How the Science of Teams Can Inform Team Science, *Small Group Research* **39**, 251 (2008), doi:10.1177/1046496408317797.

[24] Stephen M. Fiore, Catherine Gabelica, Travis J. Wiltshire, and Daniel Stokols, Training to Be a (Team) Scientist, in Kara L. Hall, Amanda L. Vogel, and Robert T. Croyle, editors, *Strategies for Team Science Success: Handbook of Evidence-Based Principles for Cross-Disciplinary Science and Practical Lessons Learned from Health Researchers*, pages 421–444, Springer International Publishing, Cham, 2019, doi:10.1007/978-3-030-20992-6_33.

[25] Steve W. J. Kozlowski and Georgia T Chao, Unpacking team process dynamics and emergent phenomena: Challenges, conceptual advances, and innovative methods., *The American psychologist* **73**, 576 (2018), doi:10.1037/amp0000245.

[26] J. C. Carver, N. Eisty, H. Nam, and I. Tezaur, Special Issue on the Future of Research Software Engineers in the United States—Part I, *Computing in Science & Engineering* **24**, 4 (2022).

[27] A. Malviya-Thakur, D. E. Bernholdt, W. F. Godoy, G. R. Watson, M. Doucet, M. A. Coletti, D. M. Rogers, M. McDonnell, J. Billings, and B. Maccabe, Research Software Engineering at Oak Ridge National Laboratory, *Computing in Science & Engineering* **24**, 14 (2022).

[28] W. F. Godoy, R. Arora, K. Beattie, D. E. Bernholdt, S. E. Bratt, D. S. Katz, I. Laguna, A. K. Maji, A. Thakur, R. M. Mudafort, N. Sukhija, D. Rouson, C. Rubio-Gonzalez, and K. Vahi, Giving Research Software Engineers a Larger Stage Through the Better Scientific Software Fellowship, *Computing in Science &; Engineering* **24**, 6 (2022).

[29] M. A. Heroux, Research Software Science: Expanding the Impact of Research Software Engineering, *Computing in Science & Engineering* , 1 (5555).

[30] Jeffrey C. Carver, Richard P. Kendall, Susan E. Squires, and Douglass E. Post, Software Development Environments for Scientific and Engineering Software: A Series of Case Studies, in *29th International Conference on Software Engineering (ICSE'07)*, pages 550–559, 2007, doi:10.1109/ICSE.2007.77.

[31] Elvira-Maria Arvanitou, Apostolos Ampatzoglou, Alexander Chatzigeorgiou, and Jeffrey C. Carver, Software Engineering Practices for Scientific Software Development: A Systematic Mapping Study, *Journal of Systems and Software* **172**, 110848 (2021), doi:10.1016/j.jss.2020.110848.

[32] Jeffrey C. Carver, Nic Weber, Karthik Ram, Sandra Gesing, and Daniel S. Katz, A survey of the state of the practice for research software in the United States, *PeerJ Computer Science* **8**, e963 (2022), doi:10.7717/peerj-cs.96.

[33] Fabien C. Y. Benureau and Nicolas P. Rougier, Re-run, Repeat, Reproduce, Reuse, Replicate: Transforming Code into Scientific Contributions, *Frontiers in Neuroinformatics* **11** (2018), doi:10.3389/fninf.2017.00069.

[34] David H. Bailey, Jonathan M. Borwein, and Victoria Stodden, *Reproducibility: Principles, Problems, Practices*, chapter Facilitating reproducibility in scientific computing: principles and practice, pages 205–232, John Wiley and Sons, New York, 2016, doi:10.1002/9781118865064.ch9.

[35] Michael Heroux (chair) and Wolfgang Bangerth and Juliana Freire and Patrick Heimbach and Ivo Jimenez and Ellen Rathje and Hakizumwami (Birali) Runesha and Victoria Stodden, Improving Trustworthiness of Computational Results: Opportunities for the NSF Office of Advanced Cyberinfrastructure to address recommendations from the National Academies Report on Reproducibility, online, `https://www.nsf.gov/cise/oac/ImprovingTrustworthiness.pdf`, 2022.

[36] Silvana M. Melo, Jeffrey C. Carver, Paulo S.L. Souza, and Simone R.S. Souza, Empirical research on concurrent software testing: A systematic mapping study, *Information and Software Technology* **105**, 226 (2019), doi:10.1016/j.infsof.2018.08.01.

[37] Nasir Eisty and Jeffrey C. Carver, Developers perception of peer code review in research software development, *Empirical Software Engineering* **27** (2022), doi:10.1007/s10664-021-10053-x.

[38] Nasir Eisty and Jeffrey C. Carver, Testing research software: a survey, *Empirical Software Engineering* **27**, 138 (2022), doi:10.1007/s10664-022-10184-9.

[39] John Rushby, The Interpretation and Evaluation of Assurance Cases, Technical Report SRI-CSL-15-01, Computer Science Laboratory, SRI International, Menlo Park, CA, 2015, Available at `http://www.csl.sri.com/users/rushby/papers/sri-csl-15-1-assurance-cases.pdf`.

[40] David J. Rinehart, John C. Knight, and Jonathan Rowanhill, Current Practices in Constructing and Evaluating Assurance Cases with Applications to Aviation, Technical Report CR-2014-218678, National Aeronautics and Space Administration (NASA), Langley Research Centre, Hampton, Virginia, 2015.

[41] W. Spencer Smith, Mojdeh Sayari Nejad, and Alan Wassyng, Raising the Bar: Assurance Cases for Scientific Computing Software, *Computing in Science and Engineering* **23**, 47 (2020), doi:10.1109/MCSE.2020.3019770.

[42] W. Spencer Smith, Lei Lai, and Ridha Khedri, Requirements Analysis for Engineering Computation: A Systematic Approach for Improving Software Reliability, *Reliable Computing, Special Issue on Reliable Engineering Computation* **13**, 83 (2007), doi:10.1007/s11155-006-9020-7.

[43] Upulee Kanewala and Tsong Yueh Chen, Metamorphic Testing: A Simple yet Effective Approach for Testing Scientific Software, *Computing in Science & Engineering* (2018), doi:10.1109/MCSE.2018.2875368.

[44] Gillian Tett, *Anthro-Vision: A New Way to See in Business and Life*, Avid Reader Press/Simon & Schuster, 2021.

[45] Nancy J. Cooke and Margaret L. Hilton, editors, *Enhancing the Effectiveness of Team Science*, The National Academies Press, Washington, DC, 2015, doi:10.17226/19007.

[46] Kara L. Hall, Amanda L. Vogel, and Robert T. Croyle, editors, *Strategies for Team Science Success*, Springer Cham, 2019, doi:10.1007/978-3-030-20992-6.

[47] Stanley McChrystal, Tantum Collins, David Silverman, and Chris Fussell, *Team of Teams: New Rules of Engagement for a Complex World*, Portfolio, 2015.

[48] Marissa L. Shuffler and Dorothy R. Carter, Teamwork situated in multiteam systems: Key lessons learned and future opportunities, *American Psychologist* **73**, 390 (2018), doi:10.1037/amp0000322.

