Community Software Facility Discovery Workshop Report:

Scientific software best practices, tools, and culture

Soudeh Kamali
Dominique Colegrove
Sheri Voelz
Katelyn FitzGerald

NCAR Technical Notes
NCAR/TN-592+PROC

National Center for
Atmospheric Research
P. O. Box 3000
Boulder, Colorado
80307-3000
www.ucar.edu

NCAR IS SPONSORED BY THE NSF

NCAR TECHNICAL NOTES

http://library.ucar.edu/research/publish-technote

The Technical Notes series provides an outlet for a variety of NCAR Manuscripts that contribute in specialized ways to the body of scientific knowledge but that are not yet at a point of a formal journal, monograph or book publication. Reports in this series are issued by the NCAR scientific divisions, serviced by OpenSky and operated through the NCAR Library. Designation symbols for the series include:

EDD – Engineering, Design, or Development Reports
Equipment descriptions, test results, instrumentation, and operating and maintenance manuals.

IA – Instructional Aids
Instruction manuals, bibliographies, film supplements, and other research or instructional aids.

PPR – Program Progress Reports
Field program reports, interim and working reports, survey reports, and plans for experiments.

PROC – Proceedings
Documentation or symposia, colloquia, conferences, workshops, and lectures. (Distribution maybe limited to attendees).

STR – Scientific and Technical Reports
Data compilations, theoretical and numerical investigations, and experimental results.

The NSF National Center for Atmospheric Research (NCAR) is operated by the nonprofit University Corporation for Atmospheric Research (UCAR) under the sponsorship of the National Science Foundation. Any opinions, findings, conclusions, or recommendations expressed in this publication are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

NSF National Center for Atmospheric Research
P. O. Box 3000
Boulder, Colorado 80307-3000

NCAR/TN-592+PROC

NCAR Technical Note

2026-04

Community Software Facility Discovery Workshop Report: Scientific software best practices, tools, and culture

Soudeh Kamali
Dominique Colegrove
Sheri Voelz
Katelyn FitzGerald

NSF National Center for Atmospheric Research (NCAR), Boulder, CO, USA

NCAR Library

NATIONAL CENTER FOR ATMOSPHERIC RESEARCH
P. O. Box 3000
BOULDER, COLORADO 80307-3000
ISSN Print Edition 2153-2397
ISSN Electronic Edition 2153-2400

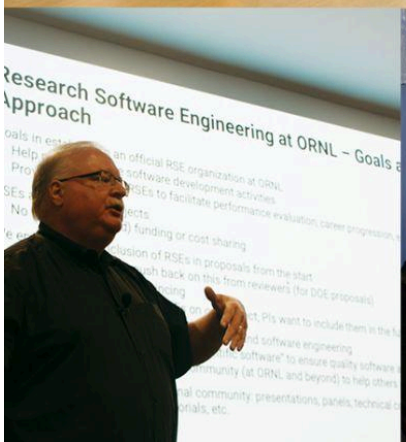
How to Cite this Document:

Soudeh Kamali, Dominique Colegrove, Sheri Voelz, Katelyn FitzGerald. (2026). Community Software Facility Discovery Workshop Report: Scientific software best practices, tools, and culture. (No. NCAR/TN-592+PROC). doi:10.5065/q5jz-k286

NCAR
OPERATED BY UCAR

Community Software Facility Discovery Workshop Report

Scientific software best practices, tools, and culture



Community Software Facility
Discovery Workshop Report

Scientific software best practices, tools, and culture

April 2026

Authors: Soudeh Kamali[1], Dominique Colegrove[1], Sheri Voelz[1], and Katelyn FitzGerald[1]

Edited and reviewed by: Thomas Hauser[1], Jon Petch[1], Gretchen Mullendore[1], Glen Romine[1], Sarah Sheard[2,3], Regina Griego[2,4], Jared Hendrickson[5], and Ian Cosden[6]

[1] U. S. National Science Foundation National Center for Atmospheric Research, Boulder, CO, USA

[2] International Council on Systems Engineering, West Lafayette, IN, USA

[3] Retired, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, USA

[4] Retired, Sandia National Laboratories, Albuquerque, NM, USA

[5] Center for Research Computing, University of Notre Dame, Notre Dame, IN, USA

[6] Princeton Research Computing, Princeton University, Princeton, NJ, USA

This material is based upon work supported by the NSF National Center for Atmospheric Research, which is a major facility sponsored by the U.S. National Science Foundation under Cooperative Agreement No. 1755088.

Acknowledgments

The authors would like to express their sincere gratitude to the following individuals and institutions for their contributions to the Community Software Facility (CSF) discovery workshop and the report:

- Editing and Visuals: Leigh Harney and Krista Laursen (proofreading and editing); and Amy Brokaw (figure design and visual assets).
- Workshop Presenters and Panelists: Simon Vosper, Charles Ewen, David Simonin, Josh Wilson, Oakley Brunt (UK Met Office); Robin Yeman (Leidos); David Bernholdt (Oak Ridge National Laboratory); Regina Griego (Sandia National Laboratories and INCOSE); Jerome Hugues (CMU Software Engineering Institute); Cena Brown, Kyle Shores, Helen Kershaw, Allison Baker, Adrianna Foster (NSF NCAR); Aaron Donahue (Lawrence Livermore); Sandra Gesing (US-RSE); Caleb Reinking (University of Notre Dame); Ian Cosden (Princeton University); and Miranda Mundt (Sandia National Laboratories) for their insights and contributions to workshop discussions.
- Workshop Planning Committee: Special thanks to the team at NSF NCAR (Cena Brown, Lalo Torres, Daniel Howard, Kate Kuzemka, Elizabeth Faircloth, and Brian Dobbins); and Jared Hendrickson (University of Notre Dame) for organizing the workshop, and to Everette Joseph, Thomas Hauser, Gretchen Mullendore, Jon Petch, and Glen Romine for their guidance and leadership.
- Administration and On-Site Support: Kailyn Kampert, Keila DeBellis, and Teresa Walz. We would also like to acknowledge the NSF NCAR Multimedia team for their audio/visual support.

Finally, we are immensely grateful to all attendees for contributing their time and expertise throughout the workshop.

Table of Contents

List of Tables

List of Figures

Executive Summary

This document synthesizes the key findings from a workshop held in August 2025 to inform software design approaches and culture norms to consider implementing within the NSF NCAR Community Software Facility (CSF). The CSF discovery workshop convened internal and external experts to address the escalating complexity of Earth system modeling, the rapid evolution of computing architectures, and the need for modern software development practices.

A major takeaway from the workshop was that improving the quality and sustainability of scientific software requires coordinated investment in people, processes, and organizational culture. This includes integrative teams that align scientific objectives with software design, focused requirements capturing and software architectural planning, structured decision-making, and broad adoption of modern software engineering practices and tools. These efforts must be enabled by deliberate culture change that explicitly recognizes software engineers as critical partners in scientific software development and design, provides dedicated time for training and collaboration, and a commitment to continuous review and improvement of processes and culture.

The CSF discovery workshop also highlighted that lasting culture change requires visible leadership commitment and the creation of organizational norms that value software quality, collaboration, learning, and shared ownership of outcomes. Additionally, effective scientific software development is grounded in consistent documentation and knowledge transfer. More information on the motivation, workshop guiding questions, and discussions can be found in the following sections. Detailed key findings can be found in the “Workshop Results: Key Findings and Recommendations” section.

1. Introduction

1.1 Motivation

Earth system modeling complexity continues to escalate as the community pursues km-scale modeling, increased representation of complex processes, ensemble simulations, and the integration of massive volumes of remotely-sensed observations. Additionally, rapid changes and innovations in processor architectures, along with the rise of Artificial Intelligence and Machine Learning (AI/ML), offer opportunities to improve code performance and add value to Earth system modeling. NSF NCAR intends to launch a Community Software Facility (CSF) to modernize how it develops, supports, and sustains community models and tools to better meet the current and future needs of staff and the broader Earth system science community. Among the facility goals are to: ensure NSF NCAR software and infrastructure is portable, interoperable, easy to maintain, and optimized across different compute systems; provide governance for informed investments and priorities; deliver new capabilities while appropriately balancing development standards; and explore new technologies and/or methodologies. It should be noted that at the time of this workshop, the CSF concept was in development, hence some of these goals may later change or be refined.

The CSF also has many cultural and best practice goals to further support our software development teams and the development of Earth system science tools. The facility will ensure training and professional development opportunities for software engineers[1], employ best practices that standardize software development

[1] Software engineers perform a wide range of work. For the purpose of this workshop, the focus is on NSF NCAR software engineers and other staff who perform research software engineer type work. A Research Software Engineer (RSE) is defined as "those who regularly use expertise in programming to advance research. This includes researchers who spend a significant amount of time programming, full-time software engineers writing code to solve research problems, and those somewhere in-between. RSEs aspire to apply the skills and practices of software development to research to create more robust, manageable, and sustainable research software." [1] It should be noted that NSF NCAR does not currently have a formal RSE job category. Responsibilities commonly associated with RSEs are performed within the existing software engineering job category and at times by individuals within other job categories.

methods across the center's models and related software, and provide a community of practice for software engineers. As part of these efforts, the CSF discovery workshop was held in August 2025 with internal and external participants to discuss scientific software best practices, tools, and culture. This workshop was organized to inform the process and framework of scientific software development within the CSF.

The goals of the CSF discovery workshop were to:

- Learn from best practices at NSF NCAR and external organizations to develop ideas that can inform the process and framework of scientific software development at NSF NCAR;
- Identify ways to improve efficiency in communications, collaboration, and decision-making during the development of scientific software between software engineers and scientists (referred to throughout the workshop documentation as co-development); and
- Hear how other organizations provide support, professional development, and develop culture and communities of practice within their software engineering teams.

The presentations, panels, and breakout discussions at the event were utilized to inform this report and answer the following guiding questions:

I. What training is needed for our scientists and software engineers?[2]
II. How do we get started, and where do we want to go with best practices?
III. What culture changes are necessary?
IV. What are models for decision making, communications, and work practices?
V. How do we empower the NSF NCAR Software Engineering Assembly (SEA) as a Community of Practice?

These guiding questions will be revisited in the "Workshop Results: Key Findings and Recommendations" section.

[2] After further discussions during the workshop this question was later revised to "What additional skillsets, roles, training, and support is needed to improve quality and sustainability of scientific software development ?".

1.2 Date and Location

The CSF discovery workshop was held on August 20-21, 2025 in a hybrid format with some participants joining online and others attending in person at the NSF NCAR Mesa Laboratory in Boulder, Colorado.

1.3 Agenda and Methodology

A planning committee consisting of project managers, research engineers, High Performance Computing (HPC) consultants, and software engineers across NSF NCAR and external to the organization prepared the agenda (Appendix A) and identified speakers to meet the goals of the workshop. Members of the planning committee were as follows:

- Cena Brown, NSF NCAR Computational and Information Systems Lab (CISL);
- Sheri Voelz, NSF NCAR CISL;
- Lalo Torres, NSF NCAR CISL;
- Soudeh Kamali, NSF NCAR High Altitude Observatory (HAO);
- Katelyn FitzGerald, NSF NCAR CISL;
- Daniel Howard, NSF NCAR CISL;
- Domi Colegrove, NSF NCAR Directorate's Office (DO);
- Kate Kuzemka, NSF NCAR DO;
- Brian Dobbins, NSF NCAR Climate and Global Dynamics Laboratory (CGD); and
- Jared Hendrickson, University of Notre Dame.

The planning committee also routinely met with NSF NCAR leadership (Glen Romine, DO; Thomas Hauser, CISL; Jon Petch, CGD; Gretchen Mullendore, Mesoscale and Microscale Meteorology (MMM) Laboratory; and Everette Joseph, DO) to ensure that the workshop objectives aligned with organizational strategies.

The workshop agenda was designed to provide numerous opportunities for participants to hear from speakers at various organizations through presentations, panels, and to provide input through breakout discussions with documented deliverables. As part of registration, participants provided information on topics they

would be most interested in and current software development challenges they are facing. This information helped inform the workshop's presentation topics and discussion questions.

1.4 Representation

A total of 104 participants attended the CSF discovery workshop, both in-person (49) and virtually (55). 68 were internal to NSF NCAR and the University Corporation for Atmospheric Research (UCAR), and 36 represented external computing centers, national laboratories, international organizations, the private sector, and academia. Job categories of attendees included senior management, scientists and researchers, software engineers and developers, systems engineers, software solution architects, product owners, faculty members, postdocs, and graduate students.

2. Sessions Synthesis

2.1 Welcome

The NSF NCAR Director, Everette Joseph, kicked the event off with introductory remarks on the history of NSF NCAR (NSF's largest Federally Funded Research and Development Center [FFRDC]) and the vision for readily configurable, interoperable software components spanning timescales from minutes to centuries, and across scientific disciplines. He spoke to NSF NCAR being a leader in the community modeling space and also discussed how the CSF concept vision will build on that success and continue democratizing access to Earth system research tools.

E. Joseph also emphasized that NSF NCAR is working to usher in a new paradigm of development at NSF NCAR, where scientific requirements, software capabilities, and solutions are developed collaboratively by integrated teams, hereinto referred to as "co-development". As the Center works toward this goal, particularly in the current environment of rapid technology development and limited resources, NSF NCAR doesn't want to reinvent the wheel. Rather, the center is looking to leverage best practices, identify opportunities, and adapt them to NSF NCAR's scope and mission.

