In Their Shoes: Persona-Based Approaches to Software Quality Practice Incentivization

Miranda R. Mundt, Reed M. Milewicz, and Elaine M. Raybourn, *Sandia National Laboratories, Albuquerque, NM, 87185, USA*

Many teams struggle to adapt and right-size software engineering best practices for quality assurance to fit their context. Introducing software quality is not 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 that 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? We explored this question through a persona-based ideation exercise at the 2021 Collegeville Workshop on Scientific Software in which we created three unique personas that represent different scientific software developer perspectives.

AMONG software engineering practitioners and researchers, it is understood that quality in software is no accident but is instead the result of intentional effort and the use of practices, processes, and norms that enable such quality. Moreover, given that time and effort are finite and there are many different software quality attributes worth targeting, achieving software quality requires strategic decision-making, and the evaluation of tradeoffs.

In the context of scientific software development, frequently software quality takes a backseat to other priorities, such as delivering scientific results. Even at institutions where quality standards are put in place, software quality is regularly framed more as a "check the box for compliance" activity rather than a cultural practice with emphasis on the values that software quality achieves. This is not to say that scientists and engineers do not value the quality of their code; certain quality attributes, such as performance and correctness, receive significant attention.

What we can say, however, is that they invest in software quality insofar as they perceive it to be worthwhile and/or easy to adopt. For example, industry quality practices are often an imperfect fit for scientific software development. Too much effort spent on topics like usability or extensibility may detract from their objectives, and even if they see that quality as worth pursuing, scientific software developers may not know how to accomplish it efficiently. While trends, such as the growth of the research software engineering (RSE) movement, may alleviate some of these concerns by embedding more software engineering expertise directly into scientific teams, researchers will always play an integral role in scientific software development and in the quality of that work. For those researchers, the case for software quality has to be compelling enough and framed in a way that aligns with their goals and needs. With that in mind, we must ask: When and how can teams be effectively incentivized to adopt and integrate meaningful and enduring software quality practices?

At the 2021 Collegeville Workshop on Scientific Software (CW'21),[a] we explored these questions through persona-based ideation exercise with workshop participants during a teatime session, which is a themed discussion activity. Personas are detailed fictional characters that often represent end users. The participants, who were all part of the scientific software development community and who came from a variety of backgrounds including academia, industry, and national laboratories, were invited to consider

1521-9615 © 2022 IEEE
Digital Object Identifier 10.1109/MCSE.2022.3170027
Date of publication 25 April 2022; date of current version 31 August 2022.

[a]https://collegeville.github.io/CW21/

three different personas: Jim, the Research Scientist; Dwight, the DevOps and Scrum Master; and Pam, the Scientific Software Developer. We presented the participants with these personas in the format of "speed dating cards" (Table 1). These cards presented quick synopses of the personas, their role, their type of project, and their current approach or objections to software quality activities. Using these dating cards as their inspiration, participants volunteered to role-play as the persona and offer arguments and responses to the other participants who attempted to probe each persona's resistance/willingness regarding software quality adoption. The volunteer role-players based their responses and reactions to participants' questions on their perceptions of the proposed persona, inspired by their own collective experiences and assumptions. The other participants were motivated with the goal of finding common objections and concerns relating to the adoption of software quality.

A RAPID REVIEW PROTOCOL IS A TIME-BOXED LITERATURE REVIEW DESIGNED TO DELIVER EVIDENCE IN A TIMELY AND ACCESSIBLE WAY.

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.[1] A study conducted by Gardner and Knowles[2] also provides support for our adaptation of television characters to allow our participants to more quickly relate to the personas. In their study, likable, familiar characters were perceived as more real, facilitating greater social connection.

