SAND2022-1691

SANDIA REPORT

Printed September 2021

Sandia National Laboratories

Incentivizing Adoption of Software Quality Practices

Elaine M. Raybourn[1,2], Reed Milewicz[2], Miranda Mundt[2]

1. Corresponding Author
2. These authors contributed equally to this work

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185
Livermore, California 94550

Issued by Sandia National Laboratories, operated for the United States Department of Energy by National Technology & Engineering Solutions of Sandia, LLC.

Available to DOE and DOE contractors from

U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@osti.gov
Online ordering: http://www.osti.gov/scitech

Available to the public from

U.S. Department of Commerce
National Technical Information Service
5301 Shawnee Road
Alexandria, VA 22312

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.gov
Online order: https://classic.ntis.gov/help/order-methods

Incentivizing Adoption of Software Quality Practices

Elaine M. Raybourn
Applied Cognitive Science
P.O. Box 5800 MS 0639
emraybo@sandia.gov

Reed Milewicz
Software Engineering and Research
P.O. Box 5800 MS 1318
rmilewi@sandia.gov

Miranda Mundt
Software Engineering and Research
P.O. Box 5800 MS 1320
mmundt@sandia.gov

ABSTRACT

Although many software teams across the laboratories comply with yearly software quality engineering (SQE) assessments, the practice of introducing quality into each phase of the software lifecycle, or the team processes, may vary substantially. Even with the support of a quality engineer, many teams struggle to adapt and right-size software engineering best practices in quality to fit their context, and these activities aren't framed in a way that motivates teams to take action. In short, software quality is often a "check the box for compliance" activity instead of a cultural practice that both values software quality and knows how to achieve it. In this report, we present the results of our 6600 VISTA Innovation Tournament project, "Incentivizing and Motivating High Confidence and Research Software Teams to Adopt the Practice of Quality." We present our findings and roadmap for future work based on 1) a rapid review of relevant literature, 2) lessons learned from an internal design thinking workshop, and 3) an external Collegeville 2021 workshop. These activities provided an opportunity for team ideation and community engagement/feedback. Based on our findings, we believe a coordinated effort (e.g. strategic communication campaign) aimed at diffusing the innovation of the practice of quality across Sandia National Laboratories could over time effect meaningful organizational change. As such, our roadmap addresses strategies for motivating and incentivizing individuals ranging from early career to seasoned software developers/scientists.

ACKNOWLEDGEMENTS

The authors would like to thank Gary Laughlin, Director; Lindsey Klennert, Sr. Manager; John Parmeter, VISTA Innovation Tournament Lead; Jeni Turgeon, and the entire VISTA Innovation Tournament Committee for the opportunity to engage in this short-term effort to explore incentivizing a practice of software quality at Sandia. In particular we appreciated our interactions with senior scientist Mike Heroux and software quality engineers Jeni Turgeon, Chris Lujan, Jeanette Johnston, and Bobby Butler. Thank you for your frequent feedback, we look forward to continuing our collaboration. The VISTA Innovation Tournament funding allowed our assembled team to build relationships and energy across Sandia that otherwise may not have occurred. Finally, Elaine would like to acknowledge the efforts of her co-authors Miranda and Reed, in particular for contributing and synthesizing an extensive bibliography of resources and literature that stands alone as a valuable artifact of this project. It is my sincere pleasure to work with you both – your passion for software quality is contagious.

This page intentionally left blank.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

This page intentionally left blank.

1. INTRODUCTION

High performance computing and computational simulation are foundational capabilities for every element of the nuclear deterrence (ND) portfolio at Sandia [10]. Through the Advanced Simulation and Computing (ASC) program, Sandia maintain a vertically-integrated R&D portfolio spanning hardware and foundational algorithms to high-consequence software applications. To sustain the ecosystem of software and to meet requirements for high-confidence decision-making, the ASC Program Office has recognized the need to invest in and promote software quality [99]. By quality, we mean the degree to which the software systems and components meet customer or user needs and expectations, both stated and implied [2, 12]. Software teams at Sandia operate on the frontiers of science and engineering and therefore should be afforded flexibility and freedom to innovate. However, at the same time, "the qualification engineering process (QER) is disjointed from the NW-related software development process, thereby adding significant time, effort, and cost with very few measurable improvements to product quality" [23].

Funded by the 6600 VISTA Innovation Tournament in May 2021, this 4-month project completed in September 2021 developed a roadmap for incentivizing and motivating software developers, engineers, and computational scientists across Sandia to adopt a practice and culture of quality. The present report and future roadmap opportunities outline initial steps toward the submission of a proposal to the FY22 ND Mission Area Integrating Initiatives (MAII) and R&D proposals to FY23 Lab Directed Research and Development (LDRD) calls in CIS (computing and information science) or GS (global security).

Ideation is the process of generating new ideas, concepts, or designs. Since our goal was to understand how and when to incentivize (intrinsic and extrinsic motivation) software developers and scientists to adopt better practices of software quality engineering (SQE), we explored different use cases by applying design thinking methodologies internally with our team at Sandia and role playing character or persona vignettes with external members of the scientific software community. The literature review of current approaches to software process improvement (SPI) informed our ideation and helped our team identify shortfalls, gaps, and obstacles to the adoption of best practices of software quality among software developers.

2. BACKGROUND

Techniques for SPI, which aim to assess, design, and realize effective development processes, have long represented an evolving area of research. The underlying assumption of SPI is that the quality of software depends upon the quality of its development. In other words, by improving the means of production — teams and their technologies and practices — we can improve the final product, the software [54]. This line of thinking has intellectual foundations in management science and organizational learning (*e.g.,* Plan-Do-Check-Act [30]), culminating in what Dybå refers to as the "quality revolution" in software development [34]. This is reflected in the literature: a systematic review by Unterkalmsteiner et al. finds that process and product quality are the most frequently measured outcomes in SPI studies [103]. In this way, SPI is intimately

connected with SQE. It is well-known that SPI efforts all too often end up being slow, having a limited impact, and/or failing to accomplish their intended objectives [76, 59]; some estimates suggest that up to 70% of SPI initiatives result in failure [74, 75]. For this reason, a major focus of research in the field has been on the factors that lead to success (or failure) of SPI. Case in point, Clark and Connor identify 44 different factors (*i.e.,* cultural, technical, business, etc.) that can influence software development processes [28]. Of particular importance to our work, an emerging theme has been the need for customization in SPI techniques.

The following excerpt is taken from a more in-depth treatment provided by this report [45] in which the authors highlight SPI customization in the context of scientific software development:

> While "classic" SPI frameworks such as CMM(I) or SPICE have existed for three decades [79, 27, 36], a comprehensive review by Kuhrmann et al. found that from 1989 to 2016 an average of 11 new or customized SPI methods were introduced per year [55]. One hypothesis put forward by the authors is that process improvement is a context-specific activity and that methods must be adapted to their context in order to be successful. This trend towards adaptation echoes recent studies that suggest most software companies today use individualized development processes that are hybrids of different methodologies and philosophies [53, 100]. Additionally, the goal of developers has been to accomplish their science and engineering objectives, not necessarily to write software [16]. In 2006, Baxter et al. observed that a lack of basic knowledge about development processes was a significant challenge for the scientific software community [18], but recent studies suggest the situation is now more nuanced. Case in point, a 2019 survey of scientific software developers by Eisty et al. found that respondents not only saw value in software process, but even "[preferred] using a defined software development process" over ad hoc approaches [35]. This may be a reaction to the increasing importance of software in science: a 2018 survey by Pinto et al. found that 8 out of 10 researchers reported spending "more time" or "much more time" developing software than they did 10 years ago [81].

Historically it is often the case that "software is valued only insofar as it progresses the science" [90]. As Dennehy and Conboy have observed, no software engineering method, practice, or tool can be studied in isolation; successful adoption requires alignment with the surrounding culture and context of the software development [31]. This implies that, for high-consequence scientific and engineering software development at Sandia, advancing the state of practices will require tailored approaches that reflect the needs, priorities, and perspectives of the community. Many developers have a background in a scientific or engineering domain and lack formal training in software development [42]. The projects they work on are funded by agencies who have traditionally focused on accelerating the creation or improvement of specific capabilities [97], a funding regime that is can be at odds with the need to maintain software quality [39]. Their projects can vary greatly in terms of their maturity and purpose, ranging from exploratory scripts to production stable codes [14]. Finally, scientific software development practices have historically been out of step with mainstream industry, with some scholars describing a "chasm" between scientific computing and conventional software engineering dating back many decades [51, 98].

Query Component	Search Strings	
Software Engineering		(“software engineering” OR “software development”)
Incentivization	AND	(“incentive” OR “drive” OR “motivation” OR “motivator” OR “human factor”)
Software Quality		(“software quality”)

Table 3-1 Search string used to collect relevant literature.

That said, the past two decades have seen significant shifts in how scientists and engineers work with software. This has coincided with the rise of the research software engineer (RSE) [17], as labs and universities have moved to create career pathways for software professionals, including at Sandia [69]. In short, as software has become critical to the work of scientists and engineers, the maturity of their software development practices and the quality of the resulting software have become increasingly important.

