Challenges and Strategies for Testing Automation Practices at Sandia National Laboratories

Miranda Mundt*
Sandia National Laboratories
Albuquerque, NM, USA
mmundt@sandia.gov

Jonathan Bisila*
Sandia National Laboratories
Albuquerque, NM, USA
jbisila@sandia.gov

Reed Milewicz*
Sandia National Laboratories
Albuquerque, NM, USA
rmilewi@sandia.gov

Joshua Teves*
Sandia National Laboratories
Albuquerque, NM, USA
jbteves@sandia.gov

Michael Buche
Sandia National Laboratories
Albuquerque, NM, USA

Jonathan Compton
Sandia National Laboratories
Albuquerque, NM, USA

Jason M. Gates
Sandia National Laboratories
Albuquerque, NM, USA

Kirk Landin
Sandia National Laboratories
Albuquerque, NM, USA

Gerald Lofstead
Sandia National Laboratories
Albuquerque, NM, USA

***Abstract*—Sandia National Laboratories is a premier United States national security laboratory which develops science-based technologies in areas such as nuclear deterrence, energy production, and climate change. Computing plays a key role in its diverse missions, and within that environment, Research Software Engineers (RSEs) and other scientific software developers utilize testing automation to ensure quality and maintainability of their work. We conducted a Participatory Action Research study to explore the challenges and strategies for testing automation through the lens of academic literature. Through the experiences collected and comparison with open literature, we identify these challenges in testing automation and then present strategies for mitigation grounded in evidence-based practice and experience reports that other, similar institutions can assess for their automation needs.**

***Index Terms*—automation, testing, scientific software development, research software engineering**

I. Introduction

US national laboratories perform fundamental research and development in the national interest, and computing is increasingly essential to supporting the needs of these efforts [1]. This is highlighted by the recent Department of Energy request for information on software stewardship [2] and the creation of the Exascale Computing Project[1]. Paired with this is the need for quality software to ensure quality results – a necessity enabled by regular testing [3]. Automation of software testing using practices such as DevOps is common practice in industry and is being adopted into scientific software development as it is necessary in order to meet the demands of the complex problems that it solves [4]. In order to even use Commercial Off-the-Shelf (COTS) products such as Jenkins[2] to automate testing, it is necessary to train the workforce to use those tools. Even with these tools, there may be gaps in functionality for research software engineering.

With this in mind, we propose two questions to investigate:

- **RQ1**: What are the challenges experienced by Research Software Engineers (RSEs) in testing automation at a large US national laboratory?
- **RQ2**: What strategies have been employed by Research Software Engineers (RSEs) to address these challenges?

In order to begin answering these questions, we use a Participatory Action Research (PAR) study to gain a foothold on answering these questions at Sandia National Laboratories (SNL). By studying RSEs and their practices at SNL, we gain information on at least one national laboratory and present a framework for conducting similar studies at other institutions for the purpose of identifying practices employed by RSEs more broadly.

In this paper, we illustrate the nature of the heterogeneous scientific software environment at SNL. We then highlight some of the current solutions and best practices employed by RSEs with regards to automation testing. We discuss some of the open challenges in automation testing for scientific software development and present strategies to mitigate these challenges. Throughout this study, we ground our findings in the open literature, comparing and contrasting our environment, solutions, and challenges with those of the larger research computing community.

II. Background

Automation has long been a topic of conversation in software development. Boehm argued extensively in the 1980's for there to be a push to automate repetitive, labor-intensive

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & 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-NA0003525.

Presented at USRSE2023, Chicago, IL, October 16-18, 2023.

* The first four authors contributed equally to this work.

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

[2] https://www.jenkins.io/

tasks in order to improve overall developer and project productivity [5], [6]. In the modern era, automation has taken a prominent role in software development. It enables reproducibility through automated workflows [7], verification through testing [8], better interdisciplinary teaming [9], and is seen as integral to successful Agile software development [10].

Within computational science and engineering (CSE), automation has a mixed history. Heaton and Carver, in a systematic literature review, found that it is widely claimed that "Scientific software developers benefit from using a wide range of testing practices from software engineering," up to and including automation for testing, refactoring, and documentation' [11]. However, they also note, "The effectiveness of the testing practices currently used by scientific software developers is limited." According to a recent study, Carver et al. determined that, in a typical mixed group of researchers and developers, only half have formal software engineering training, and only a quarter have time for further professional development [12].

To add another layer of complexity, SNL has a highly heterogeneous scientific software environment. Depending on the project and its collaborators, a CSE team may use GitHub, GitLab, and Jenkins while attempting to support several different compiler and operating system versions per piece of software. This is not entirely unique to national laboratories; in one case study, Karhu et al. detailed a team that had failed on multiple occasions to implement automated testing because "[t]he systems were large and complex, and also depended on other suppliers' systems. Especially interoperability testing was found difficult because often it could only be done at the customer's site." [13] Complicating matters further, communications methods are also often disparate, with unclear protocols on which platform to use with whom and for what. For instance, a single project may use email, Microsoft Teams, Slack, Mattermost, tapping colleagues on the shoulder, or other methods to communicate with internal and external collaborators. Teams may collaborate either in person or remotely, leading to seminars on best practices for effective communication in a dispersed teaming environment[3]. This is even mentioned as a barrier to open source software development generally [14].