[49] Suzanne T. Bell and Neal Outland, Team composition over time, in Eduardo Salas, William Brandon Vessey, and Lauren Blackwell Landon, editors, *Team dynamics over time: Advances in psychological theory, methods, and practice*, Research on Managing Groups and Teams, pages 3–27, Emerald Group Publishing, 2017.

[50] Kenneth D. Gibbs, Anna Han, and Janetta Lun, Demographic Diversity in Teams: The Challenges, Benefits, and Management Strategies, in Kara L. Hall, Amanda L. Vogel, and Robert T. Croyle, editors, *Strategies for Team Science Success: Handbook of Evidence-Based Principles for Cross-Disciplinary Science and Practical Lessons Learned from Health Researchers*, pages 197–205, Springer International Publishing, Cham, 2019, doi:10.1007/978-3-030-20992-6_15.

[51] Maritza R. Salazar, Theresa K. Lant, Stephen M. Fiore, and Eduardo Salas, Facilitating Innovation in Diverse Science Teams Through Integrative Capacity, *Small Group Research* **43**, 527 (2012), doi:10.1177/1046496412453622.

[52] Barbara Chapman, Henri Calandra, Silvia Crivelli, Jack Dongarra, Jeffrey Hittinger, Scott A. Lathrop, Vivek Sarkar, Eric Stahlberg, Jeffrey S. Vetter, and Dean Williams, DOE Advanced Scientific Advisory Committee (ASCAC): Workforce Subcommittee Letter, online, doi:10.2172/1222711, 2014.

[53] National Laboratory Directors' Council, The National Laboratories: Diversity & Inclusion, online, https://nationallabs.org/staff/diversity, 2022.

[54] The Whitehouse, FACT SHEET: Biden-Harris Administration Releases Recommendations for Advancing Use of Equitable Data, online, https://www.whitehouse.gov/briefing-room/statements-releases/2022/04/22/fact-sheet-biden-harris-administration-releases-recommendations-for-advancing-use-of-equitable-data, 2022.

[55] Mary Ann Leung, Diversity and Inclusion Through Leadership During Challenging Times, *Computing in Science & Engineering* **22**, 92 (2020), doi:10.1109/MCSE.2020.3020445.

[56] D. S. Katz, L. C. McInnes, D. E. Bernholdt, A. C. Mayes, N. P. C. 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, *Computing in Science Engineering* **21**, 8 (2019), doi:10.1109/MCSE.2018.2883051.

[57] Lynne C. Messer, Natural Experiment, Encyclopedia Britannica, https://www.britannica.com/science/natural-experiment, 2016.

[58] Michael A. Heroux, Research Software Science: A Scientific Approach to Understanding and Improving How We Develop and Use Software for Research, blog article in Better Scientific Software, online, https://bssw.io/blog_posts/research-software-science-a-scientific-approach-to-understanding-and-improving-how-we-develop-and-use-software-for-research, 2019.

[59] Michael A. Heroux, Lois McInnes, David Bernholdt, Anshu Dubey, Elsa Gonsiorowski, Osni Marques, J. David Moulton, Boyana Norris, Elaine Raybourn, Satish Balay, Roscoe A. Bartlett, Lisa Childers, Todd Gamblin, Patricia Grubel, Rinku Gupta, Rebecca Hartman-Baker, Judith Hill, Stephen Hudson, Christoph Junghans, Alicia Klinvex, Reed Milewicz, Mark Miller, Hai Ah Nam, Jared O Neal, Katherine Riley, Ben Sims, Jean Schuler, Barry F. Smith, Louis Vernon, Gregory R. Watson, James Willenbring, and Paul Wolfenbarger, Advancing Scientific Productivity through Better Scientific Software: Developer Productivity and Software Sustainability Report, Technical Report ORNL/TM-2020/1459, ECP-U-RPT-2020-0001, Oak Ridge National Laboratory, 2020, doi:10.2172/1606662.

[60] David E. Bernholdt, John Cary, Michael Heroux, and Lois Curfman McInnes, Position Papers for the ASCR Workshop on the Science of Scientific-Software Development and Use, online, doi:10.2172/1843575, 2021.

Appendix A

Workshop Description and Call for Position Papers

The following material appeared on the workshop website: `https://www.orau.gov/SSSDU2021`*. It has been lightly reformatted for inclusion here.*

A.1 Description

Software is an increasingly important component in the pursuit of scientific discovery. Both its development and use are essential activities for many scientific teams. At the same time, very little scientific study has been conducted to understand, characterize, and improve the development and use of software for science.

Computational science teams have diversified over time to include contributions from domain scientists who provide expertise in scientific and engineering disciplines, applied mathematicians who provide theoretical rigor to models, computer scientists who provide optimal algorithms and data structures, and software engineers who provide methodologies and tools adapted and adopted from other software domains. These diverse contributions have enabled tremendous advances in the pursuit of scientific discovery, even as models, computer architectures, and software environments have become more complicated.

With this increasing diversity, we believe the next opportunity for qualitative improvement comes from applying the scientific method to understanding, characterizing, and improving how scientific software is developed and used. We believe that this pursuit requires expertise from computational scientists themselves, and from the cognitive and social sciences.

As we look to increase the productivity and sustainability of the scientific-software- development-and-use cycle, especially considering the potential impact of AI-driven tools, a more systematic application of the scientific method to understanding development and use processes [58, 59] will be a valuable tool to guide future work and results in more usable and sustainable software. This workshop will bring together computer scientists, computational scientists, social scientists, cognitive scientists, and others, to explore how we can conduct such systematic investigations, what can be learned, and how doing so will benefit the scientific enterprise.

A.2 Position Paper Submission

A.2.1 Motivation

Software is an increasingly important component in the pursuit of scientific discovery. Both its development and use are essential activities for many scientific teams. At the same time, very little scientific study has

been conducted to understand, characterize, and improve the development and use of software for science.

Computational science teams have diversified over time to include contributions from domain scientists who provide expertise in scientific and engineering disciplines, applied mathematicians and computer scientists who provide optimal algorithms and data structures, and software and data engineers who provide methodologies and tools adapted and adopted from other software domains. These diverse contributions have enabled tremendous advances in the pursuit of scientific discovery, even as models, computer architectures, and software environments have become more complicated.

With this increasing diversity, we believe the next opportunity for qualitative improvement comes from applying the scientific method to understanding, characterizing, and improving how scientific software is developed and used. We believe that this pursuit requires expertise from computational scientists themselves, and from the cognitive and social sciences as well as the software engineering research community.

As we look to increase the productivity and sustainability of the scientific-software- development-and-use cycle, a more systematic application of the scientific method to understand processes for software development and use [58, 59] will be a valuable tool to guide future work and result in more usable and sustainable software. This workshop will bring together computer scientists, software engineering researchers, computational scientists, applied mathematicians, social scientists, cognitive scientists, and others, to explore how we can conduct such systematic investigations, what can be learned, and how doing so will benefit the scientific enterprise.

The workshop will be structured around a set of breakout sessions, with every attendee expected to participate actively in the discussions. Afterward, workshop attendees — from DOE, industry, and academia — will produce a report for ASCR that summarizes the findings of the workshop.

