DevOps Pragmatic Practices and Potential Perils in Scientific Software Development

Reed Milewicz, Jonathan Bisila, Miranda Mundt, Sylvain Bernard, Michael Robert Buche, Jason M. Gates, Samuel Andrew Grayson, Evan Harvey, Alexander Jaeger, Kirk Timothy Landin, Mitchell Negus, and Bethany L. Nicholson

Abstract The DevOps movement, which aims to accelerate the continuous delivery of high-quality software, has taken a leading role in reshaping the software industry. Likewise, there is growing interest in applying DevOps tools and practices in the domains of computational science and engineering (CSE) to meet the ever-growing demand for scalable simulation and analysis. Translating insights from industry to research computing, however, remains an ongoing challenge; DevOps for science and engineering demands adaptation and innovation in those tools and practices. There is a need to better understand the challenges faced by DevOps practitioners in CSE contexts in bridging this divide. To that end, we conducted a participatory action research study to collect and analyze the experiences of DevOps practitioners at a major US national laboratory through the use of storytelling techniques. We share lessons learned and present opportunities for future investigation into DevOps practice in the CSE domain.

Keywords DevOps · Scientific software development · Research software engineering

The first, second, and third authors contributed equally to this work.

R. Milewicz (✉) · J. Bisila · M. Mundt · S. Bernard · M. R. Buche · J. M. Gates · S. A. Grayson · E. Harvey · A. Jaeger · K. T. Landin · M. Negus · B. L. Nicholson
Sandia National Laboratories, Albuquerque, NM, USA
e-mail: rmilewi@sandia.gov

J. Bisila
e-mail: jbisila@sandia.gov

M. Mundt
e-mail: mmundt@sandia.gov

X.-S. Yang et al. (eds.), *Proceedings of Eighth International Congress on Information and Communication Technology*, Lecture Notes in Networks and Systems 693,
https://doi.org/10.1007/978-981-99-3243-6_51

1 Introduction

High-performance computing (HPC) today plays a central role in scientific discovery, economic competitiveness, and national security in the United States and elsewhere. On the economic front, one US-government funded study estimated that every dollar invested in HPC generated an average of $507 of new revenue and $47 in profit or cost savings [1]. Likewise, to stay at the forefront of science and engineering, the United States has made substantial investments into computing technologies to push HPC into the Exascale era [2], with the world's first supercomputer capable of over a quintillion operations per second, *Frontier*, being brought online in 2022.

As the demand for high-quality simulations and data analyses continues to grow, the computational science and engineering (CSE) community has likewise had to evolve; there has been a notable shift away from small teams working on research scripts in isolation toward community-driven, open-source software ecosystems [3–6]. The makeup of the workforce has also been rapidly diversifying, with Research Software Engineering (RSE), DevOps, and IT Service Management (ITSM) professionals allying with computational scientists and mathematicians. They bring with them modern tools, practices, and perspectives on software development and maintenance – bridging the divide between conventional and scientific computing. Integrating those professionals into the teams, institutions, and culture remains an ongoing challenge [7–9], but the historical "chasm" between software engineering and scientific computing has narrowed considerably in recent years (*c.f.,* [10]).

In this study we focus our attention on DevOps in CSE contexts. For the purposes of this work, we use the definition of DevOps coined by Leite et al.:"a collaborative and multidisciplinary effort within an organization to automate continuous delivery of new software versions, while guaranteeing their correctness and reliability" [11]. To keep pace with demand, HPC CSE software must evolve more quickly while still remaining credible and trustworthy. There is an urgent need for more capable cyberinfrastructures to develop, deploy, and maintain that software. At the same time, however, DevOps for science and engineering presents unique challenges, and it demands adaptation and innovation in tools and practices; solutions that work for a web application in industry are unlikely to perfectly fit the needs of a multi-physics HPC application. At the present, the intersection of DevOps and scientific computing is critically understudied. A deeper understanding of how DevOps work is done in CSE contexts and what needs practitioners have could (1) inform the design of better tools and techniques, (2) support effective policy-making around cyberinfrastructure sustainment, and (3) raise awareness of the critical role played by DevOps practitioners in advancing science and engineering.

For these reasons, we conducted a participatory action research study to collect and analyze the experiences of DevOps practitioners at Sandia National Laboratories [12]. The first three authors of this study recruited practitioners to share "war stories" [13], detailed narratives of challenges faced and accomplishments made in DevOps work at the laboratories. In line with the principles of action research to allow software professionals to express their own voices, all participants were co-equally involved this study and are co-authors of this paper.

2 Background

DevOps has been in existence and usage for over a decade, evolving naturally from a necessity for breaking down silos between different developers within a software's lifecycle to focus on people and processes instead of distinct outcomes [14]. The shift is primarily marked by the creation of the *devopsday* conferences[1] in 2009 and has grown into a worldwide movement over the past 13 years. The emphasis of DevOps is to merge the "makers" with the "deployers" to create a more cohesive (and iterative) product; in fact, it is a natural extension of the Agile principles to extend beyond "code checkin" [15].

We can already see an issue with this emphasis, however—it is focused on creating and deploying a *product* in a pipeline where developers and operations professionals are not one and the same. Within scientific software communities, however, these activities have stayed essentially merged for research scientists. This is because, more often than not, researchers assume all of the roles within a software lifecycle [16]. Because software now underpins nearly all realms of scientific research, scientific researchers are expected to be literate not only in their domain of expertise, but also in software engineering.