A primary role of RSEs is to improve quality and productivity within scientific software development. In the foundational paper on RSEs, Baxter et al. describe RSEs as those who "come from a research background but [are] also skilled software developer[s], and relish challenge of not just developing code to solve a problem but doing it well." [15] Maimone describes RSEs at Northwestern University as providing knowledge of "version control, testing, deployment, software design, distribution, documentation, automation, and more" to research software development [16]. Cosden describes something similar within the Princeton University RSE Group, which also provides training for research scientists on software engineering practices including automation, version control, and performance [17]. Within SNL, the Department of Software Engineering and Research employs a "Research, Develop, and Deploy" pipeline that aims to provide assistance throughout the software development lifecycle [18].

III. Methodology

For this paper, we conducted a participatory action research (PAR) study. These studies focus on direct participation of the authors as opposed to secondary participation, as with interviewing a passive participant [19]. The inspiration for this technique comes from the work of Lutter and Seaman [20], which the authors have used previously to analyze DevOps successes and challenges within SNL [21]. Using this method, we analyzed our experiences as RSEs in scientific software development teams to identify key themes. We then compared these themes to those found in the wider body of literature about the use of automation practices across both industry and scientific software development.

The main authors recruited participants for the study by reaching out to the Sandia Research Software Engineering Community of Practice (RSECOP) and other potentially interested departments at SNL (about 150 staff members in total). Interested participants were asked to detail their experiences with automation as researchers and software practitioners (through stories, data, opinions, etc.). The recruitment email included a list of topics of interest such as testing, pipelines, how decisions were made regarding automation tools, challenges and successes, and more. Interested parties were then advised to add their experience reports to a web-based, collaborative wiki page. This page included guidelines for potential contributors, which included a request for details on the underlying scientific domain as well as the automation tools in use. Recruitment yielded a total of seven participants, who together produced a set of eight stories that detail a selection of established practices for testing automation at SNL. Two authors did not contribute stories but collaborated in the analysis of the other experience reports to minimize bias and provide objective input. We present those stories in Section IV.

To better understand how our experiences map to those of RSEs and other practitioners outside of SNL, we analyze these stories through the lens of literature surrounding software quality and automation practices. In particular, we compare to several literature reviews that speak to both the impediments for testing automation [22] and on improving test automation maturity [23]. The results of this analysis can be found in Section V.

IV. Results

Following the collection of the stories, we identified four overarching themes: continuous integration, heterogeneous computing environment, interdisciplinary collaboration requirements, and lack of professionalization of software engineering practices. We present the results of those themes here, with a more in-depth discussion in Section V.

[3] https://www.exascaleproject.org/strategies-for-working-remotely/

A. Continuous Integration

Continuous Integration (CI) is a "development practice ... in which members of a team integrate and merge development work (e.g., code) frequently [and] includes automated software building and testing." [24] In this practice, a team normally does work incrementally, generally targeting a single change or related changes before merging them into the main production code base when completed. As part of this process, the incremental changes are automatically tested to ensure no breaking changes enter the main code base.

Historically, software development followed a waterfall method [25]. In 2001, the Agile methodology was introduced – a model in which a product is continuously improved in short spurts with regular feedback from the customer [26]. Following this shift to continuous improvement, DevOps was introduced as a concept in late 2009 [27]. This extension of Agile puts emphasis on CI in the software development lifecycle, including a particular focus on the automation of repetitive tasks such as testing.

CI was mentioned in most of the experience reports. As one author details:

> 💬 For one of my projects, we have the challenge of needing a specific dataset, with a specific environment, with a specific set of code we have written to support analysis. We found that using GitLab CI/CD pipelines and pytest with nbmake makes this a lot easier. In particular, this accomplishes two things for us. One, we can build Docker containers with exactly the right Python environments we need, the services attached as a microservice, and the data embedded in the container. Two, our example notebooks for our analytics can double as "tests" in the CI/CD pipeline by using nbmake and pytest! – **S1**

This author utilizes automated pipelines and testing packages to streamline the process of generating and releasing a dataset and its analysis. The pipeline expedites a previously manual process while also incorporating tools to containerize the results for rapid and seamless deployment to stakeholders.