For this article, we first distilled the results from the ideation exercise into a collection of themes that surfaced, representing common perspectives, priorities, and scenarios among scientific software developers. Next, we conducted a rapid review of the scholarly literature on software engineering practices and software quality incentivization strategies. A rapid review protocol is a 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. We used Harzing's Publish or Perish to anonymously fetch the top 1000 results from Google Scholar, and we down-

FIGURE 1. Our methodology. Personas of scientific software developers were created for use in an ideation exercise to better understand perceptions about software quality. We then conducted a review of the scholarly literature on software quality practice adoption. We compared these two sources of evidence to generate recommendations for how to incentivize scientific software developers to pursue software quality in their projects.

selected 42 articles that met our relevance and quality criteria. This down-selection process was achieved by a scan of each article's abstract and citations for relevance to the topic of "software quality incentivization," followed by the assignment of a ranking (1–5, 1 being irrelevant, 5 being perfectly relevant). From the resulting set, we selected articles on software quality incentivization that spoke directly to the issues and concerns presented in the vignettes, and we opted to further enrich our corpus with a few well-cited papers addressing related topics (e.g., a systematic review on continuous integration).

Finally, we worked to align the best available evidence from the literature with the results of our exercise to produce actionable guidance on how to encourage teams to value and achieve quality in their software. A more detailed account of our findings is available from our technical report.[3]

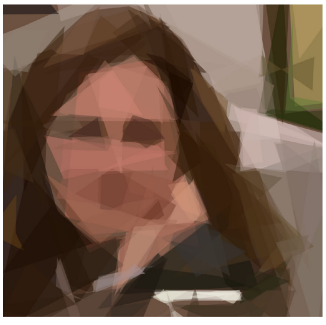
JIM, RESEARCH SCIENTIST

Jim works on a small, research-oriented scientific software team. He sees software as a means to an end in that it helps answer his research questions, and he mostly shares any code he writes with his immediate

TABLE 1. "Dating cards" generated by the authors for the 2021 collegeville workshop teatime session.

- **Name**: Jim
- **Age**: 43
- **Job/Role**: Research scientist
- **Project Maturity**: Small, research-oriented team; newly funded, short-term project
- **Project Goal**: Publishable results
- **The Purpose of Software**: "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."

- **Name**: Dwight
- **Age**: 51
- **Job/Role**: DevOps, Scrum master
- **Project Maturity**: Large, well-established software team; mature, productionized software with regular builds deliverable to government customers
- **Project Goal**: Development and maintenance of software to meet customers' needs
- **The Purpose of Software**: "Software is the end-product, and the users dictate the future development of the software."
- **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."



- **Name**: Pam
- **Age**: 42
- **Job/Role**: Research scientist and software developer
- **Project Maturity**: Split between two teams: 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
- **The Purpose of Software**: "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."

team. His goal is to create publishable results at the end of his research projects. His manager or project lead may conduct yearly assessments, but since the project is not as mature as others, the requirements are not as daunting. For the most part, software quality is not talked about on his team. It is mostly what "other software teams do," not research scientists.

Integration Into the Publication Process

Multiple times during this prompt, the participants asked those role-playing as Jim questions relating to his ultimate project deliverable: publications. Participants prodded him on his view of reproducibility, verification, and time-to-science as they relate to scientific software development. Throughout these questions, Jim's responses revealed that he will respect the requirements of his target journal or venue, but up to this point, he has done well enough. As for making software available for others to reproduce, those acting as Jim asked, "Why read my code? Read my paper. The code is fairly small, and the paper should be able to stand on its own besides." When pressed on this point, Jim concedes that if a journal requires source code be made available, he will do it. The

more that the publishing organization values the quality of the software, the more likely that Jim will invest resources into software quality practices. As noted by those role-playing as Jim, "The right answer [to software quality] is probably code inspections and people with the right background to inspect it... Maybe the person who reviews the paper could also do the code review."

These remarks align well with existing literature on scientific software development. The participants role-playing as Jim called out a frequently common gap in research scientists' knowledge: the lack of formalized software engineering training. Many times, research scientists believe their existing knowledge to be sufficient for their programming needs, and yet, at the same time, they lack familiarity with evidence-based good software quality practices, such as test-driven development and regression testing.[4] Given that some research scientists lack knowledge of the implementation of software quality practices, it is unsurprising if they may be hesitant to commit time to learn about and attempt to use these practices on their own.

Jim's comment on the link between publications and code quality is a step in the right direction to incentivize the use of software quality practices. Currently, very few venues ask for the code that produced results. Rather, peer reviewers focus on the science that resulted, and frequently the reliability of this code is judged based on personal opinion and reputation of the scientist and their institution.[5] By intertwining the article and code review processes, not only could research scientists begin to value software quality practices, but the overall reliability and trustworthiness of scientific results will increase.