Our observation is that the software culture at Sandia tends to reflect broader trends in computational science and engineering. Taken together with the findings discussed above, the timing is right to advance SQE practices at Sandia. Software is poised to take on an even more central role in the ND environment; lab leadership recognizes that improving the quality of our mod/sim (M&S) capabilities is a *sine qua non* for Sandia's strategic objectives (see Objective 2.3). This requires working with software teams to achieve quality-enabling process improvements, and significant progress has been made on this front (*e.g.,* the ASC SQE effort). However, we posit that a source of great uncertainty and risk to long-term success is in software quality incentivization. As we have outlined above, SPI is a context-specific activity that is highly influenced by culture, and without sufficient alignment with that context and culture, the effort is more likely to fail than not. For that reason, our report provides recommendations on how to structure incentives for SPI, based on best evidence from the literature and ideation exercises we conducted, that could motivate teams to value software quality and realize it in practice.

3. LITERATURE REVIEW

To collect a representative sample of the literature on software quality incentivization, we conducted a rapid review [67]. A rapid review protocol is time-boxed literature review designed to deliver evidence in a timely and accessible way. Rapid reviews simplify or omit certain steps from full systematic reviews, enabling turnaround times measured in days rather than months; specifically, rapid review protocols limit the literature search, reduce or eliminate the screening and quality appraisal steps, and present highlights from the literature without formal synthesis. Rapid reviews provide actionable results quickly with an ultimate question-and-answer format in mind [24]. After deliberating among ourselves, we arrived at a set of search terms listed in Table 3-1. On June 27th, 2021, we used Harzing's Publish or Perish (version 7.22) to anonymously fetch the top 1000 results from Google Scholar.

After extracting the search results, we followed the flow diagram in Figure 3-1 to rank and categorize each paper. We scanned the abstract of each article for the general content and ranked the relevancy and potential usage of the abstracts on a scale of 1-5 (1 being not relevant, 5 being perfectly relevant). We also added a short summary of the abstract and our justification for our

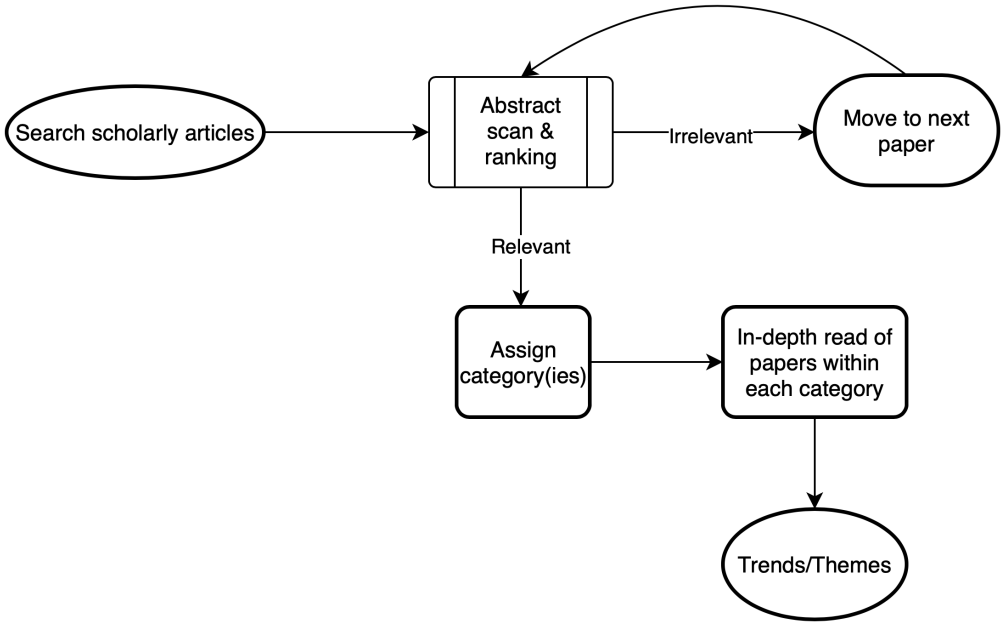

Figure 3-1 A diagram illustrating the review process for the literature presented in this study.

ranking of the content; access to this annotated bibliography is available upon request. We then refined the initial list of articles into five categories and explored within each category to find general themes or trends. The five categories that emerged from our review are: "Models and Strategies for Quality Assurance", "Economics", "Humans Factors", and "Education and Training." We present our analysis of themes below.

3.1. Findings

3.1.1. Models and Strategies for Quality Assurance

There are decades of scholarship on SPI and many established methodologies in this space (*e.g.,* CMMI, SPICE, ISO9000). A full treatment of that literature is beyond the scope of this paper; here we focus our attention on the connection between SPI methods and software quality. A 2016 systematic review of software quality management in the context of SPI by Jacobsen et al. suggests that researchers and practitioners have converged in recent years on specific practices and processes that are aligned with software quality, such as software testing and complementary activities including reviews and documentation techniques [47]. This also dovetails with our understanding the coordination challenges involved in developing complex, large-scale software (such as at Sandia). Testing helps manage awareness of how changes to code influence behaviors, code reviews promote the exchange of information and can support other quality practices, and documentation externalizes knowledge for future use. That is, while SPI methods are broadly concerned with the practices across the entire software lifecycle, there may be key practices which are aligned with software quality objectives and therefore worth focusing on.

The literature on scientific and high-consequence software is relatively limited, but it does provide us with a starting point. A 2016 study by Yost et al. investigated why critical software systems developed for the government continue to be of lower quality than expected [107]. Their survey of software development practices at FFRDCs suggested that a major driver of quality shortfalls are due to issues of coordination and knowledge management at scale. Specifically, participants reported significant challenges in finding code related to a bug or behavior to be changed, understanding code that they or someone else wrote in the past, and understanding the design rationale behind a piece of code. Projects where these barriers were considered major problems tended to rank lower in overall software quality, including maintainability, reliability, functional suitability, and evolvability. Of note, these findings were echoed in 2018 case study at Sandia by Milewicz and Raybourn using a similar survey instrument [68]; the authors noted that, for scientific software at Sandia and elsewhere, "achieving development at scale means confronting recurrent problems of awareness and understanding which pose risks to sustainability."

Top-down, policy-based strategies for quality assurance, whether based on positive or negative reinforcement, tend to produce mixed results and are notoriously difficult to design well. On the negative end of the spectrum, a multi-method field study by Wang et al. found penalizing developers for poor software quality, while moderately effective at reducing software defects, is generally harmful to morale - which is itself a driver of poor software quality [105]. For this reason, much of the literature on policy-based strategies is geared toward positive reinforcement. Case in point, Moldon et al. examined the impact of gamification strategies on the behavior of developers on GitHub – in particular streak counters, which track how many consecutive days developers contribute code [70]. While streak counters increased the volume and frequency of code contributions, they also effectively incentivized burnout and, as the authors note, "streak-chasing behavior likely had unhealthy externalities on the quality of developer outputs." This is a cautionary tale which illustrates how reinforcement strategies, whose intentions are meant to be positive, do influence developer behavior, but can end up being counter-productive. In the context of scientific software specifically, Heroux comments on this inherent tension in the design of incentives, observing that, "Development of employee incentives for quality software can be challenging. Any reward based on direct software metrics can lead to skewed behavior that may actually be counter-productive" [43]. This is not to say that such strategies are infeasible: we found a number of promising studies which indicated that, when carefully designed, gamification can lead to positive outcomes [91, 66, 93]; of note, Monteiro et al. provides an excellent review of this topic [71]. However, given that quality in scientific software is reflective of complex changes with many different drivers, it will be difficult to address it solely through policy instruments like gamification and penalization. Of note, Steffens et al. has made some progress in gamification targeting communication, coordination, and cooperation [96]; future work in this space may open up new avenues for us to consider.

Bottom-up strategies, meanwhile, emphasize ways teams can restructure their practices and processes in order to affect meaningful change. For example, Sarker et al. found that in a geographically distributed information system development team, effective knowledge transfer among team members was a function of one's communication, credibility, and culture, but not one's capability as the researchers suspected [89]. Credibility, based on trust and reputation, was measured in conjunction with one's performance and volume of conversation. The researchers found that, "[a]s expected, credibility emerged as a critical determinant that not only increased

knowledge transfer directly but also appeared to be an even more potent variable in explaining knowledge transfer in combination with a highly communicative source" (p. 213). While certainly not the only variable at play in influencing others, the evidence suggests that one's reputation (credibility) and performance can motivate effective knowledge transfer. This bottom-up strategy merits further investigation as future work. Therefore, while for this VISTA project we have conducted a targeted, rapid review of relevant literature in software assurance models, our analysis of this literature identified two R&D Roadmap bottom-up opportunities for future work:

1. Identify SPI methods or best practices currently in use at Sandia, and explore them as entry points for appropriate incentives deployed in the context of the diverse cultures at play among code teams that are responsible for yearly ASC SQE assessments [1]. This follow-on exploration would complement our understanding about the global practices expressed in the Sandia Software Quality Assurance Program documentation by further articulating "how" Sandia teams achieve these global practices and could also illuminate opportunities to incentivize team and individual behavior change.

2. Identify key Sandia opinion leaders and champions of software quality practices.

3.1.2. Software Quality Economics

When it comes to software quality, it is difficult to measure the return on investment (ROI) of quality practices. Our literature review was focused on finding both empirical and methodological approaches to measuring ROI of quality. The oldest paper we included in our annotated bibliography is by Abdel-Hamid from 1988. In this paper, the author presents an in-depth case study of the trade-off between cost and benefit of quality assurance in software engineering [11]. The author focuses on defect rates throughout the software development lifecycle as a measure of quality assurance, a process which is repeated in later studies [77, 13]. Abdel-Hamid finds that defects which escape detection at an earlier phase of the development lifecycle are significantly more costly to correct once in later stages, if those defects are found at all, and proposes methods to determine correlations between quality assurance expenditure and cost in order to find optimal expenditure to maximize ROI. This original work has been expanded upon in the following decades. More recent works point to mathematical methods for directly calculating ROI, given value assumptions [77] and finding optimal software quality assurance practices based on defect rate [13].