Gretchen Mullendore, an Associate Director of NSF NCAR and Director of the Center's Mesoscale and Microscale Meteorology (MMM) Laboratory, built on E.Joseph's message by providing specific examples of NSF NCAR's global leadership in community-developed modeling (Community Earth System Model (CESM), Model for Prediction Across Scales (MPAS), etc.) and highlighting that NSF NCAR practices creative problem solving and delivers great accomplishments despite smaller team sizes (e.g., MPAS consists of only 11 full-time positions) and budget constraints.

Thomas Hauser, Director of the Computational and Information Systems Lab (CISL) and Associate Director of NSF NCAR, presented the Center's CSF Concept. The CSF, currently on a two-year timeframe, intends to improve and grow how NSF NCAR develops, supports, and sustains community models and tools to better meet the current and future needs of staff and the broader Earth system science community.

T. Hauser emphasized the importance of learning from other disciplines and testing ideas through community input.

2.2 Keynote

Charles Ewen, Technology Director and Chief Information Officer of the Met Office, and Simon Vosper, Executive Director of Science at the Met Office, jointly presented a keynote talk titled "*Software Engineering at the Met Office*." C. Ewen emphasized that, similar to NSF NCAR, the Met Office is continuously responding to a rapidly changing technological landscape and the need for new and innovative tools to address predictability and prediction of weather and climate.

C. Ewen and S. Vosper highlighted that the number of scientific software engineers (SSEs)[3] employed by the Met Office has tripled over the past decade, with the number of SSEs now exceeding 130. There were three transformational programs that drove the demand for increased SSEs and are key to delivering on the Met Office's strategy: a supercomputing upgrade, next-generation modeling systems (NGMS), and data exploitation and pipeline design. **The SSEs are part of three main groups: dedicated SSE teams, embedded SSEs working within science teams, and deployable SSEs to support shorter-term projects.** The latter is a relatively new innovation.

A primary topic of the keynote was the Met Office's balance between enabling pioneering research and being able to deliver production-level code. They refer to their approach to this transition as a Research, Development, Operations model, which works to seamlessly move code from facilitating research (performing analysis and research using existing tooling and practices) to development (post-processing science), and finally to operations (cloud-based, rapid release). The Met Office utilizes the Spotify scalability model for their Agile development processes. **C. Ewen and S. Vosper both credited strong leadership in managing change and addressing issues in initially resistant teams who became enthusiastic adopters of the new processes.**

[3]At the Met Office the Scientific Software Engineer is the job category that most closely aligns with RSE-type work.

In terms of career progression, the Met Office offers a multitude of technology roles across all seniority levels within the organization and **provides opportunities for their SSEs to progress in both technical and leadership roles.**

2.3 Presentations

Each day of the workshop was organized around a central theme, with two invited speakers presenting under each theme. The theme for Day One was "Software Engineering Successes and Lessons Learned", and the theme for Day Two was "Design Decisions" (how to approach large-scale projects, including design, processes, and feature development). The following sections provide an overview of the presentations delivered under each theme and highlight the principal findings and insights shared by the speakers.

2.3.1 Software Engineering Successes and Lessons Learned

The focus of the presentations in this session was on understanding how other organizations have successfully executed large-scale scientific software development and the lessons they have learned along the way. The first talk in this section, "Bridging the Gap Between Scientists and Software Engineers to Accelerate Solutions," was presented by Robin Yeman, Senior Software Solution Architect from Leidos, who brought extensive experience across both academia and industry. The second talk in this section, "Research Software Development Experiences at Oak Ridge National Laboratory," was delivered by David Bernholdt, a distinguished Research and Development staff member in the Computer Science and Mathematics Division of Oak Ridge National Laboratory (ORNL).

2.3.1.1 Bridging the Gap Between Scientists and Software Engineers to Accelerate Solutions

In her presentation, Robin Yeman discussed how principles of systems engineering and Agile methodologies provide a powerful framework for improving communication, decision-making, and collaboration in large-scale software and science projects. **A central theme of her talk was the importance of viewing large-scale projects as socio-technical systems that integrate people, processes,**

and technology. For this reason, *"Industrial DevOps*[4]*"* would be more suitable than a pure software-centric Agile approach. As illustrated in Figure 1, a socio-technical system emphasizes the interdependence of technical performance and human collaboration, underscoring that success depends not only on engineering but also on organizational structure and culture.

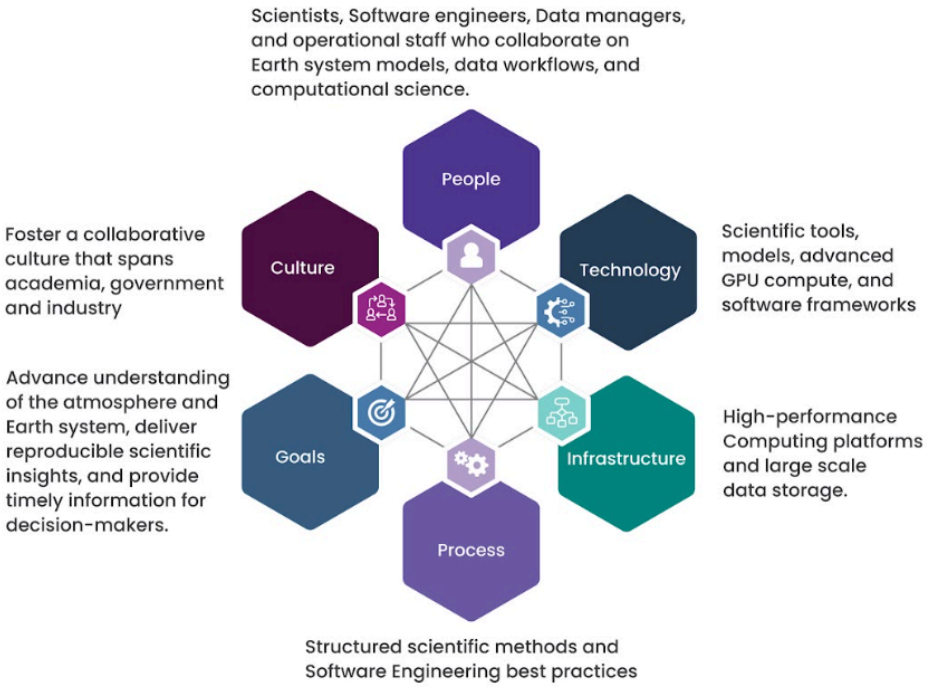


Figure 1: Socio-technical systems (credit Robin Yeman).

R. Yeman introduced "*Industrial DevOps*" as a framework for successfully delivering large-scale socio-technical systems and shared nine guiding principles or success patterns for this approach which is illustrated in Figure 2 and detailed in the following paragraphs [2].

The industrial DevOps framework was developed with the following goals in mind [2], which align well with the goals of NSF NCAR:

- Delivery of value in the shortest, sustainable lead time;

[4] DevOps is a combination of development (Dev) and Operations (Ops) and refers to a set of practices, principles, and cultural philosophies that aim to shorten the software development cycle and deliver high-quality software continuously.

- improved collaboration and knowledge sharing across functional areas;
- building a competitive advantage through rapid learning and experimentation;
- improving quality;
- improving customer/user satisfaction; and
- yielding happier, more engaged employees.

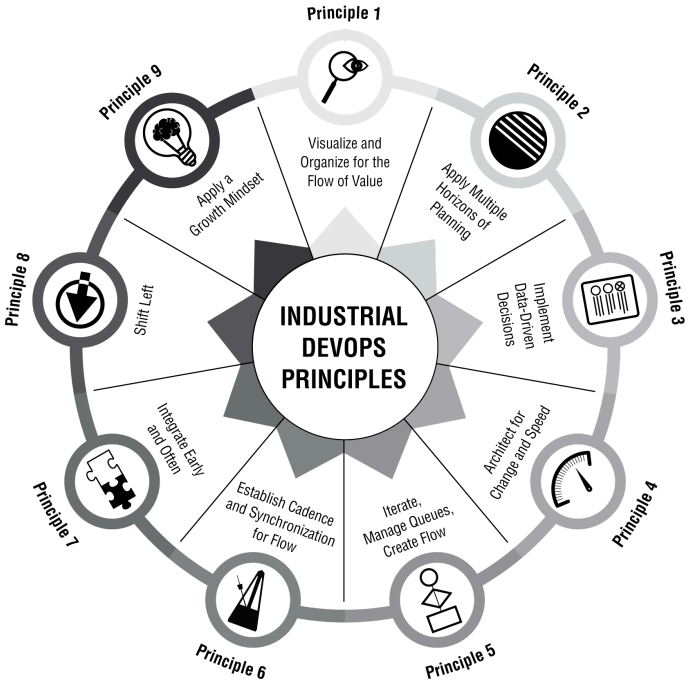


Figure 2: Industrial DevOps principles [2].

I. Organize for Flow of Value

Organizing around the flow of value means structuring teams and processes so work moves seamlessly from idea to delivery. This minimizes silos and handoffs. Organizing for flow of value replaces traditional functional departments with cross-functional teams that own end-to-end delivery, enabling continuous integration, testing, and deployment. By focusing on user value rather than internal roles, organizations become more efficient, adaptive, and collaborative, accelerating delivery and improving quality.

II. Apply Multiple Horizons of Planning

Effective organizations plan across multiple horizons to stay aligned between long-term goals and short-term execution. The “strategic horizon” defines mission outcomes and priorities, while the “operational horizon” translates these outcomes and priorities into near-term objectives. The “execution horizon” drives daily implementation through iterative delivery. Together, these horizons create a continuous feedback loop that replaces rigid long-term planning with empirical planning through the use of short cycles, data-driven insights, and minimum viable products to refine objectives and guide adaptive progress.

III. Implement Data-Driven Decisions

According to S. Johnson and R. Yeman in Industrial DevOps [2], data-driven decision-making is essential to creating transparency, reducing risk, and improving outcomes across complex, long-lifecycle systems. Rather than relying on assumptions, intuition, or rigid plans, data-driven organizations use objective evidence gathered from continuous integration, testing, and operational feedback to guide their decisions. This approach ensures that choices about design, priorities, and investments are grounded in measurable system performance and user outcomes. By integrating real-time data into decision processes, teams can identify emerging issues early, adapt quickly, and continually align their work with organizational goals and user needs, fostering a culture of learning and continuous improvement.

IV. Architect for Change and Speed

Achieving both change and speed requires designing systems with modular, loosely coupled architectures that allow components to evolve independently while maintaining overall integrity. R. Yeman pointed out that many scientific codes remain monolithic because they are developed by domain experts who may not have formal training in software architecture, which can be problematic.

V. Iterate, Manage Queues, and Create Flow

Use short sprints to move all parts of a project forward. Even if interim results are not deployable, these results generate insights that improve the overall system

development. Time-boxing is a valuable technique here, enabling predictability and consistent data collection.

VI. Establish Cadence and Synchronization

Large-scale system development involves multiple teams working in parallel on interdependent components. By establishing shared milestones and synchronization points, teams can coordinate dependencies, surface integration issues earlier, and prevent inefficiencies that arise when sub-systems evolve in isolation. Even if individual teams may appear to be more productive by working independently, consistent synchronization ensures that all components fit together into a functioning system.

VII. Integrate Early and Often

Invest in continuous integration environments early in the project. Although this may appear to be slower at first, early frequent integration ultimately accelerates system development and reduces rework. This principle carries a dual meaning. From Agile practices, it inherits continuous integration and testing of software, and from systems engineering, it inherits early and iterative integration of subsystems, models, and physical components.

VIII. Shift Left

In both Agile and systems engineering, “shift-left” means moving quality-related activities, such as testing, to an earlier stage of the development lifecycle to identify and resolve issues sooner. In Agile, this approach emphasizes embedding testing and feedback within rapid, iterative development cycles. In systems engineering, however, the concept is broader. It extends the early integration of quality practices to include compliance verification, quality assurance, security assessment, and design validation. The intent is to prevent defects and design flaws from propagating through the system and to reduce rework costs through the application of validation requirements and architectures as early as possible.

IX. Apply a Growth Mindset

Applying a growth mindset means embracing the belief that abilities and performance can improve through effort, learning, and feedback rather than being

fixed traits. This mindset encourages teams to view challenges, failures, and changes as opportunities to learn and adapt.

R. Yeman concluded her presentation by addressing audience questions about how the principles she discussed apply in environments with finite research funding or constrained resources. **She emphasized that not all principles are equally applicable to every project, and that the choice of which to adopt should always be guided by a clear understanding of the desired outcomes. In her experience, three principles, "organizing around the flow of value," "shift-left," and "time-boxing," have proven consistently valuable across projects of varying scale and complexity.** She noted that one common pitfall in research and development, particularly in government and small business settings, is building solutions in search of a problem. To avoid this, **project teams must first ensure that stakeholder needs are well understood before development begins.** Applying the shift-left approach early in the process helps clarify actual user needs and prevents wasted effort on unnecessary or misaligned features. Likewise, time-boxing helps teams focus on tangible outcomes and make iterative progress within realistic budget and schedule constraints.

When asked how communication about needs can be improved among teams that "speak different languages", whether disciplinary, organizational, or technical, R. Yeman advised **spending time at the outset to build a shared vocabulary**, or what she called a "Rosetta Stone" for the project. Investing this time during the planning phase fosters mutual understanding and greatly reduces confusion later in the project. **Establishing this common language early is one of the most effective ways to bridge communication gaps and align expectations across diverse collaborators.**

2.3.1.2 Research Software Development Experiences at Oak Ridge National Laboratory

David Bernholdt's presentation provided an overview of evolution, organizational structure, and lessons learned from ORNL's ongoing Research Software Engineering (RSE) initiative [3].