A.2.2 Invitation

We invite community input in the form of two-page position papers that identify and discuss key challenges and opportunities in the science of the scientific-software development process and the study of the use of scientific software. In addition to providing an avenue for identifying workshop participants, these position papers will be used to shape the workshop agenda, identify panelists, and contribute to the workshop report. Position papers should not describe the authors' current or planned research, contain material that should not be disclosed to the public, recommend specific solutions, or discuss narrowly-focused research topics. Rather, position papers should aim to improve the community's shared understanding of the problem space, identify challenging research directions, and help to stimulate discussion.

One author of each selected submission will be invited to participate in the workshop.

By submitting a position paper, authors consent to have their position paper published publicly.

Authors are not required to have a history of funding by the ASCR Computer Science program.

A.2.3 Submission Guidelines

Position Paper Structure and Format

Position papers should follow the following format:

- Title
- Authors (with affiliations and email addresses)
- Challenge: Identify limitations of state-of-the-art practice with examples

- Opportunity: Describe how the identified challenges may be addressed, whether through new tools and techniques, new technologies, new methodologies, or new groups collaborating in the process
- Timeliness or maturity: Why now? What breakthrough or change makes progress possible now where it wasn't possible previously? What will be the impact of success?
- References

Each position paper must be no more than two pages including figures and references. The paper may include any number of authors but contact information for a single author who can represent the position paper at the workshop must be provided with the submission. There is no limit to the number of position papers that an individual or group can submit. Authors are strongly encouraged to follow the structure outlined above. Papers should be submitted in PDF format using the designated page on the workshop website.

Notional Questions

Position papers should present views on why and how scientific-software development and use can be studied as a scientific endeavor, perhaps taking inspiration from some of the following:

- What methods can be used to study the development and/or use of scientific software?
- What methods either uniquely apply or uniquely do not apply to the development of scientific software?
- How can we best integrate social and cognitive sciences into scientific software activities?
- How can we study changes in culture affecting scientific software development and use?
- What unique changes in scientific software development would improve productivity, accuracy, trust, and/or reliability?
- How can we motivate changes in the scientific-computing community to improve practices for software development and use?
- How will AI-driven development tools affect best practices for scientific software development and use?
- How will static and dynamic analysis methods affect the practice of scientific software development?
- How are scientific software developers and users atypical from other larger software communities?
- When and how can research and methods from other software communities be adapted and adopted for scientific software?
- How is the field of computational science changing, and how will it change?
- What are the roadblocks faced by computational scientists in computational application use?
- What are the challenges faced by computational scientists in their pursuit of discoveries?

Selection

Submissions will be reviewed by the workshop's organizing committee using criteria of overall quality, relevance, likelihood of stimulating constructive discussion, and ability to contribute to an informative workshop report. Unique positions that are well presented and emphasize potentially-transformative research directions will be given preference.

Appendix B

Workshop Agenda

Links to presentations included where available. Links for breakout sessions are to the readout presentations. Some listed breakout sessions did not take place, based on participant interests. Links are to the workshop web site and should not be considered archival.

Day 1: December 13

Time	Topic
12:00pm–12:15pm	Opening Remarks - Hal Finkel, ASCR
12:15pm–12:45pm	Introduction and Logistics - Mike Heroux, SNL
12:45pm–1:30pm	Keynote: Steve Plimpton - Sandia National Laboratories Thoughts on software for science
1:30pm–2:00pm	Lightning Talks and Panel Abhinav Bhatele, Jeff Candy, Daniel Crawford, Ewa Deelman, Miranda Mundt, Slaven Peles
2:00pm–2:30pm	Lightning Talks and Panel Andy Gallo, Sandra Gesing, Kyle Harrington, Ignacio Laguna, Spencer Smith, Jana Thayer
2:30pm–3:00pm	Lightning Talks and Panel Prasanna Balaprakash, Breck Baldwin, Caroline Jay, Todd Ringler, Oceane Bel, Elizabeth Sexton-Kennedy
3:00pm–3:15pm	Break
3:15pm–4:15pm	Breakouts - Challenges, Past Experience, and Open Questions Challenges in Developing Scientific Software, Challenges in Using Scientific Software, Challenges in adoption of new practices
4:15pm–5:00pm	Readouts / Summary

Day 2: December 14

Time	Topic
12:00pm–12:05pm	Opening Remarks and Logistics
12:05pm–12:50pm	Keynote: Ian Cosden - Princeton University Research Software Engineers Success, Community, and Open Questions
12:50pm–2:00pm	Lightning Talks and Panel Christopher Lenhardt, Annika Meinecke, Boyanna Norris, Jordan Perr-Sauer, Elaine Raybourn, Phil Roth
2:00pm–2:05pm	Break
2:05pm–3:00pm	Breakouts - Methods Observations and Surveys, Large Scale Data Mining, Longevity of Best Practices, Learning from Other Domains
3:00pm–3:15pm	Break
3:15pm–4:15pm	Breakouts - Scale and Impact Scale Required for Different Kinds of Studies, Potential Opportunities for using AI, Training Impact of Better Training and Practices on Productivity, Non-Tech Road Blocks to Studying Development and Use
4:15pm–5:00pm	Readouts / Summary

Day 3: December 15

Time	Topic
12:00pm–12:05pm	Opening Remarks and Logistics
12:05pm–12:50pm	Keynote: Hannah Cohoon - The University of Texas at Austin Do science, for software's sake!
12:50pm–2:00pm	Lightning Talks and Panel Ben Dudson, Daniel S. Katz, Mary Ann Leung, Siva Rajamanickam, Ben Sims, Richard Barnes
2:00pm–2:05pm	Break
2:05pm–3:00pm	Breakouts - Priority Research Directions, Grand Challenges, and Transformational Opportunities Studying Scientific Software Development, Studying Scientific Software Use, Studying Pre-Development Processes
3:00pm–3:15pm	Break
3:15pm–3:45pm	Writing
3:45pm–5:00pm	Readouts / Summary

Appendix C

Accepted Position Papers

A total of 124 position papers were accepted for the workshop. They are listed and linked individually below and have been gathered into two collections:

- a zip file of the individual position papers, and
- a concatenation of the position papers into a single PDF which has been archived with the DOE Office of Science and Technical Information (OSTI) under the following citation [60] :

  David E. Bernholdt, John Cary, Michael Heroux, and Lois Curfman McInnes, Position Papers for the ASCR Workshop on the Science of Scientific-Software Development and Use, online, https://doi.org/10.2172/1843575, 2021.

Individual position paper links (below) and zip file link (above) refer to the workshop website and should not be considered archival. The OSTI PDF document is archival.

Table C.1: Accepted position papers.