Of note, this author details using a mix of tools to achieve the goal. This is also reflected in many of the other authors' stories (five out of eight), with some taking it a step further in describing the complex environments across which a single code must compile, build, and run. For example:

> 💬 The development of exascale codes on bleeding-edge hardware requires testing across a variety of heterogeneous machines. For each machine, there may be multiple supported programming environments, and for each environment, there may be multiple ways to configure the code. Ensuring the code clones, configures, builds, tests, installs, and runs successfully for the plethora of desired permutations is a daunting task. When considering the testing of multiple long-lived branches, and the desire to have both development and production versions of the CI infrastructure, you're looking at maintaining hundreds of jobs. – **S3**

While CI may help with this task, the sheer volume of required combinations results in added complexity and maintenance. Some may say this is akin to Software Product Lines (SPL): "a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way." [28] While SNL can be argued to have a particular mission or market segment (e.g., national security), they are not a software development institution. Rather, the software is a by-product of the mission statement. The same author speaks to a similar issue for a different project and details the way the project has stitched together a development pipeline across multiple CI tools:

> 💬 For one project, this means the following:
>
> - The main repository lives out on GitHub for the sake of collaboration.
> - On an internal, more-restricted network, a Jenkins server is used to run CI jobs on a number on physical machines (high-power servers, HPCs, test-beds).
> - Certain components of the CI infrastructure live in the main repository on GitHub.
> - Other CI components live in additional repositories on a GitLab instance on an internal, less-restricted network.
> - Parts of the CI infrastructure are tested with GitLab CI/CD on virtual machines spun up with OpenStack.
>
> – **S5**

For this case, the author details a pipeline that, while providing some functionality, is restrictive due to security concerns and the requirement to support multi-institution collaboration. Results generated on the internal Jenkins server, for example, can only post "Pass/Fail" on the GitHub repository, so external collaborators can see if their change fails tests but cannot see why without requesting more details from a direct SNL employee. These stitched-together pipelines are experienced by other authors as well. Another author tells of their performance testing pipeline:

> 💬 Using a mix of tools (GitHub, GitLab, Jenkins, Python, web servers), we have created an automated pipeline to run benchmark tests on each change to the main branch of the repository, add that data to an archive (that now has over two years' worth of previous runs), and analyze the data over time to find performance gains and losses. This data is published via plots and tables to a website, accessible publicly, with notifications of significant variations appearing at the top of the portal.
>
> Due to issues with access to hardware and web-hosting services, plus sensitive or proprietary information in some of the tests, this pipeline runs across

> a series of services. The code is hosted on GitHub; the raw data archive is hosted on an internal [SNL] GitLab instance; the performance test suite runs on Jenkins; the website is administered by a small team and is likely to be decommissioned, if possible, in the near future. – **S8**

While CI has been overall beneficial to the authors, the necessity to piece together unique toolchains to work around and satisfy corporate policy and external collaborators has effected extra complexity. This leads to the next theme - the heterogeneity of the SNL computational science and engineering environment.

B. Heterogeneous Computing Environment

By heterogeneous, we mean to say that software developers at SNL experience a computationally focused system wherein the resources available and the resources required are heavily variable depending on what and where a problem is being solved. In other words, the environment requires support for multiple institutions, software languages, tools, or hardware, and a diverse scientific computing staff with varied expertise both in software and research domains, an experience detailed by other RSE groups across national institutions [29]. Several authors have experienced struggles and successes with regards to the heterogeneous environment within SNL. For example:

> 💬 One of the projects I work on is challenging because it seeks to develop primarily in one language (Rust) but provides comprehensive APIs in two more popular/portable languages (Python, Julia). ... The varying availability and compatibility of tools across the 3 languages poses a reasonably large challenge. There are a few lucky cases. For example, Jupyter notebooks can be written in all 3 languages, allowing examples to be presented in a similar fashion in each language, even interactively (Binder). Most cases are not so lucky. Rust (via Cargo) has dedicated tools and repositories for testing, packaging, documentation, but has no LaTeX docs support yet; Python (via Pytest, PyPI) has dedicated tools and repositories for testing and packaging, but not necessarily for documentation. Julia has some limited features for testing, documentation (build tools, but have to host yourself), and packaging. – **S2**

Some scientific software developers navigate these challenges by employing several programming languages or paradigms. In this author's case, Rust is intended to provide performance; the other two languages are integrated for portability to a wider audience. This intermingling of tools, however, causes challenges in maintenance and automated testing. Another author, rather than provide a specific project example, details the utility of randomized property-based testing for large, complex systems:

> 💬 Coming up with good test coverage for software systems is extremely difficult and time consuming because the state-space of software is astronomically large. Humans are also bad at coming up with tests that cover the wide range of data values that a program could process and instead tend to only test data that is easily human understandable. So how do we leverage computers to generate more test cases for us? One approach is Randomized Property-based Testing.
>
> In randomized property-based testing, the developer specifies mathematical invariants that a given piece of code/function/interface must satisfy and provides some building blocks to allow the testing library to randomly generate input data. The testing library then runs that code on randomly-generated test data to try to find an input that falsifies the invariants. The number of random tests run is user-configurable, but is likely in the hundreds or thousands each time testing is run. – **S4**