Practices for Short-Term Projects

Another issue that those role-playing as Jim brought up was how short-lived funding places constraints on software quality practices. Particularly in government-funded facilities, scientists frequently apply for funding in one or three year periods. In order to be competitive, their proposals for research must be compelling both in content but also in cost. A quote from one volunteer was, "A week or longer is often too long for me to spend on software. My project ends in three weeks!" The current funding regime can lead to tight budgets that cannot accommodate extra personnel or "luxurious" actions, such as software quality practices.

Heroux *et al.* note that this tension is common among scientific software teams.[6] Moreover, since many research scientists do not know what they do not know when it comes to software quality practices, process improvement activities need to be clearly identified and scoped. Such activities are most likely to succeed when approached iteratively and using lightweight frameworks. For that reason, the authors present the productivity and sustainability improvement planning (PSIP) toolkit. PSIP allows projects to grade their current implementations of different software engineering practice areas (such as testing, continuous integration, and team on-boarding) using a simple "scorecard" and identify how they can improve their score in each area in small, iterative steps. In this way, short-lived projects will be able to make realistic goals backed by evidence-based recommendations that can be implemented within their minimal time-frame.

Summary

Research scientists, such as Jim, value time-to-science and ability to publish reliable and respectable results. As a result, they will value those actions that improve their publications and do not detract from their limited and valuable time. Through this role-play prompt and the resulting discussion, we identified that the integration of code reviews into the publication peer review process and lightweight tools that enable small, manageable, and natural improvements to software development processes would be the two most influential contributions to incentivize better software quality practices for those with values reflecting the persona of Jim.

DWIGHT, DEVOPS SCRUM MASTER

Dwight works on a large, well-established software team. Their software has many users, some legacy code, and is in a productionized state. Dwight's team uses Agile software development methodology and has regular builds and deliverables to government funded customers. His code is delivered to analysts and scientists, and his goal is to complete development and maintenance based on his customers' needs and their vision for future use. Yearly quality assessments are a must. Dwight is very familiar with the concept of software quality, but he does not have the time or funding to address it in his code and processes. As a result of his previous experiences, he's jaded—every push for software quality has always added time and cycles to his work that he can ill-afford.

Adopting Continuous Integration

Those role-playing as Dwight made his objective clear from the start: to meet his customers' needs by delivering the right software capabilities when they are needed. With this in mind, the discussion turned

towards testing and test coverage. With prodding from the participants, Dwight admitted that his team could and should improve. That being said, participants kept hitting a wall with Dwight as the role-playing volunteers consistently reminded everyone that with such a large project, choices must be made judiciously in order to maximize time allocated to development activities. Pulling from personal experience, one of the volunteers told a story of a staff member on a software team who proposed adding a continuous integration (CI) service into the team's workflow, and although accepted by other members, all attempts to make this a reality failed. Eventually, too much time had been lost, and the effort was abandoned.

Both inside and outside of scientific software development, these kinds of setbacks are not uncommon. A systematic review by Shahin *et al.*[7] notes a variety of challenges that can hold back adoption of CI and deployment technologies, including poor test quality, limited time, and resources to stand up new infrastructure, a lack of consensus on the benefits of CI, and a lack of training on how to use CI effectively. In Dwight's case, work needed to be done to provide a foundation for CI adoption, which would have increased his team's chances of success. This includes improving testing to take advantage of CI tools, planning ahead by establishing a formal role for team members taking responsibility for testing and CI setup and maintenance, and engaging customers directly about testing practices (building the case to allow time for CI adoption). Dwight is already a believer in developer roles, iterative development methodologies, and close engagement with customers. CI adoption needs to be framed in a way that fits those methods of working: testing needs to be made relevant to the customer, responsibilities around testing and automation need to be clearly delegated, and the team should incrementally transition to a CI-ready codebase.