D. Bernholdt began by recounting the history of RSE development at ORNL. Around 2015-2016, the term RSE began circulating in ORNL, prompting discussions about what a formal RSE capability could look like in a large research institution. In 2017, the first RSE team was incubated within one of the research groups in the Computer Science and Mathematics Division (CSMD). By 2019, the team had expanded rapidly and had become a standalone RSE group within CSMD. Participation in the RSE group was entirely voluntary, with staff being encouraged but not required to join.

The establishment of a formal RSE organization at ORNL aimed to professionalize software development across research projects and provide an institutional home for RSEs to support career progression and community building. The initiative sought to create a framework that would:

- Strengthen the interface between domain science and software engineering, emphasizing the "engineering of scientific software" to ensure quality software and quality science;
- Foster a professional community of scientific software practitioners, engaged in national and international activities such as workshops, panels, and standards development; and
- **Ensure sustainable funding through project-based support**, **with RSEs included in proposals from the outset.**

D. Bernholdt shared several key lessons learned from ORNL's experience integrating RSEs across a range of projects:

I. Design and Requirements

D. Bernholdt emphasized that obtaining high-quality requirements from domain specialists is often one of the most challenging aspects of research software development. User stories are frequently too narrow or incomplete to guide effective engineering work. In many cases, domain scientists are deeply invested in their own prototypes and fixed on specific implementation details, which can hinder broader design exploration. Additionally, they may be unaware of similar or existing solutions already developed within other organization-wide projects.

To address these challenges, RSEs often engage in design exploration and rapid prototyping to test ideas and refine system concepts, which **are essential for aligning design choices with scientific requirements and objectives.** Dr. Bernhlott noted that **RSEs should be encouraged and supported to spend time on design and prototyping.**

II. Communication and Collaboration

Frequent and effective communication between domain scientists and RSEs is critical for success. Misaligned terminology or assumptions can easily derail progress, especially in multidisciplinary teams. **Establishing shared vocabularies and expectations** helps ensure clarity and mutual understanding.

III. Recognition and Culture

Organizational culture plays a crucial role in sustaining productive RSE engagement. Project leadership must ensure that RSE contributions are visible, valued, and properly credited. **A healthy culture respects both scientific insight and software craftsmanship, recognizing that impactful science increasingly depends on reliable, well-engineered software.**

D. Bernholdt emphasized that expectations around the RSE role must be clearly defined. RSEs are development partners, and it is important for organizations to ensure they are valued for their technical contributions in scientific collaborations. Understanding RSEs' motivations, expertise, and evaluation criteria is key to integrating them effectively into research teams.

IV. Exchange of Knowledge

D. Bernholdt noted that successful multidisciplinary teams are open and able to learn from each other. RSEs need to be open to learning domain knowledge to converse with scientists and understand their scientific goals. Similarly, scientists benefit from learning and applying better software engineering practices, fostering a two-way exchange of knowledge that strengthens collaboration.

V. Sustainable Integration

Sustainability in RSE engagement depends on flexible integration models. Different projects require different approaches, some benefit from fully embedded RSEs, while others operate more effectively with centralized or shared teams. The optimal structure depends on factors such as code complexity, quality requirements, and long-term maintenance plans.

D. Bernholdt also highlighted the funding challenges associated with sustaining RSE roles. At ORNL, RSEs are project-funded, rather than centrally supported. This makes early inclusion in funding proposals essential for securing their participation. While principal investigators required some persuasion, most who collaborated with RSEs quickly recognized their technical and organizational value and continued to include them in subsequent proposals. On the other hand, while large grants typically have enough flexibility to include RSE support, smaller projects may struggle to accommodate both domain researchers and RSEs. Developing mechanisms to ensure continuity of RSE involvement across funding cycles remains an ongoing priority for ORNL and the broader RSE community.

2.3.1.3 Summary of the Software Engineering Successes and Lessons Learned Presentations

The presentations in the "Software Engineering Successes and Lessons Learned" session emphasized that **sustainable progress in large-scale scientific software development depends on combining disciplined engineering practices with adaptive, collaborative approaches, and project management across multiple horizons (strategic and operational).** R. Yeman highlighted how principles from systems engineering and Agile methodologies (e.g., modular design, iterative development, early integration, and "shift-left" testing) can enhance reliability, speed, and alignment with real user needs in complex research environments. D. Bernholdt complemented this perspective by describing ORNL's establishment of a formal RSE organization to professionalize scientific software development, strengthen collaboration between scientists and engineers, and ensure long-term software sustainability through structured processes, clear requirements, and iterative design. Both speakers emphasized that effective communication is fundamental to bridging disciplinary boundaries and ensuring alignment among

different collaborators. They also highlighted the importance of cultivating a growth mindset that encourages learning across scientific and engineering domains.

2.3.2 Design Decisions

The presentations in this section focused on approaches to designing and managing large-scale software engineering projects, with particular attention to effective processes. The overarching goal was for the sessions' outcomes to inform future discussions on technology governance and software development processes across complex scientific and engineering projects. The first talk, titled "Systems and Software Engineering," was delivered by Regina Griego, a distinguished Research and Development Systems Engineer (retired) from Sandia National Laboratories (Sandia) and a Fellow of the International Council on Systems Engineering (INCOSE). The second presentation, titled " Development of the JEDI-Based Observation Processing and Data Assimilation System" was delivered by David Simonin, Manager of the Assimilation and Surface-based observations group at the Met Office.

2.3.2.1 Systems and Software Engineering

R. Griego's talk provided a broad overview of standards, processes, and practices that support effective large-scale software and systems development. She emphasized that **understanding and applying systems engineering principles, including conceptual design, requirements engineering, and architecture, are key to improving quality, consistency, and long-term success.**

I. Standards of Systems and Software Engineering

R. Griego began by introducing two foundational standards that provide structure and best practices for developing and managing complex software and systems. The first is the Capability Maturity Model Integration (CMMI) [4], which focuses on process maturity and organizational capability. The second standard, the International Organization for Standardization/International Electrotechnical Commission/Institute of Electrical and Electronics Engineers (ISO/IEC/IEEE) 15288 [5], defines the system life cycle framework, offering guidance on how systems and software evolve from concept through retirement.

A. CMMI

The CMMI [4] is an internationally recognized framework that guides organizations in improving process quality, consistency, and performance. It integrates principles from software engineering, systems engineering, and project management to promote a structured approach to continuous improvement. At its core, CMMI emphasizes repeatability, the idea that only defined and consistent processes can be measured and improved.

The model defines five levels of organizational maturity, progressing from ad hoc practices to fully optimized processes. The description of each maturity level is summarized in Table 1. Most research-focused organizations, such as NSF NCAR, typically operate between Levels 1 and 3, with Level 3 representing a practical goal. At this level, processes are standardized, documented, and applied consistently, enabling systematic decision-making that involves stakeholders and relies on defined criteria rather than informal judgment.

Table 1: Summary of processes involved for each CMMI maturity level (Credit: Regina Griego and [4]).

Maturity Level	Focus	Process Areas
Level 1 Initial		Unpredictable and reactive.
Level 2 Managed	Basic Project Management	• Configuration Management • Measurement and Analysis • Process and Product Quality Assurance • Project Monitoring and Control • Project Planning • Requirements Management • Supplier Agreement Management
Level 3 Defined	Process Standardization	• Decision Analysis and Resolution • Integrated Project Management • Organizational Process Definition • Organizational Process Focus • Organizational Training

		• Product Integration • Requirements Development • Risk Management • Technical Solution • Validation • Verification
Level 4 Quantitatively Managed	Quantitative Management	• Organizational Process • Quantitative Project Management
Level 5 Optimizing	Continuous Process Improvement	• Casual Analysis and Resolution • Organizational Performance

B. ISO/IEC/IEEE 15288, System and Software Engineering System Life Cycle Processes

The ISO/IEC/IEEE 15288 [5] standard provides an internationally accepted framework for managing system life cycle processes, from initial concept and development through operation, maintenance, and retirement. **This standard bridges systems and software engineering**, **recognizing that software is itself a type of system**, and that both disciplines rely on shared principles of structure, verification, and traceability.

One of the motivations behind ISO/IEC/IEEE 15288 is knowledge continuity. In modern organizations, where personnel turnover is frequent, **well-documented and standardized processes are essential to maintaining organizational memory, quality, and consistent performance.** The standard also defines a Systems Engineering Management Plan (SEMP) as a tool to capture how processes are applied and managed throughout the life cycle.

II. Project Management Processes

R. Griego, briefly touched on the importance of project management processes. She recognized project management as an enabling discipline fundamental to the success of systems. **ISO/IEC/IEEE 15288 emphasizes that project management is essential for ensuring that activities are planned, resourced, monitored, and**

managed effectively throughout the system's life cycle. It provides an organizational framework that enables technical processes to achieve their intended outcomes.

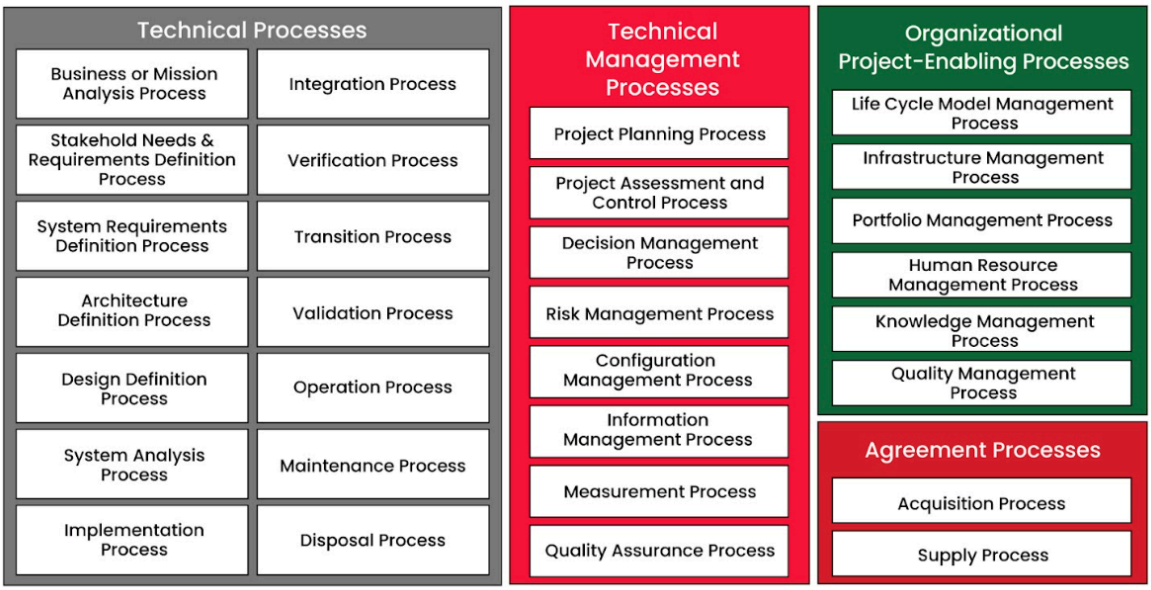


Figure 3: ISO/IEC/IEEE 15288 systems and software engineering - system life cycle processes [5].

III. Conceptual Design Process

R. Griego emphasized that conceptual design is a critical phase of any system or software project. It begins with a detailed problem analysis where we attempt to fully understand what problem is to be solved before considering how to solve it. Once the problem space is well understood, teams can explore the solution space, evaluating potential architectures against decision criteria such as performance, cost, maintainability, environmental impact, usability, etc.

Although the conceptual design phase may account for only a small fraction of the total project effort (often around 8% of the budget), **decisions made at this stage commit up to 70% of the total life cycle cost**, since they effectively determine the system's architecture and direction as illustrated in Figure 4. Thus, **early design decisions, whether explicit or implicit, have lasting financial and technical consequences.**

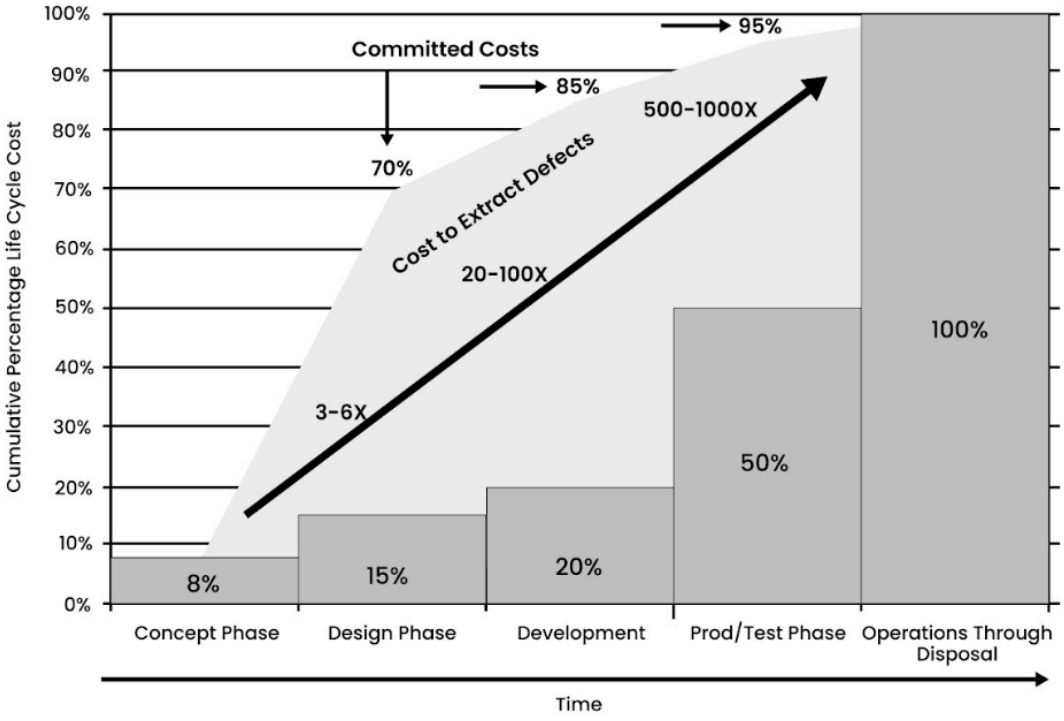


Figure 4: Cumulative percentage life cycle cost over time (credit: Defense Systems Management College (1993), as reproduced in [6]).