Authors (Corresponding Author Underlined)	**Title and Link**
Jeffrey Carver, Nasir Eisty	Scientific Software Development Laboratory: A Proposal
Anshu Dubey	Sustainability of Software Meant for Exploration
Caroline Jay, Robert Haines, Daniel Katz, Jeffrey Carver	Theory-Software Translation Challenges
Daniel S. Katz, Michelle Barker	Software sustainability is people
Barton Miller, Sean Peisert	Improving Assurance of Scientific Software
Mathieu Doucet, William F. Godoy, Srikanth Yoginath, Christopher B. Stanley	Understanding the patterns driving the development of software to support science
Andy Gallo, Eric Tucker	Coercive Software Process Improvement
Todd Ringler	Towards a More Sustainable Software Development Environment
Ganesh Gopalakrishnan, Ignacio Laguna, Ang Li, Pavel Panchekha, Cindy Rubio-Gonzalez, Zachary Tatlock	The Unfledged State of Heterogeneous System Testing

Continued on next page

Table C.1 – *continued from previous page*

Authors (Corresponding Author Underlined)	Title and Link
James M. Willenbring	A Testing Strategy that Supports Scientific Software Sustainability
Kirk Landin	Toward a Science of Abstraction Design in Software
Olga Pearce, David Boehme	Performance Diagnostics and Debugging at Scale
Xu Liu	Bridging Software Developers with Dynamic Program Analysis in IDE
Kyle Harrington	Quantitative methods for research software community management
Thorsten Sommer	Combining Repo and Process Mining for Better Sci Software
Gregory Watson, Addi Malviya Thakur	Accelerating Excellence in Scientific Software
Lynn von Kurnatowski, Martin Stoffers, Carina Haupt	Provenance based software dashboards
Xiaozhu Meng, John Mellor-Crummey	Deep Learning Assisted Performance Analysis and Diagnosis
Jin Chen, Nathaniel Ferraro, Stephen Jardin	On the need for efficient and scalable solvers for ill-conditioned sparse matrix equations
Cyrus Harrison, Arlie Capps, Richard Hornung, Mark Miller	Adventures in Modularity - Why is Sharing Hard?
Hui Zhou	Manage Complexity With Generic Meta Programming
John Wu	Beyond Open Source: A Call to Rethink Policies and Incentives for Sustainable Scientific Software Development
Breck Baldwin	Funding Strategies for Scientific Software
Sarah Poon, Drew Paine, Lavanya Ramakrishnan, Dan Gunter	Design Systems for Science
Hemanth Kolla, Marco Arienti	An experience-based perspective on scientific software integration
Dan Gunter, Lavanya Ramakrishnan, Drew Paine	Understanding Complex Software Dependencies for Better Scientific Reproducibility
Drew Paine, Lavanya Ramakrishnan, Dan Gunter, Sarah Poon	Improving Software Sustainability Through Adoption and Democratization of User
Ryan M. Richard, Theresa L. Windus	Is a Language Barrier Impeding Development of Better Scientific Software?
Sou-Cheng Terrya ChoiI, Yuhan Ding, Claude Hall Jr., Fred J. Hickernell, AlekseiI Sorokin	Four Ways to Grow Scientific Software

Continued on next page

Table C.1 – *continued from previous page*

Authors (Corresponding Author Underlined)	Title and Link
Jeff Terry	Literature Validation and Experimental Analysis Using Advanced Machine Learning Tools
Dong H. Ahn, Xiaohua Zhang, Jeffrey Mast, Francesco Di Natale, Dan Kirshner, Sam Ade Jacobs, Bronis de Supinski, Brian Van Essen, Jonathan E. Allen, Felice C. Lightstone	A large multi-disciplinary drug design team's perspective on sustainable computing software for transformative science
Chunhua Liao, Peter Pirkelbauer	Automated, Extensible Correctness Diagnostics for Scientific Computing
Chunhua Liao	Reinventing the Ecosystem to Improve HPC Software
Bill Hoffman	The Problem with Modern CI
Aaron S. Brewster, Dorothee Liebschner, Billy K. Poon, Nicholas K. Sauter, Paul D. Adams	The challenges of scientific software sustainability
Chen Zhang	AI-assisted Collaboration between Domain Scientists and Research Software Engineers
Benjamin Sims, Elaine M. Raybourn, Reed Milewicz, David Moulton, Will Sutherland	Social science research is essential to the future of the scientific
Miranda Mundt, Reed Milewicz	Working in Harmony: Towards Integrating RSEs into Multi-Disciplinary CSE Teams
Reed Milewicz, Miranda Mundt	Building Bridges: Establishing a Dialogue Between Software Engineering Research and Computational Science
Greg Eisenhauer, Jeremy Logan, Patrick Widener, Matthew Wolf	Engineering Scientific Workflows for the Scientists of the Future
Ewa Deelman, Miron Livny, Prasanna Balaprakrash, Mariam Kiran, Anirban Mandal,	The proof is in the Pudding: Detecting problems with software in production
Ignacio Laguna	Reproducibility in the Era of Heterogeneous Computing
Ignacio Laguna, Giorgis Georgakoudis, Harshitha Menon, Konstantinos Parasyris	Compiler Analysis for Early Detection of Software Defects
Nathan Tallent, Zhe Feng	Studying Data Patterns to Enable Scientific Exploration with Data-Intensive Distributed Workflows
Philip C. Roth	Viewing Scientific Software Development Through The Lens of Economics
Struan Clark, Nalinrat Guba, Rachel Hurst, Kristi Potter	Adopting Industry Software DevOps Practices to Improve Scientific Software
Spencer Smith, Jacques Carette	Long-Term Productivity Based on Science, not Preference

Continued on next page

Table C.1 – ***continued from previous page***

Authors (Corresponding Author Underlined)	Title and Link
Prasanna Balaprakash1, Romain Egele, Ewa Deelman, Mariam Kiran, Anirban Mandal	MLOps
Peter F. Peterson	The call to quantify the impact of scientific software
Derek Gaston, Logan Harbour, Fande Kong, Guillaume Giudicelli, Alexander Lindsay	Reliable, Efficient Workflows for Complex Scientific Software Development
Jeff Candy, Federico Halpern, Marta. Kostuk, David Schissel	Scientific-software Development and Lifetime
Sayan Ghosh, Anthony Skjellum, Purushotham V. Bangalore, Andrew Lumsdaine	Performance Portable Data Movement Support for MPI using C++
Slaven Peles, Ivan Celanovic, Cosmin Petra	Software for Simulations and Analysis of Complex Engineered Systems
Marshall McDonnell, Elton Cranfill, Jason Kincl, Addi Malviya-Thakur	Infrastructure-as-Code for Scientific Software and its Importance Towards Federated Infrastructure
Jai S. Sachdev	Balancing Innovation and Stability of Long Lived Scientific Software with Containers
Elizabeth Sexton-Kennedy, Graeme Andrew Stewart, Eduardo Rodrigues	HEP Software Challenges
Matthew Sottile , Markus Schordan, Dan Quinlan	Formal Methods White Paper
Sudip K. Seal	Revisiting Software Abstractions, Standards and Best Practices
Miroslav Stoyanov, Eirik Endeve	The Challenges in the Development of Math Libraries
Paul S. Crozier	How to measure and improve the quality and quantity of software developed by agile teams of scientific software
Charles Edison Tripp, Graham Johnson, Nalinrat Guba, Sagi Zisman, Jordan Perr-Sauer	Fundamental Needs for Establishing Evidence-Based Scientific Software Engineering Practices
Arnold Tharrington, Mark Berrill, Reuben D. Budiardja, Dmytro Bykov, Antigoni Georgiadou, Austin Harris, Justin Lietz	Software Engineering Mechanisms To Facilitate Best Practice Design Principles
T. Daniel Crawford, Paul Saxe, Theresa L. Windus	Engaging a Complex Domain Science in Software Engineering Best Practices: What Works and What Doesn't
Bin Dong, Kesheng Wu	High Productivity Data Analysis Infrastructure for Scientific Discoveries
Samuel D. Pollard, Ariel Kellison, John Bender, Heidi K. Thornquist, Geoffrey C. Hulette	Real(istic) Specifications of Software
Jacob McLemore, Alex C. Williams	Microproductivity in the Lab

