

# Experiences in how RSE community identity leads to improved research software practices

Daniel S. Katz

*NCSA & CS & ECE & iSchool*

*University of Illinois Urbana Champaign*

*Urbana, IL, USA*

ORCID: 0000-0001-5934-7525

**Abstract**—Research software once was a heroic and lonely activity, particularly in research computing and in HPC. But today, research software is a social activity, in the senses that most software depends on other software, and that most software that is intended to be used by more than one person is written by more than one person. These social factors have led to generally accepted practices for code development and maintenance and for interactions around code. This paper examines how these practices form, become accepted, and later change in different contexts. In addition, given that research software engineering (RSEng) and research software engineers (RSEs) are becoming accepted parts of the research software endeavor, it looks at the role of RSEs in creating, adapting, and infusing these practices. It does so by examining aspects around practices at three levels: in communities, projects, and groups. Because RSEs are often the point where new practices become accepted and then disseminated, this paper suggests that tool and practice developers should be working to get RSE champions to adopt their tools and practices, and that people who seek to understand research software practices should be studying RSEs. It also suggests areas for further research to test this idea.

## I. INTRODUCTION

Research software once was a heroic and lonely activity, particularly in research computing and in HPC. An example of this is the code I worked on as a graduate student. My advisor (Allen Taflove at Northwestern University) created his own standalone computational electromagnetics simulation software from scratch in the 1970s [1]. This was a time-marching, finite-difference method, that required boundary conditions to absorb outgoing energy and source terms to introduce energy into the system, either from the outside of the domain being studied (radar scattering) or internal (antenna analysis). It depended on compilers and operating systems (and later used PVM and then MPI for domain decomposition using ghost cells to communication with neighboring regions on parallel computers), but was otherwise complete. When I started work with him (1987), it was one FORTRAN77 file, with one main program with no subroutines or functions. It was transferred from student to student by hand (first by floppy disk, then by FTP, then by email), with him keeping the “master” with accepted changes. Coding practices were

what he said, with some pushback from students that led to some changes over time. Installation meant compiling, typically with a hand-typed command that was also shared from student to student. Internal documentation was limited to very few comments in the code, mostly identifying sections and variables, but there was also a lot of papers, many of which included algorithms, and technical reports, many of which included code snippets and explanation. Testing was mostly high-level replication of analytic results known by theory, though we would also compare with results produced by very different simulation/analysis methods. For detailed testing (debugging), we used a semi-experimental method, inserting numeric probes into the code and examining their results, either as single values vs. time (1D) or ranges of values vs. space (2D).

Today, research software is a social activity, in a couple of different senses. One is that there’s very little software doesn’t depend on other software, and another is that there’s little software is written solely by a single person, particularly for software that is intended to be used by more than one person. These two social factors, the network of software itself and the network of software developers in many overlapping and dependent projects, have led to generally accepted practices for code development and maintenance, and for interactions around code, including for version control, issues, pull requests, documentation, testing, continuous integration, packaging, installation, etc.

This paper examines how these practices form, become accepted, and later change in different contexts. In addition, given that research software engineering (RSEng) and research software engineers (RSEs) [2] are becoming accepted parts of the research software endeavor, it looks at the role of RSEs in creating, adapting, and infusing these practices. It does so by examining aspects around practices at three levels: in communities (§II), projects (§III), and groups (§IV). In each, I use a specific example to illustrate what I believe are more general points. In the conclusions and future work section (§V), the paper suggests methods to test the generality of my examples as well as these overall ideas.

## II. THE RESEARCH SOFTWARE COMMUNITY

As a way of looking at practices in the research software community, I want to examine the Journal of Open Source

Software (JOSS) [3]. We started JOSS to fill a gap: to better recognize software contributions. Specifically, we wanted to find a way to fit software into our current (paper/book-centric) system bibliographic system.

