A survey of the state of the practice for research software (#67516)

First submission

Guidance from your Editor

Please submit by **2 Dec 2021** for the benefit of the authors (and your $200 publishing discount) .

Structure and Criteria
Please read the 'Structure and Criteria' page for general guidance.

Raw data check
Review the raw data.

Image check
Check that figures and images have not been inappropriately manipulated.

Privacy reminder: If uploading an annotated PDF, remove identifiable information to remain anonymous.

Files

Download and review all files from the materials page.

31 Figure file(s)
10 Latex file(s)

Structure and Criteria

Structure your review

The review form is divided into 5 sections. Please consider these when composing your review:

1. **BASIC REPORTING**
2. **EXPERIMENTAL DESIGN**
3. **VALIDITY OF THE FINDINGS**
4. General comments
5. Confidential notes to the editor

You can also annotate this PDF and upload it as part of your review

When ready submit online.

Editorial Criteria

Use these criteria points to structure your review. The full detailed editorial criteria is on your guidance page.

BASIC REPORTING

- Clear, unambiguous, professional English language used throughout.
- Intro & background to show context. Literature well referenced & relevant.
- Structure conforms to PeerJ standards, discipline norm, or improved for clarity.
- Figures are relevant, high quality, well labelled & described.
- Raw data supplied (see PeerJ policy).

EXPERIMENTAL DESIGN

- Original primary research within Scope of the journal.
- Research question well defined, relevant & meaningful. It is stated how the research fills an identified knowledge gap.
- Rigorous investigation performed to a high technical & ethical standard.
- Methods described with sufficient detail & information to replicate.

VALIDITY OF THE FINDINGS

- Impact and novelty not assessed. *Meaningful* replication encouraged where rationale & benefit to literature is clearly stated.
- All underlying data have been provided; they are robust, statistically sound, & controlled.
- Conclusions are well stated, linked to original research question & limited to supporting results.

Standout reviewing tips

The best reviewers use these techniques

Tip	*Example*
Support criticisms with evidence from the text or from other sources	*Smith et al (J of Methodology, 2005, V3, pp 123) have shown that the analysis you use in Lines 241-250 is not the most appropriate for this situation. Please explain why you used this method.*
Give specific suggestions on how to improve the manuscript	*Your introduction needs more detail. I suggest that you improve the description at lines 57- 86 to provide more justification for your study (specifically, you should expand upon the knowledge gap being filled).*
Comment on language and grammar issues	*The English language should be improved to ensure that an international audience can clearly understand your text. Some examples where the language could be improved include lines 23, 77, 121, 128 – the current phrasing makes comprehension difficult. I suggest you have a colleague who is proficient in English and familiar with the subject matter review your manuscript, or contact a professional editing service.*
Organize by importance of the issues, and number your points	*1. Your most important issue* *2. The next most important item* *3. …* *4. The least important points*
Please provide constructive criticism, and avoid personal opinions	*I thank you for providing the raw data, however your supplemental files need more descriptive metadata identifiers to be useful to future readers. Although your results are compelling, the data analysis should be improved in the following ways: AA, BB, CC*
Comment on strengths (as well as weaknesses) of the manuscript	*I commend the authors for their extensive data set, compiled over many years of detailed fieldwork. In addition, the manuscript is clearly written in professional, unambiguous language. If there is a weakness, it is in the statistical analysis (as I have noted above) which should be improved upon before Acceptance.*

A survey of the state of the practice for research software

Jeffrey C. Carver [Corresp., 1], **Nic Weber** [2], **Karthik Ram** [3], **Sandra Gesing** [4], **Daniel S. Katz** [5]

[1] Computer Science, University of Alabama - Tuscaloosa, Tuscaloosa, AL, United States

[2] Information School, University of Washington, Seattle, WA, United States

[3] Berkeley Institute for Data Science, University of California, Berkeley, Berkeley, CA, United States

[4] Discovery Partners Institute, Chicago, IL, USA

[5] NCSA & CS & ECE & iSchool, University of Illinois at Urbana-Champaign, Urbana, IL, United States

Corresponding Author: Jeffrey C. Carver
Email address: carver@cs.ua.edu

Research software is a critical component of contemporary scholarship. Yet, most research software is developed and managed in ways that are at odds with its long-term sustainability. This paper presents findings from a survey of 1149 researchers about sustainability challenges they face in developing and using research software. Some of our key findings include a repeated need for more opportunities and time for developers of research software to receive training. These training needs cross the software lifecycle and various types of tools. We also identified the recurring need for better models of funding research software and for providing credit to those who develop the software so they can advance in their careers. The results of this survey will help inform future infrastructure and service support for software developers and users, as well as national research policy aimed at increasing the sustainability of research software.

A Survey of the State of the Practice for Research Software

Jeffrey C. Carver[1], Nicholas Weber[2], Karthik Ram[3], Sandra Gesing[4], and Daniel S. Katz[5]

[1]**Department of Computer Science, University of Alabama**
[2]**Information School, University of Washington**
[3]**Berkeley Institute for Data Science, University of California, Berkeley**
[4]**Discovery Partner Institute, University of Illinois Chicago**
[5]**NCSA & CS & ECE & iSchool, University of Illinois at Urbana-Champaign**

Corresponding author:
Jeffrey C. Carver[1]

Email address: carver@cs.ua.edu

ABSTRACT

Research software is a critical component of contemporary scholarship. Yet, most research software is developed and managed in ways that are at odds with its long-term sustainability. This paper presents findings from a survey of 1149 researchers about sustainability challenges they face in developing and using research software. Some of our key findings include a repeated need for more opportunities and time for developers of research software to receive training. These training needs cross the software lifecycle and various types of tools. We also identified the recurring need for better models of funding research software and for providing credit to those who develop the software so they can advance in their careers. The results of this survey will help inform future infrastructure and service support for software developers and users, as well as national research policy aimed at increasing the sustainability of research software.

INTRODUCTION

In almost all areas of research, from hard sciences to the humanities, the processes of collecting, storing, and analyzing data and of building and testing models have become increasingly complex. Our ability to navigate such complexity is only possible because of the existence of specialized software, often referred to as *research software*. Research software plays such a critical role in day to day research that a comprehensive survey reports 90-95% of researchers in the US and the UK rely upon it and more than 60% were unable to continue working if such software stopped functioning (Hettrick, 2014). While the research community widely acknowledges the importance of research software, the creation, development, and maintenance of research software is still *ad hoc* and improvised, making such infrastructure fragile and vulnerable to failure.

In many fields, research software is developed by academics who have varying levels of training, ability, and access to expertise, resulting in a highly variable software landscape. As researchers are under immense pressure to maintain expertise in their research domains, they have little time to stay current with the latest software engineering practices. In addition, the lack of clear career incentives for building and maintaining high quality software has made research software development unsustainable. The lack of career incentives has occurred partially because the academic environment and culture have developed over hundreds of years, while software has only recently become important, in some fields over the last 60+ years, but in many others, just in the last 20 or fewer years (Foster, 2006).

Further, only recently have groups undertaken efforts to promote the role of research software (e.g., the Society of Research Software Engineers[1], the US Research Software Engineer Association[2]) and train

[1]https://society-rse.org
[2]https://us-rse.org

researchers in modern development practices (e.g., the Carpentries[3], IRIS-HEP[4], and MolSSI[5]). While much of the development of research software occurs in academia, important development also occurs in national laboratories and industry. Wherever the development and maintenance of research software occurs, that software might be released as open source (most likely in academia and national laboratories) or it might be commercial/closed source (most likely in industry, although industry also produces and contributes to open source).

The open source movement has created a tremendous variety of software, including software used for research and software produced in academia. It is difficult for researchers to find and use these solutions without additional work (Joppa et al., 2013). The lack of standards and platforms for categorizing software for communities often leads to re-developing instead of reusing solutions (Howison et al., 2015). There are three primary classes of concerns, pervasive across the research software landscape, that have stymied this software from achieving maximum impact.

- *Functioning of the individual and team*: issues such as training and education, ensuring appropriate credit for software development, enabling publication pathways for research software, fostering satisfactory and rewarding career paths for people who develop and maintain software, and increasing the participation of underrepresented groups in software engineering.
- *Functioning of the research software*: supporting sustainability of the software; growing community, evolving governance, and developing relationships between organizations, both academic and industrial; fostering both testing and reproducibility, supporting new models and developments (e.g., agile web frameworks, Software-as-a-Service), supporting contributions of transient contributors (e.g., students), creating and sustaining pipelines of diverse developers.
- *Functioning of the research field itself*: growing communities around research software and disparate user requirements, cataloging extant and necessary software, disseminating new developments and training researchers in the usage of software.

In response to some of the challenges highlighted above, the US Research Software Sustainability Institute (URSSI)[6] conceptualization project, funded by NSF, is designing an institute that will help with the problem of sustaining research software. The overall goal of the conceptualization process is *to bring the research software community together to determine how to address known challenges to the development and sustainability of research software and to identify new challenges that need to be addressed.* One important starting point for this work is to understand and describe the current state of the practice relative to those important concerns. Therefore, in this paper we describe the results of a community survey focused on this goal.

BACKGROUND

Previous studies of research software have often focused on the development of cyberinfrastructure (Borgman et al., 2012) and the various ways software production shapes research collaboration (Howison and Herbsleb, 2011, 2013; Paine and Lee, 2017). While these studies provide rich contextual observations about research software development processes and practices, they often focus on small groups or in laboratory settings that are difficult to generalize. Therefore, there is a need to gain a broader understanding of the research software landscape.

A number of previous surveys have provided valuable insight into research software development and use, as briefly described ~~next~~ in the next subsection. Based on the results of these surveys and from other related literature, the remainder of this section motivates a series of research questions focused on important themes related to the development of research software.

Previous Surveys

The following list provides an overview of the previous surveys on research software, including the context of each survey. Table 1 summarizes the surveys.

[3] https://carpentries.org
[4] https://iris-hep.org
[5] https://molssi.org
[6] http://urssi.us

- *How do Scientists Develop and Use Scientific Software?* (Hannay et al., 2009) describes the results of a survey of 1972 scientists who develop and use software. The survey focused on questions about (1) how and when scientists learned about software development/use, (2) the importance of developing/using software, (3) time spent developing/using software, (4) hardware platforms, (5) user communities, and (6) software engineering practices.
- *How Do Scientists Develop Scientific Software? An External Replication* (Pinto et al., 2018) is a replication of the previous study (Hannay et al., 2009) conducted ten years later. The replication focused on scientists who develop R packages. The survey attracted 1553 responses. The survey asked very similar questions to the original survey, with one exception. In addition to replicating the original study, the authors also asked respondents to identify the "most pressing problems, challenges, issues, irritations, or other 'pain points' you encounter when developing scientific software." A second paper, *Naming the Pain in Developing Scientific Software* (Wiese et al., 2020), describes the results of this question in the form of a taxonomy of 2,110 problems that are either (1) technical-related, (2) social-related, or (3) scientific-related.
- *A Survey of Scientific Software Development* (Nguyen-Hoan et al., 2010) surveyed researchers in Australia working in multiple scientific domains. The survey focused on programming language use, software development tools, development teams and user bases, documentation, testing and verification, and non-functional requirements.
- *A Survey of the Practice of Computational Science* (Prabhu et al., 2011) reports the results of interviews of 114 respondents from a diverse set of domains all working at Princeton University. The interviews focused on three themes: (1) programming practices, (2) computational time and resource usage, and (3) performance enhancing methods.
- *Troubling Trends in Scientific Software* (Joppa et al., 2013) reports on the results from about 450 responses working in a specific domain, species distribution modeling, that range from people who find software difficult to use to people who are very experienced and technical. The survey focused on understanding why respondents chose the particular software they used and what other software they would like to learn how to use.
- *Self-Perceptions About Software Engineering: A Survey of Scientists and Engineers* (Carver et al., 2013) reports the results from 141 members of the Computational Science & Engineering community. The primary focus of the survey was to gain insight into whether the respondents thought they knew enough software engineering to produce high-credibility software. The survey also gathered information about software engineering training and about knowledge of specific software engineering practices.
- *"Not everyone can use Git:" Research Software Engineers' recommendations for scientist-centered software support (and what researchers really think of them)* (Jay et al., 2016) describes a study that includes both Research Software Engineers and domain researchers to understand how scientists publish code. The researchers began by interviewing domain scientists who were trying to publish their code to identify the barriers they faced in publishing their code. Then they interviewed Research Software Engineers to understand how they would address those barriers. Finally, they synthesized the results from the Research Software Engineer interviews into a series of survey questions sent to a larger group of domain researchers.
- *It's impossible to conduct research without software, say 7 out of 10 UK researchers* (Hettrick, 2018, 2014) describes the results of 417 responses to a survey of 15 Russel Group Universities in the UK. The survey focused on describing the characteristics of software use and software development within research domains. The goal was to provide evidence regarding the prevalence of software and its fundamental importance for research.
- *Surveying the US National Postdoctoral Association Regarding Software Use and Training in Research* (Nangia and Katz, 2017) reports on the results of 209 responses to provide insight into the role of software in conducting research at US universities. The survey focused on the respondents' use of research software and the training they have received in software development.
- *Towards Computational Reproducibility: Researcher Perspectives on the Use and Sharing of Software* (AlNoamany and Borghi, 2018) reports on the results from 215 respondents across a range of disciplines. The goal of the survey was to understand how researchers create, use, and share software. The survey also sought to understand how the software development practices aligned with the goal of reproducibility.