With respect to ROI, it has been anecdotally observed at Sandia that ROI is not much of an incentive for developers and is only slightly incentivizing to management. More effective drivers appear to be behavioral, which is consistent with the literature. One area of ROI that merits further exploration is the "ROI" associated with awareness of clean code practices, especially if a newly incorporated practice results from a painful experience, such as the need to redo work. We have found that as teams and software systems scale up, it becomes much harder to keep track of the overall state of the software and its quality. Although modern tech companies rely heavily on a variety of analytics tools that track various project health indicators in real-time (*e.g.,*

sourcery.ai "runs in the background and suggests realtime improvements"), it can be challenging to translate that data into actionable insights for software quality decision-making [61].

Therefore, our analysis of literature in software quality ROI identified one R&D Roadmap opportunity for future work:

1. With a team's participation and consent, we can apply data mining techniques to a code base to identify gaps and opportunities for incremental software process improvements that can be achieved to improve code quality. In the spirit of enhancing a teams' awareness of code quality, Reposcanner is a tool in the productivity & sustainability improvement planning (PSIP) toolkit developed by Reed Milewicz at Sandia that can be used for analyzing code with the purpose of helping teams improve their software practices [83].

3.1.3. Human Factors

Motivation is considered the single greatest determining factor in productivity and software quality [22, 63]. That is, developers invest in software quality (be it performance, security, usability, etc.) only insofar as they value it. Given that teams have finite time and energy and different software qualities compete for attention (or are even at odds with each other, *e.g.,* portability vs. performance), software quality management always involves strategic decision-making and trade-offs. In that sense, understanding what motivates scientific developers to make the choices they do is critical to changing the state of practice. However, as McConnell points out, "motivation is a soft factor: It is difficult to quantify, and it often takes a back seat to other factors that might be less important but are easier to measure" [63]. A 2008 systematic review by Beecham et al. on motivation in software engineering finds that there are a variety of intrinsic (*e.g.,* trust, empowerment) and extrinsic factors (*e.g.,* recognition for work, adequate resources) at play. For instance, a more recent study by Verner et al. conducted a multiple case study with software development firms across different different countries. Having the right tools, techniques, and awareness enable better decision-making around software quality, that is, having actionable data about quality of code empowers developers to create higher quality software. Unfortunately, as Guzman et al. point out, teams often lack awareness regarding the state of their software [41].

Therefore, we believe it is important to promote self-discovery tools to assist in team-directed improvement in software quality practices. Several Sandia staff members have contributed to the creation of the Productivity and Sustainability Improvement Planning (PSIP) process [6]. This process allows teams to evaluate their current processes, discover where there is room for improvement, and assists in making small, targeted changes to implement improvement plans. Another self-discovery tool that could be of use to Sandia developers is the Core Infrastructure Initiative (CII) Best Practices badge program [3]. This website clearly defines criteria for best practices and allows projects to self-report their current status and receive a badge that they can add to their project's repository to show their current status.

Our analysis of literature in human factors identified five R&D Roadmap opportunity for future work:

1. Conduct a series of "listening sessions" and informal dialog with teams/individuals who represent ASC SQE use cases to identify levels of intrinsic and extrinsic motivation.
2. Consider the role of technology-mediated or automated incentives such as "likes" and "re-tweets" in software development. For example, consider the deployment of an automated process for motivating and incentivizing by using "tokens" (which can be used to "buy" tangible items or software development/documentation support, etc. at some point in the SQE assessment review process) [93].
3. Employ interactive tools and methods in advance of ACE-SQE year-end assessments such as the PSIP toolkit comprised of rateyourproject.org [84] and Reposcanner [83] to assist software developers with decision making. These tools are lightweight and promote the practice of incremental improvements, team dialog, and communication across end users and software projects.
4. Consider future development of tools that make the software quality process easier to integrate into project workflows.
5. Wargame the possibility of incorporating more heavyweight tools that offer teams SQE dashboard capability. For example, Mayr et al. describe a passive framework called ProCon (Process execution and quality Constraint) [62]. This tool formalizes definitions, standards, and guidelines provided by an organization into a tool with an engine to automate responses ("events") and a dashboard for ease of use. It visualizes information to assist both with quality assurance practices and process progress analytics. It has the capability to connect with commercial-off-the-shelf (COTS) tools like Jira to provide easy interfaces and data gathering from existing project management processes. While the tool is incomplete and more research is needed, the capabilities described in the paper are promising for passive/easier tracking approaches to quality assurance practices.

3.1.4. Education and Training

A common theme across our selection of education and training literature was the necessity for hands-on learning resources. Several papers provided information about hands-on activities for teaching software quality best practices [85, 40, 66, 72]. These articles also point out that a key factor in incentivizing learning software quality best practices is the role students play in their own learning - that is, the student's ability to manage and self-direct their learning, as well as engage in student-driven, active learning elements. For example, there was significant empirical success when students were tasked with a real-world problem that required software quality to resolve; additionally, there was success when the activities were subject to regular peer review by other students.

Mi et. al. also pointed at gamification techniques to promote learning [66]. In this article, the authors demonstrated that a point system and leaderboard were effective gamification techniques when paired with learning. A final common theme was allocated time for learning. In his 1994 paper, Slavin speaks of the necessary quintet for instructional effectiveness: "quality, appropriateness, incentive, and time" [94]. He suggests that without adequate time allocation, the

other three factors become useless. More research is needed as gamification may have both positive and negative influences. One study, for example, speaks to the benefits of points and leader boards [66]. Other studies spin cautionary tales about adverse effects to both developer choices and software quality practices [70]. The CII Best Practices badge program utilizes badges and progress bars to incentivize software quality engineering practices [3].

Our analysis of literature in education and training identified one R&D Roadmap opportunity for future work:

1. Identify opportunities to raise software quality awareness among Sandia leadership (beginning with a pilot with a friendly audience), program managers, and software developers. Consider hands-on training that simulates best practices as well as challenges in software development. Interactive exercises such as social simulations, tabletop exercises, and games can enable learning the principles of quality practices. For example, Morales et al. describe experiential learning through playing with LEGOs™ [72]. This exercise has been conducted through several iterations of fourth-year software engineering classes and can be a fun way to reimagine software quality as the authors note "... if the playful aspect is not controlled adequately, the activity can turn into a chaos, given the euphoria of participants and the relaxed environment in which it is carried out."

4. IDEATION EXERCISES

While we were engaged in the review of the literature, we conducted a three-hour design thinking Sandia-internal workshop with the ultimate value (*i.e.*, "result"') of promoting and incentivizing an enduring practice of the valued principle, or software quality. The concept of design thinking was originally rooted in architecture and urban planning [87]. Since the creation of the methodology, it has expanded into business, information technology, medicine, and more. The ultimate goal of the design thinking methodology is to change one's approach to a problem or a situation by envisioning the ultimate value of a "what" and a "principle" [32]. That is to say, rather than the standard scientific approach to logic where the focus rests on a "what" and "how" combined to deliver a "result" (or a factual finding), design thinking is approached in reverse order; that is, the "result" is value added, but the "what" and "how" are two unknowns in the equation. Meeting our goal involved brainstorming the "what" and "how" to make that value possible. Following the internal design thinking workshop, we followed up with an interactive breakout session (teatime) with computational scientists using persona vignettes and role play at a small conference. The interactive session is another technique used in design thinking. Both activities allowed our team to more deeply understand how to motivate and incentivize certain persona archetypes intended to characterize software developers and scientists at Sandia. Details and associated and artifacts from these activities are provided as appendices to this report.

4.1. Design Thinking Workshop (Internal)

We conducted the Design Thinking workshop on June 2, 2021. Elaine Raybourn acted as organizer and facilitator; Miranda Mundt, Reed Milewicz, and student interns Jacob Moxley and Han Yong Wunrow acted as participants. All participants were internal to Sandia. The workshop took place during a 3-hour block in which the participants were directed through several exercises. The majority of the workshop focused on two activities: an empathy map and generating and analyzing an "As-Is Scenario." From these exercises, we learned that with respect to software quality, our archetypal research software scientist was primarily motivated by time commitment, funding availability, and ability to do conduct career-advancing work (*e.g.*, papers, patents).

The biggest takeaways from the design thinking workshop fell into three areas of support: social, consultation, and leadership. The need for training was also identified. For example, through addressing the persona of "Albert" by putting ourselves in his shoes (*e.g.*, perspective-taking), we recognized that there are many opportunities to address nervousness and uncertainty. "Albert" will most likely feel intimidated by software quality best practices and would need some amount of support to help alleviate this. Much of his uncertainty would likely be diminished by proper access to software quality resources, such as how-tos, formal training, consultation groups, or even something less structured such as an easy-to-use website with quick videos or a list of subject matter experts. Additionally, "Albert" may need to feel that he is not divided between the world of "performance reports" and "software quality best practices." With leadership thoroughly on-board, he could feel less pressure to choose between activities and could feel empowered to begin practicing software quality activities. Further discussion is available in the Appendix (Section 8.1).