Continued on next page

Table C.1 – *continued from previous page*

Authors (Corresponding Author Underlined)	Title and Link
Rob Crystal-Ornelas, Ann Almgren, Keith Beattie, Dan Gunter, Charuleka Varadharajan	Best practices to enable a culture of effective cross-functional scientific software collaboration
Sandra Gesing, Jeffrey C. Carver, Ian Cosden, Julia Damerow, Charles Ferenbaugh, Chris Hill, Daniel S. Katz, Christina Maimone, Lance Parsons	Accelerating Changes in the Scientific-Computing Community to Improve Practices for Software Development and Use
Ignacio Andres Aravena Solis, Nai-Yuan Chiang, Carl D. Laird, Cosmin G. Petra, Jean-Paul Watson	Maintaining Productivity and Increasing Community Outreach for Optimization Ecosystem
Yanfei Guo, Ken Raffenetti, Hui Zhou, Travis Koehring, Sudheer Chunduri, Xiaodong Yu, Rajeev Thakur	Automated Validation and Verification for Scientific Software
Gerd Heber, Elena Pourmal, Mike Folk	Scientific Software as Emergent Behavior
Shreyas Cholia, Valerie C. Hendrix, Keith Beattie	The Last Mile in Software Engineering
Leslie Carr, Simon Hettrick, Heather Packer	Explaining the Value of Research Software
Hari Krishnan , Alexander Hexemer, Padraic Shafer, Alexander Hexemer, Juliane Reinhardt, Dylan McReynolds, Padraic Shafer, Dula Parkinson, Lavanya Ramakrishnan, Shreyas Cholia	Capturing the varied complex software lifecycle processes at a scientific user facility
Jordan Perr-Sauer, Sagi Zisman, Rafael Mudafort, Robert White	Evidence-based interventions for research institutions
E. Wes Bethel, Burlen Loring, Oliver Rubel, Gunther Weber, Nicola Ferrier, Joseph Insley, Victor Mateevitsi, Silvio Rizzi	Fostering Interoperability and Increasing Scientific Productivity in Environments of Heterogeneous
Ben Dudson, Alex Friedman	The plural of anecdote is data
Sagi Zisman, Robert White, Jordan Perr-Sauer	Progress in the Study of Scientific Software Requires a Rigorous Validated Taxonomy
Jana Thayer, Jeff Shrager	Challenges and Opportunities for Scientific-Software Development Research
Addi Malviya Thakur, Audris Mockus	The New Science of Scientific Software Development
Rinku Gupta	The role of culture in scientific software teams
Sivasankaran Rajamanickam	Can Scientific Software Development Use the Outsourcing Model Successfully?
Elaine M. Raybourn	A Seat at The Table
Oceane Bel, Cimone Wright-Hamor, Joseph Manzano, Kevin Barker	Cybersecurity in High Performance Software Development: Challenges and Approaches
W. Christopher Lenhardt, Stephen M. Fiore, Shannon McKeen	Using the Science of Team Science for Better Science Software and Better Science

Continued on next page

Table C.1 – *continued from previous page*

Authors (Corresponding Author Underlined)	Title and Link
E. Wes Bethel, Zhe Bai, Talita Perciano, Robbie Sadre, John C. Wright, Gregory Wallace, Syun'Ichi Shiraiwa, Nicola Bertelli	AI/ML and Scientific Software Development and Distribution, Maintenance, and Code Teams
Ross Miller, Joshua Brown, Sarp Oral, Olga Kuchar, Mallikarjun Shankar	Emphasis on Software Maintenance for Scientific Software
Jeff Shrager, Wan-Lin Hu, Paul Fuoss, Jana Thayer	Cognitive Analytic Engines are Critical to Scientific User Facilities Reaching their Potential
Joshua Suetterlein, Joseph Manzano, Bruce Palmer, Andres Marquez, Kevin J. Barker	Software Practices for Cutting-Edge Hardware Evaluation
Justin M. Wozniak	Software Monsters: Quantifying, Reporting, and Controlling Composite Applications
Syun'Ichi Shiraiwa, Nicola Bertelli, Gregory Wallace, John C. Wright, E. Wes Bethel, Zhe Bai, Talita Perciano, Robbie Sadre	Accelerate high fidelity physics simulations for fusion energy using machine learning technology
Abhinav Bhatele, Daniel Nichols, Aniruddha Marathe	Machine Learning to Analyze and Improve the Development of Scientific Software
Jason Wiese, Mary Hall	Building Human-Centered Computing Knowledge of Scientific Software Development
Damian Rouson, Ondřej Čertík, Brad Richardson, Renée Blake, Zachary Jaggers	Expanding the Scientific Software Development Community Necessitates Studying Cognitive and Social Issues in Programming-Language Learning and Bias
Dong H. Ahn, Kyle Chard, James Corbett, Mihael Hategan, Shantenu Jha, Daniel Laney, Andre Merzky, Todd Munson, Rafael de Silva, Mikhail Titov, Matteo Turilli, Justin M. Wozniak	Community Building Blocks for Workflows
Peter Pirkelbauer, Chunhua Liao	Benchmark Suites
John Mellor-Crummey, Xiaozhu Meng	Developing a Sustainable Software Ecosystem for Tools, Runtime Libraries, and Applications
Kjiersten Fagnan, Jeffrey Johnson	Put an end to just in time planning
Tim Menzies	When Can We Trust Your Models
Mary Ann Leung, Connie L. McNeely	Transformative Profiles in Research Software Teams: DEI for Collaborative Innovation
Haowen Xu, Andy Berres, Srikanth Yoginath, Jibonananda Sanyal	Metaverse for Urban Informatics
Caifan Du	Outlining the Complexities of Scientific Software Development And Use
Elliott Slaughter, Alex Aiken, Pat McCormick	Specifying and Debugging Data Distributions at Scale