R. Griego continued this section of the presentation by sharing examples of formal concept and solution selection processes. One valuable tool presented for concept selection was the Pugh decision matrix [7], which compares a list of several design alternatives based on pre-selected evaluation criteria. This is a tool systems engineers regularly use during decision-making. **One of the benefits of using tools such as the Pugh matrix in decision-making is that they create an environment where the team works together against the problem rather than against each other, fostering collaboration and a more objective solution selection process.**

A. Importance of Requirements Engineering

Within conceptual design, requirements engineering plays a foundational role. It involves identifying, analyzing, and validating what the system must do, how well it must perform, and under what conditions. Requirements can come from diverse sources: users, legacy systems, technical constraints, regulatory standards, and environmental factors. These inputs must be normalized and traced to ensure

consistency and completeness across the system. To underscore the critical role of well-defined requirements in successful software design, R. Griego referenced Frederick Brooks's classic essay "No Silver Bullet" [8]:

"The hardest single part of building a software system is deciding precisely what to build. No other part of the conceptual work is so difficult as establishing the detailed technical requirements.... No other part of the work cripples the resulting system if done wrong. No other part is more difficult to rectify later."

R. Griego further distinguished between source requirements (needs derived directly from stakeholders or users) and system requirements (technical specification derived through analysis and formally presented). **Source or stakeholder requirements define the "what" from a user perspective, while the system requirements detail the "how" from a technical, developer perspective. If the developers jump to writing system requirements without understanding the original intent, need, and goal, they risk building a technically sound system that fails to solve the user's actual problem.** Once requirements are established, they must be traced through design, implementation, verification, and validation to ensure that the final system fulfills its intended purpose.

B. Architecture and the Role of the Systems/software Architect

Translating stakeholder needs into system and software architectures lies at the heart of software systems engineering. Bugs or implementation defects are relatively easy to fix; poorly defined requirements or flawed architectures are not. System architecture defines how components are organized and interact to achieve desired outcomes. Architecture provides multiple views and viewpoints, analogous to how building blueprints separate plumbing, electrical, and structural diagrams. The architect's role is therefore to ensure coherence across decisions, balance competing trade-offs, and preserve the technical and conceptual integrity of the system as it evolves. **R. Griego described systems and software architects as the most technically vital and strategic roles in an engineering organization as they maintain system integrity, scalability, and vision**.

R. Griego's description of system architecture and system architect aligns well with Dr. Frederick Brook's definition of software architecture and the software architect's role. F. Brooks led the development of IBM's System/360 computer family and its operating system, and his ideas have been the groundwork for the concept of software architecture. He is best known for authoring "The Mythical Man-Month" [9], and "No Silver Bullet" [8]. In "No silver bullet," he defines the architect as the person (or small group) responsible for maintaining conceptual integrity. In the "Mythical Man-Month", he describes conceptual integrity as the most important quality of great software systems.

In software and systems engineering, tools such as Systems Modeling Language (SysML) [10] and Unified Modeling Language (UML) [11] enable modeling the architecture (from both the structure and behavior viewpoints), ensuring the system is both technically sound and comprehensible to diverse stakeholders. UML is the standardized visual language for modeling the structure and behavior of software systems, helping teams design, document, and communicate software system architecture [11]. SysML extends UML for systems engineering, enabling modeling of hardware, software, data, people, and processes within an integrated, model-based framework [10].

IV. Socio-Technical Systems

R. Griego also offered a valuable perspective on the socio-technical nature of software systems. **She explained that while a piece of software or code can be viewed as the immediate system of interest, it exists within a broader context, one that includes people, tools, workflows, and organizational structures that shape its development and use.** This larger socio-technical system is inherently complex, and understanding its dynamics is essential for successfully creating and maintaining effective software systems. R. Griego's perspective closely aligns with the themes introduced earlier in the workshop by R.Yeman, who emphasized the interdependence between technical systems and the human or organizational factors that support them.

V. Incorporating Requirements and Architecture Development Process at Sandia National Laboratories

At Sandia, R. Griego led an initiative to integrate requirements and architecture development processes and tools across the organization, an effort she described as "changing the DNA of Sandia." She helped establish the Systems Engineering Excellence Team (SEET), tasked with institutionalizing systems engineering practices and cultivating a culture of continuous improvement. Key elements of this initiative included:

- Assessing existing software capabilities and identifying gaps in process maturity.
- Collaborating with internal and external education and training groups to create and deliver courses in systems engineering and requirements management.
- Introducing CMMI principles to strengthen consistency and decision-making.
- Implementing tools for requirements management, and later for architecture and model-based design.

Over several years, this team worked to align Sandia's engineering culture with systems engineering best practices, focusing on planning, traceability, and shared technical vision. The result was a measurable improvement in communication, decision consistency, and cross-team collaboration.

2.3.2.2 Development of the JEDI-Based Observation Processing and Data Assimilation System

D. Simonin's talk centered around the design and development of an observation processing and data assimilation system by his group during a critical Met Office transition of adopting JEDI as their software framework. In the late 2010s, the Met Office launched the Next Generation Modeling Systems (NGMS) program to fundamentally redesign its modeling infrastructure. The goal was to create a unified Earth environment prediction framework, known as Momentum, that would enable the Met Office and its partners to fully exploit the emerging generation of high-performance computing systems. This initiative represented a major strategic and technical shift for the organization, which required taking a different approach

than the usual one. D. Simonin summarized the approach as illustrated in Figure 5. In the following sections, details of this approach are presented.

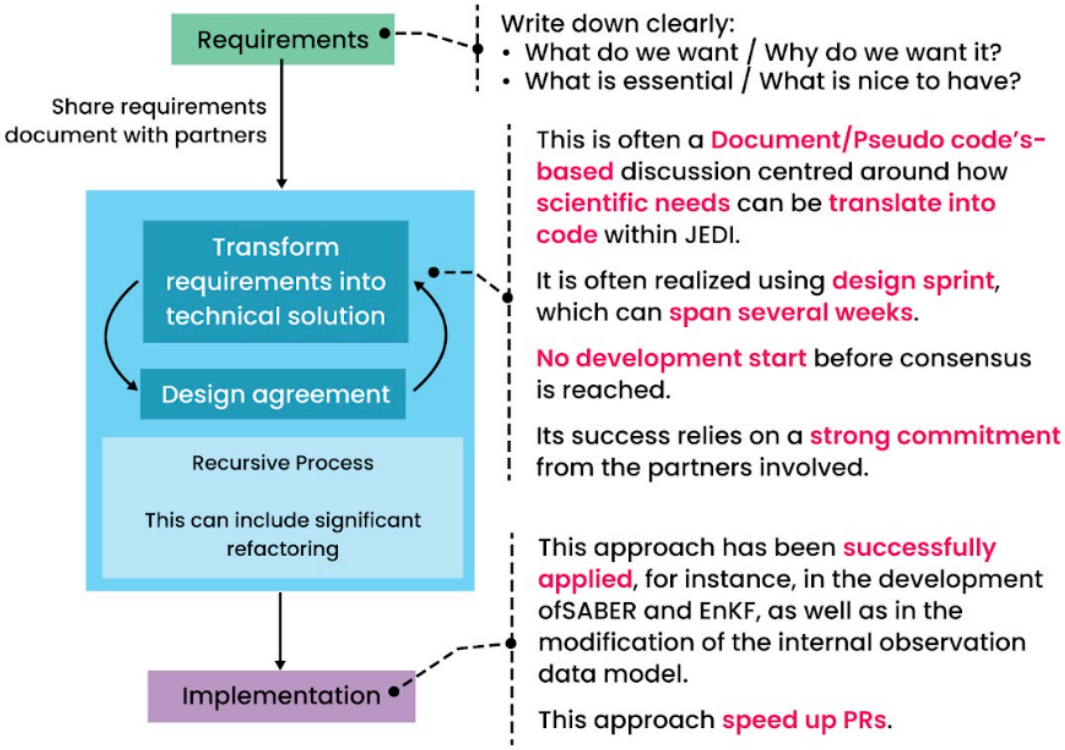


Figure 5: Schematic representation of the approach taken by the Met Office Next Generation Observation Processing and Data Assimilation Group as part of NGMS (Credit: Met Office).

I. Problem Definition and Requirements

The Met Office legacy infrastructure (built organically over more than 25 years) had become increasingly fragile and resistant to change. Although originally well-designed, decades of incremental updates had pushed the code base beyond its intended architecture. Each modification risked introducing unintended side effects, making system management complex and time-consuming.

At the same time, new challenges were emerging. These included: rapidly evolving HPC architecture (e.g., GPU and cloud computing), increasing model complexity (e.g., coupled data assimilation, kilometer-scale modeling, machine learning integration), and expanding observation and data streams (e.g., satellites and drones). D. Simonin explained that to address these challenges and needs, the team established a clear set of objectives for the new infrastructure. The system needed to be:

- Unified across applications - spanning atmosphere, ocean, land systems, from research to operations;
- Versatile - applicable to data assimilation, nowcasting, and re-analysis;
- Generic - avoiding hardcoded scientific assumptions;
- Model-agnostic - supporting everything from toy models to fully coupled Earth system models; and
- Collaborative and easy to use - promoting open development and community contribution.

Considerable effort went into defining and documenting clear, prioritized requirements, specifying what was needed, why, and which elements were essential versus optional. The resulting documentation was shared with all partners to ensure alignment before advancing to implementation.

II. Decision Making and Planning

D. Simonin emphasized that significant time was dedicated to translating the agreed requirements into technical solutions and top-level designs. Development did not begin until full consensus was reached, recognizing that success depended on a strong, shared commitment from all partners. These high-level designs and plans were then iteratively refined into actionable work packages, with clearly defined deliverables and measures of success. Because of extensive interdependencies among components, the process was inherently interactive, requiring continuous reassessment and coordination as development advanced. Following the refinement of the high-level designs into actionable work packages, each package was overseen by a domain expert and **implemented by a multidisciplinary team of scientists and software engineers.**

III. Implementation and Continuous Integration

D. Simonin shared that throughout development. The team fully embraced Agile workflows, emphasizing iterative delivery, shared objectives, and cross-team communication. Regular meetings, short development cycles, and clearly defined milestones kept the effort aligned with long-term goals.

Continuous integration and automated testing became central to ensuring code resilience. Using GitHub, the team maintained a single "source of truth" containing code, configuration, test results, and reference outputs. This was essential for ensuring full transparency and reproducibility.

IV. Methodological and Organizational Shift

Overall, the time and effort invested in clearly defining requirements, establishing a design process, and conducting thorough planning have yielded significant positive impacts. To accomplish the goals and objectives, the team took a major departure from traditional Met Office practices by:

- Embracing collaborative, open-source development;
- Transitioning from Fortran to C++, improving modularity and abstraction;
- Adopting Agile methodologies, promoting flexibility, and interactive progress; and
- Incorporating a higher level of abstraction in software design to support portability and extensibility.

2.3.2.3 Summary of Design Decisions Presentations

Both presentations emphasized that **sound design and architecture begin with well-defined requirements and disciplined processes**. R. Griego highlighted how frameworks such as CMMI and ISO/IEC/IEEE 15288 provide the foundation for structured design by promoting traceable requirements, systematic decision-making, and architecturally guided software development. She illustrated how investing early in conceptual design and requirements engineering establishes the clarity and structure needed for long-term system integrity. Similarly, D. Simonin described how the Met Office's next-generation modeling initiative relied on thorough requirements capture, consensus-driven architecture planning, and interactive design refinement. Together, the talks reinforce that an **effective architecture emerges not from coding alone but from rigorous upfront design and shared understanding of what the system must achieve and why.**

2.4 Panels

To examine scientific software development from multiple perspectives, the workshop included three panel sessions covering best practices, collaborative use of tools, and professional development and culture. The sections that follow provide a detailed account of the insights that emerged from each panel.

2.4.1 Panel A: Better Practices

The first panel session focused on how NSF NCAR and other organizations have conducted scientific software development, including some of their best practices and lessons learned. Panelists included: Miranda Mundt (Research Software Engineer, Sandia National Laboratories), Helen Kershaw (Software Engineer, NSF NCAR), Josh Wilson (Senior Software Developer, Met Office), and Jerome Hugues (Principal Researcher, CMU Software Engineering Institute).

A recurring theme throughout the session was that an organization's culture must actively value software quality. Without that cultural commitment, it quickly falls to the bottom of the priority list. This requires leadership support and a culture of psychological safety.

The session revolved around four common themes: I. Implementation of Practices, II. Organizational Structure and Roles, III. Team Dynamics and Conflict Resolution, and IV. External Contributions and Knowledge Transfer. The following sections provide details on each theme.

I. Implementation of Practices

Agile was discussed as a philosophy rather than a rigid practice. This was explained by a panelist who stated that agility is about listening to people and continuously optimizing processes. The Met Office encourages Agile practices, allowing groups to be autonomous and define the processes within which they operate. Regardless of which Agile processes are adopted, it is essential to clearly define project goals, break tasks down appropriately, and track progress as tasks are completed. **It is also important to schedule regular meetings to reflect on what was working and identify areas for improvement.**

The panelists noted that it is crucial to maintain delivery metrics and to demonstrate progress. This builds trust with the team and outside stakeholders by being transparent about the progress. It also ensures quality in the project's deliverables and alignment across the project.