- *Software Saved International Survey* (Philippe et al., 2019) reports on the results from approximately 1000 responses to a survey of research software engineers from around the world. The goal of the survey is to describe the current state of research software engineers related to various factors including employment, job satisfaction, development practices, use of tools, and citation practices.

Study	Focus	Respondents
Hannay et al. (2009)	How scientists develop and use software	1972
Pinto et al. (2018)	Replication of Hannay et al. (2009)	1553
Wiese et al. (2020)	Additional results from Pinto et al. (2018) focused on problems encountered when developing scientific software	1577
Nguyen-Hoan et al. (2010)	Software development practices of scientists in Australia	60
Prabhu et al. (2011)	Practice of computational science in one large university	114
Joppa et al. (2013)	Researchers in species domain modeling with varying levels of expertise	∼450
Carver et al. (2013)	Software engineering knowledge and training among computational scientists and engineers	141
Hettrick (2018, 2014)	Use of software in Russell Group Universities in the UK	417
Jay et al. (2016)	How scientists publish code	65
Nangia and Katz (2017)	Use of software and software development training in US Postdoctoral Association	209
AlNoamany and Borghi (2018)	How the way researchers use, develop, and share software impacts reproduciblity	215
Philippe et al. (2019)	Research Software Engineers	∼1000

Table 1. Previous Surveys

Software Engineering Practices

Based on the results of the surveys described in the previous subsection, we can make some observations about the use of various software engineering practices employed while developing software. The set of practices research developers find useful appear to have some overlap and some difference from those practices employed by developers of business or IT software. Interestingly, the results of the previous surveys do not paint a consistent picture regarding the importance and/or usefulness of various practices. Here we highlight some of the key results from these previous surveys, organized roughly in the order of the software engineering lifecycle.

Requirements The findings of two surveys (Pinto et al., 2018; Hannay et al., 2009) reported both that requirements were important to the development of research software but also that they were one of the least understood phases. Other surveys reported that (1) requirements management is the most difficult technical problem (Wiese et al., 2020) and (2) the amount of requirements documentation is low (Nguyen-Hoan et al., 2010).

Design Similar to requirements, surveys reported that design was one of the most important phases (Hannay et al., 2009) and one of the least understood phases (Pinto et al., 2018; Hannay et al., 2009). In addition, other surveys reported that (1) testing and debugging are the second most difficult technical problem (Nguyen-Hoan et al., 2010) and (2) the amount of design documentation is low (Wiese et al., 2020).

Testing There were strikingly different results related to testing. A prior survey of research software engineers found almost $2/3$ of developers do their own test, but less than 10% reported the use of formal testing Philippe et al. (2019). Some surveys ((Pinto et al., 2018; Hannay et al., 2009)) reported that testing was important. However, another survey reported that scientists do not regularly test their code (Prabhu

et al., 2011). Somewhere in the middle, another survey reports that testing is commonly used, but the use of integration testing is low (Nguyen-Hoan et al., 2010).

Software Engineering Practices Summary This discussion all leads to the first research question: *RQ1: What activities do research software developers spend their time on, and how does this impact the perceived quality and long-term accessibility of research software?*

Software tools and support

The development and maintenance of research software includes the use of standard software engineering tools such as version control systems (Milliken et al., 2021) and continuous integration(Shahin et al., 2017) as well as custom libraries developed for a specific analytic tasks, or even language specific interpreters to ease program execution.

Previous surveys have asked researchers and research software engineers about the most frequently used open-source software in development. Surveys of research software developers and users have reported the use of standard software languages and even the types of tools used in analaysis (AlNoamany and Borghi, 2018), but there has been relatively little description of the tools upon which research software developers depend, and to what extent these tools are seen by developers as supporting sustainable research software practices. We therefore seek to understand tool usage and support in a second research question that asks: *RQ2: What tools do research software developers use and what additional tools are needed to support sustainable development practices?*

Education and Training

While researchers often develop research software for the express purpose of conducting research, previous studies demonstrate that these researchers are rarely purposely trained to develop software. A 2012 survey reported that research software developers had little formal training and were mostly self-taught (Carver et al., 2013).

A UK survey (Hettrick, 2018, 2014) reported only 55% of respondents had some software development training. Of those only 40% had formal training, with 15% being self-taught. In addition, only 2% of respondents who develop their own software had no training in software development. The 2017 survey of US National Postdoctoral Association (Nangia and Katz, 2017) found similar results: while 95% of the respondents used research software, 54% reported they had not received any training in software development (Nangia and Katz, 2017). When analyzed by gender (self reported binary of men and women) these two surveys show remarkable similarities in the gap of training for men (63% in the UK and 63% in the US) and women (39% in the UK and 32% in US). The AlNoamany and Borghi (2018) survey reported similar results: 53% of respondents had formal training in coding conventions and best practices.

The Hannay et al. (2009) survey along with the Pinto et al. (2018) replication reported slightly less positive results. Regarding different mechanisms for learning about software development, 97% and 99% of the respondents thought *self-study* was important or very important, while only 13% and 22% found *formal training* to be important or very important.

The results of these prior surveys suggest that research software developers may not have access to sufficient training in software development. In addition, the results of the Joppa et al. (2013) survey indicate that most respondents want increased computational skills. The authors advocate for formal training in software engineering as part of the University science curriculum.

Therefore, we pose the following research question that guides our specific survey questions related to training – *RQ3: What training is available to research software developers and does this training meet their needs?*

Funding and Institutional Support

One of the key sustainability dilemmas for research software is the lack of direct financial support for development and maintenance. Successful research grants often focus on the merits of a new idea and the potential novel scientific or scholarly contribution of progress made on that idea. However, both institutions that support research (e.g., universities and national laboratories) and grant-making bodies that fund research (e.g., federal agencies and philanthropic organizations) often fail to recognize the central importance of software development and maintenance in conducting novel research (Goble, 2014). In turn, there is a little direct financial support for the development of new software or the sustainability of existing software upon which research depends (Katerbow et al., 2018). In particular, funding agencies

typically have not supported the continuing work needed to maintain software after its initial development. This lack of support is despite increasing recognition of reproducibility and replication crises that depend, in part, upon reliable access to the software used to produce a new finding (Hocquet and Wieber, 2021).

In reaction to a recognized gap in research funding for sustainable software, many projects have attempted to demonstrate the value of their work through traditional citation and impact analysis (Anzt et al., 2020) as well as through economic studies. An example of the latter was performed by a development team of the widely used AstroPy packages in Astronomy. Using David A. Wheeler's *SLOCCount* method for economic impact of open-source software they estimate the cost of reproducing AstroPy to be approximately $8.5 million and the annual economic impact on astronomy alone to be approximately $1.5 million (Muna et al., 2016).

There is, recently, increased attention from funders on the importance of software maintenance and archiving, including the Software Infrastructure for Sustained Innovation (SI2) program at NSF, the NIH Data Commons (which includes software used in biomedical research), the Alfred P. Sloan Foundation's Better Software for Science program, and the Chan Zuckerberg Initiative's Essential Open Source Software for Science program which provide monetary support for the production, maintenance, and adoption of research software. Despite encouraging progress there is still relatively little research that focuses specifically on how the lack of direct financial support for software sustainability impacts research software engineers and research software users. We seek to better understand this relationship through two specific research questions that focus on the impact of funding on software sustainability: *RQ4a: What is the available institutional support for research software development?* and *RQ4b: What sources of institutional funding are available to research software developers?*

Who is "we" here? The current paper's author, or Philippe et al.? Since the RQs are still being introduced, hopefully this does not actually refer to the current paper's authors.

Career Paths

While most of previous surveys did not address the topic of career paths, the survey of research software engineers (Philippe et al., 2019) did briefly address this question. Because the results differ across the world, we only report results for respondents in the US. First, 57% of respondents were funded by grants and 47% by institutional support. Second, respondents had been in their current position for an average of 8.5 years. Last, 97% were employed full-time.

Because of the lack of information from prior surveys, we focus the rest of this discussion on other work to provide background. In 2012, the Software Sustainability Institute (SSI) organized the Collaborations Workshop[7] that addressed the question: *why is there no career for software developers in academia?* The work of the participants and of the SSI's policy team led to the foundation of the UK RSE association and later to the Society of Research Software Engineering. More and more national RSE associations are being founded, such as the US Research Software Engineer Association (US-RSE)[8], and have gained momentum recently.

awkward wording

Current evaluation and promotion processes in academia and national labs typically follow the traditional pattern of rewarding activities that include publications, funding, and advising students. However, there are other factors that some have considered. Managers of RSE teams state that when hiring research developers, it is important that those developers are enthusiastic about research topics and have problem-solving capabilities[9]. Another factor, experience in research software engineering, can be evaluated by contributions to software in platforms like GitHub. However, while lines of code produced, number of solved bugs, and work hours may not be ideal measures for developer productivity, they can provide insight into the sustainability and impact of research software, i.e. the presence of an active community behind a software package that resolves bugs and interacts with users is part of sustainability of software and impact on research [10]. In addition, CaRCC (the Campus Research Computing Consortium) has defined job families and templates for job positions that can be helpful both for hiring managers and HR departments that want to recognize the role of RSEs and HPC Facilitators in their organizations[11].

However, there is still not a clearly defined and widely accepted career path for research software engineers in the US. We pose the following research question that guides our specific survey questions

If it comes out later that the scope is the US - this should be stated in the introduction

What about other countries?

[7] http://software.ac.uk/cw12

[8] http://us-rse.org/

[9] https://cosden.github.io/improving-your-RSE-application

[10] https://github.com/Collegeville/CW20/blob/master/WorkshopResources/WhitePapers/gesing-team-organization.pdf

[11] https://carcc.org/wp-content/uploads/2019/01/CI-Professionalization-Job-Families-and-Career-Gui pdf

related to career paths – *RQ5: What factors impact career advancement and hiring in research software?*

Credit

While most of the previous surveys did not address the topic of credit, the survey of research software engineers (Philippe et al., 2019) does contain a question about how researchers are acknowledged when their software contributes to a paper. The results showed that 47% were included as a co-author, 18% received only an acknowledgement, and 21% received no mention at all. Therefore, we focus on other work to provide the necessary background.

Considering credit leads to a set of interlinked research questions, which we can investigate by asking individuals who, with roles as software developers and software project collaborators, directly have important information. Here we take a white box approach and examine the inside of the box.

- How do individuals want their contributions to software projects to be recognized, both as individuals and as members of teams?
- How do software projects want to record and make available credit for the contributions to the projects?

In addition, the same individuals can express opinions that help investigate additional questions, in particular where these opinions provide a view about organizations into which they may not have direct insight. Here we can only take a black box approach and examine the box from the outside. (A white box approach would require a survey of different participants.)

- How does the existing ecosystem, based largely on the historical practices related to contributions to journal and conferences papers and monographs, measure, store, and disseminate information about contributions to software, and what is being missed?
- How do institutions (e.g. hiring organizations, funding organizations, professional societies) use the information about contributions to software that exists, and what information is being missed?

We also recognize that there are not going to be simple answers to these questions (CASBS Group on Best Practices in Science, 2018; Albert and Wager, 2013), and that any answers will likely differ to some extent between disciplines (Dance, 2012). Many professional societies and publishers have specific criteria for authorship of papers (e.g., they have made substantial intellectual contributions, they have participated in drafting and/or revision of the manuscript, they agree to be held accountable for any issues relating to correctness or integrity of the work (Association for Computing Machinery, 2018)), typically suggesting that those who have contributed but do not meet these criteria be recognized via an acknowledgment. While this approach is possible in a paper, there is no equivalent for software, other than papers about software. In some disciplines, such as those where monographs are typical products, there may be no formal guidelines. Author ordering is another challenge. The ordering of author names typically has some meaning, though the meaning varies between disciplines. Two common practices are alphabetic ordering, such as is common in economics (Weber, 2018) and ordering by contribution with the first author being the main contributor and the last author being the senior project leader, as is occurs in many fields (Riesenberg and Lundberg, 1990). The fact that the contributions of each author is unclear has led to activities and ideas to record their contributions in more detail (Allen et al., 2014; The OBO Foundry, 2020; Katz, 2014).