Our solution was based on making it as easy as possible for authors to write software papers, where paper preparation (and submission) for well-documented software should take no more than an hour. These JOSS papers are intended to be a high-level object that enables citation credit to be given to authors of research software, though the actual full object that is reviewed and published by JOSS is both a short paper and a full software package in its repository and environment. JOSS papers explicitly describe software, and while they often describe how the software is used, they are not allowed to include results of using of software; such results should be published in discipline specific journals that focus on new knowledge. We aimed to make accepted software (and a short paper about it) equivalent to an accepted paper in other journals, and based on the number of papers we have published (see Figure 1), we seem to be meeting a community need.

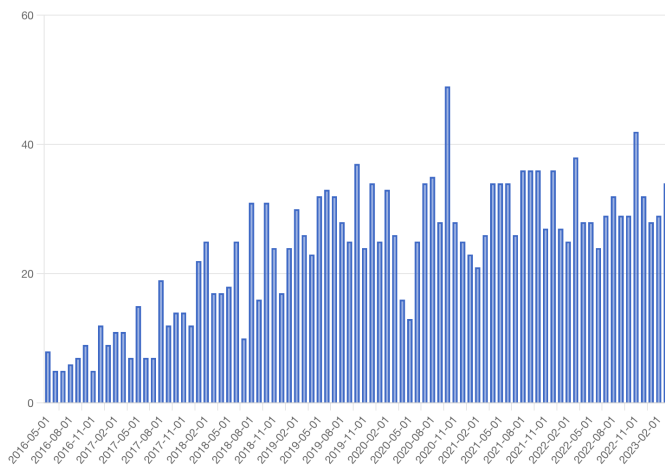


Fig. 1. Number of papers published in JOSS monthly, through March 2023.

JOSS's process is worth mentioning here, as it's quite different than most journals. JOSS reviews happen in public GitHub issues, and as such, they are open and non-anonymous. They are also collaborative: a discussion between multiple reviewers and authors. The role of the editor in the review process is also somewhat different than in other journals: it's to find reviewers, and to help them and author come to an agreement. The goal of a JOSS review is not to make an accept/reject decision on a submission but to help the authors get it to the point where it can be published.

JOSS reviews themselves are checklist-driven, containing five main groups of items. First, the reviewer must agree to Conflict of Interest and Code of Conduct guidelines. Second, there are a set of general checks, including the repository URL, the license, and contribution and authorship. Third, there are functionality checks, including installation, functional claims, and performance (if such claims are part of the submission.) The fourth set of checks are on the documentation of the

software and its repository, including a statement of need, installation instructions, example usage, functionality documentation, automated tests, and community guidelines. The last set of checks are on the software paper itself, specifically that it has a good summary, statement of need, describes the state of the field, has quality writing, and sufficient and appropriate references.

Many of the JOSS review criteria have associated guidance, and some, for example installation, API documentation, community guidelines, and automated testing have three levels: good, ok, and bad, where the criterion should be marked as satisfied if either the good or ok guidelines are met. For example, for installation, the guidelines for good quality are that the software is simple to install and that it follows established distribution and dependency management approaches for the language being used. The guidelines for ok quality are that there is a list of dependencies to install, together with some kind of script to handle their installation (e.g., a Makefile). The guidelines for bad quality, which means that the submission is not acceptable, is that the dependencies are unclear, and/or the installation process lacks automation.

Because JOSS is part of the RSE ecosystem (many JOSS authors, reviewers, and editors are RSEs), its practices (and these levels) have influenced other reviewers and developers (including RSEs) in terms of defining what's good and what's minimally acceptable. This is similar to rOpenSci's influence in the R community [4].

In general, means by which cultures change include rules and incentives. Here, JOSS provides rules, and at a high-level, tries to nudge incentives. Over time, the research software community changes, and JOSS levels also change: they typically become stricter, where what was "good" becomes the default standard and what was previously "ok" is no longer acceptable.

When JOSS began, it was primarily intended to influence the research community by rewarding software developers and maintainers with a published unit of work that was equivalent to a paper, since papers are one of the main units of credit in research. During this same period, there has been a push for and a gradual increase in recognition of the role of research software as a scholarly object itself [5], [6] that has led to an increase in software itself being published and cited, rather than papers about the software. Some of the JOSS founders thought of JOSS as a temporary project, which would no longer be needed once software itself was published.