In an attempt to overcome problem complexity and humans' limited ability to find all potential corner cases, this author has turned to a relatively unknown procedure for generating test cases. This has proved generally successful for the author. For other authors, however, the complexity of attempting to automate testing for special use cases has been more futile:

> 💬 We developed one set of testing routines for Stitch-IO in Python that focused on ensuring that things functioned (not quite unit testing, but slightly more complex). We also had a test written in C that was supposed to represent how the application works in practice, but without the physics so it would be fast.
>
> In spite of both the Python and C tests all working correctly, the application was having data corruption issues. The tests should have revealed the source of the errors, but they did not. After some analysis, we determined that the C application representation was not moving through the computational space exactly as it would for a production run. The simplification should not have mattered, but it turns out that it did. – **S6**

From this story, we see that some attempts to simplify and accelerate testing (intended to help reduce costs and time to science) can sometimes have unintended side effects.

C. Interdisciplinary Collaboration Requirements

US national laboratories are well-known for their collaborative efforts. They participate in endeavors such as the Exascale Computing Project (ECP)[4] and the Institute for the Design of Advanced Energy Systems (IDAES)[5]. These collaborations do not come without their difficulties, however. Many of the authors detailed struggles collaborating with multi-institutional teams of individuals with diverse educational and technical backgrounds. As one example:

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

[5] https://idaes.org

> 💬 We chose these tools for two reasons. One, we already had a team member who was familiar with Docker and GitLab CI/CD and the tools that are supported at [SNL]. Two, we already had all of our examples in Jupyter notebooks, so we wanted to find a "testing" solution that would suit what we already built. – **S1**

Due to varying levels of expertise in interdisciplinary teams, some projects have to restrict their tool usage to those that are already most widely known within the organization. The authors and their team members, much like those represented in Carver et al.'s survey, have minimal time for professional development and training [12]. Other authors have adopted similar practices to minimize complexity for their teams:

> 💬 Creating a "one build script to rule them all" in Python removes cognitive load from scientific subject-matter expert developers. Making it easy for them to do the right thing helps everyone. Also, providing a means for the team to contribute back to the "one script" allows flexibility to explore outside the box while still controlling things as much as possible. – **S3**

This author restricts the configuration options to those represented in a single script. This minimizes overall complexity for the group and cuts down on the "doesn't work on my machine" issue frequently experienced in interdisciplinary, distributed teams.

Ultimately, however, many of the above issues can be traced to a lack of professionalization across these teams.

D. Lack of Professionalization

In this context, "professionalization" refers to the process of "improv[ing] the capability of members to enhance the quality of service which is provided." [30] In other words, we use the term to describe the level of formalized training and capability of members of a team. In CSE teams, it is rare that all members have formalized training in software development. [12] Many learn on the job and from other CSE team members, who were also self-taught. The authors have a similar experience. One author talks about what they've gleaned about testing in Machine Learning through the years:

> 💬 While unit testing initially seemed less effective for data science projects than for web applications, I've learned a few things to adapt my testing for the needs of data science. First, most data science projects involve plain old software. APIs, functions with clear returns, internal interfaces, and other critical portions of code can be tested.
>
> Perhaps the area of machine learning code that feels hard to unit test at first is what you usually are trying to do: training and inference. Here are some things I've learned that are pretty close to unit testing (in that they are fast and exercise the important parts of your ML system):
>
> - Do an end-to-end test on a single sample such that your model ingests, trains on, and performs inference on a single sample. This feels like the definition of an integration test, but if you have your sample on the same system where you are doing the tests, it typically is fast enough not to be annoying.
> - Use seeds to lock down the randomness for most tests. Remember to set the seed for all pieces that introduce randomness since it can come from several different libraries.
> - When you want to test that randomness is still within a certain range, use a delta and subtract from and check if it is within bounds.
>
> – **S7**

Though striving to apply software engineering practices, this author still had to learn on the job how to properly introduce testing at the correct level in their domain-specific software. These customized skills are often not taught in a traditional programs. One author explicitly points this out:

> 💬 Randomized property-based testing, despite all of its successes, is still not widely known in the software engineering world. I think that is mainly due to lack of education, and our schools need to do a better job of including it in their curricula. It is still viewed as an "Advanced Topic" despite being very accessible. I think that part of this view is that successfully employing this testing requires the developers to formulate invariants, etc., and this is another skill that is not taught very well in schools. – **S4**

This author points out that, even though the process to automate test generation is accessible, many developers lack knowledge about specialized testing strategies like formulating mathematical invariants for numerical codes—a skill rarely taught in academic software engineering curricula.

V. Analysis

The stories in the above Section paint a picture of the state of scientific software testing and automation practiced and implemented by RSEs at SNL. In this Section, we present the results of our analysis, organized by research question. A full summary of the stories is presented in Table I.