Software in general has not been well-cited (Howison and Bullard, 2016), in part because the scholarly culture has not treated software as something that should be cited, or in some cases, even mentioned. The recently-perceived reproducibility crisis (Baker, 2016) has led to changes, first for data (which also was not being cited (Task Group on Data Citation Standards and Practices, 2013)) and more recently for software. For software, these changes include software papers being published, both in general journals and in journals that specialize in software papers (e.g., the Journal of Open Source Software (Smith et al., 2018)), and principles calling for direct software citation (Smith et al., 2016) and guidance on how to do so (Katz et al., 2021). Software, as a digital object, also has the advantage that it is usually stored as a collection of files, often in a software repository. This fact means that it is relatively simple to add an additional file that contains metadata about the software, including creators and contributions, in one of a number of potential styles (Wilson, 2013; Druskat, 2020; Jones et al., 2017). This effort has recently been reinforced by GitHub, who have made it easy to add such metadata to repositories and to generate citations for those repositories (Smith, 2021).

Therefore, we pose the following research questions to guide our specific survey questions related to credit – *RQ6a What do research software projects require for crediting or attributing software use?* and *RQ6b: How are individuals and groups given institutional credit for developing research software?*

Diversity

Previous research has found that both gender and tenure (length of commitment to a project) are positive and significant predictors of productivity in open source software development (Vasilescu et al., 2015). Using similar data, Ortu et al. (2017) demonstrate that diversity of nationality among team members is a predictor of productivity. However, this demographic characteristic of a team also leads to less polite and civil communication (via filed issues and discussion boards).

Nafus (2012)'s early qualitative study of gender in open source, through interview and discourse analysis of patch notes, describes sexist behavior that is linked to low participation and tenure for women in distributed software projects.

The impact of codes of conduct (CoC) - which provide formal expectations and rules for the behavior of participants in a software project - have been studied in a variety of settings. In open-source software projects codes of conduct have been shown to be widely reused (e.g. Ubuntu, Contributor Covenant, Django, Python, Citizen, Open Code of Conduct, and Geek Feminism have been reused more than 500 times by projects on GitHub) (Tourani et al., 2017).

There are few studies of the role and use of codes of conduct in research software development. Joppa et al. (2013) point to the need for developing rules which govern multiple aspects of scientific software development, but specific research that addresses the prevalence, impact, and use of a code of conduct in research software development have not been previously reported.

The 2018 survey of the research software engineer community across seven countries (Philippe et al., 2019) showed the percentage of respondents who identified as male as between 73% (US) and 91% (New Zealand). Other diversity measures are country-specific and were only collected in the UK and US, but in both, the dominant group is overrepresented compared with its share of the national population.

Therefore, we pose the following research question to guide our specific survey questions related to diversity – *RQ7: How do current Research Software Projects document diversity statements or codes of conduct, and what support is needed to further diversity initiatives?*

METHODS

To understand sustainability issues related to the development and use of research software, we developed a Qualtrics[12] survey focused on the seven research questions defined in the Background section. This section describes the design of the survey, the solicited participants, and the qualitative analysis process we followed.

Survey Design

We designed the survey to capture information about how individuals develop, use, and sustain research software. The survey first requested demographic information to help us characterize the set of respondents. Then, we enumerated 38 survey questions (35 multiple choice and 3 free response). We divided these questions among the seven research questions defined in the Background Section. This first set of 38 questions went to all survey participants, who were free to skip any questions.

Then, to gather more detailed information, we gave each respondent the option to answer follow-up questions on one or more of the seven topic areas related to the research questions. For example, if a respondent was particularly interested in *Development Practices* she or he could indicate their interest in answering more questions about that topic. Across all seven topics, there were 28 additional questions (25 multiple choice and 3 free response). Because the follow-up questions for a particular topics were only presented to respondents who expressed interest in that topic, the number of respondent to these questions is significantly lower than the number of respondents to first set of 38 questions. This discrepancy in the number of respondents is reflected in the data presented below.

In writing the questions, where possible, we replicated the wording of questions from the previous surveys about research software (described in the Background Section).

[12] http://www.qualtrics.com

Survey Participants

We distributed the survey to potential respondents through two primary venues:

1. **Email Lists**: To gather a broad range of perspectives, we distributed the survey to 33,293 NSF and 39,917 NIH PIs whose projects were funded for more than $200K in the five years prior to the survey distribution and involve research software and to mailing lists of research software developers and research software projects.
2. **Snowball Sampling**: We also used snowballing by asking people on the email lists to forward the survey to others who might be interested. We also advertised the survey via Twitter.

The approach we used to recruit participants makes it impossible to calculate a response rate. We do not know how many times people forwarded the survey invitation or the number of potential participants reached by the survey.

Research Ethics

We received approval for the survey instruments and protocols used in this study from the University of Notre Dame Committee Institutional Review Board for Social and Behavioral Responsible Conduct of Research (protocol ID 18-08-4802). Prior to taking the survey, respondents had to read and consent to participate. If a potential respondent did not consent, the survey terminated. To support open science, we provide the following information: (1) the full text of the survey and (2) a sanitized version of the data (Carver et al., 2021).

ANALYSIS

After providing an overview of the participant demographics, we describe the survey results relative to each of the research questions defined in the Background section.

Participant Demographics

We use each of the key demographics gathered on the survey to characterize the sample. Note that because some questions were optional, the number of respondents differs across the demographics.

Respondent Type We asked each respondent to characterize their relationship with research software as one of the following:

- *Researcher* - someone who only uses software
- *Developer* - someone who only develops software
- *Combination* - both of the above roles

The respondents were fairly evenly split between *Researchers* at 43% (473/1109) and *Combination* at 49% (544/1109). Note that depending on how the respondent answered this question, they received different survey questions. If a respondent indicated they were a *Researcher*, they did not receive the more development-oriented questions. We report the number of respondents for each question along with the results below. For the remainder of this analysis, we use these subsets to analyze the data.

Organization Type Next, respondents indicated the type of organization for which they worked. The vast majority 86% (898/1048) worked for *Educational Institutions*. That percentage increased to 93% (417/447) for Researcher type respondents

Geographic Location Because the focus of the URSSI project is the United States, we targeted our survey to US-based lists. As a result, the vast majority of responses (990/1038) came from the United States. We received responses from 49 states (missing only Alaska), plus Washington, DC, and Puerto Rico.

Job Title People involved in developing and using research software have various job titles. For our respondents, *Faculty* was the most common, given by 63% (668/1046) of the respondents and 79% (354/447) of the Researcher type respondents. No other title was given by more then 6% of the respondents.

Respondent Age Overall, 77% (801/1035) of the respondents are between 35 and 64 years of age. The percentage is slightly higher for Researcher type respondents (370/441 - 84%) and slightly lower for Combination type respondents (378/514 - 74%).

Respondent Experience The respondent pool is highly experienced overall, with 77% (797/1040) working in research for more than 10 years and 39% (409/1040) for more than 20 years. For the Researcher type respondents, those numbers increase to 84% (373/444) with more than 10 years and 44% (197/444) with more than 20 years.

Gender In terms of self-reported gender, 70% (732/1039) were Male, 26% (268/1039) were Female, with the remainder reporting Other or Prefer not to say. For Researcher respondents, the percentage of Females is higher (151/443 - 34%)

Discipline The survey provided a set of choices for the respondents to choose their discipline(s). Respondents could choose more than one discipline. Table 2 shows the distribution of respondents by discipline. Though the respondents represent a number of research disciplines, our use of NSF and NIH mailing lists likely skewed the results towards participants from science and engineering fields.

Discipline	Total	Researchers	Developers	Combination
Biological sciences	325	149	20	156
Mathematical and physical sciences	480	174	36	269
Engineering	207	74	18	115
Computer & information science	268	61	33	174
Medicine	105	49	5	51
Dentistry and health	17	8	1	8
Social sciences	98	55	2	41
Humanities and language based studies	24	4	1	18
Administrative & business studies	12	10 0	2	
Agricultural	23	10	0	13
Forestry and veterinary science	11	5	1	5
Education	61	29	2	30
Architecture and planning	7	1	1	5
Design	12	4	1	7
Creative & performing arts	7	0	0	7

Table 2. Disciplines of Respondents

Software Engineering Practices

This section focuses on answering *RQ1: What activities do research software developers spend their time on, and how does this impact the perceived quality and long-term accessibility of research software?* based on relevant survey questions.

Where Respondents Spend Software Time We asked respondents what percentage of their time they currently spend on a number of software activities and what percentage of time they would ideally like to spend on those activities. As Figure 1 shows, there is a mismatch between these two distributions. Overall, respondents would like to spend more time in *design* and *coding* and less time in *testing* and *debugging*.

Which aspects of the software development process are more difficult than they should be? Figure 2 illustrates the results for this question. Interestingly, the aspects most commonly reported are those that are more related to people issues rather than to technical issues (e.g. *finding personnel/turnover*, *use of best practices*, *project management*, and *keeping up with modern tools*). The only one that is technical is *testing*.

Quantify the mismatch, which two distributions?
The mismatch seem small.
Design isn't shown in the Figure
agree the difference b/w ideal and actual doesn't appear very dramatic.
The explanation could be that the technical aspects are difficult, but the respondents expected them to be difficult

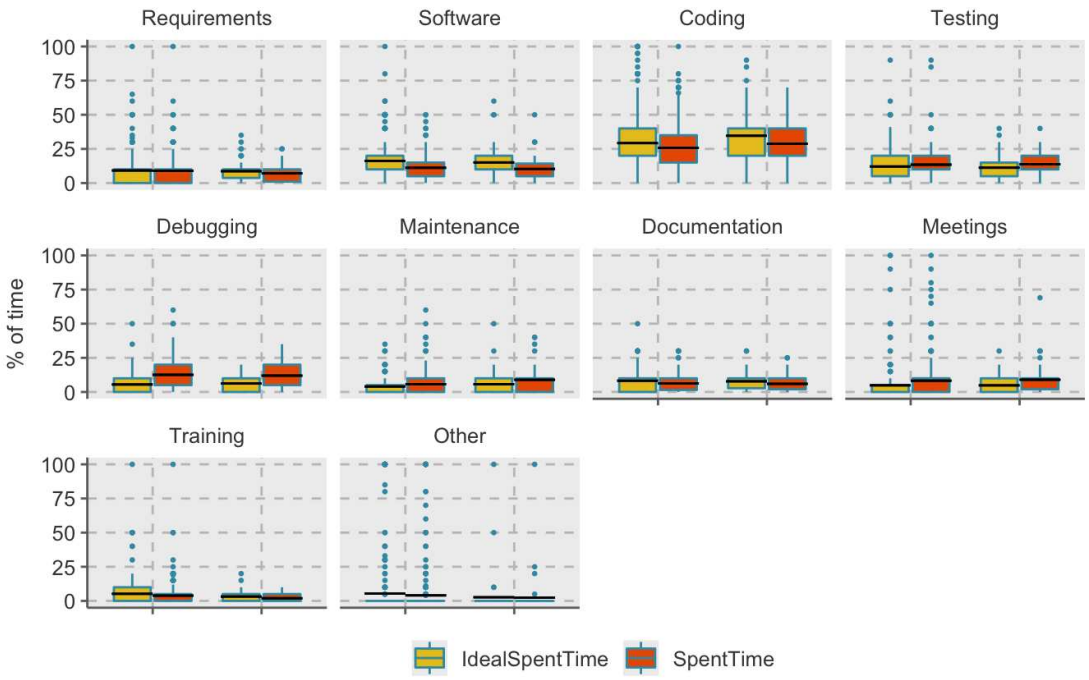

Figure 1. Where Respondents Spend Software Time

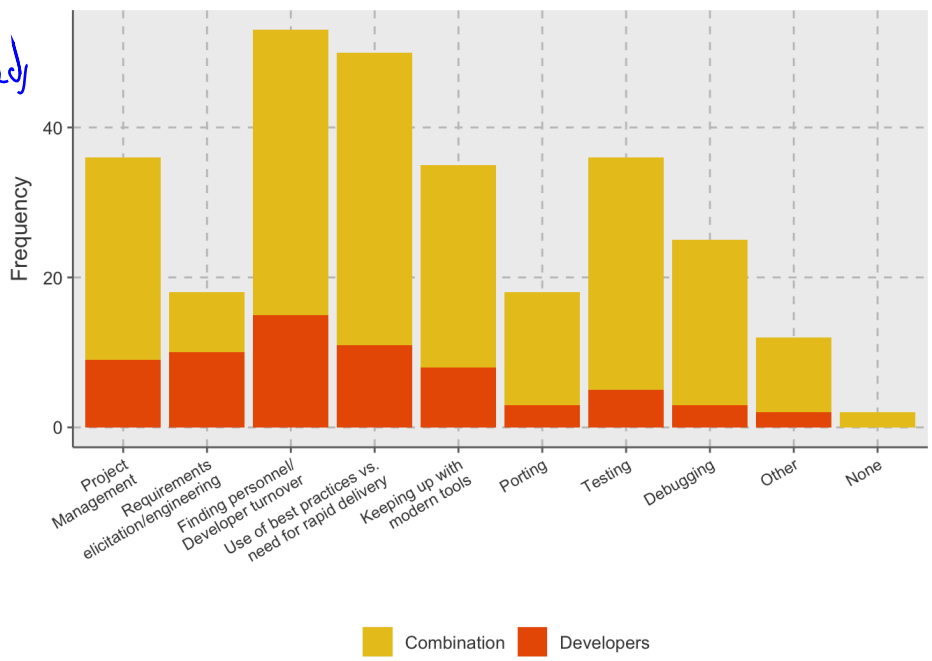

Figure 2. Aspects of software development that are more difficult than they should be

Use of Testing Focusing on the one technical aspect that respondents perceived to be more difficult than it should be, we asked the respondents how frequently they employ various types of testing, including: Unit, Integration, System, User, and Regression. The respondents could choose from *frequently*, *somewhat*,

rarely, and *never*. Figure 3 shows the results from this question. The only type of testing more than 50% of respondents used frequently was **Unit testing** (231/453 – 51%). On the other extreme only about 25% reported using **System** (118/441 – 27%) or **Regression** testing (106/440 – 24%) frequently.