Continued on next page

Table C.1 – *continued from previous page*

Authors (Corresponding Author Underlined)	Title and Link
Terry Cojean, Aditya Kashi, Tobias Ribizel, Hartwig Anzt	Continued Investment to Secure Future-Readiness of Scientific Software
Jack Deslippe, Nicholas Wright, Rechard Gerber	Utilizing Facility Data and Best Practices to Guide Sustainable
Annika Meinecke, David Heidrich, Katharina Dworatzyk	Considering Activity-Centered Design in Scientific Software Development
Edoardo Serra, Mahantesh Halappanavar	Enhancing Scientific Software Development with AI-Capabilities
Pratik Nayak, Terry Cojean, Aditya Kashi, Tobias Ribizel, Hartwig Anzt	Introducing RSE as a practice in CS courses
Jay Lofstead, Matthew Glickman	The Opportunity and Challenge of Serverless Computing
Todd Gamblin	Automating Software Integration for Sustainability
Younghyun Cho, James W. Demmel, Xiaoye S. Li, Yang Liu, Hengrui Luo	Enhancing Software Performance Portability and Reproducibility
Vanessa Sochat, David Beckingsale, Robert Neely	Improving Development Practices through Culture and Collaboration
Carlo Graziani	Data Lifecycle Assurance in Scientific HPC
Roger P. Pawlowski, Eric C. Cyr	Developing Objective and Quantifiable Processes for Assessing Software Component Usefulness
Valeri N. Vasquez, Richard Barnes	Using big data to study and improve scientific software
Kelley Ruehl, Katherine Klise	Improving adoption of scientific software best practices through the development of a software Impact Factor
Vivek Kale	More Frequent and Regular Engagement for Success of Scientific-Software
Boyana Norris	Joint Analysis of Development Processes and Code Quality
David M. Rogers	Investigating Software Developer Roles Supporting Reproducibility
Kate Keahey	Software as a Scientific Instrument
Karol Kowalski, Erdal Mutlu, Ajay Panyala, Sotiris Xantheas	Sustainability of Scalable Computational Chemistry Software at PNNL
Sandra Gesing, Maytal Dahan, Linda Hayden, Marlon Pierce, Claire Stirm, Michael Zentner	SGCI: Advancing Access to Scientific Software via Science Gateways Improves the Trust within the Computational Sciences Landscape
Anita Carleton	Architecting the Future of Software Engineering

Continued on next page

Table C.1 – ***continued from previous page***

Authors (Corresponding Author Underlined)	**Title and Link**
Jim Kowalkowski, Marc Paterno, Saba Sehrish	High Energy Physics and Scientific Software Development

Appendix D

Workshop Participants

Table D.1: Summary of registered participants. Company types are self-reported and have not been verified.

Category	Type	Count
Total registrations		264
Registration type	Invited	150
	Observers	113
	Support staff	1
Company type	U.S. DOE	22
	Industry	18
	Laboratory	146
	Other federal	10
	Support organization	1
	University	67
Additional roles	Organizers	4
	U.S. DOE point of contact	1
	Keynote speakers	3
	Lightning speakers	30
	Breakout leaders	31
	Report contributors	34

Table D.2: Registered Workshop Participants. Type denotes invited participants (I), observers (O), or support staff (S). Roles include: organizer (O), U.S. DOE Point of Contact (POC), keynote speaker (KS), lightning speaker (LS), breakout leader (BL), and contributor to the final report (R).

Name	Company	Type	Roles
Agarwal, Deb	Lawrence Berkeley National Laboratory	I	
Ahern, Sean	Ansys, Inc.	I	
Ahn, Dong	Lawrence Livermore National Laboratory	I	BL
Allu, Srikanth	Oak Ridge National Laboratory	O	
Anzt, Hartwig	University of Tennessee	I	BL, R
Appling, Alison	U.S. Geological Survey	O	

Continued on next page

Table D.2 – *continued from previous page*

Name	Company	Type	Roles
Arndt, Daniel	Oak Ridge National Laboratory	O	
Arthur, Richard	GE Research	I	
Balaprakash, Prasanna	Argonne National Laboratory	I	LS
Baldwin, Breck	Safety 3rd	I	LS
Bangalore, Purushotham	University of Alabama	O	
Bangerth, Wolfgang	Colorado State University	I	
Barker, Michelle	Research Software Alliance	I	R
Barnes, Richard	Lawrence Berkeley National Laboratory	O	LS
Beattie, Keith	Lawrence Berkeley National Laboratory	I	
Bel, Oceane	Pacific Northwest National Laboratory	I	LS
Bernholdt, David	Oak Ridge National Laboratory	I	O, R
Bethel, Wes	Lawrence Berkeley National Laboratory	I	
Bhatele, Abhinav	University of Maryland	I	LS
Blake, Renee	New York University	O	
Bradley, Pete	Pratt & Whitney	I	
Brewster, Aaron	Lawrence Berkeley National Laboratory	I	BL, R
Brown, Benjamin	U.S. DOE Office of Advanced Scientific Computing Research	I	
Brown, Joshua	Oak Ridge National Laboratory	O	R
Budiardja, Reuben	Oak Ridge National Laboratory	O	
Byna, Suren	Lawrence Berkeley National Laboratory	O	
Candy, Jeff	General Atomics	I	LS
Carleton, Anita	Carnegie Mellon University Software Engineering Institute	I	
Carr, Leslie	University of Southampton, U.K.	I	R
Carver, Jeff	University of Alabama	I	BL, R
Cary, John	Tech-X Corporation	I	O
Chang, Iris	SLAC National Accelerator Laboratory	O	
Chen, Jin	Princeton Plasma Physics Laboratory	I	
Chiang, Nai-Yuan	Lawrence Livermore National Laboratory	O	
Cho, Younghyun	University of California, Berkeley	O	
Choi, Sou Cheng Terrya	Illinois Institute of Technology	O	R
Cholia, Shreyas	Lawrence Berkeley National Laboratory	I	
Chue Hong, Neil	Software Sustainability Institute / University of Edinburgh	I	
Clark, Struan	Computational Science Center, National Renewable Energy Laboratory	I	
Cohoon, Johanna	University of Texas at Austin	I	KS
Cojean, Terry	Karlsruhe Institute of Technology	I	
Conzelmann, Craig	U.S. Geological Survey	O	
Cosden, Ian	Princeton University	I	KS
Cowart, Julie	University of Delaware	O	
Crawford, Daniel	Virginia Tech/Molecular Sciences Software Institute	I	LS
Crivelli, Silvia	Lawrence Lawrence Berkeley National Laboratory	O	