There is a natural shift to respond to this need—including the application of DevOps practices to existing and new scientific software development teams. Through a combination of gray literature, peer-reviewed literature, and personal "war stories," we have observed that the adoption of these practices has large potential—but also potential pitfalls. We aim in this paper to discuss some of these successes and failures, contrasting how our stories compare with the top critical challenges to DevOps culture adoption as found in a recent systematic review by Khan et al. [17] and prioritized practices as discussed by Akbar et al. [18], to provide actionable suggestions for changes to the current paradigm, and to highlight areas where more research must be done.

3 Related Work

Almost all of the scholarly literature concerning DevOps for CSE has come from the perspective of researchers interested in applying DevOps tools and techniques rather than from DevOps practitioners. Conversely, there is a significant amount of gray literature (e.g., whitepapers, blog posts) that comes from the practitioners' perspectives.

In part, this is due to the recent up-trend of conferences and workshops geared toward scientific software development. Here is a non-exhaustive list of examples: the Collegeville workshop series[2]; the Tri-lab Advanced Simulation & Computing

[1] https://devopsdays.org/.

[2] https://collegeville.github.io/Workshops/.

Sustainable Scientific Software Conference[3]; and the Workshop on the Science of Scientific Software Development and Use[4]. Software sustainability has also been at the forefront of consideration for the Department of Energy as shown by the recent Request for Information on Stewardship of Software for Scientific and High-Performance Computing [19]. Cross-institutional projects such as the Exascale Computing Project (ECP)[5] also place importance on developer productivity and better development practices for scientific software, with information distributed through webinars, tutorials, and the website Better Scientific Software (BSSw)[6].

DevOps is a recurring topic in this space. In their BSSw blog post, Beattie and Gunter detail the adaptations of DevOps practices that have been applied to the Institute for Design of Advanced Energy Systems (IDAES), which include weekly standup meetings, incremental improvements to automated testing, and "soapboxing" (frequent discussions about the importance of software engineering practices with leadership) [20]. The 2020 Collegeville workshop's theme was "Developer Productivity" which yielded whitepapers that discussed Agile practices, challenges and successes related to automated testing, and a mapping of difficulties and recommendations for each stage of the software delivery lifecycle from the lens of scientific software engineering [21–24]. The 2022 Tri-lab Advanced Simulation & Computing Sustainable Scientific Software Conference had two tracks for "DevOps Infrastructure Development" and "DevOps CI/CD Pipeline Development."

de Bayser et al. has argued that DevOps concepts and practices should be integrated into the activities of researchers to help increase productivity and quality of the resulting software [25, 26]. Whitepapers and blogs from the DevOps community, however, argue for more specialized roles. Gesing argues for the implementation of well-defined roles in teams rather than researchers acting as "all-rounders" [27]. Adamson and Malviya Thakur second this view in their whitepaper on the operationalization of scientific software from a DevSecOps perspective [28]. This is further supported by the rise of the RSE professional designation which aims to represent the unique role of software engineering expertise applied directly into research software development.

4 Methodology

To collect and analyze the experiences of software practitioners doing DevOps work in CSE contexts, we used storytelling techniques to draw together an ensemble of challenges and triumphs in DevOps for CSE. We then analyzed that data through a participatory action research lens to build consensus among participants around their needs and values.

[3] https://s3c.sandia.gov/.

[4] https://web.cvent.com/event/1b7d7c3a-e9b4-409d-ae2b-284779cfe72f/summary.

[5] https://www.exascaleproject.org/.

[6] https://bssw.io.

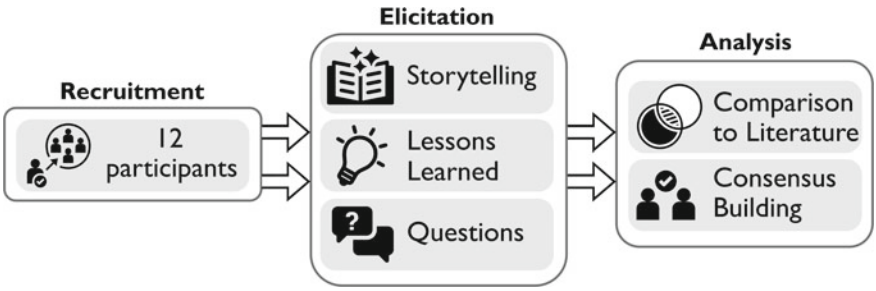

Fig. 1 An illustration of the methodology used to collect and assess evidence gathered in our study. Participants were recruited to share stories of challenges and triumphs in DevOps, report lessons learned, and reflect on their questions about best practices. For our analysis, we compared reported experiences to trends in the scholarly literature and then iterated with our participants as co-researchers to refine the contents of this study.

Storytelling is a qualitative data collection technique where participants are asked to recount detailed events from their own experiences [29]. The use of storytelling to analyze participants' experiences is a popular technique in the social sciences. A 2022 publication of *Communications of the ACM*, Barik et al. speak to the importance of applying storytelling techniques in scientific settings to promote better communication and overall understanding [30]. Moreover, as noted by Polletta et al., as a method for representing the views, attitudes, and experiences of a community, storytelling is often seen as more authentic and democratic in character (*i.e.*, "everybody has a story") [31].