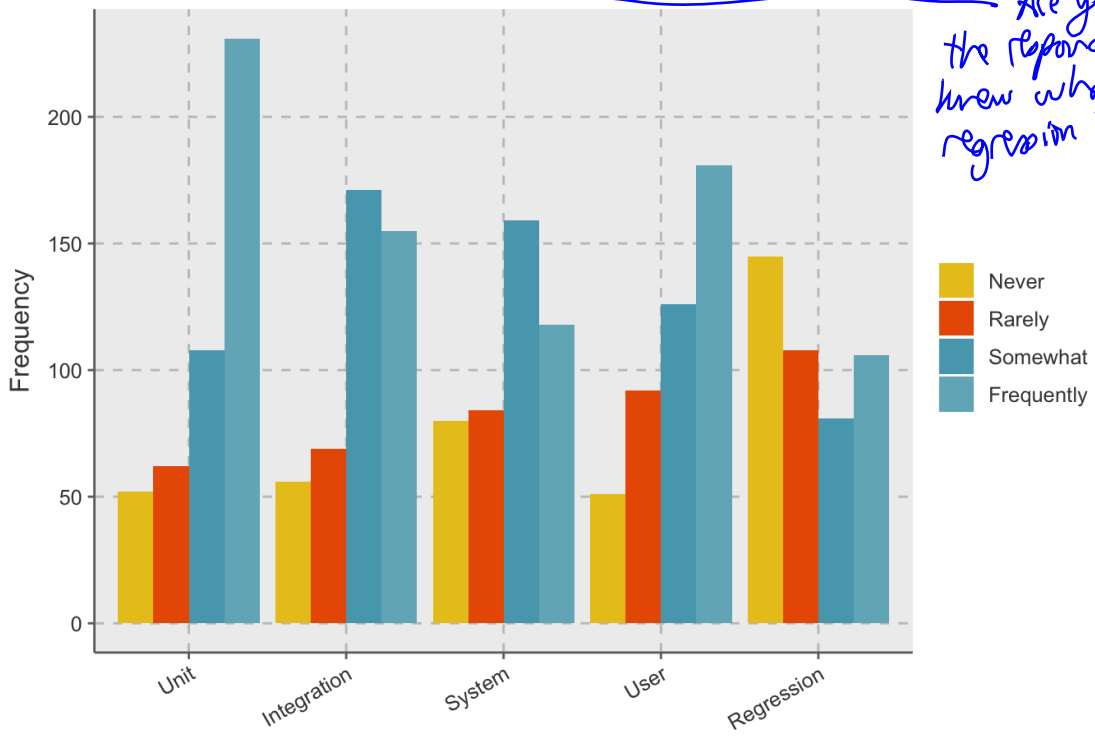

Figure 3. Use of Testing

Use of Open-Source Licensing Overall 74% (349/470) of the respondents indicated they used an open-source license. This percentage was consistent across both combination and developer respondents. However, this result still leaves 26% of respondents who do not release their code under an open-source license.

Frequency of using *best* practices As a follow-up question, we asked the respondents how frequently they used a number of standard software engineering practices. The response options were *Never*, *Sometimes*, *Half of the time*, *Most of the time*, *Always*. The following list reports those who responded *Most of the time* or *Always* for the most commonly used practices (in decreasing order):

- Continuous Integration – 54% (54/100)
- Use of coding standards – 54% (54/100)
- Architecture or Design - 51% (52/101)
- Requirements – 43% (43/101)
- Peer code review – 34% (34/99)

Documentation In terms of what information respondents document, only 55% (56/101) develop **User manuals or online help** either *Most of the time* or *Always*. However, 86% (87/101) **Comment code** and 95% (96/101) **Use descriptive variable/method names** either *Most of the time* or *Always*. Interestingly, even though a very large percentage of respondents indicated that they comment their code, when we look in more detail at the types of information documented, we see a different story. The following list reports those who responded *Most of the time* or *Always* for each type of documentation:

- Requirements – 49% (49/100)
- Software architecture or design – 34% (34/100)

- Test plans or goals – 25% (25/100)
- User stories/use cases – 24% (24/100)

Tools

This section focuses on answering *RQ2: What tools do research software developers use and what additional tools are needed to support sustainable development practices?* based on relevant survey questions.

Tools support for development activities The results in Figure 4 indicate that a large majority of respondents (340/441 – 77%) believe **Coding** is *Extremely supported* or *Very supported* by existing tools. Slightly less than half of the respondents find **Testing** (196/441 - 44%) and **Debugging** (188/441 – 43%) to be *Extremely supported* or *Well supported*. Less than 30% of the respondents reported **Requirements**, **Architecture/design**, **Maintenance**, and **Documentation** as being well-supported. Because coding is the only practice where more than half of the respondents indicate *Extremely supported* or *Very supported*, these responses indicate a clear opportunity for additional (or better) tool support in a number of areas.

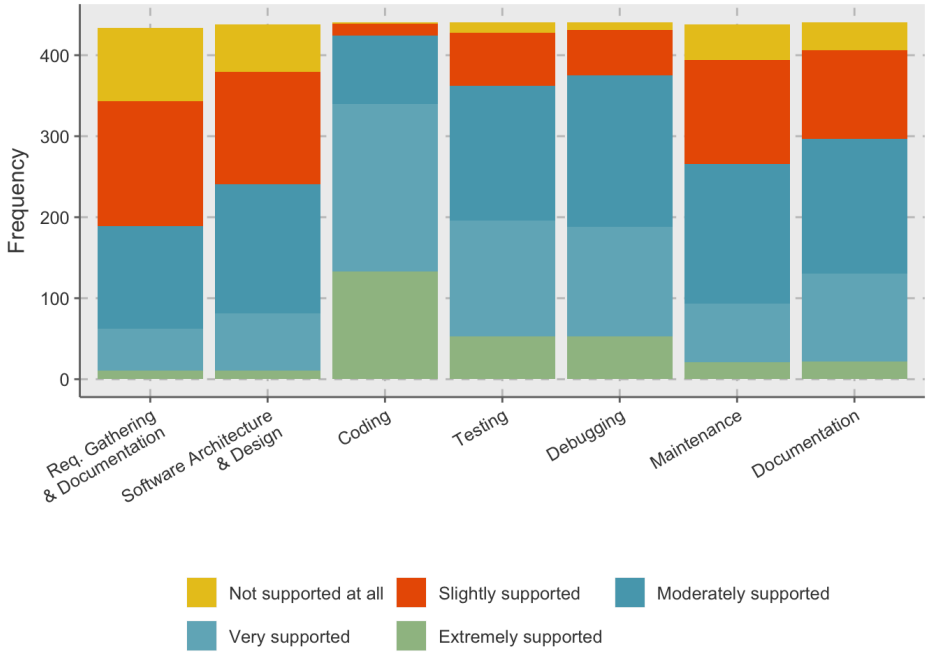

Figure 4. Availability of tool support

Version control and continuous integration In a follow-up question, almost all of those who responded (83/87) indicated they do use version control. In addition, a slightly lower but still very large percentage of respondents (74/83) indicated they used Git either *Always* or *Most of the time*. Git was by far the most commonly used version control system. Finally, almost all respondents (76/83) check their code into the version control system either *after every change* or *after a small group of changes*. However, when we investigate further about the version control practices, we find 29% of the Combination type respondents (26/56) indicated they use *copying files to another location* and 10% (6/56) used *zip file backups* as their method of version control either *Always* or *Most of the time*. The lack of use of standard version control methods is an area where additional training could help.

Regarding continuous integration, less than half of the respondents (39/84) indicated they used it either *Always* or *Most of the time*. This result suggests another area where additional training could help.

Training

This section focuses on answering RQ3: *What training is available to research software developers and does this training meet their needs?* based on relevant survey questions.

Have you received training? The percentage of respondents answering *yes* depends upon the type of respondent: Developers - 64% (39/61), Combination - 44% (170/384), and Researchers - 22% (93/421). When we examine the responses to this question by gender we also see a difference: 30% (69/229) of the Female respondents received training compared with 37% (220/601) of the Male respondents.

Where have you received training? Of those who reported receiving training, 82% had received this training from a *Class/School* and 48% received it *Online/Self-directed*. [Note that respondents could report more than one source of training.] Interestingly, only 10% of the respondents had *Software Carpentry* training. When we examine these responses by gender, we find approximately the same percentage of Male and Female responds received training in a *Class/School* or *Online/Self-directed*. However, the percentage of Female respondents who reported *Software Carpentry* or *Other* training was lower than for Male respondents.

Do the respondents know what Software carpentry means?

Are there sufficient opportunities for training? When we turn to the availability of relevant training, an interesting picture emerges. As Figure 5 shows, slightly more half half of the respondents indicate there is sufficient training for obtaining new software skills. However, when looking at the response based upon gender, there is a difference with 56% of male respondents answering positively but only 43% of the female respondents. But, as Figure 6 indicates, approximately 75% of the respondents indicated they do not have sufficient time to take advantage of these opportunities. These results are slightly higher for female respondents (79%) compared with male respondents (73%). So, while training may be available, respondents do not have adequate time to take advantage of it.

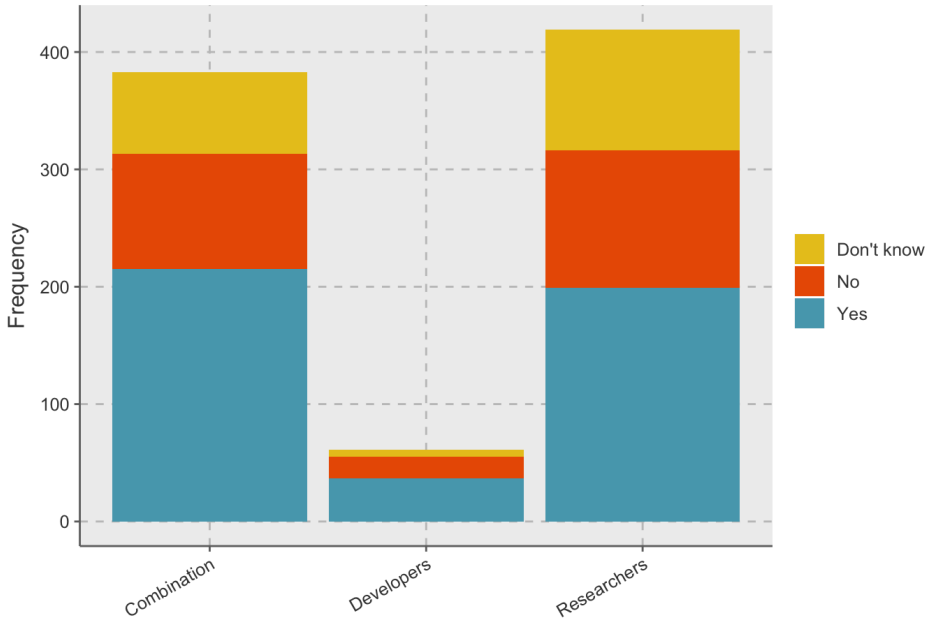

Figure 5. Sufficient opportunities for training

Preferred modes for delivery of training The results showed that there is not a dominant approach preferred for training. Carpentries, Workshops, MOOCs, and On-site custom training all had approximately the same preference across all three topic ares (Development Techniques, Languages, and Project Management). This result suggests that there is benefit to developing different modes of training about important topics, because different people prefer to learn in different ways.