The panel emphasized the importance of understanding what is right for each project and determining the appropriate level of process. M. Mundt's team at Sandia has developed a tiered approach to software quality. **Their approach involves applying different software practices based on the maturity level of the software package.** For example, a software package in its early stages of development is recommended to be under version control. As the project matures, it is recommended that regular testing be done within an automated framework. As it matures further, an additional recommendation is to have documentation in place. A full description of their tiered approach can be found in their paper "A Tiered Approach to Scientific Software Quality Practices" [12].

II. Organizational Structure and Roles

It is essential to create balanced teams with the experience and expertise necessary to meet project goals including areas such as project management and systems engineering in addition to scientific and software engineering expertise. **This is achieved by bringing the right people to a project based on their strengths, rather than solely on their job titles.** This point, raised by the panel, aligned well with the industrial DevOps principle of "organizing for the flow of value" that was brought up in the "Bridging the Gap Between Scientists and Software Engineers to Accelerate Solutions" talk by R. Yeman earlier in the first day of the workshop.

During this discussion, the panelists also emphasized **the importance of establishing an informal structure, in which software engineers/RSEs rotate roles and responsibilities over time to ensure redundancy in project knowledge.** They also suggested that roles should flex according to project needs and where individuals want to grow.

With regards to code ownership and the complexities of open-source software, the panel discussed how on many occasions there could be multiple owners for different

pieces of software. This leads to lack of clarity in maintenance responsibilities of interfaces; which is particularly problematic for large open-source projects. In addition, it was acknowledged that turnover and transition are inevitable, and it is important to have multiple experts and clear documentation for sustainability.

III. Team Dynamics and Conflict Resolution

Strong team culture was highlighted as essential for sustaining effective software engineering practices. **Panelists emphasized the value of collective responsibility, in which coding decisions are never left to a single individual and ownership is shared across the group.** This approach fosters accountability while also promoting psychological safety, enabling team members to take risks and contribute ideas without fear of retribution or blame. Additionally, regular team-building and social activities help further strengthen trust.

The panelists noted that teams thrive when given autonomy to shape their own processes, provided they have clear role definitions and high-level guidance to keep projects aligned. On large projects that involve multiple teams, cross-team workshops and architectural discussions offer opportunities to step back, establish shared direction, and prevent projects from going off track.

Regarding disagreements in decision-making, the panel recommended focusing on "team vs. the problem" and not on individuals. However, for larger disagreements, Sandia refers to Project Management Committees to resolve issues. The importance of clear decision-making processes brought up by the panelists was also emphasized in the "Systems and Software Engineering" talk by R. Griego.

IV. External Contributions and Knowledge Transfer

The panel emphasized the importance of having clear contributor guides to help onboard new contributors and establish expectations for involvement. **Providing documentation was seen as essential for lowering barriers and encouraging sustained contributions to a project**. Successful teams not only manage contributors' expectations but also prioritize strategic support for key users and contributors, **recognizing that investing time in individuals who can, in turn, assist others accelerates community growth.**

Effective knowledge transfer was identified as crucial to maintaining long-term project health. Documenting knowledge prevents critical expertise from remaining siloed and known only to a select few. The Met Office highlighted their practice of encouraging team members to take on tasks outside their immediate expertise, fostering cross-training and reducing the risk of single points of failure. By actively sharing knowledge and broadening participation, teams build resilience and ensure that projects remain maintainable even as personnel and priorities change.

2.4.2 Panel B: How Can We Use Tools to Work More Effectively Together?

The second panel focused on tools that facilitate better collaboration and efficient work between software developers, researchers, and other roles involved in scientific software development. Panelists included: Kyle Shores (Software Engineer, NSF NCAR), Aaron Donahue (Staff Research Scientist, Lawrence Livermore), Allison Baker (Project Scientist, NSF NCAR), Adrianna Foster (Project Scientist, NSF NCAR), and Oakley Brunt (Scientific Software Engineer, Met Office). Figure 6 shows the panelists online and in-person discussing tools and collaboration strategies during this session. The discussion led to insights on core collaborative tools, technology, and language debates, thoughts on Artificial Intelligence (AI) tool integration, and collaboration strategies. More information on the tools discussed during this panel and the workshop can be found in Appendix B.

I. Core Collaboration Tools

GitHub, a web-based platform built around the version control system Git, was recognized as a powerful platform for managing and collaborating on code. It allows teams to track changes, review contributions, maintain clear version histories, and provides a platform for project management. Teams and projects that were present for the discussion described it as a central part of their collaborative software development. While the panel agreed it is an excellent tool, they also noted that advanced features can be difficult for mixed-experience teams, often requiring additional training or support from software engineers/RSEs. GitHub Actions was highlighted as particularly valuable, as it enables automated testing and continuous integration, ensuring quality and consistency across contributions.

Asynchronous communication tools such as Slack and Google Chat help bridge communication gaps, providing real-time updates and discussions, which can lead to fewer and shorter meetings. Use of Confluence and Google Docs further supports these workflows by documenting decisions, collecting requirements, and organizing project details.

For development workflows, a range of environments and visualization tools support collaboration and problem-solving. Visual Studio Code (VS Code) provides a streamlined approach to coding, integrates directly with GitHub, and enhances developer productivity. Excalidraw, a virtual whiteboard tool, has proven especially useful during virtual meetings, allowing teams to sketch workflows, illustrate software dependencies, and collaborate in real time.

Figure 6. Panelists online and in-person discussing tools and collaboration strategies.

II. Technology and Language Debates

Fortran continues to be recognized for its strengths, particularly its readability and forgiving nature, which make it approachable for scientific computing tasks. However, several limitations were raised, including the long-term availability of compiler development and the challenges of debugging Fortran applications, which

may affect the sustainability of projects that continue to rely heavily on the language.

The discussion on porting and modernization highlighted the complexity of working with legacy code. Panelists and the audience weighed the pros and cons of rewriting software entirely versus directly porting existing codebases. While rewriting can yield cleaner and more maintainable solutions, it also requires significant time and resources. One proposed way forward was to create abstract C objects to facilitate integration with Fortran, potentially offering a middle path between full rewrites and direct ports.

Performance portability emerged as another key theme, with Kokkos identified as a promising approach for porting codes to run on emerging architectures, such as GPUs.. By providing an abstraction layer for parallel programming, Kokkos enables applications to run efficiently across various types of architectures without requiring code rewrite. One caveat is that this approach requires training for software engineers/RSEs as well as scientists to enable them to contribute productively.

III. AI Tool Integration

AI-assisted development tools are becoming an integral part of coding workflows, providing valuable support for tasks like code completion and debugging. GitHub Copilot, ChatGPT, and other AI tools can generate useful code suggestions that help developers quickly address routine or boilerplate tasks. It was also noted that GitHub Copilot can provide initial reviews to expedite the process of merging contributions. Together, these tools create a more streamlined workflow, allowing teams to focus on higher-level design decisions while automation handles repetitive tasks. However, their use also comes with some caveats. Developers must remain knowledgeable about the requests they make to these systems and take responsibility for validating and testing the proposed solutions. The output from Copilot or ChatGPT, for example, may not always be correct or optimal, therefore it is essential to have a human in the loop for a review before adoption.

The panelists agreed that when used carefully, these tools can significantly improve efficiency while also serving an educational role by clarifying language abstractions

and offering insights into alternative coding approaches. Overall, they are best viewed as accelerators for simple tasks and learning aids, rather than a replacement for expertise or rigorous testing.

It should be noted that since the workshop discussions on this topic, the integration of AI assistants and agents in development workflows, has rapidly matured. Many Integrated Development Environments (IDEs) now incorporate AI agents directly into the workflow. These tools can assist developers by leveraging knowledge of team repositories, common coding practices, and broader codebase structures to support development tasks. In addition, AI agents can be valuable conversation partners when brainstorming new problem-solving strategies. AI can generate alternative solutions by detecting interdependencies that may not be immediately apparent to humans. Moreover, it is now possible to generate software largely via textual prompt.

Given current limitations in AI agents and tools including context processing limitations, incomplete or undocumented behavioral constraints, probabilistic output variability, limited long-horizon reasoning, and non-verifiable outputs, it remains necessary to actively guide, validate, and review AI-generated code. Effective use of such systems requires human oversight to ensure correctness, security, maintainability and alignment with project-specific architectural and operational standards. Nevertheless, the capacity of AI systems to provide rapid access to complex technical and domain knowledge and local context is clearly powerful and actively changing how developers work. Moving forward, it will be critical to remain engaged in broader discussions regarding the development and application of these technologies in order to provide necessary support and guidance in these areas.

IV. Collaboration Strategies

Collaboration within scientific software projects often requires carefully managed dynamics to balance expertise and requirements from both scientists and software engineers/RSEs. A parallel development strategy, where scientific requirements are gathered while technical implementation plans are developed in a shared environment, fosters mutual learning. **In this environment, scientists become more comfortable with coding practices, and engineers gain insight into the scientific**

context of the work. Bridge roles (e.g., project managers, product owners, systems engineers/architects) play a critical part in facilitating dialogue between these groups, ensuring that communication gaps are minimized and project goals remain aligned.

Software engineers/RSEs also noted that productivity improves when they are provided with clear needs rather than prescriptive instructions on how to implement a solution. This approach allows software engineers/RSEs , as well as the broader team, to participate fully in designing and shaping the solution. This aligns well with points raised by D. Simonin, R. Griego, and D. Bernholdt in their talks, where they emphasized the importance of defining needs and requirements as one of the most important steps in software design. In addition, comprehensive code documentation was emphasized, not only to explain implementation details but also to record the sources and rationale behind algorithms, thereby increasing the accessibility of the code and making it easier to contribute.

To support this collaborative model, training is seen as essential, particularly when introducing new technologies and methodologies. The panelists noted the benefits of structured training approaches such as Software Carpentries workshops, which focus on foundational computing skills, and hands-on tutorials that allow learners to engage directly with tools in a practical setting. Such training helps level the playing field with mixed-experience teams, ensuring all team members can contribute effectively and continue to build their expertise over time.

2.4.3 Panel C: Professional Development and Culture

The third panel focused on addressing and offering solutions to the unique challenges that RSEs[5] may face when entering the workforce, including professional

[5] A reminder that RSEs are defined as "those who regularly use expertise in programming to advance research. This includes researchers who spend a significant amount of time programming, full-time software engineers writing code to solve research problems, and those somewhere in-between. RSEs aspire to apply the skills and practices of software development to research to create more robust, manageable, and sustainable research software." [1] As mentioned in the Motivation section, NSF NCAR does not currently have a formal RSE job category, responsibilities commonly associated with RSEs are performed within the existing software engineering job category and at times by individuals within other

development and access to training. Another topic discussed by the panel was communities of practice and the facilitation of organizational culture change. The panelists included Ian Cosden (Princeton), Sandra Gesing (US-RSE, San Diego), Caleb Reinking (Notre Dame), and Katelyn FitzGerald (NSF NCAR, Earth System Data Science). This section began with a short presentation from each panelist, followed by an open fishbowl activity, providing opportunities for workshop participants and the panelists to contribute to the discussion.

2.4.3.1 Princeton RSE Group

The first part of I. Cosden's presentation focused on the RSE group within Princeton. This group has grown steadily since its founding, expanding from just two members in 2016 to over thirty-five members at the time of the workshop in 2025. The Princeton RSE group is housed centrally within Research Computing and supports research efforts across the entire university. Its mission is to help researchers develop efficient, scalable, and sustainable research software that enables new scientific discoveries.
Princeton follows an embedded approach to collaboration. RSEs work directly within research groups for a minimum of three months, with some collaborations extending into multi-year commitments. The RSE group prefers to assign as many as three RSEs to a single project, though many projects contain a single RSE. The RSEs focus on software development, including applying coding standards, designing algorithms, and optimizing performance. This model of embedding RSEs within scientific teams enables RSEs to develop domain-specific knowledge and establish trust with researchers.

The teams themselves are composed of members of various backgrounds. Some join directly after completing their undergraduate computer science degrees, while others come from domain-specific science programs and transition into software development, and others bring experience from industry. Some challenges they've seen include managing growth, maintaining a strong team identity, and effective knowledge transfer. I. Cosden stressed the importance of developing a shared

---

job categories. For the purpose of this workshop, the focus is on NSF NCAR software engineers and other staff who perform RSE-type work.

vocabulary within a team to ensure effective communication, which was also emphasized by speakers D. Bernholdt and R. Yeman earlier in the workshop.
During his presentation, I. Cosden described his involvement in the INTERSECT Training Program, an NSF-funded program created to address software engineering skill gaps within the research community. This program is designed for domain researchers from non-computer science backgrounds who are interested in strengthening their software engineering skills. The program is delivered as a week-long bootcamp using a Carpentries-style lesson format. Its curriculum covers core topics such as testing, CI/CD, and software design. All instructional materials are maintained as open-source resources on GitHub, ensuring accessibility and long-term value.

Beyond technical training, INTERSECT works to build communities. Networking opportunities encourage participants to connect with peers who share a commitment to advancing research through software. With new funding secured, the program will continue with four additional bootcamps scheduled between 2026 and 2029. Each session is expected to support 30 to 40 fully funded participants.