Our study draws inspiration from the work of Lutter and Seaman, who collected "war stories" concerning documentation usage during software maintenance [13]. A key methodological difference in our work is that we seek to apply methodological techniques from participatory action research (PAR) to guide our data collection and analysis. PAR is an approach to action research that emphasizes direct participation in the research process by the members of the community whose interests the research is meant to serve [32]; as explained by Baum et al., "PAR advocates that those being researched should be involved in the process actively" [33]. In our work, we take the position that software practitioners doing DevOps for CSE are qualified subject matter experts who can speak credibly to the challenges they face, and that all participants in our study (the first three authors included) are co-researchers. For that reason, rather than collecting data from participants and independently performing qualitative analysis on that data, we used an iterative, consensus-based approach to draw out themes among our experiences. To help lend greater validity to the work and mitigate bias, we draw upon the peer-reviewed literature to compare and contrast our experiences with those of other DevOps practitioners (see Fig. 1).

All the authors of this study are employed at Sandia National Laboratories, a US federally-funded research and development center (FFRDC)[7]. As national secu-

[7] https://www.sandia.gov.

rity laboratory, Sandia relies heavily on computational simulation and data analysis to achieve its science and engineering objectives; this is made possible through a complex ecosystem of scientific software libraries and applications, some developed internally and others community-owned and hosted on the open web. Orchestrating the development, deployment, and maintenance of those software stacks is a significant DevOps research and development (R&D) challenge and an active area of interest for US national laboratories. During the Fall of 2022, the first three authors recruited participants through the institution's Research Software Engineering Community of Practice (RSE-COP) mailing list and directly from the first three authors' departments (roughly 150 people in total). Participants confirmed their participation over email and submissions for stories were collected using a collaboratively-edited, web-based corporate wiki. The contribution page included guidelines for potential contributors.

In particular, potential contributors were asked to provide stories about their experiences conducting DevOps in scientific software development (successes, failures, challenges, changes, etc.) in topics such as testing, team policies and procedures, technology stack modernizations, tradeoffs (e.g., maintainability for performance), etc. To seed the discussion, the first three authors provided stories and open questions. Recruitment yielded 9 additional participants, and together we produced a set of 13 stories that reflect different aspects of DevOps work at the labs. In addition to the stories, contributors added open questions relating to their story or the DevOps culture and ecosystem. We present those stories and questions in Sect. 5.

To better understand how our experiences map to those of DevOps practitioners outside of CSE contexts, we analyze their challenges and lessons learned through the lens of the scholarly literature on DevOps in industry contexts. In particular, we use two systematic reviews of the literature to frame our analysis: a review of common cultural challenges to DevOps adoption in organizations by Khan et al. [17] and a review of best practices in DevOps by Akbar et al. [18]. We present findings from our analysis in Sect. 6.

5 Results

Following the collection of the stories, we identified four overarching themes: Software Development Lifecycle, Testing, Team Policies and Processes, and Institutional Support. We present the results of those themes here, including open questions posed by the authors, and will discuss them in more detail in Sect. 6.

5.1 Software Development Lifecycle

Software development of all forms will execute, whether explicitly or implicitly, a software development lifecycle (SDLC) model [34]. Popular models are Agile,

Kanban, and (the topic of this paper) DevOps. At their core, SDLC models provide defined structure for software development activities.

CSE teams also employ SDLC models, such as Use Case Driven Development. "Use Cases" are the fundamental piece of business value in most software. They can help clearly communicate the business needs from the customer to the development and research staff. Use Case Driven Development is a methodology focused on using "Use Cases" as the central component to writing software [35].

The application of this methodology to scientific software development can be quite natural. As one author details:

> At the start of a new project, we knew that the customer would have a set of research questions related to their data and domain. To design the software, we modeled each research question as a use case and had a few planning meetings to further define entities and relationships. ... Each use case became a command line tool and separate python module that the customer could use directly. We developed a common set of libraries that define and work with the domain entities and relationships—**S1**.

In practice, this author found that applying Use Case Driven Development strategies to their research resulted in much more cohesive conversations around and development of the research software. Each feature could be directly mapped to a "Use Case," and because of the modularity of the design, it became much easier to augment when new questions were added to the domain.

While Use Case Driven Development is useful for creating software, there is still the consideration of deploying it. A common challenge particularly across national laboratories is how to ensure cohesive usage across differing customers, networks, computing systems, and architectures. Two of the authors describe their solutions to this challenge:

> We have started using Docker containers to rapid and flexibly deploy software to our customers. ... Using this set of Docker containers removes the concern about customizing an environment on the customer machine(s). Instead, we can customize everything on our end and then send them the set of Docker containers as a zip file. In practice this has increased our success rate of deployment and allowed us to ensure that our development environment is nearly identical to our deployment environment—**S2**.

While this solution is tuned to an external customers' needs, another of the authors instead looked at how to resolve differing build environments within the same development team:

> One of the consistent workflow problems that my colleagues and I have run into, especially when on-boarding new team members, is to get someone set up with the proper development environment, the correct versions of libraries, etc. for a particular project. ... In the last few years, a number of my colleagues and I have managed our build environments using the Nix system, and it has been quite beneficial to our projects. When on-boarding new people, all they have to do is execute the command `nix develop` in the root of the source tree. Occasionally there is a hiccup, but the vast majority of the time they get a fully configured development environment without any additional effort. This saves us days or weeks of frustration—**S3**.

These stories detail successful application of industry-standard DevOps solutions to scientific software development needs.

5.2 Testing

The challenges surrounding testing scientific software have been well-documented. In Kanewala and Bieman's literature review, they detailed that these challenges come in two forms: technical and cultural [36]. The authors have experienced the problems in both of these major categories.

Technical In the category of technical challenges, Kanewala and Bieman further categorize into four sections: (1) test case development, (2) producing expected test case output values, (3) test execution, and (4) test result interpretation. Independent of this literature review, the authors provided stories (some failures, some successes) that fall into each of the four sub-categories.