Funding

This section uses the relevant survey questions to answer the two research questions related to funding:

- *RQ4a: What is the available institutional support for research software development?*
- *RQ4b: What sources of institutional funding are available to research software developers?*

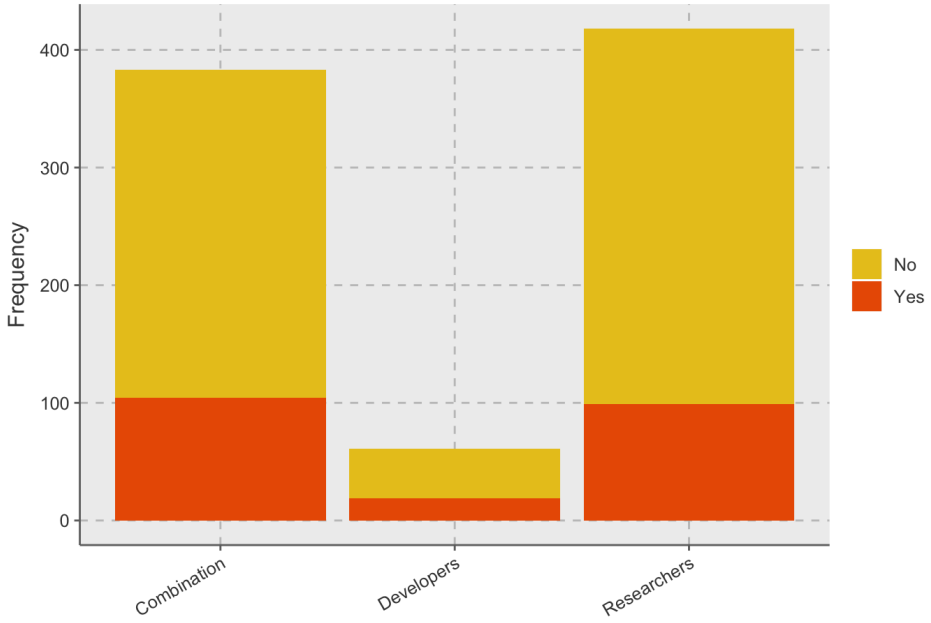

Figure 6. Sufficient time for training

First, 54% (450/834) of the respondents reported they have included funding for software in their proposals. However, that percentage drops to 30% (124/408) for respondents who identify primarily as Researchers.

When looking at the specific types of costs respondents include in their proposals, 48% (342/710) include costs for *developing new software*, 22% (159/710) for *reusing existing software* and 29% (209/710) for *maintaining/sustaining software*. [Note that respondents could provide more than one response.] It is somewhat surprising to see such a large number of respondents who include funding for maintaining and sustaining software.

In examining the source of funding for the projects represented by the survey respondents, the largest funder is NSF, at 36%. But, as Figure 7 shows, a significant portion of funding comes from the researchers' own institutions. While other funding agencies provide funding for the represented projects and may be very important for individual respondents, overall, they have little impact. This result could have been impacted by the fact that we used a mailing list of NSF projects as one means of distributing the survey. However, we also used a list of NIH PIs who led projects funded at least at $200K, so it is interesting that NSF is still the largest source.

In terms of the necessary support, Figure 8 indicates that, while institutions do provide some RSE, financial, and infrastructure support, it is inadequate to meet the respondents' needs, overall. In addition, when asked in a follow-up question whether the respondents have sufficient funding to support software development activities for their research the overwhelming answer is no (Figure 9).

When asked about whether current funding adequately supports some key phases of the software lifecycle, the results were mixed. Respondents answered on a scale of 1-5 from *insufficient* to *sufficient*. For *Developing new software* and *Modifying or reusing existing software* there is an relatively uniform distribution of responses across the five answer choices. However, for *Maintaining software*, the responses skew towards the *insufficient* end of the scale.

For respondents who develop new software, we asked (on a 5-point scale) whether their funding supports various important activities, including *refactoring*, *responding to bugs*, *testing*, *developing new features*, and *documentation*. In all cases less than 35% of the respondents answered 4 or 5 (*sufficient*) on the scale.

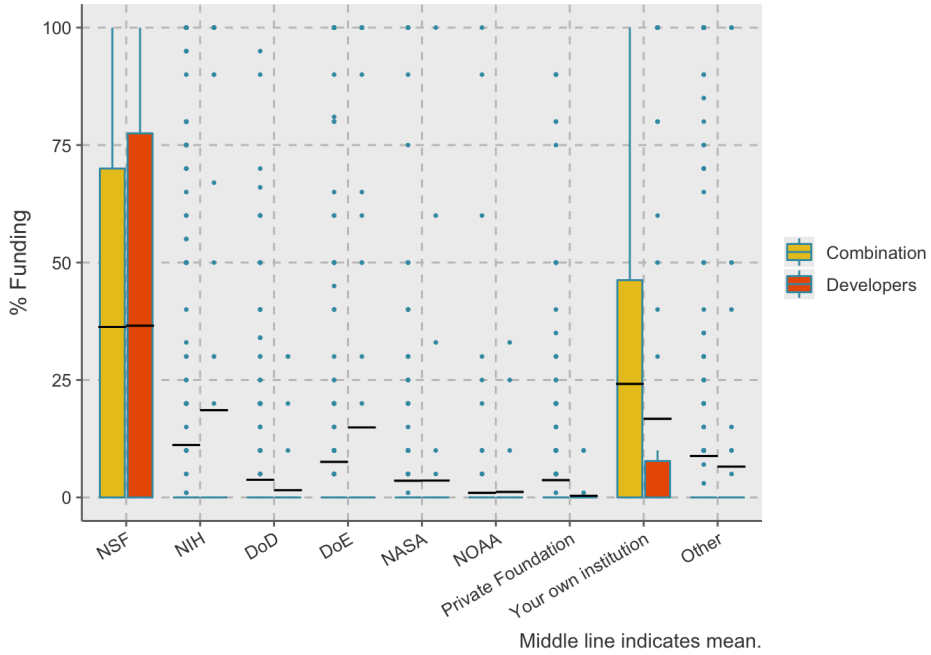

Figure 7. Sources of funding

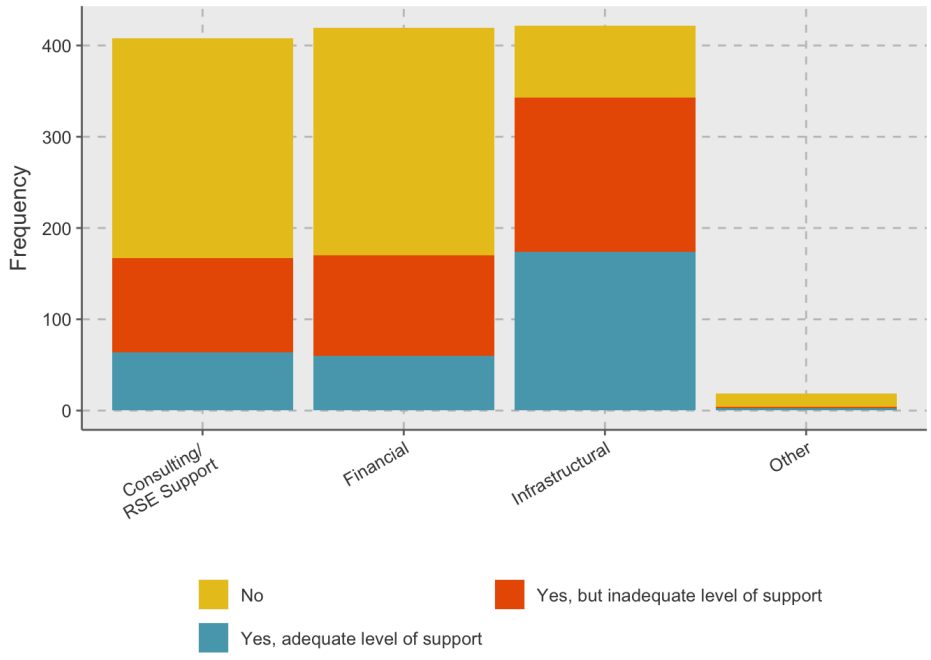

Figure 8. Sufficiency of institutional support

Career Paths

This section focuses on answering *RQ5: What factors impact career advancement and hiring in research software?* based on relevant survey questions.

Institutions have a number of different job titles for people who develop software. The most frequently reported title is *postdoc* (411), with other titles including *Research Software Engineers* (223), *Research Programmers* (251), *Software Developer*(253), and *Programmer* (253). There were also a good number of respondents who were *Faculty* (215) or *Research Faculty* (242). Note that people could provide more

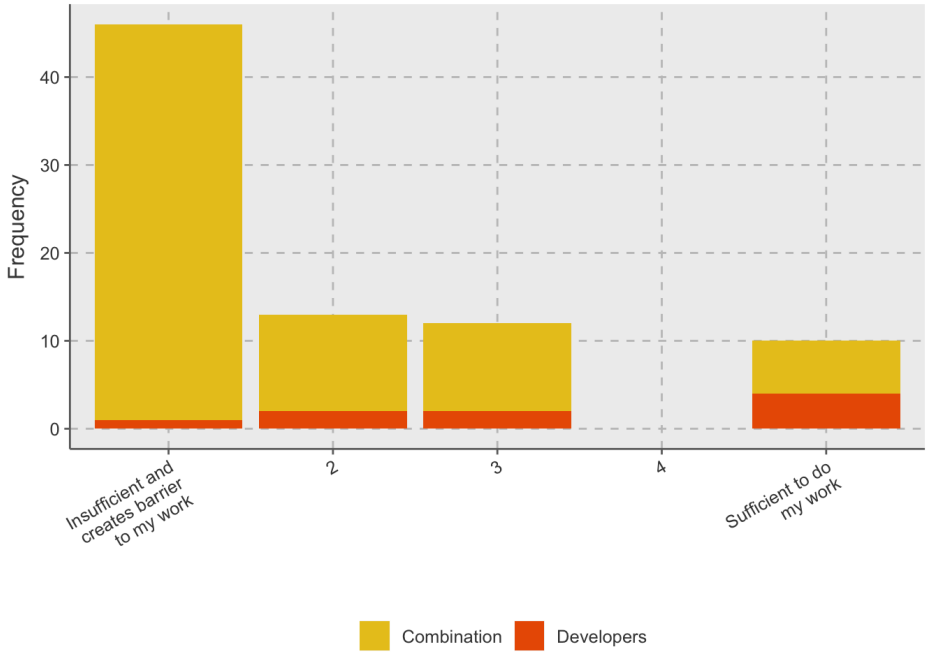

Figure 9. Necessary funding to support software development activities

than one answer, so the total exceeds the number of respondents.

While there are a number of job titles that research software developers can fill, unfortunately, as Figure 10 shows, the respondents saw little chance for career advancement for those whose primary job is software development. Only 21% (153/724) of the *Combination* and *Researcher* respondents saw opportunity for advancement. When we look at the result by gender, only 16% (32/202) of the female respondents see an opportunity for advancement compared with 24% (139/548) for the male respondents.

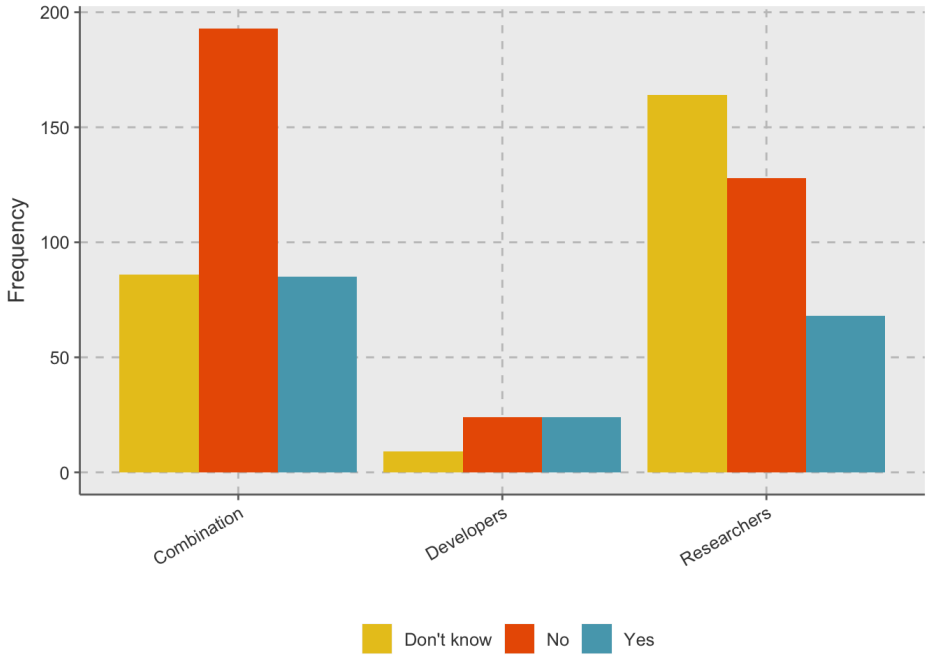

Figure 10. Opportunities for career advancement for software developers

We asked respondents who have the responsibility for hiring people into software development roles to indicate the importance of the following concerns:

- Identifying a pipeline for future staff
- Attracting staff from underrepresented groups
- Ability of staff to work across disciplines
- Competing with industry for high performers
- Offering a viable career path
- Opportunities to outsource skilled work

As Figure 11 shows, most factors were at least moderately important, with the exception of opportunities to outsource work.