A. RQ1: Challenges in Testing Automation

When it comes to challenges in test automation, we compare our experiences to those described by Wiklund et al. [22] (Table II).

We see a universal alignment of experiences around limitations in externally sourced tools and the existence of a steep learning curve for testing automation skills. Both of these categories are represented in some form in every experience report from the authors. The steep learning curve may be supported by a result of the survey by Carver et al. speaking to the lack of formalized training in scientific software developers [12]. The survey shows that fewer than 50% of members of scientific software development teams have received any formal training

Story	Topic	Summary
S1	**Testing, Tools**	Using pytest, nbmake, and Gitlab pipelines to handle a specific dataset and environment.
S2	**Testing, Documentation, Multiple Languages**	Using Rust to provide a computational back-end while providing an interface for Python and Julia.
S3	**Automation and Pipeline Design**	Creating a pipeline layer and a machine orchestration layer to manage a separation between the two.
S4	**Randomized Property Testing**	Creating randomized tests that check an invariant property.
S5	**Multiple Repositories**	Distributing software components across two hosting services.
S6	**Pytest and C**	Writing a simplified test with no compute for speed.
S7	**Machine Learning, Unit Testing**	Testing machine learning code by using small unit tests and checking expected invariants; running the program to see if it will crash as a basic test.
S8	**Performance Testing**	Creating an automated pipeline to get performance benchmarks.

TABLE I
SUMMARIES OF STORIES GENERATED FROM RECRUITMENT CALL

Challenge	Discussed in Stories
(Behavioural) Process adherence	S3, S5
(Behavioural) Organizational change	–
(Behavioural) Too high expectations	S1, S3, S4, S6, S8
(Behavioural) Process deviations	S3, S5
(Business and Planning) Cost of ownership and operation	S1, S3, S4, S5, S8
(Business and Planning) Automation too expensive for small projects	S1
(Business and Planning) Lack of time, people, and funding	S1
(Skills) Diversity	S1, S2, S3, S5, S8
(Skills) Steep learning curve	S1, S2, S3, S4, S5, S6, S7, S8
(Test System) Inadequate development practices	S1, S3, S4, S5, S7
(Test System) Testware architecture	S1, S4, S3, S5, S8
(Test System) Untested test environment	S1, S3, S6
(Test System) Limitations in externally sourced tools	S1, S2, S3, S4, S5, S6, S7, S8
(Test System) Environment configuration	S3, S5
(System Under Test) SUT Speed	S3, S4, S6
(System Under Test) SUT testability	S1, S2, S3, S5, S6, S7
(System Under Test) Platform limitations	S1, S2, S3, S5, S6, S8

TABLE II
CHALLENGES IN TEST AUTOMATION AS IDENTIFIED BY WIKLUND ET AL. [22]

in software engineering practices; many learn on the job in between their other responsibilities. Two experience reports call this out explicitly (**S4**, **S7**).

Next in order of substance, there are significant limitations that result from the "system under the test," or, in other words, the infrastructure upon which the automated tests are built. In six of the eight stories, authors reported difficulties resulting from platform restrictions and limitations. Many of these stemmed from an overly complex combination of operating system, compiler, and software versions, all queued for builds and tests per each change (**S3**, **S5**). Some of these instead originated in institutional policy and sensitivity restrictions (**S1**, **S5**, **S8**).

In the next cluster of represented challenges, the authors' stories (**S1**, **S3**, **S4**, **S8**) all spoke of having to deal with too high expectations, the cost of ownership and operation, challenges with skill diversity, and the testware architecture. These four topics are likely correlated. For example, a team with greater skill diversity may also tend to have higher expectations on what is possible within their testing infrastructure. This is reflected in [22] where skill diversity and too high expectations are mentioned in [22]'s papers marked [IP1], [IP3], [IP4], [IP10], [IP16], and [IP26]. Likewise, a complex testware architecture may lead to higher cost of ownership and operation. Diversity is also linked to inadequate development practices across these stories, a correlation likely related to the varied educational background of such diverse, interdisciplinary teams, a sentiment reflected in numerous responses to the recent Department of Energy Request for Information on Software Stewardship [2].

Only one story speaks to the cost associated with automation (**S1**). This author, however, represents the smallest of the projects across all responses; as such, it is unsurprising that this was called out explicitly in their story.

B. RQ2: Strategies to Address Challenges

In our analysis of the stories, we identified two thematic arcs that run throughout: (1) adapting testing technologies to meet CSE project needs and (2) aligning human skills and relationships to accomplish test-related tasks effectively. By mapping out the challenges teams face on these fronts in **RQ1**, we can recommend evidence-based interventions to help address those challenges.

To guide our recommendations, we draw upon the findings of a recent multivocal literature review on strategies for improving test automation maturity by Wang et al. In Table III we propose a mapping between frequently mentioned challenges and best practice recommendations supported by the literature.