4.2. Collegeville Workshop Teatime Theme (External)

Later in our project, we conducted a targeted, small-group discussion, otherwise known as "teatime" at the Collegeville Workshop on scientific software teams [4]. Our abstract for this teatime was: "Many teams struggle to adapt and right-size software engineering best practices in quality to fit their context. Introducing software quality isn't usually framed in a way that motivates teams to take action, thus resulting in it becoming a 'check the box for compliance' activity instead of a cultural practice which values software quality and the effort to achieve it. When and how can we provide effective incentives for software teams to adopt and integrate meaningful and enduring software quality practices? To kick off the conversation the presenters will use breakout rooms and provide short vignettes. This teatime session is open to anyone who wants to brainstorm, share ideas, and learn from others!"'

As the abstract describes, we created three personas ("vignettes") to match three different archetypes contributing to software engineering projects: high-consequence software, research-oriented software, and something in the middle (*e.g.*, research software or prototype code that needs to adapt to production code per customer last minute request). Using these personas, we encouraged role-playing from the participants to promote empathy for the different requirements of each role and therefore address the motivation of each persona. The participants for the Collegeville Teatime came from a mix of industry, academia, and national labs. We

presented the participants of the teatime discussion with three personas in the format of "speed dating cards." These cards had quick synopses of the persona, their role, their type of project, and their current approach or objections to software quality activities. Following the exercise, we conducted a large-group debrief on what we learned throughout the role-playing activity. A clear theme across all three personas was the power of word of mouth - that is, given a vociferous advocate, more people would be likely to adopt processes and practices. Another general consensus of the group was that at the level of leadership, a stronger "hammer" approach is needed - that is, clear direction, expectations, and requirements from leadership, in addition to clear consequences should those requirements not be met. For the practitioners, though, they reported needing access to appropriate funding as well as easy to obtain resources and support that lower the barrier to entry to software quality. Further discussion of findings and artifacts is available in the Appendix (Section 8.2).

5. RECOMMENDATIONS

As discussed earlier, our ultimate goal is to influence change and promote a culture of quality within organizations and enterprises. Currently, software quality assurance and software quality engineering are perceived as one-and-the-same: a check-box activity in which practitioners frequently do minimal work in order to appease corporate or center standards. These two activities, however, are separate. Software quality assurance involves confirming that practices are being adequately followed as defined. Software quality engineering involves the development and implementation of those practices; that is, it is the way that work is defined and completed in order to efficiently develop and deliver a reliable software package. Regarding software quality engineering as the same as software quality assurance results in poor software quality practices which, as literature shows, contributes to a vicious cycle: "poor software quality begets poor software quality" [88].

To truly change the culture of quality, the first shift must come at a systemic level. For example, Sandia policy IT008 outlines the expectations for software quality assurance practices at a corporate level; that is, it identifies activities to ensure appropriate engineering efforts [7]. The policy identifies categories of practices such as validation, risk management, and configuration management, to name a few. Perhaps specific recommendations as to how these activities could be implemented would benefit software developers who are unfamiliar with the practice of quality. Requirements for further quality assurance methodologies are defined in Sandia policy QA001 to ensure proper quality practices [8]. These policies, along with interviews conducted with staff and managers demonstrate that at several levels, Sandia supports and recognizes the importance of software quality engineering. This is a step in the right direction. Literature supports the necessity for a high-ranking champion to begin the push towards the implementation of software quality engineering best practices [50, 102].

That being said, more weight is given to a vision when supported and practiced by someone who is personally known and trusted [86]. Through our design thinking workshop activity, Collegeville teatime discussion, and interviews with personnel from two distinct Centers of the authors' organization, this necessity for a well-known, approachable champion was empirically

supported. The biggest takeaway from the Collegeville teatime discussion, in fact, was that the highest likelihood for change comes from external sources and a main, influential advocate. Participants expressed emphatically that word-of-mouth recommendations for practices and processes easily holds the biggest sway for any substantive change. At Sandia, we recommend this take the form of the identification of an influential voice within each Center or team to act as a local advocate for software quality best practices. This advocate would be able to better understand and appreciate the work styles and culture of that Center or team and recommend appropriate quality practices, much like the advocacy championed by the Center for Computing Research Director with the introduction of the Center Software Quality Standardization document. Across the laboratories there may be many other examples we are unaware of. By championing the creation and implementation of this "right-sized" approach to software quality practices, a spotlight was placed on quality engineering within the Center for Computing Research. More research is needed to measure its long-term impact and efficacy.

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, a challenge supported in the literature [49, 107], 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." When it comes to requesting funding, proposals must be competitive compared to other applicants. Applicants must try to maximize their requested funds, and without a proper emphasis on allocated funding for software quality engineering activities, this will frequently be the first item cut from a proposal. From a managerial and funder standpoint, software quality activities only make economic sense. As noted in the literature review, defects that slip through the software development lifecycle are significantly more costly to correct, if they are ever found [11]. The CIS LDRD proposal process has already taken a step in this direction with a new reproducibility requirement for LDRD proposals as of late 2020 [44]. To this point, we recommend three paths:

1. A new allocation category within funding proposals specifically geared towards software quality activities. This category is the most direct form of supplying monetary resources. Funding sources can directly provide some percentage of funds for quality engineering activities for software projects and remove the perceived stress of being forced to choose between research and software quality.

2. Center-supplied overhead funds that any software project may use towards software quality practices. This category would take the impetus away from funding sources and provide indirect funding that can be utilized for any and all software quality activities at a systemic, high-level. These funds would be allocated at the beginning of the year and free to use.

3. Center-supplied resources. This category points towards the scenario where funds exist but the researcher is unsure of the work to be done. Funding here ideally could be applied towards quality engineers but could also be dedicated to training and consultations.

In parallel, we can also address the general lack of formalized software education training across software developers within the national labs. This phenomena is not unique to national labs, of course. Even given a traditional software engineering education, universities often lag behind the

needs of industry [48] and rarely provide the specific knowledge and skills needed for scientific computing, let alone software quality engineering best practices. Through our literature review, we explored several hands-on approaches to training software quality activities and learned protocols for proper education techniques. Due to the generally split-time allocation of software developers at Sandia, however, it would be difficult to provide training that takes large chunks of time away from already overbooked schedules. On this note, we make three recommendations:

1. Apply the LEGO™software quality training method introduced by Morales-Trujillo [72] in a small pilot. After the pilot, an adjusted version (based on pilot feedback) would be rolled up to higher-level tiers in management, thus creating thorough software quality understanding and stronger, well-advised advocates and champions.

2. Create a resource website or database that allows for asynchronous access to software quality resources in one easy-to-access location. Efforts to do just this have been attempted in the past [9]. In this effort, we look into previous iterations for insight on what succeeded and failed and work closely with existing personnel to conduct usability studies and garner feedback through all phases of the website development.

3. Introduce a software quality mentoring network comprised of software quality practitioners and advocates. Sandia has recently revived a mentoring network effort. We ideally will be able to incorporate ourselves into this effort and carve out a use-case specifically geared towards quality engineering practices.

Finally, much more understanding could come from gathering data on the motivating factors within the Sandia culture. While our VISTA project has yielded the beginnings of a strategic plan, of sorts, our short-term effort only scratches the surface of what could be accomplished through more research including experimentation, exploration through social simulation, and mixed-method data analysis. Our team is poised to follow on this VISTA effort with research and development as requested.

To this end, we intend to explore software developers' motivations. The core activity of this research initiative could involve surveys of a sample of the population, focus group "listening sessions," piloting the efficacy of experiential learning, etc. each taking into account different job titles and expectations, to inform inquiry. Some of our potential research questions are:

- **RQ1**: What motivating factors for quality are most prevalent in the Sandia workforce?

- **RQ2**: Do different job titles and descriptions contribute to variations in software quality engineering practices?

- **RQ3**: Does the degree of intrinsic motivation account for a developer's software quality engineering practices?

- **RQ4**: What is the role of one's team, management, and Sandia leadership in the diffusion of the practice of quality as an innovation?

Upon better understanding what motivates our developers, we can explore some "gamification" (strategic engagement to heighten motivation) or transmedia learning [82] techniques to maximize our incentivization. There have been few studies conducted within a national laboratory setting to investigate the effect of gamification practices. To this end, we intend to seek funding to investigate what gamification techniques would be most appropriate in this setting. This study would incorporate human research subjects in order to gauge effectiveness of gamification elements to software quality engineering practices.

6. LIMITATIONS, COMPLICATIONS, AND CONFOUNDS TO INCENTIVES

In the case of software quality, several cautionary themes arose during the literature review. First and foremost is the potential negative effect of gamification in software engineering practices. We've seen in several previously cited papers that gamification can be used to great benefit [66], but in the case of Moldon et. al., the authors argue that certain gamification factors (such as streak counters) can lead to adverse effects [70]. In this natural case study, the gamification element actually promoted both poor software quality practices and work-life balance.

Additionally, incentives can only go so far given the existing barriers in software development. Particularly in national laboratory settings, software developers are split across multiple projects, frequently with differing goals. This results in frequent context switches, causing developers to have to "spin up" several times in a given day [49, 107]. These frequent shifts in focus understandably cause lapses in quality practices or processes. Additionally, most software projects are plagued by missing or insufficient documentation [107]. This lack of documentation reduces the ability of developers to maintain or increase quality as it lowers understanding of usage or design decisions for legacy code.