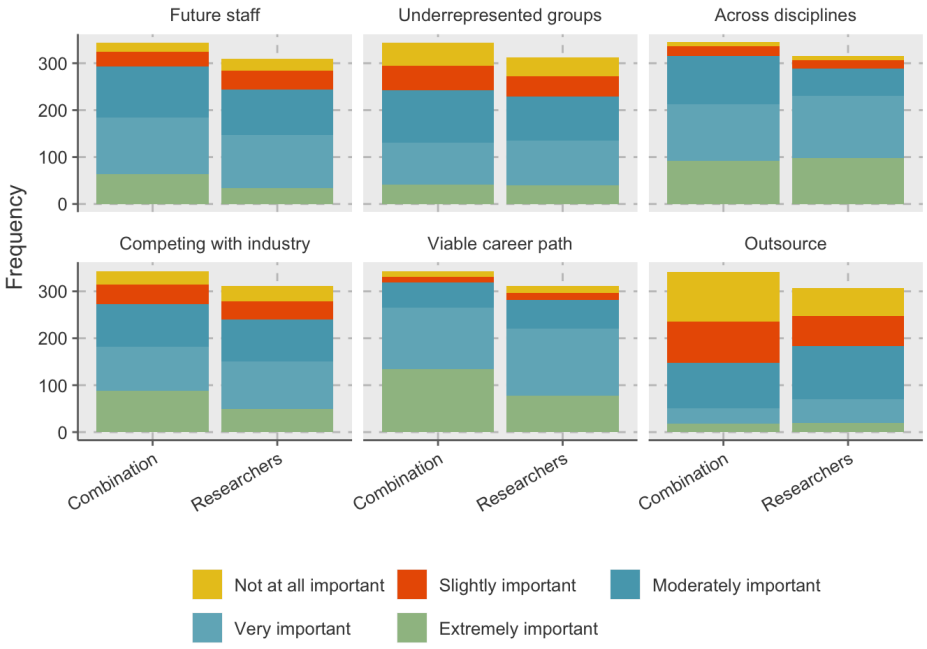

Figure 11. Concerns when trying to hire software development staff

When examining the perceptions of those that have been hired into a software development role, we asked a similar question. We asked respondents the importance of the following concerns when they were hired into their current role:

- Diversity in the organization
- Your experience as a programmer or software engineer
- Your background in science
- Your knowledge of programming languages
- Your knowledge and capabilities across disciplines
- Your potential for growth

As Figure 12 shows, for job seekers, their background in science, ability to work across disciplines, and opportunities for growth were the most important factors. Interestingly, the respondents saw less importance in their experience as a programmer or software engineering or their knowledge of programming languages.

Besides these factors, we asked a follow-up question about the importance of the following factors when deciding on whether to accept a new job or a promotion:

- Title of the position
- Salary raise
- Responsibilities for a project or part of a project

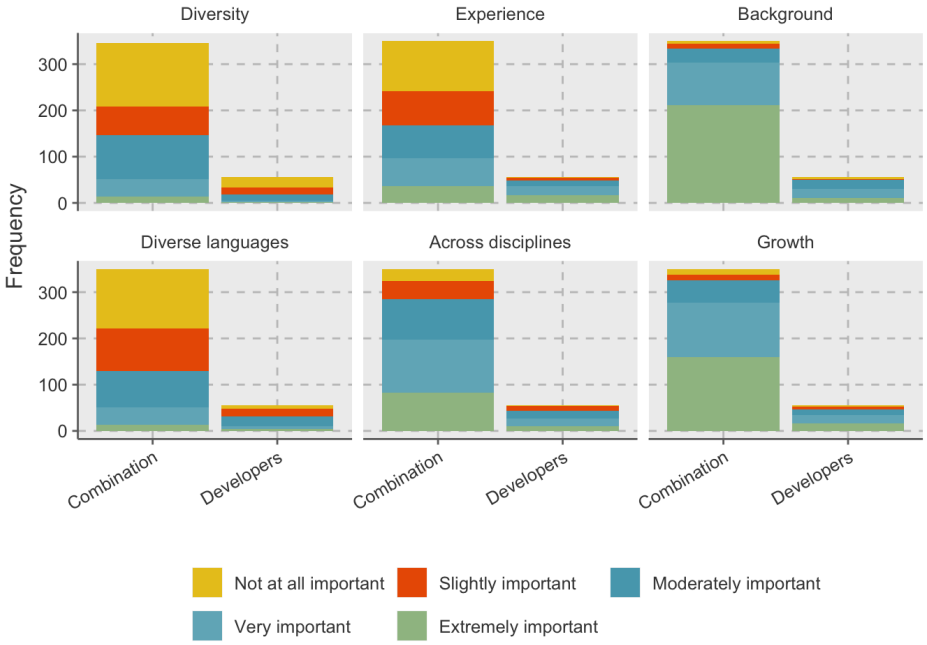

Figure 12. Concerns when being hired as software development staff

- Leading a team
- Available resources such as travel money

As Figure 13 shows *Salary*, *Responsibility*, and *Leadership* are the most important factors.

Figure 13. Importance of factors when taking a job or promotion

Lastly, in terms of recognition within their organization, we asked respondents to indicate whether other people in their organization use their software and whether other people in their organization have contacted them about developing software. Almost everyone (54/61) that responded to these follow-up

questions indicated that other people in the organization use their software. In addition just over half of the Combination respondents (27/51) and 72% (8/11) of the Developer respondents indicated that people in their organization had contacted them about writing software for them.

Credit

This section uses the relevant survey questions to answer the two research questions related to credit:

- *RQ6a: What do research software projects require for crediting or attributing software use?*
- *RQ6b: How are individuals and groups given institutional credit for developing research software?*

When asked how respondents credit software they use in their research, as Figure 14 shows, the most common approaches are either to *cite a paper about the software* or to *mention the software by name*. Interestingly, authors tended to *cite the software archive itself*, *mention the software URL*, or *cite the software URL* much less frequently. Unfortunately, this practice leads to fewer trackable citations of the software, making it more difficult to judge its impact.

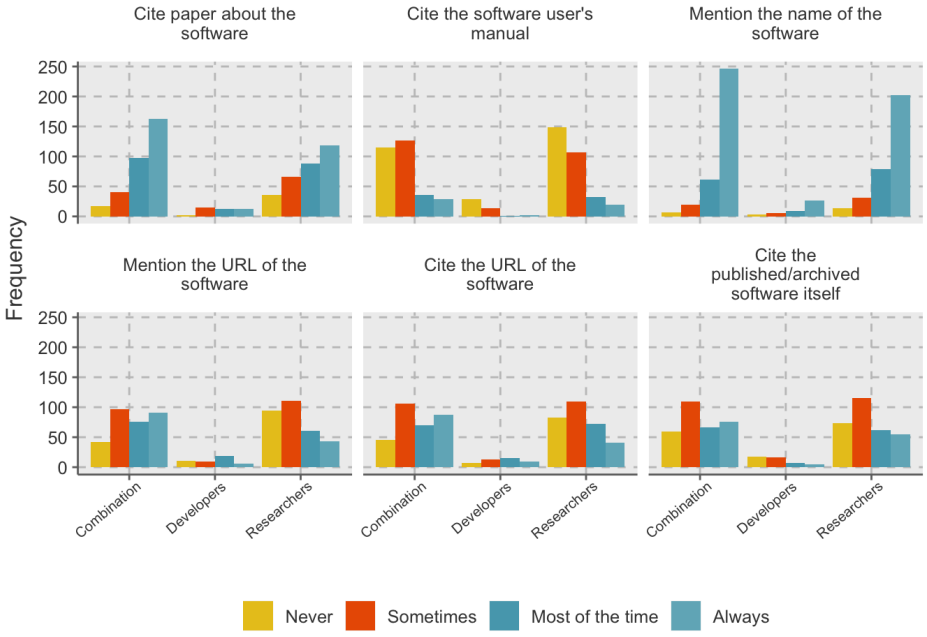

Figure 14. How authors credit software used in their research

Following on this trend of software work not being properly credited, when asked how they currently receive credit for their own software contributions, as Figure 15 shows, none of the standard practices appear to be used very often.

An additional topic related to credit is whether respondent's contributions are valued for performance reviews or promotion within their organization. As Figure 16 shows, approximately half of the respondents indicate that software contributions are considered. Another large percentage say that it *depends*.

While it is encouraging that a relatively large percentage of respondents' institutions consider software during performance reviews and promotion some or all of the time, the importance of those contributions is still rather low, especially for respondents who identify as *Researchers*, as shown in Figure 17.

Diversity

This section focuses on answering *RQ7: How do current Research Software Projects document diversity statements and what support is needed to further diversity initiatives?* based on relevant survey questions.

When asked how well their projects recruit, retain, and include in governance participants from underrepresented groups, only about $1/3$ of the respondents thought they did an "Excellent" or "Good" job. Interestingly, when asked how well they promote a culture of inclusion, 68% of the respondents (390/572) indicated they did an "Excellent" or "Good" job. These two responses seem to be at odds with

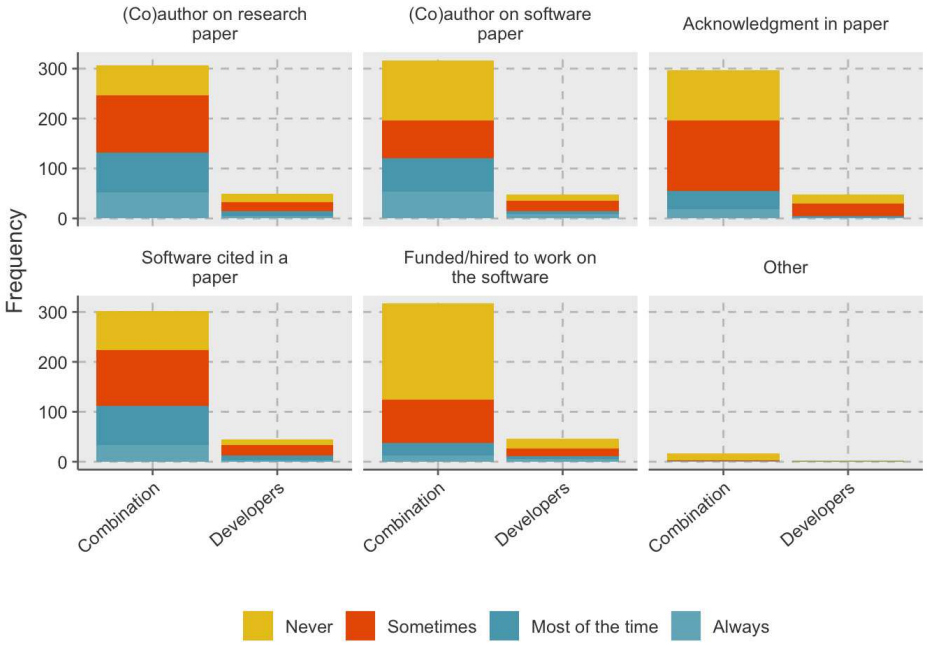

Figure 15. How respondents current receive credit for their software contributions

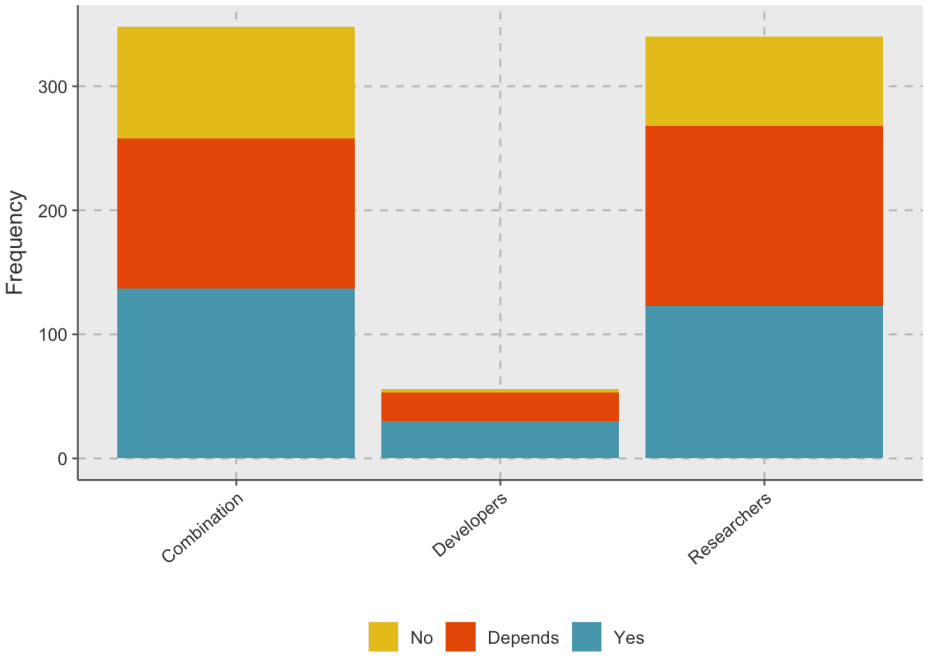

Figure 16. Does institution consider software contributions in performance reviews or promotion casess?

each other, suggesting that perhaps projects are not doing as well as they think they are. Conversely, it could be that projects do not do a good job of recruiting diverse participants, but do a good job of supporting the ones they do recruit. Figure 18 shows the details of these responses.