Continued on next page

Table D.2 – *continued from previous page*

Name	Company	Type	Roles
Crozier, Paul	Sandia National Laboratories	I	
Crystal-Ornelas, Robert	Lawrence Berkeley National Laboratory	I	
Deelman, Ewa	University of Southern California	I	LS
Denny, Joel	Oak Ridge National Laboratory	O	
Ding, Yuhan	Illinois Institute of Technology	O	
Doak, Peter	Oak Ridge National Laboratory	O	
Donato, David	U.S. Geological Survey	O	
Dong, Bin	Lawrence Berkeley National Laboratory	I	R
Doucet, Mathieu	Oak Ridge National Laboratory	I	
Du, Caifan	The University of Texas at Austin	I	
Dubey, Anshu	Argonne National Laboratory	I	BL
Dudson, Benjamin	Lawrence Livermore National Laboratory	I	LS, R
Dyadechko, Vadim	ExxonMobil	I	
Eisty, Nasir	Boise State University	O	
Elwasif, Wael	Oak Ridge National Laboratory	O	
Fadel, Nur	CSCS	O	
Fagnan, Kjiersten	Lawrence Berkeley National Laboratory	I	R
Ferenbaugh, Charles	Los Alamos National Laboratory	O	
Ferreira da Silva, Rafael	Oak Ridge National Laboratory	O	
Finkel, Hal	U.S. DOE Office of Advanced Scientific Computing Research	O	POC
Fiore, Stephen	University of Central Florida	O	R
Foertter, Fernanda	NextSilicon	I	
Fortney, Jon	Oak Ridge National Laboratory	O	
Friedman, Alex	Lawrence Livermore National Laboratory	O	
Gallo, Andy	GE Research	I	LS
Gamblin, Todd	Lawrence Livermore National Laboratory	I	BL
Gaston, Derek	Idaho National Laboratory	I	
Georgakoudis, Giorgis	Lawrence Livermore National Laboratory	O	
Georgiadou, Antigoni	Oak Ridge National Laboratory	I	
Gesing, Sandra	Discovery Partners Institute, University of Illinois Chicago	I	LS, R
Ghosh, Sayan	Pacific Northwest National Laboratory	I	
Giudicelli, Guillaume	Idaho National Laboratory	O	
Godoy, William	Oak Ridge National Laboratory	O	
Gopalakrishnan, Ganesh	University of Utah	I	BL
Gordon, Janice	U.S. Geological Survey	O	
Graziani, Carlo	Argonne National Laboratory	I	
Guillen, Donna	Idaho National Laboratory	O	
Gunter, Dan	Lawrence Berkeley National Laboratory	I	BL
Guo, Yanfei	Argonne National Laboratory	I	
Gupta, Rinku	Argonne national laboratory	I	
Halappanavar, Mahantesh	Pacific Northwest National Laboratory	O	
Hall, Claude	Illinois Institute of Technology	O	

Continued on next page

Table D.2 – *continued from previous page*

Name	Company	Type	Roles
Halper, Andrew	U.S. Geological Survey	O	
Harbour, Logan	Idaho National Laboratory	O	
Harrington, Kyle	Oak Ridge National Laboratory	I	LS
Harrison, Cyrus	Lawrence Livermore National Laboratory	I	
Haupt, Carina	German Aerospace Center (DLR)	I	
Hawk, Carol	U.S. DOE Office of Advance Scientific Computing Research	O	
Heber, Gerd	The HDF Group	I	
Heroux, Michael	Sandia National Laboratories	I	O
Hettrick, Simon	Software Sustainability Institute	I	R
Hickernell, Fred J.	Illinois Institute of Technology	I	R
Hoffman, Bill	Kitware	I	BL
Hovland, Paul	Argonne National Laboratory	O	
Howison, James	University of Texas at Austin	I	
Hsu, Darren	Oak Ridge National Laboratory	O	
Huang, Tsung-Wei	University of Utah	O	
Jacob, Robert	Argonne National Laboratory	O	
Jansen, Gustav	Oak Ridge National Laboratory	O	
Jardin, Stephen	Princeton Plasma Physics Laboratory	O	
Jay, Caroline	University of Manchester	I	LS
Kale, Vivek	Brookhaven National Laboratory	I	BL
Kapadia, Anuj	Oak Ridge National Laboratory	O	
Kashi, Aditya	Karlsruhe Institute of Technology	O	
Kasiraju, Sashank	University of Delaware	O	
Katz, Daniel S.	University of Illinois at Urbana-Champaign	I	LS, BL, R
Keahey, Kate	Argonne National Laboratory	I	
Klasky, Scott	Oak Ridge National Laboratory	O	
Klise, Katherine	Sandia National Laboratories	O	
Kolla, Hemanth	Sandia National Laboratories	I	
Kong, Fande	Idaho National Laboratory	O	
Kowalkowski, Jim	Fermi National Accelerator Laboratory	O	
Krishnan, Hari	Lawrence Berkeley National Laboratory	I	BL
Laguna, Ignacio	Lawrence Livermore National Laboratory	I	LS, R
Laird, Carl	Carnegie Mellon University	I	
Landin, Kirk	Sandia National Laboratories	I	
Laney, Daniel	Lawrence Livermore National Laboratory	I	BL
Lebrun-Grandie, Damien	Oak Ridge National Laboratory	O	
Lee, Steven	U.S. DOE Office of Advanced Scientific Computing Research	O	
Lenhardt, Chris	Renaissance Computing Institute (RENCI) - University of North Carolina-Chapel Hill	I	LS
Lesmes, David	U.S. Geological Survey	O	
Leung, Mary Ann	Sustainable Horizons Institute	I	LS, BL
Li, Sherry	Lawrence Berkeley National Laboratory	I	BL

Continued on next page

Table D.2 – *continued from previous page*

Name	Company	Type	Roles
Li, Ang	Pacific Northwest National Laboratory	O	
Li, Xingpeng	University of Houston	O	
Liao, Chunhua	Lawrence Livermore National Laboratory	I	
Liebschner, Dorothee	Lawrence Berkeley National Laboratory	O	
Lin, Huey-Wen	Michigan State University	I	
Lindberg, John	Electric Power Research Institute	O	
Littlewood, David	Sandia National Laboratories	I	
Liu, Yang	Lawrence Berkeley National Laboratory	O	
Liu, Xu	North Carolina State University	I	
Lofstead, Jay	Sandia National Laboratories	I	BL
Luszczek, Piotr	University of Tennessee	O	
Maleki, Morteza	Tarbiat Modares University	O	
Malik, Abid	Brookhaven National Laboratory	O	
Malviya-Thakur, Addi	Oak Ridge National Laboratory	I	BL
Marathe, Aniruddha	Lawrence Livermore National Laboratory	O	
Marques, Osni	Lawrence Berkeley National Laboratory	I	
Marsico, Carli	University of Washington	O	
Mateevitsi, Victor	Argonne National Laboratory	O	
McDonnell, Marshall	Oak Ridge National Laboratory	I	
McInnes, Lois	Argonne National Laboratory	I	O, R
McKeen, Shannon	Renaissance Computing Institute (RENCI)	O	
McLemore, Jacob	University of Tennessee	I	
Mehta, Kshitij	Oak Ridge National Laboratory	O	
Meinecke, Annika	German Aerospace Center (DLR)	I	LS
Mellor-Crummey, John	Rice University	O	
Meng, Xiaozhu	Rice University	I	
Menon, Harshitha	Lawrence Livermore National Laboratory	O	
Menzies, Tim	North Carolina State University	I	
Mervin, Brenden	Electric Power Research Institute	O	
Mikaitis, Mantas	University of Manchester	O	
Milewicz, Reed	Sandia National Laboratories	I	BL, R
Miller, Mark	Lawrence Livermore National Laboratory	I	
Miller, Ross	Oak Ridge National Laboratory	I	
Mohror, Kathryn	Lawrence Livermore National Laboratory	O	
Montenegro, Davis	Electric Power Research Institute	I	
Moussa, Jonathan	Molecular Sciences Software Institute, Virginia Tech	O	
Mudafort, Rafael	National Renewable Energy Laboratory	O	
Mundt, Miranda	Sandia National Laboratories	I	LS
Munson, Todd	Argonne National Laboratory	I	R
Mutlu, Erdal	Pacific Northwest National Laboratory	I	
Nayak, Pratik	Karlsruhe Institute of Technology	O	R
Newson, Jeremy	U.S. Geological Survey	O	
Nichols, Daniel	University of Maryland, College Park	O	
Norris, Boyana	University of Oregon	I	LS