1) Toolchain Alteration: Wang et al. notes "Test tools" as a key area in test automation, with best practices of "Select the right tools" and "Properly use the tools." [23] These suggestions seem straightforward when considering a more homogeneous computing environment. In SNL, however, we have shown that the environment is layered and complex, resulting in extra challenges with the tools themselves. When the available tools do not work on their own, there seem to be two distinct strategies: combination and creation.

Combination in this context refers to using multiple tools to offset the inefficiencies in individual tools. For example, in **S8**, the author stitches together multiple tools (GitHub, GitLab, Jenkins) to an overall greater purpose. Wang et al. calls out "Resources" as a main strategy consideration – that is, consider the resources required and available. In certain situations, a single tool may not serve all particular requirements because of, e.g., security, policy, or external collaboration, to name a few. We see this in practice in multiple stories but also in gray and peer-reviewed literature. In a 2017 blog post, the author debunks the myth that "All automation tools are equal:" "Every test automation tool has its 'sweet spot'. Depending on the level of complexity, you may use a blend of test automation tools to achieve your short and long-term testing goals. Furthermore, the automation tool(s) you use will vary depending on the application that you are testing and the skills of those responsible for testing." [31]

Creation, on the other hand, is the strategy of making a new tool to address a gap. In **S3**, for example, the author made two API wrappers to enable easier usage of a tool due to limitations in wider knowledge. This particular strategy is not directly addressed by Wang et al.; however, in their analysis on criteria for selecting the right tool, constraints in environment and organization are contributing factors. In the case of the authors' experiences, creation can often be the response to those constraints.

Calling out several of the recommendations from Table III (in order of importance):

- *Design the system under test for automated testability.* Better test automation starts with the code itself. This recommendation speaks to inspecting the code and ensuring that it is testable in a systematic way.
- *Select the right test tools.* Much like selecting the right language for a programming task, the right test tools must also be chosen. The process to find the right set of tools may take time; however, the right tools are a significant step towards resolving automation challenges.
- *Adjust the test automation strategy to the changes.* The right tool now may not be the right tool forever. Teams are encouraged to continually evaluate their testing infrastructure for areas of improvement. If a team experiences chronic infrastructure issues, it may be time to readdress the strategy and tools in use.

2) Human Factors: Managing expectations about what testing automation can do and what it costs to achieve is key to preventing abandonment and short-term thinking. Wang et al. identifies several relevant strategies on this front. First, teams should formally define their test automation strategy, including the goals, test scope, risks, resources, costs and benefits, and effort needed to succeed. Securing buy-in from the development team is vitally important here; as noted by Fewster and Graham, "the best automation tool in the world will not help test efforts if your team resists using it." [32] Likewise, teams should directly involve stakeholders in developing this testing automation strategy; this includes managers, users, or anyone else who may be affected by the test automation. Finally, as the software project and the surrounding circumstances change, the test automation plan should remain flexible and evolve with them.

Another common challenge reported by our storytellers are the gaps in testing automation skills due to the diversity of backgrounds among CSE practitioners, the steep learning curve of testing automation tools, and inconsistent and inadequate practices around test automation among CSE teams. Wang et al. emphasizes the need for teams to share their experiences with one another to enable organizational learning and promote collaboration; this could include wikis, group discussion forums, and creating reference repositories. This should also include formal training programs offered by the organization to help ease the learning curve, and support from both the team and management for individuals to spend time in that training in order to develop those skills more fully. Within teams, there should be well-defined roles for test-related activities, and the responsibilities for standing up and maintaining test automation should be distributed evenly so that all team members, not just RSEs, have a working knowledge of the testing systems. Managing these human factors may help teams to address the challenges of test automation more effectively.

Again, we refer to several specific recommendations from Table III (in order of importance):

- *Have competent test professionals.* A test professional is expected to: "[i]dentify stakeholders and tools for testing the software, develop a test plan for testing the software, and collect and report data resulting from testing/demonstration, etc." [33] Without team members who have adequate skills in these activities, testing will likely suffer.
- *Involve key stakeholders in strategy development.* Key stakeholders are necessary for the success of testing. They will be able to ensure that the tests cover their needs and provide insight and support.
- *Keep test professionals motivated about test automation.* Testing cannot succeed without the team's motivation. If morale is falling, steps should be taken to discover and fix the root cause.