This thinking missed the larger community impact that JOSS has made by influencing software developers and maintainers and their practices as mentioned above. If software was cited directly, JOSS papers would no longer be needed, but JOSS as a mechanism for reviewing and improving software packages, and for contributing to and improving software and RSE practices would still be important in shaping community values.

### III. RESEARCH SOFTWARE PROJECTS

As an example of a research software project, I’m going to look at one I work on: Parsl, a parallel scripting library for Python [7]. The idea behind Parsl is that it lets a developer define opportunities for parallelism, which can be “Python apps” that call Python functions, “Bash apps” that call external applications. In either case, these are marked using an annotation in Python; see Figure 2 These apps then immediately return “futures” rather than waiting to return result values, where a future is a proxy for a result that might not yet be available. This lets the apps run concurrently, while respecting dataflow dependencies, providing natural parallel programming. It also means that Parsl scripts are independent of where they run, as the logic of what parts of the program can be run in parallel is separate from the definition of which resources can be used for running the program.

```
@python_app
def hello():
    return 'Hello World!'

print(hello().result())
```




Hello World!

```
@bash_app
def echo_hello(stdout='echo-hello.stdout'):
    return 'echo "Hello World!"'

echo_hello().result()

with open('echo-hello.stdout', 'r') as f:
    print(f.read())
```



Hello World!

Fig. 2. Parsl Python and Bash app examples.

Parsl was originally developed and then maintained with support from an NSF SI2 award during 2016-2022. The funded development team has been 2-4 people/FTEs per year. Parsl is open source and has significant community support; see Figure 3. The project released version 1.0 in 2020, and has now moved to a weekly release model. The focus since v1.0 has mostly been maintenance rather than adding new features. Parsl’s current funding includes a new NSF collaborative award (2209919, 2209920), a Chan Zuckerberg Initiative (CZI) award, and contributions from projects that depend on Parsl.

The initial work on Parsl was based on previous work that had been done in a project called Swift, which had started much earlier [8]. Swift was aimed at providing a simple tool (language/runtime) for fast, easy scripting on big machines. We basically asked ourselves if we were starting from scratch at that time (2016), what different decisions would we make than Swift had, based on the changed environment, with one main difference being the growth and acceptance of Python.

One developer (Yadu Babuji) did the initial exploration of this issue and developed the initial proof-of-concept that became Parsl, with a few more people “supervising”. He then built the initial usable system, and was the main developer,

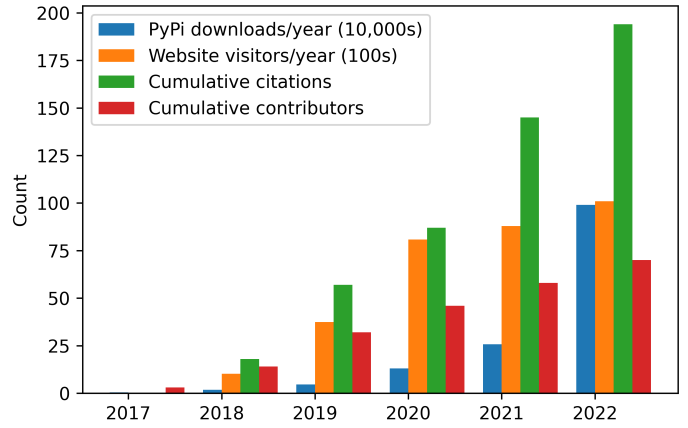


Fig. 3. Growth of Parsl’s usage and contributions.

choosing the processes to use for development himself based on his own experience.

Once the project grew a bit and a second developer (Anna Woodard) was brought on, the two developers then needed to agree on the processes that would be used, and to document them so that others in the community could also contribute. As Parsl moved to become a more open community project, with 2 to 4 main developers and 73 contributors, having and following these processes became much more important.

The processes include:

- How to make design/architecture decisions?
- What code style to use?
- What testing is sufficient?
- What documentation is sufficient?
- How to engage with and support users?
- What properties do contributions and changes need to have?
- How do contributions and changes get accepted?
- How to encourage/develop contributors?
- How to mix CS research and software product development?
- Who writes papers and who is listed as co-authors?