Continued on next page

Table D.2 – *continued from previous page*

Name	Company	Type	Roles
Oliver, Samantha	U.S. Geological Survey	O	
Oryspayev, Dossay	Brookhaven National Laboratory	O	
Paine, Drew	Lawrence Berkeley National Laboratory	I	
Parashar, Manish	Scientific Computing and Imaging (SCI) Institute, University of Utah	I	R
Parasyris, Konstantinos	Lawrence Livermore National Laboratory	O	
Paterno, Marc	Fermi National Accelerator Laboratory	I	BL
Pawlowski, Roger	Sandia National Laboratories	I	
Pearce, Olga	Lawrence Livermore National Laboratory	I	BL
Peisert, Sean	Lawrence Berkeley National Laboratory	I	
Peles, Slaven	Oak Ridge National Laboratory	I	LS, R
Perciano, Talita	Lawrence Berkeley National Laboratory	I	
Perr-Sauer, Jordan	National Renewable Energy Laboratory	I	LS
Peterson, Peter	Oak Ridge National Laboratory	I	BL
Pino, Robinson	U.S. DOE Office of Advanced Scientific Computing Research	O	
Pirkelbauer, Peter	Lawrence Livermore National Laboratory	I	
Plimpton, Steve	Sandia National Laboratories	I	KS
Podgorney, Robert	Idaho National Laboratory	O	
Pollard, Samuel	Sandia National Laboratories, California	I	
Poon, Sarah	Lawrence Berkeley National Laboratory	I	
Poon, Billy	Lawrence Berkeley National Laboratory	O	
Pourkargar, Davood	Kansas State University	O	
Pourmal, Elena	The HDF Group	O	
Prokopenko, Andrey	Oak Ridge National Laboratory	O	
Rajamanickam, Siva	Sandia National Laboratories	I	LS
Ram, Karthik	University of California, Berkeley	I	
Raybourn, Elaine M.	Sandia National Laboratories	I	LS, BL, R
Richard, Ryan	Ames Laboratory	I	
Riley, Katherine	Argonne National Laboratory	I	
Ringler, Todd	Science and Technology Policy Institute	I	LS
Rogers, David	Oak Ridge National Laboratory	I	BL, R
Romero Alcalde, Eloy	Thomas Jefferson National Accelerator Laboratory	O	
Roth, Philip	Oak Ridge National Laboratory	I	LS, R
Rouson, Damian	Lawrence Berkeley National Laboratory	I	BL
Ruehl, Kelley	Sandia National Laboratories	I	BL
Sachdev, Jai	Princeton Plasma Physics Laboratory	I	
Sattasathuchana, Tosaporn	Ames Laboratory	I	
Saxe, Paul	Molecular Sciences Software Institute, Virginia Tech	O	
Schissel, David	General Atomics	O	
Seal, Sudip	Oak Ridge National Laboratory	I	
Sehrish, Saba	Fermi National Accelerator Laboratory	O	
Serra, Edoardo	Boise State University and Pacific Northwest National Laboratory	I	

Continued on next page

Table D.2 – *continued from previous page*

Name	Company	Type	Roles
Sexton-Kennedy, Elizabeth	Fermi National Accelerator Laboratory	I	LS
Shiraiwa, Syun'ichi	Princeton Plasma Physics Laboratory	I	
Shrager, Jeff	xCures, Stanford Symbolic Systems Program (adjunct), and SLAC National Accelerator Laboratory/Linac Coherent Light Source (consulting)	I	
Simon, Horst D.	Lawrence Berkeley National Laboratory	I	
Sims, Benjamin	Los Alamos National Laboratory	I	LS, R
Sjaardema, Gregory	Sandia National Laboratories	O	
Skjellum, Anthony	University of Tennessee at Chattanooga	O	
Slaughter, Elliott	SLAC National Accelerator Laboratory	I	
Smith, Jared	U.S. Geological Survey	O	
Smith, Spencer	McMaster University	I	LS, R
Sochat, Vanessa	Lawrence Livermore National Laboratory	I	
Sommer, Thorsten	German Aerospace Center (DLR)	I	
Sorokin, Aleksei	Illinois Institute of Technology	O	
Sottile, Matthew	Lawrence Livermore National Laboratory	I	BL, R
Stewart, Graeme	CERN	O	
Suetterlein, Joshua	Pacific Northwest National Laboratory	I	
Suset-Bennett, Ceren	U.S. DOE Office of Advanced Scientific Computing Research	O	
Sutherland, Will	University of Washington	O	
Sutherland, Kevin	Apogee Engineering, contractor to U.S. Geological Survey Advanced Research Computing team	O	
Tallent, Nathan	Pacific Northwest National Laboratory	I	BL
Terry, Deneise	Oak Ridge Institute for Science and Education	S	
Terry, Jeff	Illinois Institute of Technology	I	
Thakur, Rajeev	Argonne National Laboratory	O	
Tharrington, Arnold	Oak Ridge National Laboratory	I	
Thayer, Jana	SLAC National Accelerator Laboratory	I	LS
Tracey, Jeff	U.S. Geological Survey	O	
Tripp, Charles	National Renewable Energy Laboratory	I	
Turcksin, Bruno	Oak Ridge National Laboratory	O	
van Oostrum, Rene	HPE/Cray	O	
Vasquez, Valeri	University of California, Berkeley	O	
Vu, Elizabeth	Sloan Foundation	O	
Walter, Peter	SLAC National Accelerator Laboratory	O	BL
Watson, Gregory	Oak Ridge National Laboratory	I	R
Weber, Nic	University of Washington	I	R
White, Robert	National Renewable Energy Laboratory	O	
Whitfield, Ross	Oak Ridge National Laboratory	O	
Whitney, Ben	Oak Ridge National Laboratory	O	
Wiese, Jason	University of Utah	I	R
Willenbring, James	Sandia National Laboratories	I	BL
Wolf, Matthew	Oak Ridge National Laboratory	I	BL

Continued on next page

Table D.2 – *continued from previous page*

Name	Company	Type	Roles
Wong, Emma	Electric Power Research Institute	O	
Wozniak, Justin	Argonne National Laboratory	I	R
Wu, John	Lawrence Berkeley National Laboratory	I	BL
Xu, Haowen	Oak Ridge National Laboratory	I	
Yoginath, Srikanth	Oak Ridge National Laboratory	O	
Zhang, Chen	Oak Ridge National Laboratory	I	R
Zhang, Xiaohua	Lawrence Livermore National Laboratory	O	
Zhou, Hui	Argonne National Laboratory	I	
Zhou, Wenduo	Oak Ridge National Laboratory	O	
Zisman, Sagi	National Renewable Energy Laboratory	I	

Appendix E

Pre-Workshop Report

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.