VI. Threats to Validity

We have identified four potential threats to validity in our study: (1) generalization; (2) qualitative nature of the data; (3)

Challenges	Recommendations
(Behavioural) Too high expectations	✓Involve key stakeholders in strategy development ✓Keep test professionals motivated about test automation ✓Define an effective test automation strategy ✓Adjust the test automation strategy to the changes
(Business and Planning) Cost of ownership and operation	✓Provide enough resources
(Skills) Diversity (Skills) Steep learning curve (Test System) Inadequate development practices	✓Share available test automation knowledge ✓Allow time for training and the learning curve ✓Have competent test professionals ✓Promote collaboration
(Test System) Testware architecture (Test System) Limitations in externally sourced tools	✓Select the right test tools ✓Arrange testware in good architecture
(System Under Test) SUT testability	✓Design the system under test for automated testability
(System Under Test) Platform limitations	✓Define test automation requirements ✓Have control over changes of test automation requirements ✓Arrange testware in good architecture

TABLE III
PROPOSED ALIGNMENT OF CHALLENGES ATTESTED BY HALF OR MORE OF THE STORYTELLERS TO HIGH-LEVEL CATEGORIES OF BEST PRACTICES IN TEST AUTOMATION IDENTIFIED IN A REVIEW BY WANG ET AL. [23].

lack of discussion or data on model evolution; and (4) personal biases.

First, the authors in this study are all members of the same institution and act as RSEs or RSE allies. This presents a threat in the wide generalization of the results and analysis. We cannot say, for example, that all practitioners across the institution will have the same experiences and views. We have mitigated this risk to the best of our ability, however, by comparing our experiences with those recorded in scholarly literature.

Second, the data in this study is entirely qualitative. Due to the lack of quantitative data, we cannot guarantee the trends and conclusions will hold true for all cases. Nevertheless, we deem this to be an acceptable trade-off in order to demonstrate a more general picture of our established test automation state of practice at Sandia National Laboratories. As with the previous threat, we have also attempted to mitigate this risk by grounding our experiences with those found in external scholarly literature, thus adding to the well of experiences and reports in similar and different institutions.

Third, the data in this study does not account for the research software phenomenon that tests may model the behavior that is expected, only for the field to discover that this model is incorrect. In practice, this may mean that even programs which pass a well-designed automated test suite may still deliver incorrect results. We cannot mitigate this threat to validity as it is not enumerated as a barrier in the framework we use to categorize barriers to automation, and it is not present in user stories either. However, this model instability may be a part of the complexity of the system under test, and described only implicitly as "a complexity" rather than explicitly called out.

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. As a primary attempt to mitigate this bias, two of the authors provided no experience reports and focused solely on the summary and analysis of those stories contributed by the other seven authors. They then reviewed these perspectives and analyses with the seven authors to ensure consistency of beliefs and experiences. In this way, we aimed to minimize possible bias while still preserving the expressed opinions.

VII. CONCLUSION

Automation for software development has become a common practice across industry and scientific software development. In this article, the nine authors, all members of the Research Software Engineering Community of Practice at Sandia National Laboratories, shared their experiences with testing automation in their scientific software projects. Using a Participatory Action Research approach, we examined these stories to identify common challenges and strategies employed and compared those against peer-reviewed literature.

We found that there are a number of challenges reflected in both the experience reports and literature. Some were universal, such as system under test complexity and steep learning curves, suggesting that effective mitigation strategies should address these in particular. This in itself presents an inherent obstacle since highly complex problems will most likely generate highly complex solutions. The problems broadly fall under both technical and human factor categories, with strategies mirroring these. These findings suggest that while existing literature may offer some successful strategies, further innovation is needed to continue enabling and accelerating automated testing at SNL.

REFERENCES

[1] U. D. of Energy, "The state of the DOE national laboratories (2020 edition)," U.S. Department of Energy, Tech. Rep., Jan 2021. [Online]. Available: https://www.energy.gov/downloads/state-doe-national-laboratories-2020-edition

[2] H. Finkel, B. Brown, R. Pino, S. Hier-Majumder, and B. Spotz, "Responses to the request for information on stewardship of software for scientific and high-performance computing," USDOE Office of Science (SC), Tech. Rep., 2021.

[3] B. W. Boehm, J. R. Brown, and M. Lipow, "Quantitative evaluation of software quality," in *Proceedings of the 2nd international conference on Software engineering*, 1976, pp. 592–605.

[4] M. De Bayser, L. G. Azevedo, and R. Cerqueira, "ResearchOps: The case for DevOps in scientific applications," in *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*. IEEE, 2015, pp. 1398–1404.

[5] B. W. Boehm, M. H. Penedo, E. D. Stuckle, R. D. Williams, and A. B. Pyster, "A software development environment for improving productivity," *Computer*, vol. 17, no. 06, pp. 30–44, 1984.

[6] B. W. Boehm, "Improving software productivity," *Computer*, vol. 20, no. 09, pp. 43–57, 1987.

[7] D. Georgakopoulos, M. Hornick, and A. Sheth, "An overview of workflow management: From process modeling to workflow automation infrastructure," *Distributed and parallel Databases*, vol. 3, pp. 119–153, 1995.

[8] E. Dou and C. Reascos, "The next step in verification testing of complex systems is automation," in *2015 IEEE AUTOTESTCON*. IEEE, 2015, pp. 194–198.