We made a set of specific decisions on these processes, though alternative decisions could have been equally valid. Understanding and using these practices is part of what makes the development team a community, and allows it to grow. As new developers come in, there’s often discussion about these practices and if they can or should be improved, sometimes leading to changes.

Many of the Parsl developers and contributors are RSEs, and some are undergrad/grad students or faculty. All of them bring in ideas, though new ideas about software engineering are mostly from RSEs and students.

Parsl is a one of many projects in the research software

space<sup>1</sup>. Its developers also typically either are working, or will work on other research software projects as well. As such, Parsl's software processes impact other projects' processes as well, either in terms of processes to follow or in terms of processes not to follow. Because a part of the developer community are RSEs, these processes also influence their groups and the larger RSE community. And where these developers are students, Parsl's processes influence their groups, leading to other students at least learning from them and potentially taking them up. Some of these students also will become RSEs, creating another pathway by which a particular project's processes can be infused into the RSE community.

#### IV. RESEARCH SOFTWARE GROUPS

The third level at which research software practices are developed, infused, and accepted is in research software engineering groups (see Figure 4. Many institutions, such as universities, national laboratories, and companies, are starting to have such groups [13], [14]. In one sense, this is a new role, as the concept of RSEs only started in 2012. However, some institutions, particularly those with HPC centers, have had such groups for a much longer time, for example, NCSA and the Edinburgh Parallel Computer Center, both of which started in the mid 1980s. Given that most of these institutions are older than the idea of RSE groups, it's important to initially consider how these RSE groups were created, as well as how this impacts the ways in which they choose acceptable and appropriate research software processes.

RSE groups typically start based on someone becoming aware of the RSE concept and thinking about how it impacts them, their projects, and their institutions. This might be an individual research software developer, a team of research software developers, or someone at a higher level in the institution.

**1. Individual-created:** In the case of an individual, typically they hear about RSE concept, and realize they are one. They often then talk about this with colleagues in their organization, leading to creation of an informal community. Someone in this community then formalizes it, though the details of this depend on the organization. This might include either becoming a unit in the institution or a virtual community that spans multiple units. A virtual community also usually includes some type of communication mechanisms, such as a mailing list, a Slack channel, and some type of meetings, perhaps lunches. These two choices can overlap, as a unit can be the center of a larger virtual community in the institution. Within their institution, they need to work with the human resources unit to ensure that there is correct recognition of their roles, including job titles, salaries, and career paths.

**2. Group-created:** In this case of a group of developers, the process is roughly similar. Typically, a set of software

<sup>1</sup>Research software is not well-categorized, but can be found by examining software that has been funded as research software, for example, under NSF's SI2 [9] and CSSI [10] solicitations, or CZI's Essential Open Source for Science (EOSS) [11] calls. It can also be found via its publications in journals like JOSS [3], or its membership in funding organizations such as NumFOCUS [12]



Fig. 4. The UK currently has over 30 RSE groups. (Captured from <https://society-rse.org/community/rse-groups/> 10 April 2023.)

developers, perhaps already part of a computing organization, group together to address a problem that they are having. For example, in a university, they may want to more effectively manage changing projects/staffing needs over time and provide some semblance of a career path in a soft-funded environment. In a company or national laboratory, they may be more concerned with job titles and skill recognition, as a part of career path development. In such a group, one or more members become aware of the RSE concept, and realizes this is what they are. They then follow a process similar to the one when the group was started by an individual.

However, the focus on a formal RSE group is stronger here. In particular, the RSE group needs to develop a mission, working both internally and with stakeholders, and it needs to define a method for institutional funding support for the group. This support can be full (the RSEs are supported directly by the institution), partial (the RSEs are supported by a combination of the institution and projects), or none (the RSEs are fully supported by projects).

**3. Management-created:** The third case is when the institution creates an RSE group from scratch. Unlike the first two cases, which are in some sense organic or bottom-up, this is directed and top-down. This case may occur because the institution hears about others who have RSE groups, and decides that they need one too, either to compete with those other institutions in work or in staff, such as if software staff are difficult to hire or are leaving for other institutions. To create and formalize such a group, a champion and other stakeholders will develop a mission for the group, determine

how the group will be funded, determine appropriate titles, salaries, and career paths, and then hire a leader and group members or move people internally into new roles.