Top–Down Vision

Two clear imperatives that the role-play volunteers emphasized were that 1) the customer is always right and 2) the chain-of-command's wishes should be respected. Frequently throughout the discussion, the volunteers noted that if management or customers expressed a desire for better software quality, Dwight would prioritize it; otherwise, "We have been delivering software successfully for years, and there haven't been any issues that we couldn't correct." Likewise, he pays close attention to requirements that flow downstream from leadership and participates in formal software quality assessments. So long as he gets a passing grade, however, he wants to avoid unnecessarily altering how his project operates and stagnate the work the customers have demanded, even if it would make the work go more smoothly. Stepping into Dwight's shoes, one volunteer remarked, "Is my job easy? No. Could my job be easier? Yes. But if what I have been doing so far has worked, why do I need to change what I am doing? I have been able to get the features in and ship the product on time. Changing our process would mess everyone up."

A systematic review of software process improvement success factors by Niazi[8] finds that senior management commitment (above all else), customer satisfaction, and positive cultural attitudes toward process improvement are important to success. Dwight would agree. He is not being intransigent; he is simply a rational actor who is carefully reading the environment in which he works. He does not want to expend effort on activities for which neither his customers nor his supervisors are asking. This dovetails with the recommendations of participants in Niazi's study who call for leadership to be engaged and conversant in software process improvement activities and for sufficient time and resources to be provided to staff carrying out improvement activities.

These remarks also echo the results of a survey by Yost *et al.* on barriers to software quality among government-funded, high-consequence software projects.[9] Survey respondents noted that not getting enough time for software development, having to convince managers that they should spend time refactoring their code, and not getting clear specifications from customers on what a product should do were among the top barriers. Barriers such as these were found to have statistically significant correlations with reduced maintainability and evolvability—that is, teams end up having to work a lot harder to accomplish their objectives.

Summary

Team leads for highly productionized science and engineering codes, such as Dwight, prioritize meeting deadlines and minimizing risks. They strategically value software quality insofar as it is expected of them but can rationally choose to do work in "inefficient" ways because they cannot spare the overhead or the risk of transitioning to a new state of practice. Many have seen first-hand what can happen when improvement activities are carried out injudiciously. Those like Dwight would benefit most from improving software quality practices in a structured way and, more than anything else, having clear expectations for quality from customers and management.

PAM, SCIENTIFIC SOFTWARE DEVELOPER

Pam is a scientific software developer who is split in her tasks. She is a team lead for a project that is entering its second year and a team member for a another, well-established software team. From her perspective, software has multiple uses, depending on the project type, but the ultimate goal is the same: operable software that answers relevant scientific questions. She wants to implement some software quality in her team but does not know how to go about it. She knows that introducing quality practices will set her teams back a bit, and she is not sure she can afford that or be able to convince other team members that the practices are worthwhile.

RSEs BY DESIGN ARE VERSED BOTH IN SOFTWARE ENGINEERING BEST PRACTICES AS WELL AS FLUENT IN ONE OR MORE DOMAIN SCIENCES.

Access to Resources and Expertise

Frequently during this discussion, the participants role-playing as Pam pointed out that while she was interested by software quality and the value it could bring to her projects, she had little or no access to resources that would help her get started. Participants felt that Pam and others on her team would benefit from mentorship and training opportunities in addition to bringing in someone who was an expert in software engineering practices for consultation and one-on-one help, as necessary. One role-play volunteer emphatically expressed, "Neither of the teams that I am on know much about software quality practices at all, so we're lost. What I *really* want is someone sitting with me and helping me."

This need for support in the form of training, mentorship, and staff time and resources is echoed in literature. Work by Niazi[8] found that the availability of sufficient training and resources were important positive factors to implementing software process improvement. As research scientists and scientific software developers will always be directly involved in the production of any scientific software, they must have access to the proper knowledge in order to implement good software quality practices.

In addition to having at least a working knowledge, access to those with software engineering expertise is essential to implementing new processes and procedures effectively. The introduction of RSEs into the scientific research community is a step in this direction. RSEs by design are versed both in software engineering best practices as well as fluent in one or more domain sciences.[10] As a result, these individuals are able to better understand and anticipate the needs of scientific software developers and provide targeted advice on tools, activities, processes, and procedures that could improve a project's overall software quality. Their integration directly into the organizational structure would also empower scientific software developers to take advantage of these resources which would overall save time for scientific research.