With regards to (1), one author specifically calls out difficulties relating to number of potential parameters:

> Some bugs were able to slip past our CI/CD. These bugs were usually missed because the CI/CD did not fully exercise the parameter space (e.g., build options). The team is looking to fix this by either running a full factorial parameter matrix or by decomposing the behavior into independent units which can be tested independently rather than compositionally—**S4**.

Another author also alludes to configuration options in their testing infrastructure:

> I am member of a software package that consists of dozens of sub-packages with thousands of configuration options, all of which application teams rely on for their specific sub-package. Continuous integration testing entails vetting that the code-base works with specific configuration settings; toolchains such as GCC, CLANG, and CUDA; and HPC architectures—**S5**.

The same author also details challenges with regards to (3):

> The CI testing infrastructure is currently limited to using a custom automation tool that pulls the proposed code changes into [our institution's] networks. The tool must then launch and monitor the tests. With build and test times averaging six hours and up to 11 builds run per change, there is a huge maintenance and resource cost. Frequently, something goes wrong during the average build and results in test results never being reported back to the developer—**S5**.

While not explicitly stated within (3), modernization of testing infrastructures was another consideration of the authors with respect to test execution.

> I am a DevOps contributor to a scientific Python package aimed at optimization modeling. ... In early 2022, a downstream dependency requested support for Python 3.10. This request revealed a problem ... [that] required an entire refactor of the testing suite to use the popular and regularly maintained package pytest. On the surface, this refactor seemed simple. The issue: because of the age of the scientific package, ample homegrown infrastructure had been built specifically around nosetests that needed to be preserved (e.g., dynamic categorization, dynamic test creation). What was anticipated to be a quick and simple fix took over a month of dedicated, time-intensive work to convert in order to maintain expected functionality—**S6**.

Multiple authors have had to contend with one of the largest plagues in scientific software: repeated code (4).

> Often times in engineering science software libraries, there are many similar implementations (models, etc.) that could share a lot of the same core tests. This includes integration tests in addition to unit tests. The problem is that the many similar implementations rarely share the same tests, since they are typically developed in series rather than in parallel. ... This causes issues related to inconsistent testing of implementations when certain tests are included somewhere and not elsewhere, and a lack of testing efficiency when the tests are copied in multiple places. Theses issues can inhibit the quality of the software and cause further development to become less straightforward—**S7**.

In some cases, the repeated code led to the developers struggling with the same bug for over a decade:

> While running an important application deployed to an HPC cluster, it was discovered that there was a scalability bug in a math library we develop, resulting in a nearly 30% drop in performance. ... What was interesting, however, was the unusually long-lived history of the bug. The team of developers who found the bug discovered that the exact same bug had been introduced, found, and fixed in the math library multiple times over the years. The offending code was first introduced in three packages between 1998–2000 and fixed in 2005, copied line for-line into a fourth package in 2004 and fixed again in 2015, and finally introduced into the last package in 2014 and fixed in 2017. In each case, the discovery and solutions were socialized, comments were made in the code, and notes were left in an issue tracker, but that information did not flow to the right parties in each subsequent incident—**S8**.

While the DevOps challenges surrounding testing in scientific software development are well-known and commonly experienced, testing has long-reaching implications on the success and stability of the software. One author describes the benefit of formal verification (2):

> I regularly contribute to a large code base in a domain that is notorious for begetting intricate software systems that contain all sorts of subtle bugs, some which can live for decades. The particular code that I contribute to is unique because it has a formal, end-to-end, proof of correctness, which is mechanically checked against the code at compile-time. [In o]ne of the improvements that I worked on ... I hit a wall in the proof and could not proceed further. When investigating why the proof wouldn't work, I realized that my implementation was incorrect. I had missed a corner case. In the end, I might have been able to catch this corner case with tests, but the application domain is complex enough where that bug could easily have gone unnoticed. I was able to fix this bug before it ever made it into the code-base—**S9**.

Some of these experiences, however, crossed the line from technical into cultural.

Cultural Examples of technical challenges experienced by the authors are endless. They are not, however, strangers to the cultural concerns as well. While story **S9** talks about the benefit of formal verification, the author also notes a significant problem—formal education:

> The overwhelming majority of scientists and software engineers have absolutely no experience in formal verification. It just seems like a black art. How do we educate our workforce about practical formal verification? Just as with "design-for-test," how do we integrate "design-for-verification" into our programming curricula?—**S9**

One author calls out specific ways that education can be applied:

> The greatest difficulty I've had as a DevOps practitioner in the research software world has been getting decision makers to understand the complexity involved in designing, building,

> maintaining, and extending infrastructures to accelerate the delivery of value from development into operations, amplify feedback loops, and enable a culture of continual learning and experimentation. Building shared understanding is a prerequisite for culture change. Applying this to DevOps in the research software space means algorithmists, simulations experts, analysts, managers, etc., must dedicate time away from regular milestone-driven activities to learn what the DevOps paradigm shift actually is, what changes in thinking it requires, and what kind of activities it entails. This can be done, in part, through studies and discussions of books such as *The DevOps Handbook*, *The Phoenix Project*, *The Unicorn Project*, or *Continuous Delivery*—**S10**.

This author points out a secondary problem—software development activities fall below research priorities. In reference to story **S6** regarding the conversion from `nosetests` to `pytest`:

> As software projects mature, so, too, should their support for modern technology. In this case, once the main test driver was announced to no longer be supported or updated, it would have benefited the scientific software development team to begin the transition to a newer, regularly maintained test driver. Instead, the team relied on the hope that it would continue to work... Until it didn't. It is essential for teams to preemptively address these concerns rather than wait—**S6**.