The last issue related to these groups is that the initial group structure may need to be changed over time, for a variety of reasons. If the group grows, a multi-layer management structure will likely be needed. This may also lead to new discussions about funding, as group management is difficult to charge to projects, and typically is either paid by the institution or by some type of tax on projects.

An RSE group in a research institution will often have some type of matrix management, where the RSEs work on projects led by PIs (who could be the RSEs themselves, faculty members, or other senior researchers) who are primarily concerned about their projects, and where the RSEs report to their group manager, who is concerned with the RSEs as a group, including hiring, promotion, and developing their skills and career.

Some institutions, such as NCSA [15], try to avoid having RSEs work full-time on one project, so that the RSEs are primarily members of the RSE group who work on multiple projects, rather than being primarily a member of a single project who happens to be in an RSE group. If a project needs the equivalent of one FTE, the RSE group will project two half-time people. This is often also better for projects, as it avoids a single point of failure where one person leaving could cause the project to stall, and it's better for the RSEs as their peers are other RSEs from whom they can learn and with whom they can share information and lessons.

Because RSEs generally work on multiple projects, and each project has its own choice of practices, multiple RSEs in a group can collectively think about these practices as a metalevel. They can compare them, determine the advantages and disadvantages of each in particular contexts, and decide which should be used in new projects.

The RSE unit or virtual community and its members often join international, national, and regional RSE groups. In these communications, as well as local communication, they share experiences, and learn from each other. In this way, they influence their institution and each other. For example, RSEs work with faculty, staff, and students in their institution and those in other institutions on multi-institution projects. Via these collaborations and interactions with national RSE organizations, they infuse good practices into the overall research community.

This is not completely new, as developers in computing centers in the pre-RSE era (before 2012) certainly did work with each other, such as at the annual SC conference, vendor user group meetings, and other computer science, math, and disciplinary conferences and workshops, but these conversations were not as planned or intentional as the ones today. Having a shared RSE identity and explicitly working together within the RSE movement have catalyzed a dramatic increase in social interaction between developers, complementing the previous level of mostly technical interactions.

## V. CONCLUSIONS AND FUTURE WORK

All practices, including research software practices, are tied to the communities that perform them. Today, RSEs increasingly lead the research software development community. They often are the point where new practices become accepted and then disseminated. This happens in projects, RSE groups, and the community as a whole.

As such, I believe that tool and practice developers who are creating and proposing new methodologies and are looking for quick impact should be working to get RSEs to adopt them by convincing champions who are RSEs. And people who seek to understand research software practices should be studying RSEs.

However, while this belief is based on my experience over 35+ years of research software development and involvement with the research software developer and RSE community, it is just based on my own experience, and could be invalid. Additionally, while I believe JOSS and Parsl are generally representative of communities and research software projects, this may not be the case. Readers may agree with some points and disagree with others. To better and more formally test and understand these issues, a set of research activities could be carried out, including the following:

- Via interviews or focus groups, investigating the role of review criteria such as those used by JOSS, rOpenSci, etc. How do these criteria relate to formal and informal training taken by RSEs and non-RSEs? How are they accepted by the developers of research software as good practices in general vs requirements that only need to be met for the purpose of publication?
- Analyze research software that has been reviewed by JOSS or similar and research software in common use that has not gone through such a review process to understand difference in practices.
- Studying research software by surveys or by artifact analysis that is developed by RSEs vs research software developed by graduate students or faculty member, looking at both tools and practices and understanding the differences in which are used by each group, and then, assuming there are differences, using interviews or focus groups to test if RSE identity is the cause of these differences.
- Studying different types of RSE communities, for example, at universities, including those that have emerged organically from a leader in the institution's developer community, from a group of developers, and from the institution's management. Important topics would again be about practices and tools, and how these are formally or informally communicated within the community. This could also be compared with standard measures of well-functioning teams.
- In addition, it would be useful to examine how these different types of RSE communities make use of larger national and international RSE communities, e.g., do these RSEs communities that started with RSEs partic-



ipate more in their national community than those that were management-created?