Squazzoni and others present an experimental study to gauge quality of peer reviews in scientific publications in incentivized or not-incentivized scenarios [95]. The ultimate conclusion in this study is that, in this particular case, no incentive is simply the best incentive. In the realm of scientific work, tangible incentives reduced the commitment of reviewers as they felt that it took away from the idea of "good science leads to good reputation." In this scenario, the only incentive required was the perceived notion that good quality reviews would lead to a better, more reputable journal, free from any potential scandal. This notion of "good quality for good reputation" cannot be ignored in the scenario of national labs.

7. CONCLUSION

It is also broadly understood that the state of practice in computational science has yet to reach a level of maturity commensurate with traditional, physical experimentation. However, by incentivizing software developers and scientists to prioritize software quality and to adopt practices and processes that promote that quality, we believe teams will be empowered to deliver results with reduced uncertainty and more reproducibility. Meeting this challenge demands a shift

both in our ways of working with software and how the culture of the labs understands and values software quality. Software at Sandia is the product of world-class scientists and engineers, and annual appraisals by the ASE SQE team suggest that the labs have made significant progress to date in adopting software quality practices [101]. In collaboration with the ASC SQE team, we have identified areas of improvement, and we maintain that raising the bar will require addressing cultural issues that are known and unknown at this time. Finally, we have presented findings and recommendations in the form of a R&D roadmap and report of interventions through which Sandia can augment the process of quality engineering with accessible and more persuasive scalable tools/methods/resources that motivate and reward developers.

REFERENCES

[1] Asc sqe code teams. https://sharepoint.sandia.gov/sites/ASCSQE/Appraisals/SitePages/Code\%20Teams.aspx. Accessed: 2021-09-20.

[2] Asc sqe glossary. https://sharepoint.sandia.gov/sites/ASCSQE/SitePages/Glossary.aspx. Accessed: 2021-08-11.

[3] Cii best practices badge program. https://bestpractices.coreinfrastructure.org/en. Accessed: August 24, 2021.

[4] Collegeville 2021 workshop. https://collegeville.github.io/CW21/. Accessed: 2021-08-11.

[5] Policy and requirements management policy. https://lps.web.sandia.gov/policy/PRM001. Accessed: 2021-08-11.

[6] Productivity and sustainability improvement planning (psip). bssw.io/psip. Accessed: August 24, 2021.

[7] Provide quality software policy. https://lps.web.sandia.gov/policy/IT008. Accessed: 2021-08-11.

[8] Quality assurance policy. https://lps.web.sandia.gov/policy/QA001. Accessed: 2021-08-11.

[9] Software quality implementation group. https://sharepoint.sandia.gov/sites/squig/SitePages/Home.aspx. Accessed: September 10, 2021.

[10] Creating the future: Sandia national laboratories' strategic direction. Technical report, Sandia National Laboratories (SAND2018-9740R), September 2018.

[11] Tarek K Abdel-Hamid. The economics of software quality assurance: A simulation-based case study. *MIS Quarterly*, pages 395–411, 1988.

[12] Alain Abran, James W Moore, Pierre Bourque, Robert Dupuis, and L Tripp. Software engineering body of knowledge. *IEEE Computer Society, Angela Burgess*, 2004.

[13] Omar Alshathry and Helge Janicke. Optimizing software quality assurance. In *2010 IEEE 34th Annual Computer Software and Applications Conference Workshops*, pages 87–92. IEEE, 2010.

[14] Roscoe A Bartlett. A roadmap for sustainable ecosystems of cse software. *Proceedings of the Computational Science and Engineering Software Sustainability and Productivity Challenges (CSESSP) Workshop*, 2015.

[15] Victor R Basili and Gianluigi Caldiera. Improve soft-ware quality by reusing knowledge and experience. *Sloan management review*, 37:55–55, 1995.

[16] Victor R Basili, Daniela Cruzes, Jeffrey C Carver, Lorin M Hochstein, Jeffrey K Hollingsworth, Marvin V Zelkowitz, and Forrest Shull. Understanding the high-performance-computing community. *IEEE Software*, 2006.

[17] Rob Baxter, N Chue Hong, Dirk Gorissen, James Hetherington, and Ilian Todorov. The research software engineer. In *Digital Research Conference, Oxford*, pages 1–3, 2012.

[18] Susan M Baxter, Steven W Day, Jacquelyn S Fetrow, and Stephanie J Reisinger. Scientific software development is not an oxymoron. *PLoS Computational Biology*, 2(9):e87, 2006.

[19] Sarah Beecham, Nathan Baddoo, Tracy Hall, Hugh Robinson, and Helen Sharp. Motivation in software engineering: A systematic literature review. *Information and software technology*, 50(9-10):860–878, 2008.

[20] Sarah Beecham, Tracy Hall, and Austen Rainer. Software process improvement problems in twelve software companies: An empirical analysis. *Empirical software engineering*, 8(1):7–42, 2003.

[21] Janel Bloch and Sandra E Spataro. Lessons from scranton: Using scenes from the television series the office to teach topics in professional communication. *IEEE Transactions on Professional Communication*, 59(3):274–287, 2016.

[22] BW Boehm. Software engineering economics (prentice-hall, inc., englewood cliffs, new jersey, usa). 1981.

[23] Bobby Butler, Jeanette Johnston, Nick Blazier, Ron Dulaney, John Englemann, Jennifer Turgeon, and Charlene Coriz. Software qualification engineering release (qer) improvement project, October 2019.

[24] Bruno Cartaxo, Gustavo Pinto, and Sergio Soares. The role of rapid reviews in supporting decision-making in software engineering practice. In *Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering 2018*, pages 24–34, 2018.

[25] Marly M Carvalho, André Fleury, and Ana Paula Lopes. An overview of the literature on technology roadmapping (trm): Contributions and trends. *Technological Forecasting and Social Change*, 80(7):1418–1437, 2013.

[26] Jeffrey Carver, Dustin Heaton, Lorin Hochstein, and Roscoe Bartlett. Self-perceptions about software engineering: A survey of scientists and engineers. *Computing in Science & Engineering*, 15(1):7, 2013.

[27] Mary Beth Chrissis, Mike Konrad, and Sandy Shrum. *CMMI guidlines for process integration and product improvement*. Addison-Wesley Longman Publishing Co., Inc., 2003.

[28] Paul Clarke and Rory V. O'Connor. The situational factors that affect the software development process: Towards a comprehensive reference framework. *Information and Software Technology*, 54(5):433–447, 2012.

[29] Paul Clarke and Rory V O'Connor. The situational factors that affect the software development process: Towards a comprehensive reference framework. *Information and software technology*, 54(5):433–447, 2012.

[30] William Edwards Deming. *Elementary principles of the statistical control of quality: a series of lectures*. Nippon Kegaku Gijutsu Remmei, 1950.

[31] Denis Dennehy and Kieran Conboy. Going with the flow: An activity theory analysis of flow techniques in software development. *Journal of Systems and Software*, 133:160–173, 2017.

[32] Kees Dorst. The core of 'design thinking'and its application. *Design studies*, 32(6):521–532, 2011.

[33] Anshu Dubey and Lois Curfman McInnes. Proposal for a scientific software lifecycle model. In *Proceedings of the 1st International Workshop on Software Engineering for High Performance Computing in Computational and Data-enabled Science & Engineering*, pages 22–26. ACM, 2017.

[34] Tore Dyba. An empirical investigation of the key factors for success in software process improvement. *IEEE transactions on Software Engineering*, 31(5):410–424, 2005.

[35] Nasir U Eisty, George K Thiruvathukal, and Jeffrey C Carver. Use of software process in research software development: A survey. In *Proceedings of the Evaluation and Assessment on Software Engineering*, pages 276–282. ACM, 2019.

[36] Khaled El Emam, Walcelio Melo, and Jean-Normand Drouin. *SPICE: The theory and practice of software process improvement and capability determination*. IEEE Computer Society Press, 1997.

[37] Luis Fernández-Sanz and Sanjay Misra. Influence of human factors in software quality and productivity. In *International Conference on Computational Science and Its Applications*, pages 257–269. Springer, 2011.

[38] Wendi L Gardner and Megan L Knowles. Love makes you real: Favorite television characters are perceived as "real" in a social facilitation paradigm. *Social Cognition*, 26(2):156–168, 2008.

[39] Ian Gorton. Cyberinfrastructures: Bridging the divide between scientific research and software engineering. *Computer*, 47(8):48–55, 2014.

[40] Olly Gotel, Christelle Scharff, and Andrew Wildenberg. Teaching software quality assurance by encouraging student contributions to an open source web-based system for the assessment of programming assignments. In *Proceedings of the 13th annual conference on Innovation and technology in computer science education*, pages 214–218, 2008.

[41] Liliana Guzmán, Marc Oriol, Pilar Rodríguez, Xavier Franch, Andreas Jedlitschka, and Markku Oivo. How can quality awareness support rapid software development?–a research preview. In *International Working Conference on Requirements Engineering: Foundation for Software Quality*, pages 167–173. Springer, 2017.

[42] Dustin Heaton and Jeffrey C Carver. Claims about the use of software engineering practices in science: A systematic literature review. *Information and Software Technology*, 67:207–219, 2015.

[43] Michael A Heroux. Sustainable & productive: Improving incentives for quality software. Technical report, Sandia National Lab.(SNL-NM), Albuquerque, NM (United States), 2016.