We asked follow-up questions about whether the respondents' projects have a diversity/inclusion statement or have a code of conduct. As Figure 19 and 20 show, most projects do not have either of these, nor do they plan to develop one. This answer again seems at odds with the previous answer that most

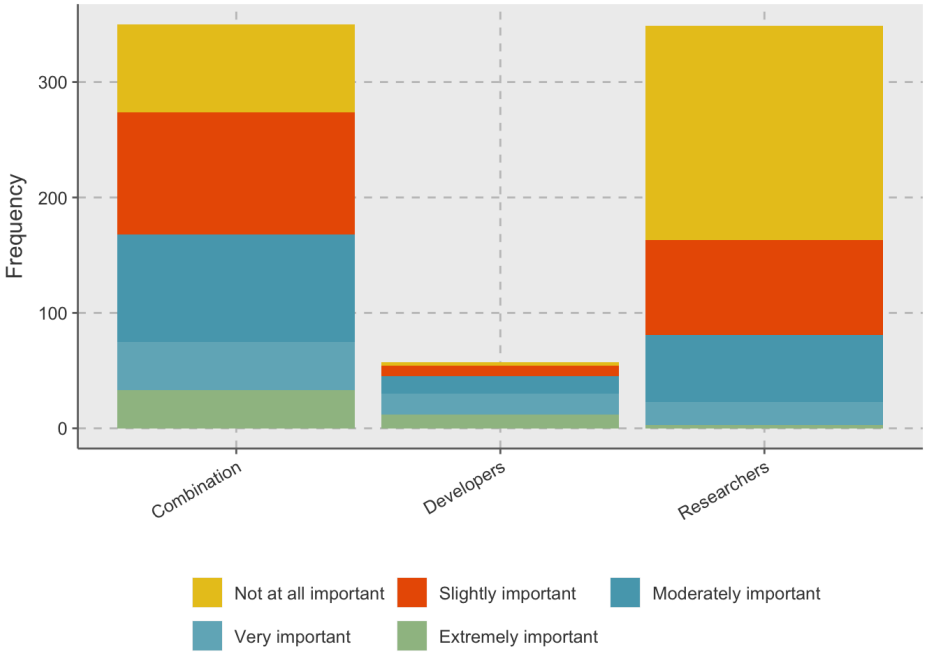

Figure 17. Importance of software contributions during performance review or promotion

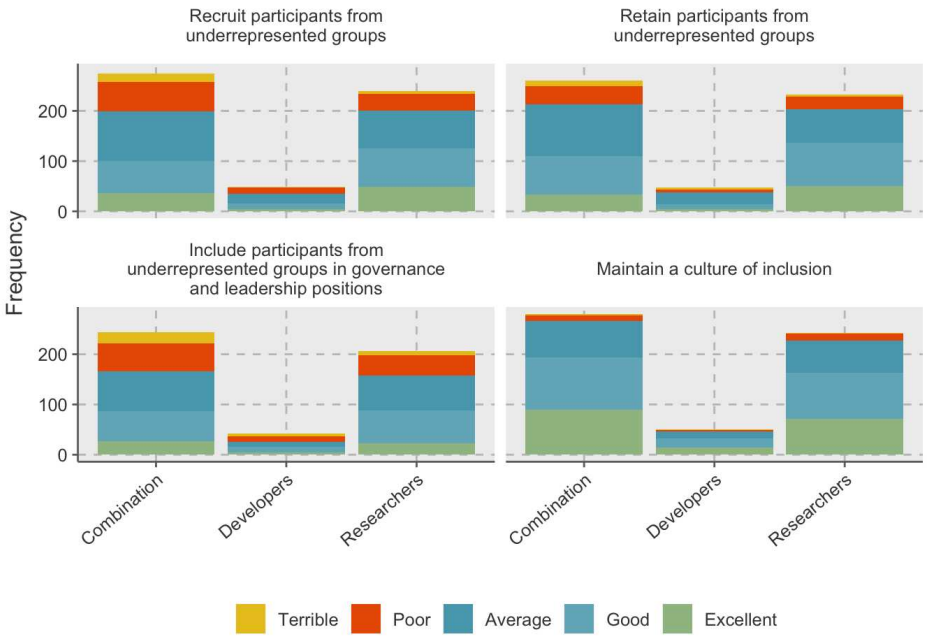

Figure 18. How respondents think their project does with inclusion

people thought their projects fostered a culture of inclusions.

Lastly, we asked respondents to indicate the aspects of diversity or inclusion for which they could use help. As Figure 21 shows, respondents indicated they needed the most help with recruiting, retaining, and promoting diverse participants. They also need help with developing diversity/inclusion statements and codes of conduct.

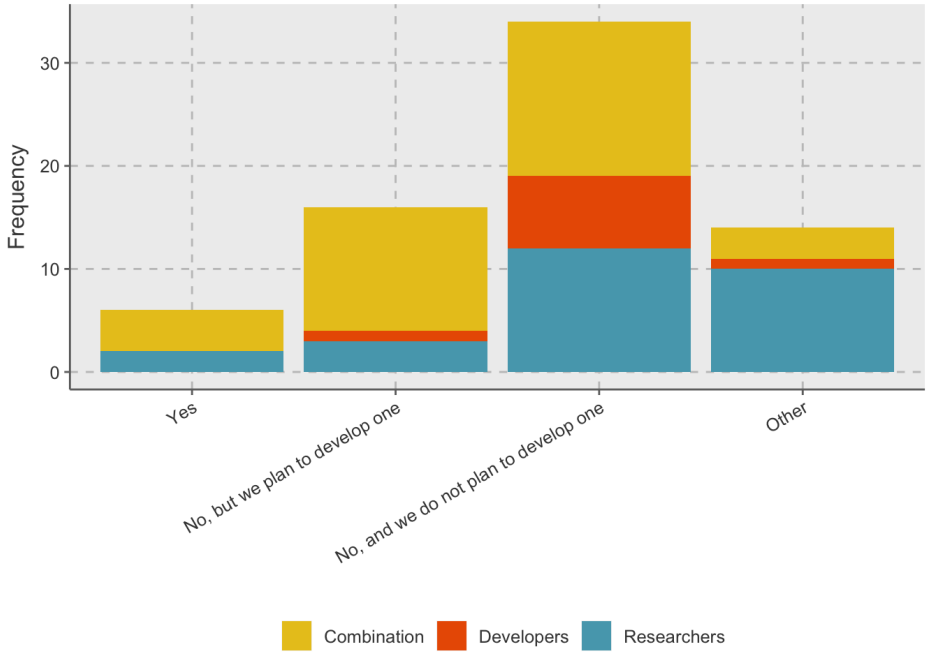

Figure 19. Whether the respondents' projects have a diversity/inclusion statement

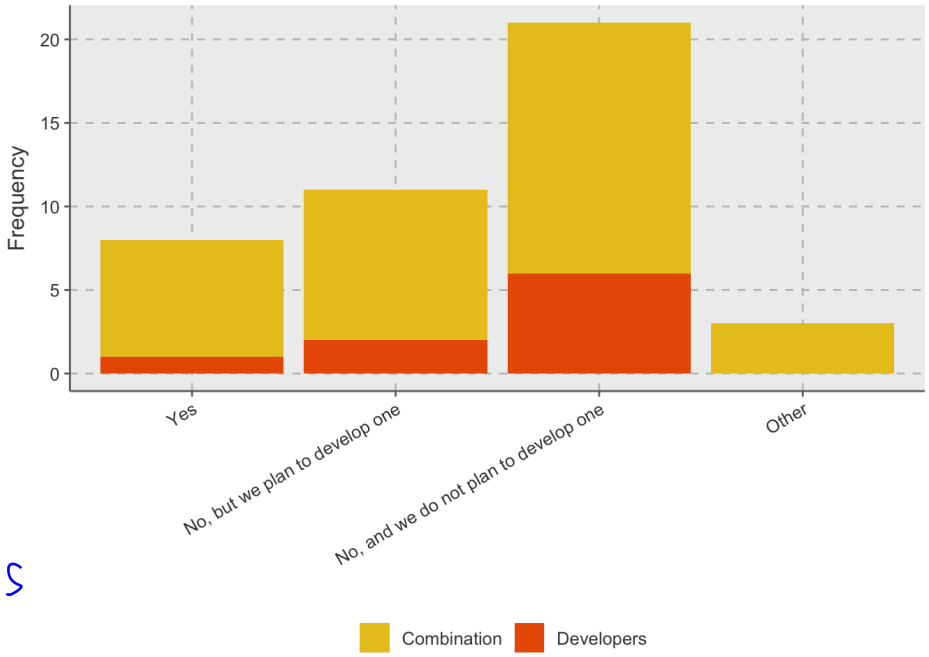

Figure 20. Whether the respondents' projects have a code of conduct

Threat to validity? Future work?

Conclusions

DISCUSSION

In this section, we discuss our results within the context of the findings from the previous surveys described in the Background section.

It is interesting to observe that the most difficult aspects of the software development process focused mostly on topics related to people, rather than on more technical topics, with the exception of testing and keeping up with tools. This result suggests that there is a need for training on the social and human

the question didn't ask most difficult - it asked more difficult than expected.

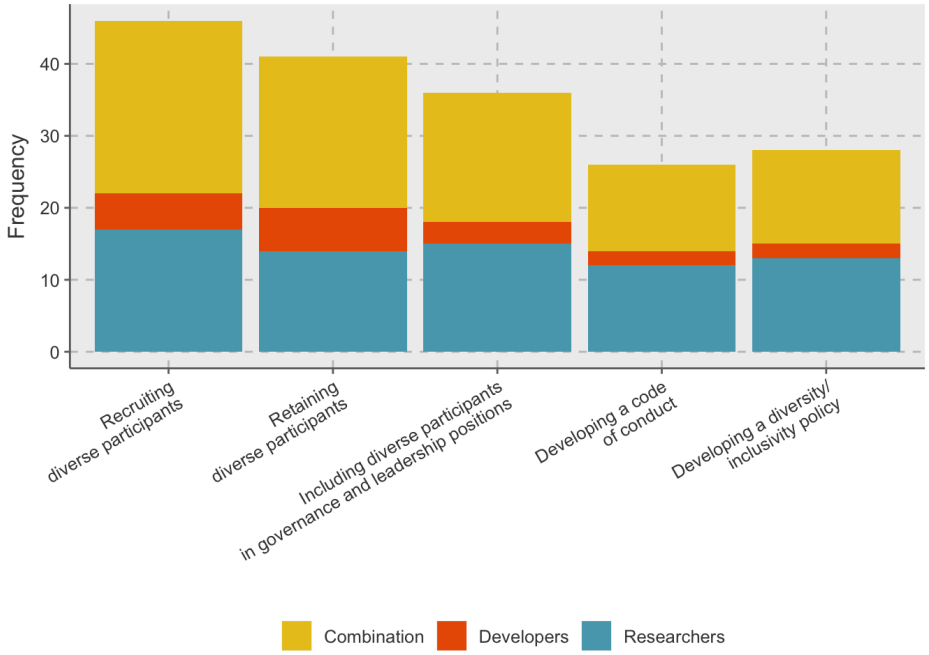

Figure 21. Type of support needed for inclusion and diversity

aspects of software development rather than only on the technical aspects.

Regarding software testing, previous surveys reported mixed results, with some finding that software testing was important and highly used while others showed that scientists did not test their code. Our results found testing to be one of the more difficult aspects of the software process. Our results also showed fairly high use of unit testing, but much lower for other types of testing.

One more observation about the software development process relates to where people spend their time. For most activities, the respondents said they were spending about the same amount of time as they would ideally like to spend. However, one notable exception regards *debugging*, where people reported spending more time than they would have liked. Our study did not focus on the reason(s) for this, but it could be a combination both of the availability of tools and the lack of training on how to effectively debug complex software code.

Lastly, overall there is a low level of use for commonly accepted best practices from software engineering. Of the practices we included in our survey (Continuous Integration, Coding Standards, Architecture/Design, Requirements, and Peer Code Review), none were used by more than 54% of the respondents. This result indicates a need for additional work to gather information about how these practices can and are being using in the development research software and disseminate that information to the appropriate communities. There may also be a need for additional training in these areas.

Regarding the use of software tools, previous surveys reported few results. Our results showed that respondents found only the *coding* activity to be very well supported by tools. This result suggests the need for additional tools to support important activities like Requirements, Design, Testing, Debugging, and Maintenance. The availability of useful tools that can fit into developers current workflows can increase the use of these key practices for software quality and sustainability.