The overarching consensus for cultural issues throughout these stories: the expertise of the DevOps developers needs to be given the same priority as those of the domain scientists.

5.3 Team Policies and Processes

Software quality is dependent on the effectiveness of a project's DevOps practices, but this goes further than just quality—culture also plays a critical role [37]. As stated by Perera et al.: "Culture is another important factor because it changes the way in which teams work together and share the responsibility for the end users of their application."

As detailed above, one author contributes to a package with dozens of subpackages and thousands of configuration combinations. In their case, the difficulties in testing lead directly to poor teaming dynamics:

> Developers are frustrated while the DevOps team has an endlessly growing backlog of work. While this automation is better than no CI testing, it has resulted in poor teaming dynamics and a large maintenance burden. Additionally, this DevOps team is so busy maintaining configurations and keeping the infrastructure running that they have no time (or available options) to improve the infrastructure. As a result, teaming dynamics continue to digress, and it is difficult to retain team members—**S5**.

Teaming dynamics can be strained more with changing policies and procedures, though the risk may pay off in the long run. One author details their package's shift from subversion to git and hosting on GitHub:

> The more mature a project is, the more likely there has been turmoil over changing technologies. For example, one package created by [laboratory] scientists started on subversion

> and a [internally]-hosted repository for many years before transitioning to GitHub. In the subversion days, developers would commit directly to the main branch, which led to frequent bugs, breakage, or repository pollution. ... When the team eventually transitioned to GitHub, pull requests and code reviews were added into the development workflow. Initially this caused conflict on the team as it slowed down the development speed and introduced "extra overhead." This change, however, improved the overall stability of the code base. ... The code reviews also generally have raised the quality of the code base. This has allowed developers and maintainers to shift focus to improving existing infrastructure and modernizing the code and its dependencies and has allowed a larger community of contributors to add their contributions without lowering the quality of the package—**S11**.

This example highlights two main points: (1) change can be difficult but overall bring about better processes and stability, and (2) buy-in is essential for adopting team policies. Once fully adopted, solid team policies can save a team from disaster. Another author provides a perfect example:

> A graduate student intern collaborating with [laboratory] researchers was traveling to a customer meeting to demonstrate their software product, including very recent updates to the tool and presentation. During travel, the student's laptop—the primary machine used for development—was stolen. Fortunately, members of the team had diligently maintained comprehensive remote version control systems. Upon arrival at the meeting location, the presentation, software tool, and demonstration were downloaded to a colleague's system, reverted to a stable version, and operations proceeded with minimal disruption. Customers who were unfamiliar with the team's version control practices were thoroughly impressed at the team's resilience given the circumstances—**S12**.

This same author goes further to say, "Incorporating DevOps best practices into workflows hedges against unforeseen catastrophe. The initial investment and learning curve associated with applying these strategies routinely—especially for scientists who may feel little need to otherwise learn "software development" skills—has the potential for serious payoff in the long run."

5.4 Institutional Support

Providing institutional support for development opportunities (professional and technical) is a key contributor to retention of staff and staff happiness [7]. In particular, providing opportunities for staff recognition, growth, and ability to influence the organization's direction through implementing cultural changes, teaching new best practices, and advancing an organization's shared understanding of DevOps best practices can boost retention and create a sense of belonging.

Professional development opportunities can come in different forms: taking training, developing training, community building, etc. There has long been a history of software developers within CSE teams being fragmented in their work [38]. This type of fragmentation primarily affects access to professional development opportunities that would improve the overall state and trustworthiness of scientific software. As one author points out:

> The problem is we have a long history of insufficiently funding and staffing the [software development] activities that would solve our issues and prevent them from happening again in the future—**S10**.

In their study, Raybourn et al. found much of the same: "The next opportunity area for incentivizing software quality engineering as part of a culture's practice comes in the allocation of funding for quality in software projects. In interviews with personnel from two distinct Centers, lack of monetary resources was the most-frequently stated challenge facing developers. Participants mentioned fragmentation, competition for funding, lack of rewards for development work, etc. As one participant plainly stated, "You can't have quality without the money to pay for it."" [39]

Institutional support is necessary not only for direct development activities, but also training. One such author was fortunate to be financially supported in creating and delivering a training for Git:

> Running a 2-hour long workshop and teaching others about Git was a first for me. Once the reality sank in and I was getting ready for the talk and the hands-on activity, I realized that I didn't know the tool as well as I ought to to be able to present to others with any sense of authority. I asked around for resources and compiled the important topics to cover. From practicing Git for about a decade now, I knew how to use it and navigate its documentation to get the work done, but I did not feel comfortable answering questions on the spot about its nuances. Being confronted with my lack of knowledge forced me to better my understanding of Git and it allowed me to become a more proficient user—**S13**.

This activity provided the author with the ability to not only inform others, but also strengthen themselves as a developer.

5.5 Open Questions

- How can we document and standardize our deployment process to make it even easier to release and deploy software for researchers without DevOps training?
- How do you find the time/funding to make such drastically large changes?
- Is it possible to get funding agencies to start requiring technical debt reduction plans as part of the proposal process? How do we promote a culture of mutual ownership of technical debt?
- How do we build a culture among those who identify as scientists first that software best practices should be adopted and adhered to, even when the initial investment is high?
- How do you appropriately allocate time and resources for making small incremental changes in an effort to avoid technical DevOps debt accumulation and poor teaming dynamics?
- Can knowledge sharing among development teams be encouraged by creating formal roles for people doing DevOps work? What would that look like?
- Are there tools that can help with code clone detection at production-scale?
- Is there a well-defined process that already exists (code agnostic, of course) that tackles the problem of duplicate tests?