[44] Michael A Heroux. Driving computational science and engineering excellence through reproducibility requirements. Technical report, Sandia National Lab.(SNL-NM), Albuquerque, NM (United States), 2020.

[45] Michael A Heroux, Elsa Gonsiorowski, Rinku Gupta, Reed Milewicz, J David Moulton, Gregory R Watson, Jim Willenbring, Richard J Zamora, and Elaine M Raybourn. Lightweight software process improvement using productivity and sustainability improvement planning (psip). In *Tools and Techniques for High Performance Computing*, pages 98–110. Springer, 2019.

[46] Watts S Humphrey. *Managing the software process*. Addison-Wesley Longman Publishing Co., Inc., 1989.

[47] Jan Wiedemann Jacobsen, Marco Kuhrmann, Jürgen Münch, Philipp Diebold, and Michael Felderer. On the role of software quality management in software process improvement. In *International Conference on Product-Focused Software Process Improvement*, pages 327–343. Springer, 2016.

[48] Mehdi Jazayeri. The education of a software engineer. In *Proceedings. 19th International Conference on Automated Software Engineering, 2004.*, pages xviii–xxvii. IEEE, 2004.

[49] Daniel S Katz, Kenton McHenry, Caleb Reinking, and Robert Haines. 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.

[50] Daniel S Katz, Lois Curfman McInnes, David E Bernholdt, Abigail Cabunoc Mayes, Neil P Chue Hong, Jonah Duckles, Sandra Gesing, Michael A Heroux, Simon Hettrick, Rafael C Jimenez, et al. Community organizations: changing the culture in which research software is developed and sustained. *Computing in Science & Engineering*, 21(2):8–24, 2018.

[51] Diane F Kelly. A software chasm: Software engineering and scientific computing. *IEEE Software*, 24(6):120–119, 2007.

[52] Arif Ali Khan and Jacky Keung. Systematic review of success factors and barriers for software process improvement in global software development. *IET software*, 10(5):125–135, 2016.

[53] Jil Klünder, Regina Hebig, Paolo Tell, Marco Kuhrmann, Joyce Nakatumba-Nabende, Rogardt Heldal, Stephan Krusche, Masud Fazal-Baqaie, Michael Felderer, Marcela Fabiana Genero Bocco, et al. Catching up with method and process practice: An

industry-informed baseline for researchers. In *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice*, pages 255–264. IEEE Press, 2019.

[54] Andrzej Kobyliński. The relationships between software development processes and software product quality. In *International Conference on Business Informatics Research*, pages 161–169. Springer, 2013.

[55] Marco Kuhrmann, Philipp Diebold, and Jürgen Münch. Software process improvement: a systematic mapping study on the state of the art. *PeerJ Computer Science*, 2:e62, 2016.

[56] Marco Kuhrmann and Jürgen Münch. Spi is dead, isn't it?: clear the stage for continuous learning! In *Proceedings of the International Conference on Software and System Processes*, pages 9–13. IEEE Press, 2019.

[57] Mathieu Lavallée and Pierre N Robillard. The impacts of software process improvement on developers: A systematic review. In *2012 34th International Conference on Software Engineering (ICSE)*, pages 113–122. IEEE, 2012.

[58] Sungjoo Lee and Yongtae Park. Customization of technology roadmaps according to roadmapping purposes: Overall process and detailed modules. *Technological forecasting and social change*, 72(5):567–583, 2005.

[59] Hareton KN Leung. Slow change of information system development practice. *Software quality journal*, 8(3):197–210, 1999.

[60] Meira Levy and Chen Huli. Design thinking in a nutshell for eliciting requirements of a business process: A case study of a design thinking workshop. In *2019 IEEE 27th International Requirements Engineering Conference (RE)*, pages 351–356. IEEE, 2019.

[61] Silverio Martínez-Fernández, Anna Maria Vollmer, Andreas Jedlitschka, Xavier Franch, Lidia López, Prabhat Ram, Pilar Rodríguez, Sanja Aaramaa, Alessandra Bagnato, Michał Choraś, et al. Continuously assessing and improving software quality with software analytics tools: a case study. *IEEE access*, 7:68219–68239, 2019.

[62] Christoph Mayr-Dorn, Michael Vierhauser, Stefan Bichler, Felix Keplinger, Jane Cleland-Huang, Alexander Egyed, and Thomas Mehofer. Supporting quality assurance with automated process-centric quality constraints checking. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 1298–1310. IEEE, 2021.

[63] Steve McConnell. Avoiding classic mistakes [software engineering]. *IEEE Software*, 13(5):112, 1996.

[64] Xianhai Meng and Brendan Gallagher. The impact of incentive mechanisms on project performance. *International journal of project management*, 30(3):352–362, 2012.

[65] Erika S Mesh. Supporting scientific se process improvement. In *Proceedings of the 37th International Conference on Software Engineering-Volume 2*, pages 923–926. IEEE Press, 2015.

[66] Qing Mi, Jacky Keung, Xiupei Mei, Yan Xiao, and WK Chan. A gamification technique for motivating students to learn code readability in software engineering. In *2018 International Symposium on Educational Technology (ISET)*, pages 250–254. IEEE, 2018.

[67] Reed Milewicz. Towards evidence-based practice in scientific software development. `https://collegeville.github.io/CW20/WorkshopResources/WhitePapers/milewicz-evidence-based-practice.pdf`. Accessed: 08-11-2021.

[68] Reed Milewicz and Elaine Raybourn. Talk to me: A case study on coordinating expertise in large-scale scientific software projects. In *2018 IEEE 14th International Conference on e-Science (e-Science)*, pages 9–18. IEEE, 2018.

[69] Reed Milewicz, James Willenbring, and Dena Vigil. Research, develop, deploy: Building a full spectrum software engineering and research department. *Research Software Engineers in HPC (RSE-HPC-2020)*, 2020.

[70] Lukas Moldon, Markus Strohmaier, and Johannes Wachs. How gamification affects software developers: Cautionary evidence from a natural experiment on github. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 549–561. IEEE, 2021.

[71] Rodrigo Henrique Barbosa Monteiro, Maurício Ronny de Almeida Souza, Sandro Ronaldo Bezerra Oliveira, Carlos dos Santos Portela, and Cesar Elias de Cristo Lobato. The diversity of gamification evaluation in the software engineering education and industry: Trends, comparisons and gaps. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*, pages 154–164. IEEE, 2021.

[72] Miguel Ehécatl Morales-Trujillo. Learning software quality assurance with bricks. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*, pages 11–19. IEEE, 2021.

[73] Jürgen Münch, Stefan Trieflinger, and Dominic Lang. Product roadmap–from vision to reality: a systematic literature review. In *2019 IEEE International Conference on Engineering, Technology and Innovation (ICE/ITMC)*, pages 1–8. IEEE, 2019.

[74] Mahmood Niazi. An exploratory study of software process improvement implementation risks. *Journal of software: Evolution and Process*, 24(8):877–894, 2012.

[75] Mahmood Niazi. A comparative study of software process improvement implementation success factors. *Journal of Software: Evolution and Process*, 27(9):700–722, 2015.

[76] Mahmood Niazi, David Wilson, and Didar Zowghi. A maturity model for the implementation of software process improvement: an empirical study. *Journal of systems and software*, 74(2):155–172, 2005.

[77] Borislav Nikolik. Software quality assurance economics. *Information and Software Technology*, 54(11):1229–1238, 2012.

[78] Charles Owen. Design thinking: Notes on its nature and use. *Design Research Quarterly*, 2(1):16–27, 2007.

[79] Mark C Paulk, Bill Curtis, Mary Beth Chrissis, and Charles V Weber. Capability maturity model, version 1.1. *IEEE software*, 10(4):18–27, 1993.

[80] Robert Phaal, Clare JP Farrukh, John F Mills, and David R Probert. Customizing the technology roadmapping approach. In *PICMET'03: Portland International Conference on Management of Engineering and Technology Technology Management for Reshaping the World, 2003.*, pages 361–369. IEEE, 2003.

[81] Gustavo Pinto, Igor Wiese, and Luiz Felipe Dias. How do scientists develop scientific software? an external replication. In *25th International Conference on Software Analysis, Evolution and Reengineering, SANER 2018, Campobasso, Italy, March 20-23, 2018*, pages 582–591, 2018.

[82] Elaine M Raybourn. A new paradigm for serious games: Transmedia learning for more effective training and education. *Journal of computational science*, 5(3):471–481, 2014.

[83] Elaine M. Raybourn, Reed Milewicz, David M. Rogers, Benjamin H. Sims, Greg Watson, Elsa Gonsiorowski, , and Jim Willenbring. A data-driven approach to rethinking open source software organizations as a team of teams. *SERP'21 - The 19th Int'l Conf on Software Engineering Research and Practice*, 2021.

[84] Elaine M. Raybourn, Greg Watson, Elsa Gonsiorowski, Reed Milewicz, David M. Rogers, Benjamin H. Sims, and Jim Willenbring. Automating software productivity planning: Lightweight tools for upgrading team practices. *SERP'21 - The 19th Int'l Conf on Software Engineering Research and Practice*, 2021.

[85] Ita Richardson, Louise Reid, Stephen B Seidman, Bob Pattinson, and Yvonne Delaney. Educating software engineers of the future: Software quality research through problem-based learning. In *2011 24th IEEE-CS Conference on Software Engineering Education and Training (CSEE&T)*, pages 91–100. IEEE, 2011.