- RSEs do not exist in isolation, and are influenced by (and perhaps influence) traditional software developers<sup>2</sup>. The software engineering community has studied the traditional software developer community to a large extent, but has only studies that RSE community to a much smaller extent. This points to an area of needed research: understanding the differences in practices between RSEs and traditional software engineers, as well as understanding the communication and infusion of practices and tools between the two communities.

Better understanding this could lead to changes in policies and practices in teaching institutions, research performing institutions, and funding organizations, as well as in national and international RSE organizations, which in turn would lead to better research software and better research.

#### ACKNOWLEDGMENT

Thanks to Neil Chue Hong for feedback on an earlier version of this paper. It is based on a talk I gave at the SIAM CSE 2023 Conference in Amsterdam, 2 March 2023 [16].

Some of the JOSS discussion is from a Feb 2022 talk [17], with contributions from the other JOSS editors at that time: Gabriela Alessio Robles, Mikkel Meyer Andersen, Katy Barnhart, Juanjo Bazán, Sebastian Benthall, Eloisa Bentivegna, Monica Bobra, Frederick Boehm, Jed Brown, Pierre de Buyl, Patrick Diehl, Elizabeth DuPre, Vissarion Fisikopoulos, Martin Fleischmann, Dan Foreman-Mackey, Jarvis Moore Frost, Nikoleta Glynatsi, Jeff Gostick, Richard Gowers, Hugo Gruson, Olivia Guest, David Hagan, Jayaram Harihan, Chris Hartgerink, Bitá Hasheminezhad, Christina Hedges, Luiz Irber, Mark A. Jensen, Prashant K. Jha, Vincent Knight, Rachel Kurchin, Hugo Ledoux, Christopher R. Madan, Brian McFee, Melissa Weber Mendonça, Kevin M. Moerman, Kyle Niemeyer, Juan Nunez-Iglesias, Lorena Pantano, Stefan Pfenniger, Viviane Pons, Kristina Riemer, Amy Roberts, Marie E. Rognes, Ariel Rokem, Will Rowe, Kelly Rowland, David P. Sanders, Mehmet Hakan Satman, Fabian Scheipl, Jacob Schrieber, Adi Singh, Arfon Smith, Charlotte Soneson, Øystein Sørensen, Andrew Stewart, Fabian-Robert Stöter, Yuan Tang, George K. Thiruvathukal, Kristen Thyng, Tim Tröndle, Leonardo Uieda, Chris Vernon, Marcos Vital, Lucy Whalley, Bruce E. Wilson, and Frauke Wiese. That talk was in turn based on a previous talk by Arfon Smith [18].

Discussion about Parsl includes contributions from the Parsl team [7], which has significantly included Yadu Babuj, Kyle Chard, Ryan Chard, Ben Clifford, Ian Foster, Zhuazhao Li, Mike Wilde, and Anna Woodard.

<sup>2</sup>We cannot simply separate software engineers based on where they work, saying that research software engineers work in research institutions such as universities and national laboratories, and traditional software engineers work in industry, as there are RSEs in industry and traditional software engineers in universities and national labs; the distinction is based on the software that is being developed and maintained, not the institution where the work is being performed.