Natural Evolution of Practices

Because of the nature of their positions, those role-playing as Pam noted that she would be concerned about attempting to implement practices that would be able to scale and change according to the team's maturity and specific needs. The large team she is on likely adopted a set of tools and practices long ago, and even though their circumstances have changed, the old solutions have accrued inertia over time. One volunteer noted about Pam's projects,"I assume that if they're using something, it's outdated. I'm not sure we can even convince them something new would be good for them because they are coming from old, legacy code projects with established practices." As for the newer team with the smaller codebase, "Now may be the right time to instill new practices with the team. They aren't as invested so you can change directions. But I still don't know where should I start. How do we get it going?" Change is a fact of life, and software team dynamics are no exception, but a primary consideration is how to adapt to that change.

There is a large body of open questions relating to exactly this topic. Although over 15 years old, many of the questions presented by Mens *et al.* regarding challenges in software (and team) evolution lack satisfactory answers,[11] such as how best to manage the challenges between the coevolution of different software artifacts and the practices and processes around those artifacts. What we can recommend, however, for organizations like Pam's is to develop and socialize tiered software quality standardization models that capture recommended practices at different levels of maturity. A systematic review of quality models by Nistala *et al.*[12] notes that there is a rich body of literature around this subject, although quality realization—connecting specific software practices to quality outcomes—often goes unaddressed. We see this as an

opportunity for co-design of practice standards with scientific software teams to determine what practices are appropriate for different projects.

In the absence of comprehensive guidance, we can recommend a set of baseline practices for scientific software teams. For open science projects, use of open-source version control platforms, such as GitHub, allow the centralization of source code, testing, internal and external documentation, project management, and more. Other tools, such as Sphinx, automate documentation generation and integrate its creation into the software development life cycle. Although there is a lack of literature to rigorously answer the specific question of practices that evolve with a team, there is empirical evidence to suggest that the usage of the correct tools that automate certain quality practices (e.g., documentation and testing) would meet Pam's need.

Summary

Pam is interested in software quality and knows that it will bring value to her projects, but she struggles to find appropriate resources and practices that answer her questions and evolve naturally with her project. She needs to feel like the gap in her expertise is addressed, as well as that her choices will continue to be beneficial throughout the life of her software project. For Pam, the largest incentive would be empowerment through training and mentorship, access to hands-on, face-to-face expertise, potentially in the form of an RSE, and the use of tools and services that integrate software quality practices into a team's workflow.

DISCUSSION

Personas are a helpful tool for promoting empathy and understanding and generating authentic scenarios; when paired with other forms of evidence, they can aid in the design of adoption interventions.[3] In this article, we considered six occurring threads for barriers to implementing software quality practices, and we matched them with peer-reviewed literature to generate recommendations for how to incentivize ways of working that would lead to improved software quality. Taking these results together, we have identified a set of common themes that connect the scientific software developer's motivations to pursue software quality to strategies which would empower them to do so (see Figure 2).

Incentivizing quality through better policies: A common theme among all three role-playing prompts is the ways in which the choice to invest in software quality is influenced by institutional environments and the policies that shape them. Jim would benefit from