[86] Everett M Rogers. *Diffusion of innovations*. Simon and Schuster, 2010.

[87] Peter G Rowe. *Design thinking*. MIT press, 1987.

[88] Rien Sach, Helen Sharp, and Marian Petre. Software engineers' perceptions of factors in motivation: The work, people, obstacles. In *2011 International Symposium on Empirical Software Engineering and Measurement*, pages 368–371. IEEE, 2011.

[89] SAONEE Sarker, DB Nicholson, and Kshiti D Joshi. Knowledge transfer in virtual systems development teams: An exploratory study of four key enablers. *IEEE transactions on professional communication*, 48(2):201–218, 2005.

[90] Judith Segal. Scientists and software engineers: A tale of two cultures. 2008.

[91] Vibhu Saujanya Sharma, Vikrant Kaulgud, and Pradeepkumar Duraisamy. A gamification approach for distributed agile delivery. In *Proceedings of the 5th International Workshop on Games and Software Engineering*, pages 42–45, 2016.

[92] Peng Shen, Xiaoming Ding, Wenjun Ren, and Chujun Yang. Research on software quality assurance based on software quality standards and technology management. In *2018 19th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, pages 385–390. IEEE, 2018.

[93] Kapil Singi, Vikrant Kaulgud, RP Jagadeesh Chandra Bose, Swapnajeet Gon Choudhury, Sanjay Podder, and Adam P Burden. Are software engineers incentivized enough? an outcome based incentive framework using tokens. In *2020 IEEE International Workshop on Blockchain Oriented Software Engineering (IWBOSE)*, pages 37–47. IEEE, 2020.

[94] Robert E Slavin. Quality, appropriateness, incentive, and time: A model of instructional effectiveness. *International Journal of Educational Research*, 21(2):141–157, 1994.

[95] Flaminio Squazzoni, Giangiacomo Bravo, and Károly Takács. Does incentive provision increase the quality of peer review? an experimental study. *Research Policy*, 42(1):287–294, 2013.

[96] Flávio Steffens, Sabrina dos Santos Marczak, CHRISTOPH TREUDE, Leif Singer, David Redmiles, Ban Al-Ani, et al. Using gamification as a collaboration motivator for software development teams: A preliminary framework. *Anais do XII Simpósio Brasileiro de Sistemas Colaborativos, 2015, Brasil.*, 2015.

[97] Craig A Stewart, William K Barnett, Eric A Wernert, Julie A Wernert, Von Welch, and Richard Knepper. Sustained software for cyberinfrastructure: Analyses of successful efforts with a focus on nsf-funded software. In *Proceedings of the 1st Workshop on The Science of Cyberinfrastructure: Research, Experience, Applications and Models*, pages 63–72. ACM, 2015.

[98] Tim Storer. Bridging the chasm: A survey of software engineering practice in scientific programming. *ACM Computing Surveys (CSUR)*, 50(4):47, 2017.

[99] Erik Strack. A message from the asc program office. `https://sharepoint.sandia.gov/sites/ASCSQE/SitePages/Home.aspx`, 2018. Accessed: 2021-08-11.

[100] Paolo Tell, Jil Klünder, Steffen Küpper, David Raffo, Stephen G MacDonell, Jürgen Münch, Dietmar Pfahl, Oliver Linssen, and Marco Kuhrmann. What are hybrid development methods made of?: an evidence-based characterization. In *Proceedings of the International Conference on Software and System Processes*, pages 105–114. IEEE Press, 2019.

[101] Jennifer Turgeon, Christopher Lujan, and Elsa M. Galloway. Sandia national laboratories advanced simulation and computing fy20 software quality engineering summary appraisal report. Technical report, Sandia National Laboratories (SAND2021-4271), April 2021.

[102] Medha Umarji and Carolyn Seaman. Predicting acceptance of software process improvement. *ACM SIGSOFT Software Engineering Notes*, 30(4):1–6, 2005.

[103] Michael Unterkalmsteiner, Tony Gorschek, AKM Moinul Islam, Chow Kian Cheng, Rahadian Bayu Permadi, and Robert Feldt. Evaluation and measurement of software process improvement—a systematic literature review. *IEEE Transactions on Software Engineering*, 38(2):398–424, 2011.

[104] June M Verner, Muhammad Ali Babar, Narciso Cerpa, Tracy Hall, and Sarah Beecham. Factors that motivate software engineering teams: A four country empirical study. *Journal of Systems and Software*, 92:115–127, 2014.

[105] Yi Wang and Min Zhang. Penalty policies in professional software development practice: a multi-method field study. In *2010 ACM/IEEE 32nd International Conference on Software Engineering*, volume 2, pages 39–47. IEEE, 2010.

[106] Dave West, Mike Gilpin, Tom Grant, and Alissa Anderson. Water-scrum-fall is the reality of agile for most organizations today. *Forrester Research*, 26, 2011.

[107] Beth Yost, Michael Coblenz, Brad Myers, Joshua Sunshine, Jonathan Aldrich, Sam Weber, Matthew Patron, Melissa Heeren, Shelley Krueger, and Mark Pfaff. Software development practices, barriers in the field and the relationship to software quality. In *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, pages 1–6, 2016.

[108] Ewa Ziemba and Monika Eisenbardt. Incentives encouraging prosumers to knowledge sharing–framework based on polish study. *Online Journal of Applied Knowledge Management (OJAKM)*, 4(2):146–166, 2016.

8. APPENDIX

8.1. Design Thinking Results And Artifacts

We conducted the Design Thinking workshop on June 2, 2021. Elaine Raybourn acted as organizer and facilitator; Miranda Mundt, Reed Milewicz, Jacob Moxley, and Han Yong Wunrow acted as participants. All participants were internal to Sandia. The workshop took place during a 3-hour block in which the participants were directed through several exercises. The majority of the workshop focused on two activities: an empathy map and generating and analyzing an "As-Is Scenario."

The empathy map activity began with creating a fictitious persona - Albert, a mid-career research software scientist. The goal of this activity was for the participants to put themselves into Albert's shoes and consider the space of software quality from his perspective. The empathy map was composed of four sections which the participants were prompted: "Considering the persona, what would Albert say, think, feel, and do in relation to software quality activities?" After a few minutes of individual brainstorming, the participants came together as a group to compare. The following commonalities were found in each section:

Says	Thinks	Feels	Does
Lack of time.	Software quality is a box-checking activity.	Nervous and fearful about time and money.	Does bare minimum, delegates to others.
Creative independence.	Research comes first.	Publish or perish.	Does it quickly.
Territoriality.	Only my team is using it, so why bother?	There is a barrier to communication.	Does the code compile? Then it's fine.

Table 8-1 Commonalities in brainstorming results for the empathy map exercise.

From this exercise, we learned that the largest considerations for the standard research software scientist in respect to software quality are time commitment, funding availability, and ability to do conduct career-advancing work (*e.g.*, papers, patents). With the primary considerations of Albert in mind, the participants were then directed to the next activity: the creation of the "As-Is Scenario." The figure below details a single participant's entries in each element. Note that SQ denotes "software quality" and SD denotes "software development."

In this activity, the participants collaborated to determine all of the steps a project may go through in the current as-is approach to software quality. This resulted in the "Phases or steps" at the top of the figure. Then using these steps and once again considering the previously created persona, participants were directed to individually brainstorm what Albert might be doing, thinking, or feeling in regards to software quality during each phase of his project. The team came back together and found the following commonalities:

Using this chart and the brainstorming as a guide, the team then thought about need-scenario questions, that is, questions in the format of "Albert needs ____ so that ____." After brainstorming a list of possibilities, the group came together and choose one need-scenario to consider with the goal of generating a list of potential action items to address pain points: "Albert needs a way to prioritize software quality alongside research so that he doesn't feel pressured to choose between the two." Considering this scenario, the group discussed and annotated concerns and actions that could be taken to address potential pain points. For example, in the "Seek funding" phase, the red text is a suggestion for a solution - that funding be directly provided by a

Acvitity	Doing, Thinking, Feeling
Seeking Funding	• **Doing**: Writing proposals, meeting with stakeholders, applying for grants. • **Thinking**: The money is better spent in other areas. How can I frame this correctly? Will they laugh at me for adding this software quality to my proposal? • **Feeling**: Nervous and out of my wheelhouse. Is the money well-spent? I'm going to be made fun of it not!
Finding Team Member	• **Doing**: Interviewing people, asking around other researchers, talking to managers, pitching ideas to software developers • **Thinking**: I need to find someone. Who fits my team? I want to maintain creative control. They should share my research values. • **Feeling**: Hesitant, relieved, uncertain, uncomfortable.
Learning Software Development	• **Doing**: Talking, reading, automated tools training • **Thinking**: I'm glad to have a team member. I don't know what to search for or ask about. I don't have time for this. • **Feeling**: Relieved to not be alone, nervous, hesitant, overly busy. Is this money well spent?
Plan Research	• **Doing**: Experimenting, creating algorithms, leading meetings, delegating tasks • **Thinking**: Hypotheses! My favorite part. What do I need to get answers? I have big ideas. Need to interpret the data. • **Feeling**: Excited! Game day! Finally, this is what I trained for.
Write Code	• **Doing**: Fixing bugs, working with legacy code, matching algorithms to code, partnering with team, dividing work • **Thinking**: How do I apply best practices? This is hard to do. I should minimize time spent on this. • **Feeling**: Glad I'm not alone. Will this work?
Do Assessment	• **Doing**: Referring to standards, receiving feedback from auditor, conferring with software developer, deciding on metrics • **Thinking**: What is good enough? How do I define quality? Good that I started early. • **Feeling**: Reassured and informed. Satisfied that we did it. Stressed about showing proof.