[9] A. Meier and J. C. Ivarsson, "Agile software development and service science: How to develop it-enabled services in an interdisciplinary environment," *GSTF Journal on Computing (JoC)*, vol. 3, pp. 1–5, 2013.

[10] M. Kropp and A. Meier, "Qualitative study of successful agile software development projects," *IMVS Fokus Report*, 2014.

[11] D. Heaton and J. C. Carver, "Claims about the use of software engineering practices in science: A systematic literature review," *Information and Software Technology*, vol. 67, pp. 207–219, 2015.

[12] J. C. Carver, N. Weber, S. Gesing, D. S. Katz, and K. Ram, "Urssi conceptualization survey questions," May 2019. [Online]. Available: https://doi.org/10.5281/zenodo.2713885

[13] K. Karhu, T. Repo, O. Taipale, and K. Smolander, "Empirical observations on software testing automation," in *2009 International Conference on Software Testing Verification and Validation*. IEEE, 2009, pp. 201–209.

[14] K. Fogel, *Producing open source software: How to run a successful free software project*. " O'Reilly Media, Inc.", 2005.

[15] R. Baxter, N. C. Hong, D. Gorissen, J. Hetherington, and I. Todorov, "The research software engineer," in *Digital Research Conference, Oxford*, 2012, pp. 1–3.

[16] C. Maimone, "Supporting research software and research software engineers," 2019.

[17] I. A. Cosden, "The Princeton University RSE group model: Operational and organizational approaches," *Computing in Science & Engineering*, 2023.

[18] R. Milewicz, J. Willenbring, and D. Vigil, "Research, Develop, Deploy: Building a full spectrum software engineering and research department," *arXiv preprint arXiv:2010.04660*, 2020.

[19] P. Reason and H. Bradbury, *Handbook of action research: Participative inquiry and practice*. sage, 2008.

[20] W. G. Lutters and C. B. Seaman, "Revealing actual documentation usage in software maintenance through war stories," *Information and Software Technology*, vol. 49, no. 6, pp. 576–587, 2007.

[21] R. Milewicz, J. Bisila, M. Mundt, S. Bernard, M. R. Buche, J. M. Gates, S. A. Grayson, E. Harvey, A. Jarger, K. T. Landin, M. Negus, and B. L. Nicholson, "DevOps pragmatic practices and potential perils in scientific software development," in *Proceedings of Eighth International Congress on Information and Communication Technology: ICICT 2023, London*. Springer, 2023.

[22] K. Wiklund, S. Eldh, D. Sundmark, and K. Lundqvist, "Impediments for software test automation: A systematic literature review," *Software Testing, Verification and Reliability*, vol. 27, no. 8, p. e1639, 2017.

[23] Y. Wang, M. V. Mäntylä, Z. Liu, J. Markkula, and P. Raulamo-jurvanen, "Improving test automation maturity: A multivocal literature review," *Software Testing, Verification and Reliability*, vol. 32, no. 3, p. e1804, 2022.

[24] M. Shahin, M. A. Babar, and L. Zhu, "Continuous integration, delivery and deployment: a systematic review on approaches, tools, challenges and practices," *IEEE access*, vol. 5, pp. 3909–3943, 2017.

[25] W. W. Royce, "Managing the development of large software systems: concepts and techniques," in *Proceedings of the 9th international conference on Software Engineering*, 1987, pp. 328–338.

[26] M. Gokarna and R. Singh, "Devops: a historical review and future works," in *2021 International Conference on Computing, Communication, and Intelligent Systems (ICCCIS)*. IEEE, 2021, pp. 366–371.

[27] E. Mueller, "What is DevOps?" Aug 2010. [Online]. Available: https://theagileadmin.com/what-is-devops/

[28] C. SEI, "Software product lines curriculum," 2020. [Online]. Available: https://resources.sei.cmu.edu/library/asset-view.cfm?assetID=650239

[29] M. Mundt, K. Beattie, J. Bisila, C. Ferenbaugh, W. Godoy, R. Gupta, J. Guyer, M. Kiran, A. Malviya-Thakur, R. Milewicz *et al.*, "For the public good: Connecting, retaining, and recognizing current and future rses at national organizations," *Computing in Science & Engineering*, 2023.

[30] E. Hoyle, "Teaching as a profession," in *International Encyclopedia of the Social & Behavioral Sciences*, N. J. Smelser and P. B. Baltes, Eds. Oxford: Pergamon, 2001, pp. 15 472–15 476. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B0080430767024505

[31] L. Le Francois, "Test automation: A reality check," 2017. [Online]. Available: https://qaconsultants.com/blog/test-automation-reality-check-1/

[32] M. Fewster and D. Graham, *Software test automation*. Addison-Wesley Reading, 1999.

[33] N. Assyne, H. Ghanbari, and M. Pulkkinen, "The state of research on software engineering competencies: A systematic mapping study," *Journal of Systems and Software*, vol. 185, p. 111183, 2022.