2.4.3.2 US-RSE - Beyond Bug Fixes: Professional Development and Communities in RSE Teams

The second panelist, Sandra Gesing, is the Executive Director of the United States Research Software Engineers Association (US-RSE) and Senior Researcher at the San Diego Supercomputing Center. **Since its founding in 2018 with 20 members, the US-RSE community has grown to more than 3,400 members in 2025.**

One of the core focuses of S. Gesing's presentation was defining the components of a strong Community of Practice. Successful communities are built on a shared purpose and clear identity. They must engage a broad range of stakeholders while maintaining an inclusive and supportive environment. Communities should also set measurable goals, track impact, and adapt to members' evolving needs. Just as important is cultivating a positive culture where members treat each other with respect, conflicts are resolved constructively, and growth is encouraged.

The presentation also outlined various career tracks and progression opportunities for RSEs as outlined by US-RSE [13, 14]. The first included the Technical Leadership track, which focuses on developing technical expertise and leadership. Growth is seen through the development of skills in areas such as architecture, frameworks, software development practices, technology, advanced processes, and languages. The second is a Research Focused Track that combines the focuses of software development and research. Growth is achieved in this track through the application of more advanced methods, combining scientific theory with a software engineering approach. In addition to these two tracks, Management and Hybrid Tracks allow individuals to take on project or people management roles, or to move laterally across technical, research, and leadership areas. Across all tracks, competency-based models emphasize a balance of technical expertise, research skills, project management, and power skills [15] (e.g. active listening, communication, leadership, and conflict management).

2.4.3.3 Culture Shifts and Building Cross Functional RSE Teams

Caleb Rainking, Associate Director of Research Software Engineering at the University of Notre Dame's Center for Research Computing, spoke about the evolution of the Notre Dame RSE team and the lessons learned in managing culture shifts and integrating cross-functional teams. The RSE group is housed alongside a research facility group and an HPC compute group, creating a collaborative environment within the center. The team includes 20 staff members, with 15 dedicated RSEs. In the past year, they supported 33 projects for 28 principal investigators across fifteen campus units. The RSE group has undergone several organizational phases, and today the team has reverted to an embedded approach, where RSEs are embedded within the research teams.

C. Rainking emphasized several cultural and management lessons. First, **effective RSE organizations must focus on people over process, particularly in complex socio-technical systems. In this process, leaders should consistently communicate the "why" behind decisions and build a common language through repeated, transparent messaging**. C. Rainking stressed the importance of investing in RSEs' domain knowledge. Without this investment, communication gaps can emerge and outputs may become misaligned.

Flexible management is also essential, where policies should not replace thoughtful, individual leadership. Instead, managers should listen, adapt, and be willing to revisit approaches that may not have worked in the past. C. Rainking recognized the importance of celebrating culture builders within the organization, as cultivating a positive and resilient team culture is just as crucial as achieving technical outcomes.

Finally, C. Rainking discussed the challenges of building cross-functional teams and stated that there is no single formula for creating high-functioning teams. They require time to progress through their natural stages of development. If alignment across teams is achieved, it can be extremely powerful. Sustaining such alignment requires ongoing effort, but it is central to maintaining long-term cultural health within the teams. It is not mentioning that funding models can add additional tension and complication to these efforts.

2.4.3.4 Communities as Agents of Change: Learning and Growing through Community

Katelyn FitzGerald, a Software Engineer at the NSF National Center for Atmospheric Research (NCAR), spoke about the role of Communities of Practice (CoP) as powerful agents of change. She emphasized how **CoPs foster innovation, drive learning, and support growth within open science contexts where collaboration and knowledge sharing are essential**.

A community of practice is built on three core components: domain, the shared area of interest; community, the group of people engaged in mutual learning; and practice, the body of knowledge, methods, and resources they develop together. When these components are aligned, CoPs can deliver a wide range of benefits. They empower individual learning and professional growth, support skill development, create leadership opportunities, strengthen networks, and promote decentralized, distributed collaboration.

Despite these benefits, several challenges were noted by K. FitzGerald. CoPs often struggle with sustaining engagement, securing adequate resources and institutional support, and maintaining long-term infrastructure. They must continually refine

goals and expectations while also managing onboarding and offboarding processes effectively. Volunteer-based communities often struggle with sustainability when key contributors leave and frequently find it challenging to convert surface-level participation into sustained engagement. At the same time, significant opportunities exist: CoPs can strengthen learning within and across communities, facilitate sharing of challenges and strategies, and serve as advocates for their members and their broader fields.

K. FitzGerald highlighted related examples at NSF NCAR that parallel the CoP model. The UCAR Software Engineering Assembly (SEA), first established in 2004 and refreshed in 2022, provides a professional forum for advancing software engineering activities across UCAR/NCAR. Similarly, the Earth System Data Science (ESDS) Initiative convenes the data science community through bi-weekly forums, virtual hours, training sessions, and experimental collaborative work time. These initiatives illustrate both the potential and the challenges of CoPs, especially when they rely on volunteer contributions and must adapt to changes in participation over time.

2.4.3.5 Panel Discussion

The panel discussion emphasized that **the most essential skills for RSEs are tied to personal qualities such as curiosity, problem-solving, empathy, active listening, and adaptability. These traits enable RSEs to build trust, reduce communication barriers, and work effectively alongside researchers.**

During the activity, it was mentioned that project management practices should be intentional and flexible, allowing teams to select approaches and tools that best fit the needs of a specific project and research group, rather than relying on one-size-fits-all solutions. It was also mentioned that adoption of improved practices should not be imposed from the top down. Instead, adoption of improved practices works best when tied to productivity outcomes, introduced through incremental changes, and supported by success stories and external advocates.

Finally, participants stressed that research institutions should embrace a collaboration model over a service model, positioning RSEs as partners rather than service providers. Mutual trust, respect, and shared goals are essential for sustaining

this model, as is clear communication that meets people where they are. Organizational agility, consistency, and elevating good examples can help lower barriers, align groups toward common objectives, and make lasting improvements to research software practices.

2.4.4 Summary of the Workshop Panel Discussions

The three panel sessions surfaced a consistent message: **improving software requires alignment across practices, tools, and people.** Panel 1 emphasized that sustained software quality depends on culture, clear goals, appropriately scaled practices, and intentional knowledge transfer. Panel 2 highlighted the central role of shared collaboration tooling (e.g., version control, CI/CD, documentation, and communication platforms), along with the need for training and thoughtful integration of emerging AI capabilities. Panel 3 focused on professional development and culture, underscoring the importance of communities of practice, career pathways, partnership-based collaboration between software engineers/RSEs and scientists, and integrative/bridge roles in sustaining collaboration and shared ownership over time. **Together, the panels reinforced that software sustainability is shaped as much by communication, training, governance, and coordination as by technical choices.**

2.5 Breakout Sessions

In order to capture participants' thoughts and feedback on topics from the workshop, 45-minute long breakout sessions were organized toward the end of each day. Participants were assigned to an in-person or virtual breakout group based on their mode of attendance, with both internal (NSF NCAR and UCAR staff) and external participants represented in each group. For the second day, participants were shuffled between groups with the intention of bringing new perspectives to the discussion. A set of questions (Appendix C) drafted by the organizing committee was provided for each session, and groups were encouraged to designate a scribe to record notes. Figure 7 shows the workshop participants in discussion during a breakout session.

Figure 7. CSF discovery workshop participants in discussion during a breakout session.

2.5.1 Day 1 Breakout Session: Capturing Feedback from Talks and Panels

The first breakout session focused on topics covered in Day 1 of the workshop, including "Software Engineering Successes and Lessons Learned" and "Better Practices." Discussion in the breakout groups centered on topics such as challenges related to funding and resource allocation, project management methodologies,

team collaboration, and technical practices for scientific software development. Specific points are summarized under each category below.

2.5.1.1 Challenges in Funding and Resource Allocation

A central theme across multiple groups was the difficulty of operating scientific software projects in environments characterized by limited resources and uncertain funding:

- **Resource constraints:** Many projects were noted to operate as single-person operations or in otherwise highly resource constrained environments.
- **Impact on morale:** Uncertainty regarding whether a project might continue in the near future can deeply affect morale, particularly for staff whose work is tied to a single project.
- **Need for additional funding:** There was a strong consensus on the need for more resources dedicated to software engineering and support. Discussions included exploring funding mechanisms and grants explicitly dedicated to software development and shifting the culture of institutions to include funding for software engineers/RSEs in relevant proposals.
- **Full cost of ownership:** Discussions highlighted the need to understand and properly resource efforts throughout the software lifecycle, considering the full cost of ownership.

2.5.1.2 Adopting Flexible Project Management Methodologies

Attendees discussed adopting practices that allow for flexibility and optimization rather than strict adherence to rigid frameworks:

- **Hybrid/Flexible Approaches:** The ideal approach depends upon project needs and team preferences. Suggestions to consider included a multiple horizon approach to project management, where longer-term plans are updated based upon experience gained from completing shorter horizons, and a hybrid approach, using Waterfall for the larger scope and Agile for the smaller components.
- **Agile Methodology:** Agile methodologies were viewed as valuable in many cases, especially when adopted in a less rigid manner. In particular, the aspects of introspection and retrospection were seen as beneficial.

- **Shift Left and Planning:** A key takeaway for larger projects, especially, was to "shift left," meaning requirements and testing should be addressed earlier in the development lifecycle rather than waiting until the end.
- **Time-boxing:** Time-boxing certain activities was noted as helpful but may require initial adjustments to create realistic estimations where there are competing demands.
- **Maturity Models:** The idea of "right sizing" processes and practices and leveraging maturity models or tiers of readiness came up in several groups.
- **Optimization:** Optimizing at the system level, e.g., thinking about overall time to solution rather than localized efficiency, was highlighted along with the importance of identifying bottlenecks and friction points.

2.5.1.3 Fostering Collaboration and Knowledge Sharing

To overcome organizational friction and high project risk, several methods for collaboration and knowledge transfer were highlighted:

- **Addressing the Semantic Gap:** Bridging the gaps in language and conceptual understanding between domain scientists and software engineers was noted as an important challenge. Successful project outcomes require shared understanding.
- **Resilience:** Practices learned from groups like the Met Office emphasized spreading knowledge to mitigate the risk of a single person holding critical information. One approach is to have the person with the least amount of experience pick up the task, and have everyone on the team review the code before it is integrated.
- **Communities of Practice:** Social infrastructure, such as communities of practice and champion programs, can be effective mechanisms for collaboration and knowledge sharing.
- **Collaborative Working:** Other mechanisms such as co-working, pairing, and hack sessions were also seen as effective and particularly helpful when starting a new project.
- **Common Goals:** Several suggestions were made to improve alignment within groups through emphasizing common goals. For example, focusing on shared

goals and challenges rather than emphasizing disciplinary divides or interpersonal disagreements.

- **Early Adopters and Champions:** The power of supporting and leveraging early adopters and champions came up several times. One successful example was working with interested scientists to model new tools and act as early adopters of software engineering practices.
- **Software Catalogs:** Improved searchability and discovery through software catalogs (e.g., the Sandia Software Portal and Stanford OSPO)

2.5.1.4 Technical Quality, Testing, and Infrastructure

Recommendations focused heavily on improving code quality, testing rigor, and developing shared technical resources:

- **Increased Adoption of Testing:** There needs to be a paradigm shift to more testing for both new and legacy codes moving beyond regression testing and incorporating methodologies relevant for the application.
- **Engaging Scientists:** Scientific input and perspectives were recognized as important for testing and several approaches including scientific reviews and testing suites catered to scientific needs.
- **Common Infrastructure:** There was interest in mechanisms to build and share common infrastructure and practices, specifically for CI/CD, testing, and deployment. Developer platforms (e.g., Backstage) were also mentioned.
- **Automation and Tooling:** CI/CD practices should be implemented where possible with automated formatting, code style, and testing to promote and maintain code quality.
- **Standards and Documentation:** Documentation, programming standards, and contributing guides were noted as important particularly for projects with larger contributor bases and to allow developers to more easily move between projects.
- **Separation of Concerns:** Leveraging a modular approach to software development, particularly in large codes, where distinct groups may own and/or engage more directly with different portions of the code to allow for flexibility in approaches and practices.

2.5.2 Day 2 Breakout Session: Reflecting on the Workshop

The second breakout session prompted participants to reflect on the overall workshop with an emphasis on topics covered in the talks and panel discussions from Day 2 of the workshop. Discussions highlighted key challenges, desired organizational changes, training needs, and reflected on current software development practices.

2.5.2.1 Leadership, Vision, and Cultural Change

A primary takeaway was the **critical need for leadership commitment and cultural buy-in to successfully implement collaborative software development strategies.**

- Many participants felt that the discussions held would be most valuable for management and sponsors to hear.
- Leading with "why" was suggested as a way to build understanding, trust, and morale along with starting with smaller, more focused efforts in order to show early success.
- Communities of practice, champion programs, and other types of social infrastructure were suggested as ways to break down silos, promote knowledge sharing, and empower staff.
- Several groups noted the importance of creating a positive culture of review and feedback and to share lessons learned.

2.5.2.2 Scientific Software Practices and Tools

The groups discussed various methods and tools to improve scientific software development:

- Being able to clearly explain the reasoning for adopting new practices was recognized as important for adoption.
- Systems engineering was frequently mentioned as a valuable viewpoint for tackling problems, including analyzing an entire software project from start to finish.
- Adopting a tiered approach to software processes and practices was suggested alongside frameworks like maturity models and readiness levels.
- **The importance of software architecture was a common topic of discussion.**

- Software testing and correctness were recognized as important and developing practices.
- Tools mentioned included Excalidraw, Unified Modeling Language (UML) diagramming tools, Mermaid, and Vernier (refer to Appendix B for more information on effective tools highlighted during the workshop)
- AI tools such as GitHub Copilot or Cursor were noted as valuable for experienced developers who can write clear instructions, but perhaps less so for junior or inexperienced developers who may not be able to understand or evaluate the code generated.