6 Analysis

The stories recorded above offer a window into DevOps practice at a major scientific institution. In this section, we seek to ground those experiences in the scholarly literature around DevOps in industry to draw out similarities and differences to DevOps in CSE contexts. In Table 1, we provide summaries of the key lessons learned in each story; these recommendations may be valuable for computational scientists, engineers, and those doing DevOps work in CSE contexts.

We first compare the challenges faced and addressed by our participants to those commonly attested in the scholarly literature on DevOps practice. Using Khan et al.'s systematic review of challenges to adoption of a DevOps culture as a guide, we found support in our stories for eight out of the ten most-frequently mentioned challenges in the literature (see Table 2). That is, many of the obstacles encountered and overcome by our participants are not unique to CSE software development, which lends credibility to the generalizability of our findings and recommendations. Successful implementation of DevOps tools and practices requires addressing quality in the systems-under-test and the tests themselves (S4, S7, S9, S13), and a lack of knowledge about DevOps can hinder implementation (S9, S10, S12) or lead to suboptimal decision-making (S1, S2). Where DevOps infrastructure exists, it must be actively maintained to keep it up-to-date (S6, S11) and to manage complexity creep (S3, S5). Having the support of leadership and buy-in from development teams (S10, S11) is critical to success, as is promoting communication and collaboration across teams and organizations (S8).

Next we compare the successes and better practices experienced by our participants to Akbar et al.'s "prioritization-based framework of the DevOps best practices based on evidence collected from industry experts." We found support in our stories for ten of the twelve top-ranked practices (see Table 3). It is worth noting that Akbar et al. divide their rankings into global vs. local ranks (e.g., overall highest priority vs. priority within a common category). The top ten globally *all* fall within the "Culture" category; we opt to use the top three ranked practices in each local category instead to diversify the conversation. As shown, besides two relatively industry-centric prioritizations ("microservices" and "tools to capture requests"), all of the other top-ranked practices are reflected in the stories. DevOps adoption and implementation is seen as successful within a more collaborative culture with a shared value system (S8, S10, S11, S13), which is bolstered by the education of both staff and leadership (S12). The practices succeed only insofar as there is standardized buy-in and adherence (S1, S8, S10, S11), and constant communication is seen as necessary to minimize the potential for issues and inefficiencies (S5, S8, S10, S13). Only one story touches on rapid deployment as a way to receive constant feedback from customers (S2). When it comes to automation, there is a strong emphasis on the importance of continuous integration and testing (S4, S5, S6, S7, S9), but this cannot be applied unless a team can decide what it wants to actually achieve first (S1, S6, S11). Taking this one step further, DevOps practices are seen as overall more effective when adopted early while a project is still small (S8, S10, S11, S12) and are more successful with

Table 1 Summaries of lessons learned from the stories generated during the storytelling exercise.

Story	Topic	Summary
S1	*Align Tools and Methodologies*	Teams can be highly productive when their DevOps tooling matches with and supports their development methodology.
S2	*Embrace Virtualization and Interactive Media for Deployment*	Technologies like containerization (e.g., Docker) and interactive computing media (e.g., Jupyter notebooks) enable projects to rapidly deploy CSE software to customers.
S3	*Adopt Dependency Managers*	As CSE software ecosystems continue to grow and mature, emerging dependency management solutions like Nix and Spack can help manage that complexity.
S4	*Use a Mix of Testing Strategies to Ensure Code Quality*	While CI/CD is effective at catching certain kinds of bugs, it is not a panacea. It is important to test across different environments and configurations with different tiers of testing and to design software to be more testable.
S5	*Maintain Good Tooling to Have Good Teaming*	Inefficient, fragile, and complex infrastructure drains the energy and morale of DevOps practitioners. Refreshing and incrementally improving infrastructure is vitally important for effective teaming.
S6	*Manage Test Infrastructure*	Teams must proactively maintain their build and test infrastructure, as it is guaranteed to fail eventually.
S7	*Design Software for Testing*	CSE software modules and components should be built with testing in mind, such as by having common interfaces against which tests can be written.
S8	*Address Human Communication Bugs*	As projects scale up, it becomes essential not only to put in place DevOps infrastructure but also to build out processes and practices to facilitate coordination between teams.
S9	*Leverage Static Analysis and Formal Methods*	Some of the worst, most complicated software bugs can be prevented through automated software analyses, ranging from built-in type-checking to robust theorem proving tools.
S10	*Promote Shared Understanding for Culture Change*	For DevOps efforts to succeed, there needs to be a shared understanding throughout the organization of what the DevOps paradigm shift actually is, what changes in thinking it requires, and what kind of activities it entails.
S11	*Build Consensus Around DevOps Tools and Practices*	Mature projects accrue inertia around existing tools and practices, and securing buy-in on new tools and practices is essential for adoption to succeed.
S12	*Teach DevOps Practices and Principles*	DevOps practitioners should educate their peers on best practices, both to reinforce their own knowledge and to improve the state of practice in their institutions.
S13	*Plan for Resiliency*	Getting CSE teams to adopt DevOps tools and practices helps guard against unexpected catastrophes, and this should be emphasized as a benefit of having those tools and practices in place.

Table 2 Common cultural challenges to implementing DevOps in organizations (as identified by Khan et al. [17]) attested and/or mitigated in the stories we collected.