#### REFERENCES

- [1] A. Taflove, *Computational Electrodynamics: The Finite-Difference Time-Domain Method*. Norwood, MA, USA: Artech House, 1995.
- [2] J. Cohen, D. S. Katz, M. Barker, N. Chue Hong, R. Haines, and C. Jay, “The four pillars of research software engineering,” *IEEE Software*, vol. 38, no. 1, pp. 97–105, 2021. [Online]. Available: <https://doi.org/10.1109/MS.2020.2973362>
- [3] A. M. Smith, K. E. Niemeyer, D. S. Katz, L. A. Barba, G. Githinji, M. Gymrek, K. D. Huff, C. R. Madan, A. Cabunoc Mayes, K. M. Moerman, P. Prins, K. Ram, A. Rokem, T. K. Teal, R. Valls Guimera, and J. T. Vanderplas, “Journal of Open Source Software (JOSS): design and first-year review,” *PeerJ Computer Science*, vol. 4, p. e147, Feb. 2018. [Online]. Available: <https://doi.org/10.7717/peerj-cs.147>
- [4] K. Ram, C. Boettiger, S. Chamberlain, N. Ross, M. Salmon, and S. Butland, “A community of practice around peer review for long-term research software sustainability,” *Computing in Science & Engineering*, vol. 21, no. 2, pp. 59–65, 2019. [Online]. Available: <https://doi.org/10.1109/MCSE.2018.2882753>
- [5] A. M. Smith, D. S. Katz, K. E. Niemeyer, and F. S. C. W. Group, “Software citation principles,” *PeerJ Computer Science*, vol. 2, p. e86, Sep. 2016. [Online]. Available: <https://doi.org/10.7717/peerj-cs.86>
- [6] D. S. Katz and M. Barker, “The Research Software Alliance (ReSA),” Apr. 2023. [Online]. Available: <https://doi.org/10.54900/zwm7q-vet94>
- [7] Y. Babuji, A. Woodard, Z. Li, D. S. Katz, B. Clifford, R. Kumar, L. Lacinski, R. Chard, J. M. Wozniak, I. Foster, M. Wilde, and K. Chard, “Parsl: Pervasive parallel programming in Python,” in *Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing*, ser. HPDC ’19. New York, NY, USA: Association for Computing Machinery, 2019, p. 25–36. [Online]. Available: <https://doi.org/10.1145/3307681.3325400>
- [8] M. Wilde, M. Hategan, J. M. Wozniak, B. Clifford, D. S. Katz, and I. Foster, “Swift: A language for distributed parallel scripting,” *Parallel Computing*, vol. 37, no. 9, pp. 633–652, 2011, emerging Programming Paradigms for Large-Scale Scientific Computing. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0167819111000524>
- [9] National Science Foundation, “Software infrastructure for sustained innovation (SSE, SSI, S2I2): Software elements, frameworks and institute conceptualizations,” [https://www.nsf.gov/publications/pub\\_summ.jsp?ods\\_key=nsf17526](https://www.nsf.gov/publications/pub_summ.jsp?ods_key=nsf17526), 2017.
- [10] —, “Cyberinfrastructure for sustained scientific innovation (CSSI),” [https://www.nsf.gov/publications/pub\\_summ.jsp?ods\\_key=nsf22632](https://www.nsf.gov/publications/pub_summ.jsp?ods_key=nsf22632), 2022.
- [11] Chan Zuckerberg Initiative, “Essential open source software for science (EOSS),” <https://chanzuckerberg.com/eoss/>, 2023.
- [12] NumFOCUS, <https://numfocus.org>.
- [13] D. S. Katz, K. McHenry, C. Reinking, and R. Haines, “Research software development & management in universities: Case studies from Manchester’s RSDS Group, Illinois’ NCSA, and Notre Dame’s CRC,” in *2019 IEEE/ACM 14th International Workshop on Software Engineering for Science (SE4Science)*, 2019, pp. 17–24. [Online]. Available: <https://doi.org/10.1109/SE4Science.2019.00009>
- [14] I. A. Cosden, “The Princeton University RSE group model: Operational and organizational approaches,” *Computing in Science & Engineering*, pp. 1–9, 2023. [Online]. Available: <https://doi.org/10.1109/MCSE.2023.3264113>
- [15] D. S. Katz, K. McHenry, and J. S. Lee, “Research software sustainability: Lessons learned at NCSA,” in *54th Hawaii International Conference on System Sciences*, 2021, pp. 7249–7256. [Online]. Available: <http://hdl.handle.net/10125/71494>
- [16] D. S. Katz, “How RSE community identity leads to improved research software practices,” Feb. 2023. [Online]. Available: <https://doi.org/10.5281/zenodo.7683966>
- [17] D. S. Katz and JOSS Editors, “Journal of Open Source Software: Developing a software review community,” Feb. 2022. [Online]. Available: <https://doi.org/10.5281/zenodo.6305241>
- [18] A. Smith, “Journal of Open Source Software: When collaborative open source meets peer review,” Oct. 2019. [Online]. Available: <https://doi.org/10.5281/zenodo.3497422>