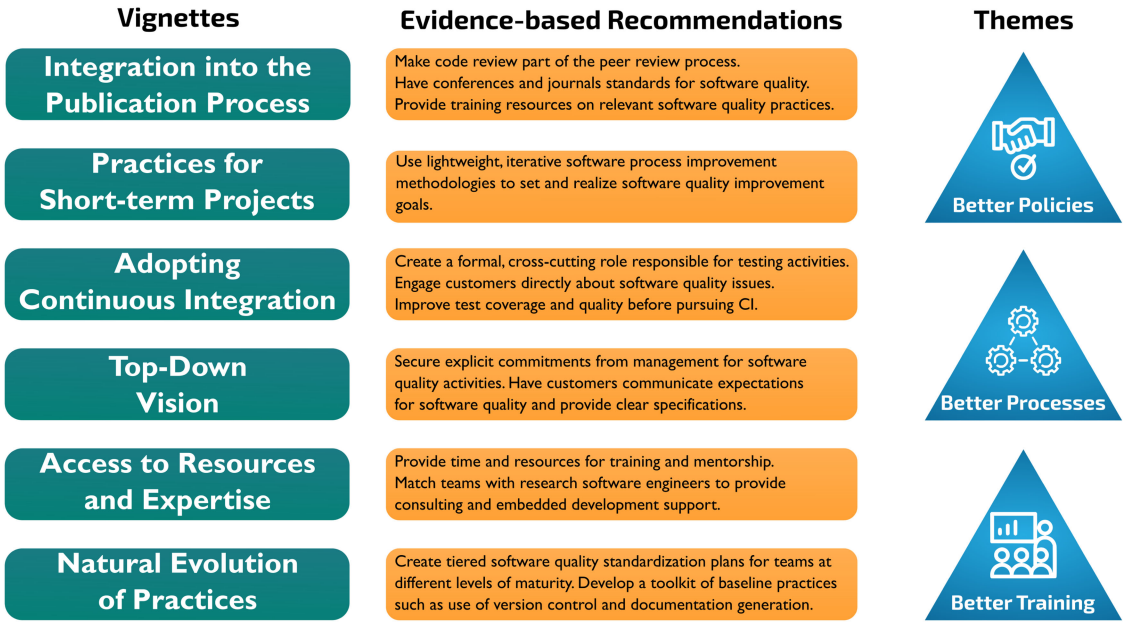


FIGURE 2. Overview of the results presented in this article. In response to the issues raised in the vignettes, we developed a set of evidence-based recommendations to incentivize the adoption of software quality practices. We distilled these recommendations into a set of overarching themes which we cover in the discussion section.

journals and conferences setting expectations for software quality and incorporating code review into the peer-review process where needed. Dwight wants to have management communicate a clear commitment to supporting quality-improving activities and to have customers engaged in conversations about software quality. Pam seeks institutional support for training and mentorship as well as providing software engineering specialists to support her projects, that is is, scientific software developers' motivations to adopt software quality practices depend, at least in part, on the extent to which those practices are valued and encouraged by their communities and institutions.

Incentivizing quality through better processes: Another emerging theme is that software quality needs to be operationalized through intentional processes that make quality an explicit goal. Jim could use a software process improvement methodology to set and realize software quality goals. Dwight could institute a formal, cross-cutting role for team members focused on test coverage and quality. Pam could enforce specific development practices through automated tools. That is, if teams want to improve on software qualities, such as usability, reliability, or performance, this has to be made explicit, measurable, and incorporated into a team's workflow.

Incentivizing quality through better training: Finally, a recurring finding is that a lack of sufficient training and/or knowledge forces scientific software developers to be conservative in their decision-making around software quality practices. Jim thinks code review could be folded in with the peer-review process to ensure quality, but he recognizes that neither he nor his peers have adequate training in software development practices. Dwight wants to adopt modern software development tools, such as continuous integration, but his team is not familiar enough with such tools to make adoption feasible. Pam would like to pick the right set of practices for each of her projects and adjust those practices as those projects evolve, but she lacks a good mental model for how to make those kinds of choices. In each of these situations, our personas see an opportunity to improve how they develop software but are stymied because they do not have the knowledge they need.

CONCLUSION

Software quality practices succeed only insomuch as developers realize their value. Through a guided ideation exercise at the 2021 Collegeville Workshop on Scientific Software, three personas with different value systems were presented and participants, all existing members of the scientific software development community, explored common barriers that different roles experience with respect to software quality. Overall from this discussion and a following rapid literature review, we identified that opportunities to incentivize software quality practices come in the form of better policies, better processes, and better training. Scientific software developers require top–down expression of value from management, publishers, and customers. These developers also need measurable and integrated practices as an intrinsic portion of their workflows. Finally, scientific software developers lack access to formalized training and knowledge sources that would enable better software quality decisions. With proper action taken in all three categories, we would achieve more reliable, robust, and respected scientific software.

ACKNOWLEDGMENTS

The authors would like to thank the organizers of Collegeville 2021 and the participants of the "Incentivizing Software Quality Practices" ideation exercise for the opportunity to explore this subject. They would also like to thank Sandia's VISTA Innovation Tournament for supporting the larger effort from which this article was created. *Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DENA0003525*. SAND No. SAND2022-5143 J