Common cultural challenges	Discussed/mitigated in stories
Lack of collaboration and communication	S8
Lack of skill or knowledge about DevOps	S9, S10, S12
Culture of blame (criticism)	–
Lack of intentional DevOps approach	S1, S2
Lack of management support	S10
Trust and confidence issues	S10, S11
Complicated infrastructure	S3, S5
Poor quality	S4, S7, S9, S13
Security issues	–
Legacy infrastructure	S6, S11

Table 3 Top-ranked DevOps practices (as identified by Akbar et al. [18]) mirrored in the stories we collected

Prioritized cultural practices	Discussed in stories
Collaborative culture with shared goals	S8, S10, S11, S13
Readiness to utilize a microservices architecture	–
Education of executives	S12
Prioritized sharing practices	Discussed in stories
Standardized processes and procedures	S1, S8, S10, S11
Continuous feedback to address issues and inefficiencies	S5, S8, S10, S13
Reduce batch size to increase communication	S2
Prioritized automation practices	Discussed in stories
Decide what to do first	S1, S6, S11
Continuous integration and testing	S4, S5, S6, S7, S9
Use tools to capture every request	–
Prioritized measurement practices	Discussed in stories
Effective and comprehensive measurement/monitoring	S4, S9
Start DevOps on small projects	S8, S10, S11, S12
Integrated configuration management	S2, S3

frequent monitoring and adaption of practices as a project matures (S4, S9). They can also lead to better ease-of-use when there is a cohesive and integrated system for managing dependencies and environments (S2, S3).

Overall, we see a trend that DevOps practices and perils align across industry and CSE contexts—however, within CSE, there are necessary adjustments. For example,

while our participants agreed with the necessity for standardized processes and procedures, the scope differs. In industry, this standardization is desired across the entire organization; for CSE, it's enough for the standardization to be across a project or projects that collaborate. They also do not need to be completely standardized; rather, they should be appropriately scaled as funding and expertise allow. CSE teams can succeed by applying appropriately scaled practices to their projects, but they must avoid the perils that come with either trying to do too much or too little.

Open questions remain surrounding how to do this adaptation in a scaleable, reproducible manner. For CSE projects that start as completely exploratory and state-of-the-art, at what point does the team "scale up" their DevOps practices? What tools and methods for continuous integration should be applied at each stage of maturity? How often should a research code release and deploy, and what does operations and maintenance really look like after the fact? These are all potential future avenues for research.

7 Threats to Validity

As with every study, this one has its threats to its validity. We will discuss three such potential threats here: (1) generalization; (2) qualitative nature of the data; and (3) personal biases.

The authors (and participants) in this study all come from the same institution and have similar job types. This presents a threat in being able to widely generalize the results. In particular, we cannot say for certain that all DevOps practitioners who collaborate with CSE teams will have these same views; however, we have aimed to mitigate this by pairing the authors' experiences with support from peer-reviewed and gray literature.

As for the second threat, all data presented here is purely qualitative. It can be difficult to establish concrete trends and conclusions; however, as with all studies, we consider this a trade-off. For this study, we believe the stories encapsulate a fuller, richer, and more complete picture of what DevOps work looks like in practice. Additionally, similar to the previous threat, we have attempted to mitigate this with peer-reviewed and gray literature to provide more breadth of experience to our own.

As a final note, we want to call out the potential of our own personal biases. Because the authors themselves are the participants, we recognize the potential to skew the results based on our assumptions and feelings rather than actual fact. To mitigate this, we collectively reviewed perspectives and content contributed by each other. In this way, we aimed to minimize possible bias while still preserving the expressed opinions.

8 Conclusion

The DevOps movement may have its roots in industry, but it has branched into scientific software development. As software becomes more integral to the advancement of science, so do the processes and procedures used to create scientific software.

In this article, the twelve authors shared thirteen unique stories of their experiences as DevOps practitioners within CSE teams within Sandia National Laboratories, including lessons learned and residual open questions. Using a participatory action research approach, we combined these stories with gray and peer-reviewed literature to analyze the commonalities and differences between industry and scientific software DevOps practices.

We found that many practices and perils are mirrored. CSE teams experience the same cultural challenges as industry while emphasizing similar priorities on testing, collaboration, and starting early. With that in mind, DevOps practices cannot be perfectly applied out-of-the-box to a CSE project. There needs to be adaptation, education, and buy-in to create success.

DevOps in the CSE context is a research realm that is rich in unanswered questions. We shared some of these, as well as lessons learned, to add to and promote further conversation around pragmatic practices and potential perils in scientific software development.

Acknowledgements Illustrations used with permission from thenounproject.com by Kamin Ginkaew, Adrien Coquet, Numero Uno, and Gregor Cresnar.
Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA-0003525. SAND2022-16099 C.

References

1. Joseph E (2020) SC20 update on the ROI and ROR from investing in HPC. https://www.hpcuserforum.com/ROI/
2. Douglas K, Stephen L, Irene Q (2018) Exascale computing in the United States. Comput Sci Eng 21(1):17–29
3. Wilhelm H, Leslie C, Simon H, Heather P, Thanassis T (2020) Open source research software. Computer 53(8):84–88
4. Heise C, Pearce JM (2020) From open access to open science: the path from scientific reality to open scientific communication. SAGE open 10(2):2158244020915900
5. Arne J, Wilhelm H (2018) Software engineering for computational science: past, present, future. Comput Sci Eng 20(2):90–109
6. Tennant JP, Agrawal R, Baždarić K, Brassard D, Crick T, Dunleavy DJ, Evans T R, Gardner N, Gonzalez-Marquez M, Graziotin D et al (2020) A tale of two 'opens': intersections between free and open source software and open scholarship
7. Mundt M, Beattie K, Bisila J, Ferebaugh C, Godoy W, Gupta R, Guyer J, Kiran M, Malviya-Thakur A, Milewicz R, Sims B, Sochat V (2022) For the public good: connecting, retaining, and recognizing current and future RSEs at national organizations (under review). In: Special issue on the future of research software engineers in the US. Comput Sci Eng