Table 8-2 Commonalities in participant's as-is scenario results.

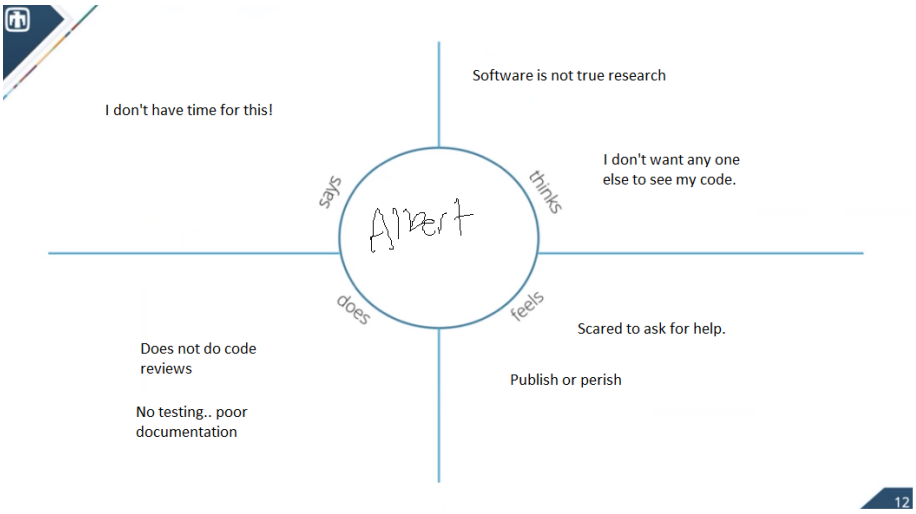

Figure 8-1 An empathy map created for a fictitious mid-career research software scientist (Albert)

Center, outside of a proposal process. Another example under the "Find SD team member" phase is posed as a question, identifying a common concern. The full results are available in figure below.

The biggest takeaways from this activity fell into three areas of support: social, consultation, and leadership. The need for training was also identified.

Through addressing the persona of "Albert" by putting ourselves in his shoes (*e.g.*, perspective-taking), we recognized that there are many opportunities to address nervousness and uncertainty. "Albert" will most likely feel intimidated by software quality best practices and would need some amount of support to help alleviate this. Much of his uncertainty would likely be diminished by proper access to software quality resources, such as how-tos, formal training, consultation groups, or even something less structured such as an easy-to-use website with quick videos or a list of subject matter experts. Additionally, "Albert" may need to feel that he is not divided between the world of "performance reports" and "software quality best practices." With leadership thoroughly on-board, he could feel less pressure to choose between activities and could feel empowered to begin practicing software quality activities.

8.2. Collegeville Teatime Results

The participants for the Collegeville Teatime came from a mix of industry, academia, and national labs. We presented the participants of the teatime discussion with three personas in the format of "speed dating cards." These cards presented quick synopses of the personas, their role, their type of project, and their current approach or objections to software quality activities. The personas were loosely based on characters of the popular television series 'The Office.' 'The Office' has been previously utilized for classroom learning with some success [21]. A study conducted by Gardner and Knowles also provides support for our adaptation of television characters to allow

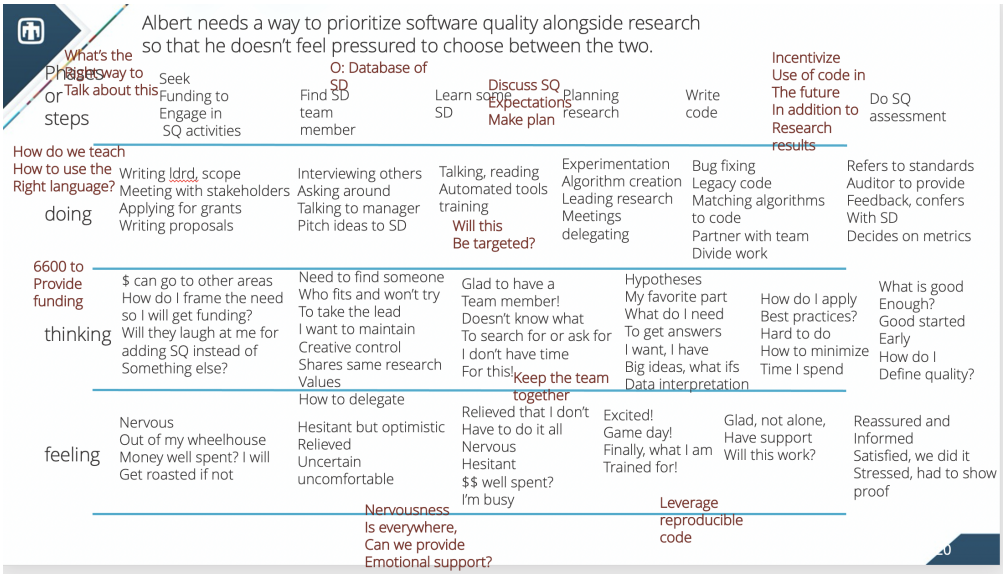

Figure 8-2 The full results from the scenario exercise.

our teatime participants to more quickly relate to the personas [38]. In their study, likable, familiar characters were perceived as more real, facilitating greater social connection. Our speed dating cards are provided below.

Speed Dating Card	
Name	Jim Halpert
Age	43
Job/Role	Research Scientist
Project Maturity	Small, research-oriented team; newly funded, short-term project
Project Goal	Publishable results
Software Purpose	Software is a means to an end. It helps answer research questions.
Opinion of Software Quality	"Software quality is what other software teams do, not research scientists."

We started our role-play with the persona of Jim Halpert, the research scientist whose ultimate goal for his project is publishable results. Throughout this role-play, a clear theme impacting adoption of software quality was lack of time or funding. Most of these projects are short-term and come from competitive funding sources, thus promoting submitters to minimize their monetary request and time spent on non-research tasks. Given extra emphasis or a dedicated pool of funding for software quality activities, Jim would more readily engage.

A secondary concern was the general size of the code. Frequently, many research projects result in a code that is only several hundred lines, which tends to result in fewer software quality practices. A recommendation from the group was to provide a resource for code inspections from an independent third party. This could be a dedicated group, such as the Software Engineering &

Research Department in the Center for Computing Research, the ASC SQE team, or a dual paper/code reviewer who does both steps as part of the peer review process.

Speed Dating Card	
Name	Dwight Schrute
Age	51
Job/Role	DevOps / Scrum Master
Project Maturity	Large, well-established software team; mature, productionized software with regular builds deliverable to goverment customers
Project Goal	Publishable results
Software Purpose	Development and maintenance to meet customers' needs.
Opinion of Software Quality	"I know what software quality is, but I don't have the time or funding for it. It adds too many cycles that we can't afford."

Our next persona was Dwight Schrute, the DevOps / Scrum master from a large, well-established software team who regularly delivers releases to government customers. Dwight was particularly difficult to sway on the concept of software quality. The general theme here was, "Our processes are already established. It would rock the boat too much to change anything." This was a sticking point in the conversation that was difficult to get past.

The group eventually came to the conclusion that, for people like Dwight, it would be crucial to have either a strong push from leadership or a large subset of the group requesting changes in the processes. Otherwise, they will continue with the "business as usual" approach.

Speed Dating Card	
Name	Pam Beesly
Age	42
Job/Role	Research Scientist / Software Developer
Project Maturity	Split-team: team lead for a project entering its second year and a team member for a well-established software team.
Project Goal	Operable software that answers research questions.
Software Purpose	It has multiple uses, depending on the project type.
Opinion of Software Quality	"I'm not sure how to implement good software quality practices on my teams. It will probably set us back, and I'm not sure if I can convince them that it's worthwhile."

Our final persona was Pam Beesly, a split-role staff member who does work that on both newly-established and well-established software projects. Pam is mostly confused about how to implement software quality best practices and where to get the help she needs.

In this scenario, Pam was amenable to change. It was clear that she mostly didn't know how to implement best practices, and she didn't have knowledge of an existing support system that would allow her to do so. Because she was on two projects, she was inclined to start small and test out any new practice so she could better advocate advocate its efficacy to the well-established project. We found that for a person like Pam, the biggest incentive was access to a group of consultants to help her iron out the details and help her see the big-picture results of her actions.

At the end of the teatime, we did a large-group debrief on what we learned throughout the role-playing activity. A clear theme across all three personas was the power of word of mouth - that is, given a loud advocate, more people are likely to adopt processes and practices. Another general consensus of the group was that at the top-level, there needs to be a stronger "hammer" approach - that is, clear direction, expectations, and requirements from leadership, in addition to clear consequences should those requirements not be met. For the practitioners, though, they needed access to appropriate funding as well as easy to obtain resources and support that lower the barrier to entry.

DISTRIBUTION

Hardcopy—Internal

Number of Copies	Name	Org.	Mailstop
1	Reed Milewicz	1424	1318
1	Miranda Mundt	1424	1320
1	Elaine Raybourn	5572	0639

Email—Internal [redacted]

Name	Org.	Sandia Email Address
Technical Library	1911	sanddocs@sandia.gov

This page intentionally left blank.

This page intentionally left blank.

Sandia National Laboratories

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.