One of the key concepts that arose in many of the items highlighted above is the need for training. Previous surveys found that less than half (and sometimes much less) of developers reported they had formal training in software development. Our results are in line, with Developers reporting a higher percentage. In either case, approximately half of the people developing research software have received no formal training. Consistent with this number, only about half of the respondents to our survey reported that there were sufficient opportunities for training. However, approximately 3/4 indicated they did not have sufficient time for training that was available. Together these results suggest two conclusions: (1) there is a need for more training opportunities (as described above) and (2) developers of research software need more time for training, either by prioritizing in their own schedule or by being given it from

their employers.

Next we turn to the question of funding. As we discussed in the Background section, historically there has not been sufficient funding dedicated to the development and maintenance of research software. The results from our survey support this trend, with just under half of the respondents including costs for developing software into their proposals, even less for reusing or maintaining software. From the perspective of software sustainability, these results are troubling. Without support for maintaining and sustaining research software, at least some of the initial investments made in software are lost over time. As a result of this funding gap, institutions may need to take on more of the responsibility for funding the maintenance and sustainability of software projects.

Related to the idea of funding is the types of career paths available for people who focus their work on developing research software. As identified in the Background section, many organizations do not adequately recognize or value contributions to research software when it comes to evaluation and promotion. The results of our survey echo this sentiment, with a very low percentage reporting that they have had opportunities for career advancement in their roles. We see changes in institutional policies as the primary remedy to this situation. We call on Universities, National Laboratories, and other organizations to revisit their evaluation and promotion criteria to include measures that capture the contributions to research that are made by developers of research software.

Closely related to the idea of career paths for research software developers is the concept of credit. As described in the Background section, the idea of obtaining credit for software work has just begun receiving recognition recently. However, that recognition is still not at the level that most in this field would like. The results of our survey are consistent with this trend showing that none of the standard methods for giving credit to software are used very frequently. There has been recent activity to help in this area, including work in software citation aimed at changing publication practices (Katz et al., 2021), in software repositories (Smith, 2021), and a proposed definition for FAIR software to add software to funder requirements for FAIR research outputs (Chue Hong et al., 2021).

Finally, in regards to diversity, there has been a problem in this area for many years, as highlighted in the Background work. Recently Codes of Conduct and other efforts have tried to improve the situation. However, as the results from our survey show, problems still exist. Our results showed that about $2/3$ of the respondents thought their organizations promoted a "culture of inclusion." Conversely, only about $1/3$ indicated that their organization did a good job of recruiting and including diverse participants. Unfortunately, the majority of respondents also indicated that their projects do not have a diversity/inclusion statement or a code of conduct, and they have no plans to create them. Clearly, we still have a long way to go in this area. Fortunately, there are a number of groups interested in this topic and working to improve the situation.

ACKNOWLEDGMENTS

We acknowledge support from NSF-1743188 "SI2-S2I2 Conceptualization: Conceptualizing a US Research Software Sustainability Institute (URSSI)." We also thank the survey participants for their time.

REFERENCES

Albert, T. and Wager, E. (2013). How to handle authorship disputes: a guide for new researchers. *The COPE Report*, pages 32–34.

Allen, L., Scott, J., Brand, A., Hlava, M., and Altman, M. (2014). Publishing: Credit where credit is due. *Nature*, 508(7496):312–313.

AlNoamany, Y. and Borghi, J. A. (2018). Towards computational reproducibility: researcher perspectives on the use and sharing of software. *PeerJ Computer Science*.

Anzt, H., Bach, F., Druskat, S., Löffler, F., Loewe, A., Renard, B. Y., Seemann, G., Struck, A., Achhammer, E., Aggarwal, P., et al. (2020). An environment for sustainable research software in Germany and beyond: current state, open challenges, and call for action. *arXiv preprint arXiv:2005.01469*.

Association for Computing Machinery (2018). Criteria for authorship.

Baker, M. (2016). 1,500 scientists lift the lid on reproducibility. *Nature*, 533(7604):452—-454.

Borgman, C., Wallis, J., and Mayernik (2012). Who's got the data? interdependencies in science and technology collaborations. *Computer Supported Cooperative Work*, 21:485–523.

Carver, J., Gesing, S., Katz, D. S., Ram, K., and Weber, N. (2021). Urssi community survey 2018 raw data.

Carver, J., Heaton, D., Hochstein, L., and Bartlett, R. (2013). Self-perceptions about software engineering: A survey of scientists and engineers. *Computing in Science & Engineering*, 15(1):7–11.

CASBS Group on Best Practices in Science (2018). Issues related to authorship, co-authorship, and collaboration. https://bps.stanford.edu/?page_id=8617.

Chue Hong, N. P., Katz, D. S., Barker, M., Lamprecht, A.-L., Martinez, C., Psomopoulos, F. E., Harrow, J., Castro, L. J., Gruenpeter, M., Martinez, P. A., and Honeyman, T. (2021). FAIR principles for research software (FAIR4RS principles). Technical report, Research Data Alliance.

Dance, A. (2012). Authorship: Who's on first? *Nature*, 489:591—593.

Druskat, S. (2020). Citation file format (CFF). https://citation-file-format.github.io.

Foster, I. (2006). 2020 computing: A two-way street to science's future. *Nature*, 440(7083):419.

Goble, C. (2014). Better software, better research. *IEEE Internet Computing*, 18(5):4–8.

Hannay, J. E., MacLeod, C., Singer, J., Langtangen, H. P., Pfahl, D., and Wilson, G. (2009). How do scientists develop and use scientific software? In *2009 ICSE Workshop on Software Engineering for Computational Science and Engineering*, pages 1–8.

Hettrick, S. (2014). It's impossible to conduct research without software, say 7 out of 10 uk researchers.

Hettrick, S. (2018). softwaresaved/software_in_research_survey_2014: Software in research survey.

Hocquet, A. and Wieber, F. (2021). Epistemic issues in computational reproducibility: software as the elephant in the room. *European Journal for Philosophy of Science*, 11(2):1–20.

Howison, J. and Bullard, J. (2016). Software in the scientific literature: Problems with seeing, finding, and using software mentioned in the biology literature. *Journal of the Association for Information Science and Technology*, 67(9):2137–2155.

Howison, J., Deelman, E., McLennan, M. J., Ferreira da Silva, R., and Herbsleb, J. D. (2015). Understanding the scientific software ecosystem and its impact: Current and future measures. *Research Evaluation*, 24(4):454.

Howison, J. and Herbsleb, J. D. (2011). Scientific software production: Incentives and collaboration. In *Proceedings of the ACM 2011 Conference on Computer Supported Cooperative Work*, CSCW '11, page 513–522, New York, NY, USA. Association for Computing Machinery.

Howison, J. and Herbsleb, J. D. (2013). Incentives and integration in scientific software production. In *Proceedings of the 2013 Conference on Computer Supported Cooperative Work*, CSCW '13, page 459–470, New York, NY, USA. Association for Computing Machinery.

Jay, C., Sanyour, R., and Haines, R. (2016). "not everyone can use git": recommendations for scientist-centred software support (and what researchers really think of them). *Journal of Open Research Software*.

Jones, M. B., Boettiger, C., Mayes, A. C., Smith, A., Slaughter, P., Niemeyer, K., Gil, Y., Fenner, M., Nowak, K., Hahnel, M., Coy, L., Allen, A., Crosas, M., Sands, A., Chue Hong, N., Cruse, P., Katz, D. S., and Goble, C. (2017). CodeMeta: an exchange schema for software metadata. https://doi.org/10.5063/schema/codemeta-2.0. Version 2.0.

Joppa, L. N., McInerny, G., Harper, R., Salido, L., Takeda, K., O'Hara, K., Gavaghan, D., and Emmott, S. (2013). Troubling trends in scientific software use. *Science*, 340(6134):814–815.

Katerbow, M., Feulner, G., Bornschein, M., Brembs, B., Erben-Russ, M., Förstner, K., Franke, M., Fritzsch, B., Fuhrmann, J., Goedicke, M., et al. (2018). Recommendations on the development, use and provision of research software. *Zenodo*.

Katz, D., Chue Hong, N., Clark, T., Muench, A., Stall, S., Bouquin, D., Cannon, M., Edmunds, S., Faez, T., Feeney, P., Fenner, M., Friedman, M., Grenier, G., Harrison, M., Heber, J., Leary, A., MacCallum, C., Murray, H., Pastrana, E., Perry, K., Schuster, D., Stockhause, M., and Yeston, J. (2021). Recognizing the value of software: a software citation guide [version 2; peer review: 2 approved]. *F1000Research*, 9(1257).

Katz, D. S. (2014). Transitive credit as a means to address social and technological concerns stemming from citation and attribution of digital products. *Journal of Open Research Software*, 2(1):e20.

Milliken, G., Nguyen, S., and Steeves, V. (2021). A behavioral approach to understanding the git experience. In *Proceedings of the 54th Hawaii International Conference on System Sciences*, page 7239.

Muna, D., Alexander, M., Allen, A., Ashley, R., Asmus, D., Azzollini, R., Bannister, M., Beaton, R.,

Benson, A., Berriman, G. B., et al. (2016). The astropy problem. *arXiv preprint arXiv:1610.03159*.

Nafus, D. (2012). 'patches don't have gender': What is not open in open source software. *New Media & Society*, 14(4):669–683.

Nangia, U. and Katz, D. S. (2017). Track 1 paper: Surveying the U.S. National Postdoctoral Association regarding software use and training in research. In *Workshop on Sustainable Software for Science: Practice and Experiences (WSSSPE5.1)*. figshare.

Nguyen-Hoan, L., Flint, S., and Sankaranarayana, R. (2010). A survey of scientific software development. In *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '10, New York, NY, USA. Association for Computing Machinery.

Ortu, M., Destefanis, G., Counsell, S., Swift, S., Tonelli, R., and Marchesi, M. (2017). How diverse is your team? investigating gender and nationality diversity in github teams. *Journal of Software Engineering Research and Development*, 5(1):1–18.

Paine, D. and Lee, C. P. (2017). "who has plots?": Contextualizing scientific software, practice, and visualizations. *Proc. ACM Hum.-Comput. Interact.*, 1(CSCW).

Philippe, O., Hammitzsch, M., Janosch, S., van der Walt, A., van Werkhoven, B., Hettrick, S., Katz, D. S., Leinweber, K., Gesing, S., Druskat, S., Henwood, S., May, N. R., Lohani, N. P., and Sinha, M. (2019). softwaresaved/international-survey: Public release for 2018 results.

Pinto, G., Wiese, I., and Dias, L. F. (2018). How do scientists develop scientific software? an external replication. In *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 582–591.

Prabhu, P., Jablin, T. B., Raman, A., Zhang, Y., Huang, J., Kim, H., Johnson, N. P., Liu, F., Ghosh, S., Beard, S., Oh, T., Zoufaly, M., Walker, D., and August, D. I. (2011). A survey of the practice of computational science. In *State of the Practice Reports*, SC '11, New York, NY, USA. Association for Computing Machinery.

Riesenberg, D. and Lundberg, G. D. (1990). The Order of Authorship: Who's on First? *JAMA*, 264(14):1857–1857.

Shahin, M., Babar, M. A., and Zhu, L. (2017). Continuous integration, delivery and deployment: a systematic review on approaches, tools, challenges and practices. *IEEE Access*, 5:3909–3943.

Smith, A. (2021). Enhanced support for citations on GitHub.

Smith, A. M., Katz, D. S., Niemeyer, K. E., and FORCE11 Software Citation Principles Working Group (2016). Software citation principles. *PeerJ Computer Science*, 2:e86.

Smith, A. M., Niemeyer, K. E., Katz, D. S., Barba, L. A., Githinji, G., Gymrek, M., Huff, K. D., Madan, C. R., Cabunoc Mayes, A., Moerman, K. M., Prins, P., Ram, K., Rokem, A., Teal, T. K., Valls Guimera, R., and Vanderplas, J. T. (2018). Journal of open source software (JOSS): Design and first-year review. *PeerJ Computer Science*, 4:e147.

Task Group on Data Citation Standards and Practices (2013). Out of cite, out of mind: The current state of practice, policy, and technology for the citation of data. *Data Science Journal*, (12):CIDCR1—CIDCR7.

The OBO Foundry (2020). Contributor role ontology.

Tourani, P., Adams, B., and Serebrenik, A. (2017). Code of conduct in open source projects. In *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 24–33. IEEE.

Vasilescu, B., Posnett, D., Ray, B., van den Brand, M. G., Serebrenik, A., Devanbu, P., and Filkov, V. (2015). Gender and tenure diversity in github teams. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pages 3789–3798.

Weber, M. (2018). The effects of listing authors in alphabetical order: A review of the empirical evidence. *Research Evaluation*, 27(3):238—245.

Wiese, I., Polato, I., and Pinto, G. (2020). Naming the pain in developing scientific software. *IEEE Software*, 37(4):75–82.

Wilson, R. (2013). Encouraging citation of software – introducing CITATION files.