8. Mundt M, Milewicz R (2021) Working in harmony: towards integrating RSEs into multi-disciplinary CSE teams. In: Proceedings of the 2021 workshop on the science of scientific-software development and use. U.S. Department of Energy, Office of Advanced Scientific Computing Research
9. Mundt M, Sochat V, Katz DS, Gesing S, Melessa Vergara VG (2021) DOE software stewardship challenges in diversity, professional development, and retention of research software engineers. In: Responses to the request for information on stewardship of software for scientific and high-performance computing. U.S. Department of Energy
10. Kelly DF (2007) A software chasm: software engineering and scientific computing. IEEE Softw 24(6):120–119
11. Leonardo L, Carla R, Fabio K, Dejan M, Paulo M (2019) A survey of DevOps concepts and challenges. ACM Comput Surv (CSUR) 52(6):1–35
12. Chevalier JM, Buckles DJ (2019) Participatory action research: theory and methods for engaged inquiry. Routledge, Milton Park
13. Lutters WG, Seaman CB (2007) Revealing actual documentation usage in software maintenance through war stories. Inform Softw Technol 49(6):576–587
14. Davis J, Daniels R (2016) Effective DevOps: building a culture of collaboration, affinity, and tooling at scale. O'Reilly Media, Inc
15. Mueller E (2010) What is DevOps?
16. Bernholdt DE, Cary J, Heroux M, McInnes LC (2021) Position papers for the ASCR workshop on the science of scientific-software development and use. Technical report, USDOE Office of Science (SC)
17. Khan MS, Khan AW, Khan F, Khan MA, Whangbo TK (2022) Critical challenges to adopt DevOps culture in software organizations: a systematic review. IEEE Access 10:14339–14349
18. Azeem Akbar M, Rafi S, Alsanad AA, Furqan Qadri S, Alsanad A, Alothaim A (2022) Toward successful DevOps: a decision-making framework. IEEE Access 10:51343–51362
19. Hal F, Ben B, Robinson P, Saswata H-M, Bill S (2021) Responses to the request for information on stewardship of software for scientific and high-performance computing. Technical report, USDOE Office of Science (SC)
20. Beattie K, Gunter D (2021) Useful practices for software engineering on medium-sized distributed scientific projects
21. Dubey A (2020) When not to use agile in scientific software development. In: The 2020 Collegeville workshop on scientific software
22. Ellingwood N, Rajamanickam S (2020) Practices and challenges of software development for a performance portable ecosystem. In: The 2020 collegeville workshop on scientific software
23. Finkel H (2020) The many faces of the productivity challenge in scientific software. In: The 2020 Collegeville workshop on scientific software
24. Windus T, Nash J, Richard R (2020) Scientific software developer productivity challenges from the molecular sciences. In: The 2020 Collegeville workshop on scientific software
25. De Bayser M, Azevedo LG, Cerqueira R (2015) ResearchOps: the case for DevOps in scientific applications. In: 2015 IFIP/IEEE international symposium on integrated network management (IM), pp 1398–1404
26. de Bayser M, Segura V, Azevedo LG, Tizzei LP, Thiago R, Soares E, Cerqueira R (2022) DevOps and microservices in scientific system development: experience on a multi-year industry research project. In: Proceedings of the 37th ACM/SIGAPP symposium on applied computing, pp 1452–1455
27. Gesing S (2020) Increasing developer productivity by assigning well-defined roles in teams. In: The 2020 collegeville workshop on scientific software
28. Adamson R, Malviya Thakur A (2021) Perspectives on operationalizing scientific software. In: The 2021 collegeville workshop on scientific software
29. Patton MQ (2002) Qualitative research & evaluation methods. Sage, Newcastle upon Tyne
30. Titus B, Sumit G, Mario J (2022) Storytelling and science. Commun ACM 65(10):27–30
31. Polletta F, Chen PCB, Gardner BG, Motes A (2011) The sociology of storytelling. Ann Rev Sociol 37(1):109–130

32. Reason P, Bradbury H (2008) Handbook of action research: participative inquiry and practice. Sage, Newcastle upon Tyne
33. Fran B, Colin MD, Danielle S (2006) Participatory action research. J Epidemiol Commun Health 60(10):854
34. Ralf K (2017) Sixty years of software development life cycle models. IEEE Annal Hist Comput 39(3):41–54
35. Alexander IF, Beus-Dukic L (2009) Discovering requirements: how to specify products and services. Wiley, New York
36. Kanewala U, Bieman JM (2014) Testing scientific software: a systematic literature review. Inform Softw Technol 56(10):1219–1232
37. Perera P, Silva R, Perera I (2017) Improve software quality through practicing DevOps. In: 2017 seventeenth international conference on advances in ICT for emerging regions (ICTer), pp 1–6
38. Katz DS, McHenry K, Reinking C, Haines R (2019) Research software development & management in universities: case studies from Manchester's RSDS group, Illinois' NCSA, and Notre Dame's CRC. In: 2019 IEEE/ACM 14th International Workshop on Software Engineering for Science (SE4Science), pages 17–24. IEEE, 2019
39. Raybourn E, Milewicz R, Mundt M (2022) Incentivizing adoption of software quality practices. Technical report, Sandia National Laboratories (SNL-NM), Albuquerque, NM