2.5.2.3 Training and Resource Gaps

Training and having the bandwidth and support needed to dedicate time to training were common topics of discussion:

- The primary limitation for many software engineers engaging in training was time or bandwidth, rather than a lack of training opportunities. Along these lines, there was a lot of interest in the Met Office approach of having 20% of time available for professional development with the ability to bank time for workshops.
- Supervisor support was mentioned as an important factor in training participation.
- Specific training needs in systems engineering, testing, and automation were noted along with Agile methodologies and project management.
- Self-directed learning or co-working was recognized as being more valuable in some cases than formal training opportunities.
- The importance of having mechanisms to share high quality learning materials and resources was noted.

2.5.2.4 Co-development and Organizational Support

Participants reflected on the challenges of collaborative development between scientists and software engineers/RSEs and organizational support needed for co-development:

- The term “co-development” was still not entirely understood by many, despite its frequent use during the workshop.

- A desired form of support was providing opportunities for scientists and software engineers/RSEs to collaborate. One suggestion was to adapt the Met Office structure of deploying software engineering teams to projects or labs.
- Software engineers/RSEs noted a desire to have more input on decisions related to sustainability and maintainability of software.
- Encouraging scientists to communicate "what" they need rather than "how" they would like it implemented was suggested as a way to improve project outcomes.
- The importance of organization-wide communities, learning opportunities, and celebration and sharing of relevant successes was noted.
- The piloted Developer Exchange[6] program at NSF NCAR or similar programs were suggested as a mechanism to promote collaboration and knowledge sharing.

2.5.2.5 Workshop Coverage Gaps

Several topics were noted as missing or needing deeper exploration. Participants expressed interest in further discussion of the following areas:

- Details on the nature of the proposed Community Software Facility.
- Software lifecycle management, including project transitions and sunsetting.
- Co-development of scientific software with more representation from scientists was mentioned several times.
- Perspectives from people and teams working in more similar contexts and in particular more resource-constrained environments.
- More time spent discussing challenges and lessons learned.
- Greater depth regarding software testing practices.

[6] The "Developer Exchange Program" is a pilot program at NSF NCAR inspired by the existing "Visitor Programs" (student and faculty scientific and professional visits that range from a few days to sabbatical). During the pilot program two software engineers at UCAR and NSF NCAR dedicated portions of their time working on software development projects across the organization. With the successful completion of the pilot program there is interest in expanding this program so that other software engineers at NSF NCAR and UCAR could participate.

3. Workshop Results: Key Findings and Recommendations

As introduced earlier in this report, the goals of this workshop were to:

- Learn from best practices at NSF NCAR and external organizations to develop ideas that can inform the process and framework of scientific software development at NSF NCAR.
- Identify ways to improve efficiency in communication, collaboration, and decision making during the development of scientific software.
- Understand how other organizations provide support, professional development and foster culture and communities of practices within their software engineering teams.

To address these goals, the workshop focused on several guiding questions that were previously stated in the "Introduction" section. The key findings below synthesize insights from presentations, panels, and breakout discussions. These findings are organized under each guiding question and inform the recommendations for CSF program design and development. For all the recommendations, the CSF team is encouraged to establish **measurable indicators of success and impact to evaluate effectiveness over time.** In addition, more detailed findings can be found below each question's section.

I. What additional skillsets, roles, training, and support is needed to improve quality and sustainability of scientific software development?[7]
Key Findings *Advances in scientific software quality and sustainability depend on focused system-level training for select roles, widespread modern software engineering skills and tools, and integrative roles that help align scientific goals with software design. This needs to be supported by organizational structures that acknowledge and provide opportunities for dedicated staff time for training.*

[7] Based on discussions during the workshop, this question was revised to this current form to better align with the workshop goals and reflect the intended scope.

Recommendations for CSF

★ *Broaden the focus beyond software engineers* - Development teams are inherently multi-disciplinary, it is recommended that the CSF expand its focus to include teams organized around flow of value rather than roles and that the CSF encompass not only software engineers but also scientists, project managers, product managers, and systems engineers.

★ *Provide training and dedicated time* - It is recommended that the CSF provide specific training opportunities and dedicated time for software development teams to attend professional activities, as participation is often limited by competing workload demands rather than lack of interest.

★ *Revisit and assess software engineering resources* - It is recommended that the CSF reassess and evaluate software engineering resources and capacity to ensure adequate support for current operations. This recommendation stems from trends observed at peer institutions, including the Met Office, which have recently increased software engineering staff capacity.

Detailed Findings:

- There is a need to emphasize and provide training to a sub-set of personnel in systems-level competencies, including requirements engineering, conceptual design, architecture development, and structured decision frameworks (e.g., Pugh matrices [7]).
- Modern software engineering training is essential, focusing on testing beyond regression tests, CI/CD, software architecture and design considerations, distribution and packaging, documentation, performance portability, and version control workflows.
- "Bridge roles" such as project managers, product owners, systems engineers/architects, and software architects play a critical part in facilitating dialogue between different technical groups, ensuring that communication gaps are minimized and project goals remain aligned.
- Practical tools training should include UML/SysML, diagramming tools, GitHub/GitHub Actions, performance profilers, abstraction frameworks (e.g., Kokkos), documentation frameworks and knowledge bases, software-focused project management tooling, integrated development environments, and AI-assisted development tools (e.g., GitHub Copilot, ChatGPT, Cursor).

- The primary constraint on training in many cases is staff time and not availability of learning resources and opportunities. It should be noted that NSF NCAR has a policy that allows all staff 15% time for professional development, committee meetings, or other work-related activities. However, not all staff or their supervisors are aware of this benefit, especially if the staff member is primarily funded by non-NSF sources. Additionally, the 15% is not always taken into account during proposal development for external funding.
- A list of professional development resources, including training programs and community networks, that support continued learning and skill advancement is provided in Appendix D of this report. These resources were identified throughout the workshop.

II. How do we get started with best practices, and where do we want to go?

Key Findings

Providing best practices for scientific software begins with intentional early-stage design, emphasizing problem definition, requirements capturing, architecture, and structured decision-making through the adoption of process models. In practice, this means establishing shared expectations for how decisions are made and documented, architectural intent is defined and revisited, and responsibilities are coordinated across teams. It is also important to adopt a mindset to continuously improve best practices through review, refinement, and iteration.

Recommendations for CSF

★ *Establish and refine processes* - The CSF should study, adapt, and hybridize systems engineering, agile, DevOps and project management methodologies, to fit NSF NCAR's specific culture and operational needs. This will result in a tailored model that supports requirements tracking, organizational agility, communications, and efficiency. To effectively accomplish this, inputs from software engineering and scientific teams should be elicited.

★ *Invest in conceptual design and software architecture* - Architecture and conceptual design is necessary to develop complex, long-lived software systems. The CSF should identify and cultivate individuals who identify as software architects and big picture thinkers, involving them in early design and decision-making and investing in their continued technical development.

★ *Establish clear processes for requirements engineering* - Establishing clear

requirements is considered the most challenging part of building a software system. Project teams should invest time early in the process to understand and listen to stakeholder needs to set realistic expectations around scope, resources, and prioritization before moving to solutions.

★ *Establish recommendations for coding standards and practices* - The adoption of common coding standards and practices can create a more consistent experience allowing for the automation of tasks and facilitating smoother onboarding and transitions between projects (recommendations should come with guidance based upon project maturity level and other criteria). This should be done with input from different software engineering teams and communities of practice at NSF NCAR.

Detailed Findings:

- Start with early stage and upfront design processes which includes spending time on defining clear problem definition, requirements capturing, architectural design, and exploration of alternatives. Architecture and conceptual design includes early decisions that have a strong impact on long term cost and complexity. Strong facilitation is needed in this early stage to keep the focus of the team on problem definition rather than transitioning to solution selection.
- Well-documented and standardized processes are essential to maintaining organizational memory, quality, and consistent performance.
- Apply and choose the appropriate practices and processes based on the maturity level of the software package. This came up during the workshop in terms of already available industry standards (e.g., CMMI) as well as institutional practices in some groups (e.g., Sandia).
- Adopt flexible, hybrid process models that fit well with the project, team, and organization. This includes elements of Agile, industrial DevOps, and systems engineering processes to combine long term planning with short, iterative execution cycles.
- Separation of concerns was brought up as a practical approach when dealing with large codes where distinct groups may have governance over portions of the code. This translates to a modular approach to software development allowing for flexibility needed when different stakeholders are involved.

III. What culture changes are necessary?
Key Findings *Lasting improvements in scientific software practices require deliberate cultural change, including visible leadership commitment, shared understanding of purpose, and organizational norms that value software engineering contributions, software quality, psychological safety, learning and experimentation, and shared ownership of scientific outcomes.*
Recommendations for CSF ★ *Empower software engineers as equal partners* - It is recommended that the CSF work with software engineers and scientists at NSF NCAR to establish processes, workflows, and a governance structure where software engineers are empowered as collaborative partners in the software development lifecycle. ★ *Establish multiple career pathways for software engineers, including senior leadership positions within the organization* - It is recommended that CSF and NSF NCAR as a whole consider developing diverse and clear career pathways that allow software engineers to advance in both technical and leadership roles. Providing structured opportunities for progression will strengthen retention, motivation, and professional growth and will ensure that software engineers' voices are represented at all levels within the organization.

Detailed Findings:

- Without cultural alignment, even well designed processes, tools, and software will not be adopted effectively.
- Early adopters and internal champions accelerate change and should be supported and recognized.
- Leadership must communicate the "why" behind the change clearly. Cultural change is most successful when leaders clearly articulate purpose, benefits, and expected outcomes of the new practices.
- An organization as a whole must actively value software quality and software craftsmanship.
- **Software engineers/RSEs must be treated as collaborative partners, not service providers.** Delivering scientific capability and impact is a shared responsibility of the whole team, regardless of the roles.

- Software delivery teams need to recognize the value of experimentation, openly discussing where ideas have not worked as planned and embrace them as opportunities to learn.
- Collaboration-oriented skills, such as code review, shared vocabulary development, stakeholder engagement, interdisciplinary communication, and empathy are as important as coding technical skills. Any cultural change in this area requires organization-wide action supported by strong leadership.
- Organizational development has many angles to it, including leadership, vision, resources, workforce, processes, and organizational culture. Change requires supporting social and cultural structures (e.g. networks, relationships, champions). Investing in these should be an early priority.

IV. What models for decision making, communication, and ways of working are effective?

Key Findings

Effective scientific software teams rely on explicit decision-making frameworks and tools, intentional communication structures and collaborative ways of working (that emphasize early engineering involvement), shared governance, and continuous knowledge transfer. Together these practices increase transparency, reduce ad hoc decisions, and limit team divergence over time.

Recommendations for CSF

★ *Promote approaching challenges as " the team versus the problem", not individual conflicts* - The CSF should explore tools and methods that promote this mindset, such as the Pugh decision matrix [7], to foster collaborative, transparent decision-making processes. These processes should be adopted at all levels of decision making and grounded in an understanding of socio-technical systems, power dynamics, conflict, and teamwork.

★ *Explore deployable and fixed team structures* - Other organizations spoke to the benefits of maintaining different types of software teams, some deployable across projects and others fixed within specific laboratories. The CSF should evaluate these models to determine if similar flexibility could enhance organizational responsiveness and collaboration.

★ *Enhance knowledge transfer and documentation* - The CSF should identify and implement best practices for documenting expert knowledge and mental models,

ensuring sustained project progress, effective knowledge transfer, and continuity even when team members change or projects evolve.

★ *Develop glossaries of shared vocabulary and ontology* - To bridge disciplinary boundaries, it is recommended that the CSF team invest time in building a shared vocabulary (a "Rosetta Stone") between scientists and engineers. This will improve collaboration, align expectations, reduce misunderstandings, and strengthen interdisciplinary integration.

Detailed Findings:

- Structured decision-making frameworks such as the Pugh matrix [7] and systems engineering management plans (SEMPs) improve transparency and reduce ad hoc decision processes.
- Knowledge transfer must be intentional, with documentation of design rationale, onboarding guides, and shared CI/CD and test infrastructure.
- Early and regular synchronization and integration points reduce divergence between components and highlight interaction issues early.
- Co-development models with clear governance, shared planning, and joint ownership of outcomes, are essential and require ongoing refinement and communication.
- Engage software engineers early in the development process so that decisions related to architecture, maintainability, and scalability are addressed from the outset.

V. How do we empower the NSF NCAR Software Engineering Assembly (SEA) as a community of practice?

Key Findings

Specifically at NSF NCAR, the SEA could be strengthened through dedicated funding to support staff training and time put in by committee leadership, and providing opportunities for recurrent engagement with outside organizations. The SEA should be recognized as a strategic organizational asset for cultural change, workforce development, and improved knowledge transfer in support of high-quality, sustainable scientific software.

Recommendations for CSF

- ★ *Strengthen and expand Communities of Practice (CoP)*- Empowering existing CoPs are vital for fostering innovation, driving learning, and maintaining alignment across teams. There is also room for CSF to continue engagement with external CoP through workshops or virtual discussions.
- ★ *Continue engagement with external experts through a formal committee* - This workshop underscored the importance of engaging with external organizations, groups, and individuals who possess expertise and experience in this area. It is recommended that the CSF team explore forming an external advisory committee to provide periodic guidance.

Detailed Findings:

- Communities of practice such as the SEA play a vital role in spreading cultural change, developing talent, and aligning teams with modern practices.
- Communities of practice should be recognized as strategic organizational assets, not volunteer side efforts.
- The SEA should be provided with additional institutional support, including funding SEA leadership time to participate (versus all time put in by staff being volunteer work) and providing opportunities for the SEA to give feedback on CSF concepts.
- Intentional onboarding materials, documentation of best practices, and regular exchange with external organizations will help strengthen the SEA.