REFERENCES

1. J. Bloch and S. E. Spataro, "Lessons from scranton: Using scenes from the television series the office to teach topics in professional communication," *IEEE Prof. Commun.*, vol. 59, no. 3, pp. 274–287, Sep. 2016, doi: 10.1109/TPC.2016.2583300.
2. W. L. Gardner and M. L. Knowles, "Characters are perceived as 'real' in a social facilitation paradigm," *Social Cogn.*, vol. 26, no. 2, pp. 156–168, 2008, doi: 10.1521/soco.2008.26.2.156.
3. E. M. Raybourn, R. Milewicz, and M. Mundt, "Incentivizing adoption of software quality practices," Sandia National Lab., Albuquerque, NM, USA, Tech. Rep. SAND2022-1691703441, 2022, doi: 10.2172/1845193.
4. J. Carver, D. Heaton, L. Hochstein, and R. Bartlett, "Self-perceptions about software engineering: A survey of scientists and engineers," *Comput. Sci. Eng.*, vol. 15, no. 1, pp. 7–11, 2013, doi: 10.1109/MCSE.2013.12.

5. L. N. Joppa *et al.*, "Troubling trends in scientific software use," *Science*, vol. 340, no. 6134, pp. 814–815, 2013, doi: 10.1126/science.1231535.
6. M. A. Heroux *et al.*, "Lightweight software process improvement using productivity and sustainability improvement planning (PSIP)," in *Tools and Techniques for High Performance Computing*. Cham, Switzerland: Springer, 2020, pp. 98–110, doi: 10.1007/978-3-030-44728-1_6. [Online]. Available: https://link.springer.com/book/10.1007/978-3-030-44728-1
7. 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, doi: 10.1109/ACCESS.2017.2685629.
8. M. Niazi, "A comparative study of software process improvement implementation success factors," *J. Softw., Evol. Process*, vol. 27, no. 9, pp. 700–722, 2015, doi: 10.1002/smr.1704.
9. B. Yost *et al.*, "Software development practices, barriers in the field and the relationship to software quality," in *Proc. 10th ACM/IEEE Int. Symp. Empirical Softw. Eng. Meas.*, 2016, pp. 1–6, doi: 10.1145/2961111.2962614.
10. R. Baxter, N. C. Hong, D. Gorissen, J. Hetherington, and I. Todorov, "The research software engineer," in *Proc. Digit. Res. Conf.*, 2012, pp. 1–3. [Online]. Available: https://www.research.ed.ac.uk/en/publications/the-research-software-engineer
11. T. Mens, M. Wermelinger, S. Ducasse, S. Demeyer, R. Hirschfeld, and M. Jazayeri, "Challenges in software evolution," in *Proc. 8th Int. Workshop Princ. Softw. Evol.*, 2005, pp. 13–22, doi: 10.1109/IWPSE.2005.7.
12. P. Nistala, K. V. Nori, and R. Reddy, "Software quality models: A systematic mapping study," in *Proc. IEEE/ACM Int. Conf. Softw. Syst. Process.*, 2019, pp. 125–134, doi: 10.1007/s11432-018-9608-3.

MIRANDA R. MUNDT is a research software engineer and member of technical staff with the Department of Software Engineering and Research, Sandia National Laboratories, Albuquerque, NM, 87185, USA, with an interdisciplinary background in applied mathematics and economics. Within the Exascale Computing Project (ECP), she is a member of the Interoperable Design of Extreme-Scale Application Software (IDEAS) project and part of the Productivity and Sustainability Improvement Planning (PSIP) team. She is also an active member of the US-RSE Association, where she helps lead outreach and diversity, equity, and inclusion efforts. Contact her at mmundt@sandia.gov.

REED M. MILEWICZ is a computer scientist and software engineering researcher with the Department of Software Engineering and Research, Sandia National Laboratories, Albuquerque, NM, 87185, USA. His research focuses on developing better practices, processes, and tools to improve software development in the scientific domain. Contact him at rmilewi@sandia.gov.

ELAINE M. RAYBOURN is a social scientist with the Statistics and Human Systems Group, Sandia National Laboratories, Albuquerque, NM, 87185, USA. She worked from the U.K. (British Telecom), Germany (Fraunhofer FIT), and France's National Institute for Research in Digital Science and Technology (INRIA). Her research interests include virtual teams, software productivity methods, immersive virtual environments, and transmedia learning. She is a fellow of the European Research Consortium in Informatics and Mathematics (ERCIM). Contact her at emraybo@sandia.gov.