It should be noted that while these insights provide a foundation for advancing sustainable, collaborative, and forward-looking scientific software development at NSF NCAR, this CSF discovery workshop serves only as the first step in gathering feedback for the CSF. The workshop participants emphasized the need for broader, ongoing discussions between scientists and software engineers to support co-development of scientific software. To ensure we gather input from a wider community, the center is planning to host further listening sessions and events to capture perspectives from NSF NCAR scientists as well as our broader research community.

4. Closing Remarks

The CSF discovery workshop reinforced that NSF NCAR's software ecosystem is not merely a collection of tools and codebases but a socio-technical system: an interconnected network of people, processes, and technologies. The success of scientific software development depends as much on communication, trust, and shared understanding as it does on good software and technical rigor. **Recognizing this interdependence is essential as NSF NCAR moves forward with the CSF and broader efforts to modernize scientific software development.** Sustained progress will require building and leveraging organizational structures and cultural context that enable people to work effectively together across disciplines and support the adoption of technical best practices. **Advancing these objectives will require clear and thoughtful communication and leadership.**

While process and structure are important, the workshop underscored that **cultural change will determine the long-term success of initiatives such as CSF.** NSF NCAR's culture must continue to evolve towards one that prizes collaboration and continuous learning. Leadership plays a critical role in modeling these values, communicating the purpose behind new practices, and recognizing collaborative achievements.

The CSF will serve both as a **technical and a cultural catalyst**, demonstrating that when development processes are designed with people, collaboration, and feedback at their center, the result is not only better software but also a more resilient, innovative, and connected research community.

* * *

5. References

[1] United States Research Software Engineer Association. Accessed 20th January, 2026. https://us-rse.org/

[2] Johnson, S., and Yeman, R. (2023). *"Industrial DevOps: Build Better Systems Faster"*. IT Revolution.

[3] Malviya-Thakur, A., Bernholdt, D, E., Godoy, W, F., Watson, G, R., Doucet, M., and Coletti, M, A. (2023). *"Research Software Engineering at Oak Ridge National Laboratory"*. IEEE. DOI: 10.1109/MCSE.2023.3260211

[4] CMMI Institute. *"Capability Maturity Model Integration (CMMI) Version 3.0"*. (2023).

[5] International Organization for Standardization (ISO), International Electrotechnical Commission (IEC), and Institute of Electrical and Electronics Engineers (IEEE). *"Systems and Software Engineering-System Life Cycle Processes, ISO/IEC/IEEE 15288"*. (2023). Geneva, Switzerland, ISO/IEC/IEEE.

[6] Thomson, J. *"The Development of Strategic Capability Through Multi-National Corporations "* (2009). Retrieved from ResearchGate October 2024.

[7] Pugh, S. (1991). *"Total Design: Integrated Methods for Successful Product Engineering"*. Addison-Wesley.

[8] Brooks, F, P., *"No Silver Bullet: Essence and Accidents of Software Engineering"*. (1987). IEEE Computer, vol. 20, no. 4, pp. 10-19.

[9] Brooks, F. P., *"The mythical man-month: Essays on software engineering"*. (1975). Addison-Wesley.

[10] Object Management Group (OMG). *"OMG Systems Modelling Language (OMG SysML)"*. (2019). OMG document number formal/2019-11-01.

[11] Object Management Group (OMG). *"Unified Modeling Language (UML) Version 2.5.1"*. (2017). OMG document number formal/2017-12-01.

[12] Mundt, M. R., Burgess, P. W., and Vigil, D. V. (2022). *"A Tiered Approach to Scientific Software Quality Practices"*. In proceedings of the 2022 improving scientific software conference. [Online]. Available at https://www.osti.gov/servlets/purl/2004212

[13] United States Research Software Engineer Association and Academic Data Science Alliance. (2023). *"Hiring, Managing, and Retaining Data Scientists and Research Software Engineers in Academia: A Career Guidebook from ADSA and US-RSE"*. https://doi.org/10.5281/zenodo.8264152

[14] Katz, D.S., McHenry, K., and Lee, J. S. (2021). *"Senior Level RSE Career Paths (with an S)"*. SeptembRSE proceedings. https://doi.org/10.5281/zenodo.8264152.

[15] Project Management Institute, Inc. *"Power Skills"*. Accessed 13 January, 2026. *https://www.pmi.org/disciplined-agile/people/powerskills*

Appendix A: Agenda

Table A1: CSF discovery workshop agenda for day one.

Day 1: Wednesday, August 20, 2025	
9:00 am (MDT)	**Welcome and Framing**, introductions, Met Office Keynote, and ice-breaker activity • Welcome, framing, and Code of Conduct (Domi Colegrove, Strategic Initiative Projects Lead, NSF NCAR) • NSF NCAR Introduction (Everette Joseph, NSF NCAR Director) • Acknowledging our Current Successes (Gretchen Mullendore, MMM Lab Director, NSF NCAR) • Community Software Facility Introduction (Thomas Hauser, CISL Lab Director, NSF NCAR) • Introduction to Met Office (Jon Petch, CGD Lab Director, NSF NCAR) • Scientific Software Engineering at the Met Office (Simon Vosper, Executive Director of Science, Met Office) • Software Engineering Discipline and Approach (Charles Ewen, Technology Director and Chief Information Officer, Met Office)
10:30am	**Break**
10:45am	**Presentations: Scientific Software Engineering Successes and Lessons Learned** • Bridging the gap between Scientists and Software Engineers to Accelerate Solutions (Robin Yemen, Senior Software Solution Architect, Leidos) • Research Software Development Experiences at Oak Ridge National Laboratory (David Bernholdt, Distinguished R&D Staff Member, Computer Science and Mathematics Division, Oak Ridge National Laboratory)
12:00pm	**Lunch**

1:00pm	**Panel A: Better Practices**
	• Facilitator: Jared Henrickson (Product Owner, Center for Research Computing, Notre Dame)
	• Panelists:
	  ○ Miranda Mundt (Research Software Engineer, Sandia National Laboratories)
	  ○ Helen Kershaw (Software Engineer, NSF NCAR)
	  ○ Josh Wilson (Senior Software Developer, Met Office)
	  ○ Jerome Hugues (Principal Researcher, CMU Software Engineering Institute)

2:15pm	**Break**
2:30-3:15pm	**Day 1 Breakout Sessions: Capturing Feedback From Today's Talks and Panels**
3:15 pm	**Closing Remarks**

Table A2: CSF discovery workshop agenda for day two.

Day 2: Thursday, August 21, 2025	
9:00am (MDT)	**Presentations: Design Decisions - How to Approach Large Project Design, Processes, and Features** • Systems and Software Engineering (Regina Griego, INCOSE Fellow [previously Distinguished R&D Systems Engineer at Sandia National Laboratories]) • Developments of the JEDI-based observation processing and data assimilation system (David Simonin, Manager of the Assimilation of Surface-based Observations Group, Met Office)
10:30am	**Break**

10:45am	**Panel B: How Can We Use Tools to Work More Effectively Together?**
	• Facilitator: Cena Brown (Software Engineer, NSF NCAR)
	• Panelists:
	  ○ Kyle Shores (Software Engineer, NSF NCAR)
	  ○ Aaron Donahue (Staff Research Scientist, Lawrence Livermore)
	  ○ Allison Baker (Project Scientist, NSF NCAR)
	  ○ Adrianna Foster (Project Scientist, NSF NCAR)
	  ○ Oakley Brunt (Scientific Software Engineer, Met Office)

12:00pm	**Lunch**

1:00pm	**Panel C: Professional Development and Culture**
	• Facilitator: Daniel Howard (HPC Consultant, NSF NCAR)
	• Presentations and Panelists:
	  ○ Princeton RSE Group, INTERSECT (Ian Cosden, Senior Director, Research Software Engineering, Princeton)
	  ○ US-RSE - Beyond Bug Fixes: Professional Development and Communities in RSE Teams (Sandra Gesing, Executive Director, US Research Software Engineers Association (US-RSE) and Senior Researcher at San Diego Supercomputer Center)
	  ○ Culture Shifts & Building Cross Functional RSE Teams (Caleb Reinking, Associate Director RSE, University of Notre Dame Center for Research Computing)
	  ○ Communities as agents of change: learning and growing through communities (Katelyn FitzGerald, Software Engineer, NSF NCAR)

2:30pm	**Break**
2:30-3:15pm	**Day 2 Breakout Sessions: Reflecting on the Workshop**
3:15pm	**Closing Remarks**

Appendix B: Effective Tools

The following is a list of tools that were highlighted during the workshop by panelists, presenters, and participants as valuable for improving development practices and workflows for scientific software.

Table B 1: List of effective tools highlighted in the CSF discovery workshop for scientific software development practices and workflows.

Tool	Description	URL
Backstage	An open source framework for building developer portals (e.g. internal software catalogs, templates, and documentation).	https://backstage.io/
ChatGPT	A GenAI tool used to generate human-like text to assist in writing and problem-solving.	https://chatgpt.com/
Confluence	A collaborative and documentation platform used to share knowledge in one centralized location.	https://www.atlassian.com/software/confluence
Excalidraw	An online whiteboard tool for creating hand-drawn style diagrams for collaborative visuals.	https://excalidraw.com/
GitHub	A web-based platform for version control and collaborative software development.	https://github.com/
GitHub Actions	An automation platform within GitHub that lets you build, test, and deploy code through a customizable workflow triggered by repository events.	https://docs.github.com/en/actions
GitHub Copilot	A GenAI-powered coding assistant	https://github.com/features/copilot
Cursor	An AI-powered Integrated Development Environment (IDE) built as a fork of Visual Studio Code.	https://cursor.com/
Google Docs	A word processing application that allows users to create, edit, and collaborate on documents.	https://docs.google.com/
Google Spaces	A collaboration tool that enables teams to chat in organized spaces.	https://chat.google.com

Slack	A messaging platform that enables teams to communicate through organized channels and direct messages.	https://slack.com/
Visual Studio Code	An Integrated Development Environment (IDE) that is used for writing code, building, and debugging across multiple programming languages.	https://code.visualstudio.com/
Mermaid	A diagramming and charting tool capable of producing Unified Modeling Language (UML) diagrams.	https://mermaid.js.org/
Vernier	A profiling tool for scientific codes on HPC platforms developed by the UK Met Office.	https://metoffice.github.io/Vernier
Cameo Systems Modeller	A cross-platform collaborative Model-Based Systems Engineering (MBSE) environment that offers capability to create SysML and UML models.	https://www.3ds.com/products/catia/no-magic/cameo-systems-modeler
MyST Markdown	An ecosystem of tools that extend Markdown for scientific and technical communication including functionality such as glossaries and dynamic links, references, and visualizations.	https://mystmd.org/
Kokkos	An open-source C++ performance portability library that allows software to run on different architectures such as CPUs and GPUs.	https://kokkos.org/

Appendix C: Breakout Session Discussion Questions

Day 1 Questions

1. What are key points on scientific software development that you learned today that you would bring back to your organization and hope that they would adopt?
2. What is a scientific software development best practice that you would suggest that didn't come up today?
3. What is something you're doing in scientific software development that you didn't hear today or how does it look different from what you heard today?
4. If you were starting a scientific software development project, what processes would you incorporate that you heard about in presentations and panels today?

Day 2 Questions

1. Have you learned about any new techniques, tools, or experiences that will help you address the challenges you're facing in scientific software development?
2. Is there anything we didn't cover at this workshop that you wish we had?
    a. What speakers or organizations would you recommend for next time?
3. In your current role, do you have access to the training you think you need? Would you engage in training if it were available?
4. What steps do you think we could take to bring about cultural change in an organization?
    a. As an individual, how do you cope with cultural change?
    b. How would you describe your ideal culture for collaborative software development within a research organization?
5. How do you see yourself contributing to scientific software development? What's your role in this?
    a. Has your definition of "co-development" changed over the past few days?

6. What kind of support do you wish your organization provided to enable scientific software development?

Appendix D: Resources

This section provides a list of professional development resources, including training programs and community networks, that support continued learning and skill advancement for software engineers, research software engineers, and software systems engineers.

Table D1: List of professional development resources highlighted during the CSF discovery workshop.

Resource	Description	URL
Earth System Data Science (ESDS)	A community of practice within NSF NCAR and UCAR with a common interest in advancing the use of modern approaches and technology to support Data Science.	https://ncar.github.io/esds/
INTERSECT	An NSF-funded initiative designed to provide advanced training in research software engineering through community-driven, modular educational programs.	https://intersect-training.org/
The Carpentries	An educational community dedicated to equipping researchers with skills for computational and data-intensive research.	https://carpentries.org/
The Turing Way	A community-driven guide to reproducible, ethical and collaborative data science.	https://book.the-turing-way.org/
Software Engineering Assembly (SEA)	A community of practice within NSF NCAR and UCAR to serve the interests of the software engineering community. Open membership is available to all staff.	https://sea.ucar.edu/
US-RSE	A professional community that supports and advocates for research software engineers.	https://us-rse.org/
International Council on Systems Engineering (INCOSE)	A professional community that develops and disseminates the transdisciplinary principles and practices that enable the realization of successful systems. INCOSE is designed to connect systems engineering professionals with educational, networking, and career advancement opportunities in the interest of developing the global community of systems engineers and systems approaches to	https://www.incose.org/

	problems.	
Object Modeling Group	Provides a neutral forum where best practices from a wide range of fields can be discussed and standards can be generated that drive the adoption and innovation of cutting-edge technology spanning industries worldwide. OMG is responsible for the UML and SysML standards.	https://www.omg.org