Research Software Sustainability: Lessons Learned at NCSA

Daniel S. Katz
University of Illinois
at Urbana-Champaign
d.katz@ieee.org

Kenton McHenry
University of Illinois
at Urbana-Champaign
mchenry@illinois.edu

Jong S. Lee
University of Illinois
at Urbana-Champaign
jonglee1@illinois.edu

Abstract

This paper discusses why research software is important, and what sustainability means in this context. It then talks about how research software sustainability can be achieved, and what our experiences at NCSA have been using specific examples, what we have learned from this, and how we think these lessons can help others.

1. Introduction

Software is essential for research. Research data relies on research software for interpretation, understanding, analysis, and sometimes generation. Software makes modeling and simulation of physical and social phenomena at scale possible. Good research software can make the difference between valid, sustainable, reproducible research outputs and short-lived, potentially unreliable or erroneous outputs.

- The National Science Foundation has identified software as "directly responsible for increased scientific productivity and significant enhancement of researchers' capabilities" [1].
- Many of the 100 most-cited papers of all time identified in a 2014 article describe research software, from fields such as bioinformatics (BLAST, ClustalW), phylogenetics (MEGA4, MrBayes), and crystallography (SHELX, CNS) [2].
- A UK survey of 1,000 randomly chosen researchers showed that more than 90% of researchers acknowledged software as being important for their own research, and about 70% of researchers said that their research would not be possible without software [3].
- A study of Nature papers from Jan–March 2016 reveals that "32 of the 40 papers examined mention software, and the 32 papers contain 211 mentions of distinct pieces of software, for an average of 6.5 mentions per paper." [4]
- The US National Science Foundation made 18,592 awards totaling $9.6 billion to projects that included the topic of "software" in their abstracts between 1995–2016 [5].

Additional evidence for the importance of research software can be found through a community effort led by the Research Software Alliance [6].

2. Research software sustainability

But for research software to continue to be usable over time, ongoing human activity is needed to maintain it. Research software today involves a complicated stack of components, ranging from the operating system to general infrastructure (e.g., compilers) to domain-independent scientific infrastructure (e.g., linear algebra and I/O libraries) to domain-specific scientific tools (e.g., community applications and libraries) to project-specific code (e.g., scripts, workflows). Each layer depends on the layers below it, which can and do change in response to bugs, new needs, or changes in underlying layers, all the way down to the hardware itself [7].

We can define research software sustainability as the process of developing and maintaining software that continues to meet its purpose over time, which includes that the software adds new capabilities as needed by its users, responds to bugs and other problems that are discovered, and is ported to work with new versions of the underlying layers, including software as well as new hardware.

In order to sustain research software, we can

1. Take actions that reduce the amount of work needed
2. Take actions that increase the available resources

URI: https://hdl.handle.net/10125/71494
978-0-9981331-4-0

3. Take actions that both reduce the amount of work needed and increase the available resources

In order to reduce the amount of work needed to sustain the software, we can train its developers, which involves finding or developing training material, and use best practices, which involves finding or developing best practices.

In order to increase the available resources to sustain the software, we can create incentives so that people want to work on the software, e.g. getting citations that help in existing career paths, adjusting existing career paths to ensure that they reward software work, and creating new career paths; and we can increase available funding by first making the role of software in research clear to research funders, and then by clearly making the case for them to increase funding for new software, and to provide funding for software maintenance.

Finally, the main action that both reduces work and brings in new resources is collaboration. Using the work of others rather than reimplementing a function or package reduces what a software team (or its developers) needs to do themselves, even without assuming that the collaborators contribute to the software, which also may happen. Similarly, if others use a team's software, and contribute to maintaining it, the team has less they need to do. To make this work, the software has to be designed from the start to be modular and reusable, and it must also be clearly documented and explained to potential users, even those in fields other than the developer's. And the team has to put effort into engaging and working with the potential user and contributor community.

A related concept is FAIR, meaning Findable, Accessible, Interoperable, and Reusable. A set of FAIR principles were created [8], primarily for data, but these four foundational principles are also important for software. And community work has recently started to formalize a definition and metrics for FAIR for research software [9], in part starting with a recent paper by Lambrecht et al. [10]. The more FAIR software is, the more likely it is that it will be found, accessed, and used, which in turn leads to both a need for sustaining it, which can be filled by the original developers, users, or others who see an opportunity to contribute.

3. Sustainability is people

Overall, we recognize that nothing will change the fact that someone needs to maintain the software, whether they do this as a volunteer or as a paid activity.

We may be able to use institutional resources for this, if the software is considered sufficiently important to the organization, either or both operationally or reputationally. NCSA (the National Center for Supercomputing Applications at the University of Illinois Urbana-Champaign) has done this in the past. For example, NCSA provided support to the Ergo framework, described in the next section, because it saw an emerging community around the software and wanted to bridge it to the next project that would leverage it. In order to increase this, we need to be able to make a case to our management that this provides a good return on investment, in terms of needing to spin up fewer future projects and being more efficient in applying existing software to new projects, both of which can lead to increased success in collaboration and in proposals.

At NCSA, our primary experience has been in bringing in collaborators through joint funding opportunities, where either we write a proposal to develop software that includes funding others as users and developers to drive the development and expand it to support more use cases, or where users write proposals to do their research that includes funding for us and our software as an efficient means of enabling that research. In some cases, those collaborators are us, in a follow-on project that depends on the initial project and thus dedicates resources to maintaining it. We discuss this in more detail in the next section.

But first, we also want to briefly discuss the experiences and lessons about bringing in volunteers. We have some experience with volunteers, such as users who find bugs and tell us about them, or users who provide fixes to bugs, or developers who want to incorporate their software into ours, but here we highlight a couple of useful ideas from others who have focused on this type of interaction.

If we want to bring in volunteers or collaborators, we need to think about why they will choose to put effort into our software project, or how we can engage them. In the context of community activities and organizing, Joseph Porcelli has defined engagement as $motivation + support - friction$ [11]. Porcelli further splits motivation into intrinsic and extrinsic components, and defines each term as follows:

- Intrinsic motivation = self-fulfillment, altruism, satisfaction, accomplishment, pleasure of sharing, curiosity, real contribution to science
- Extrinsic motivation = job, rewards, recognition, influence, knowledge, relationships, community membership
- Support = ease, relevance, timeliness, value
- Friction = technology, time, access, knowledge

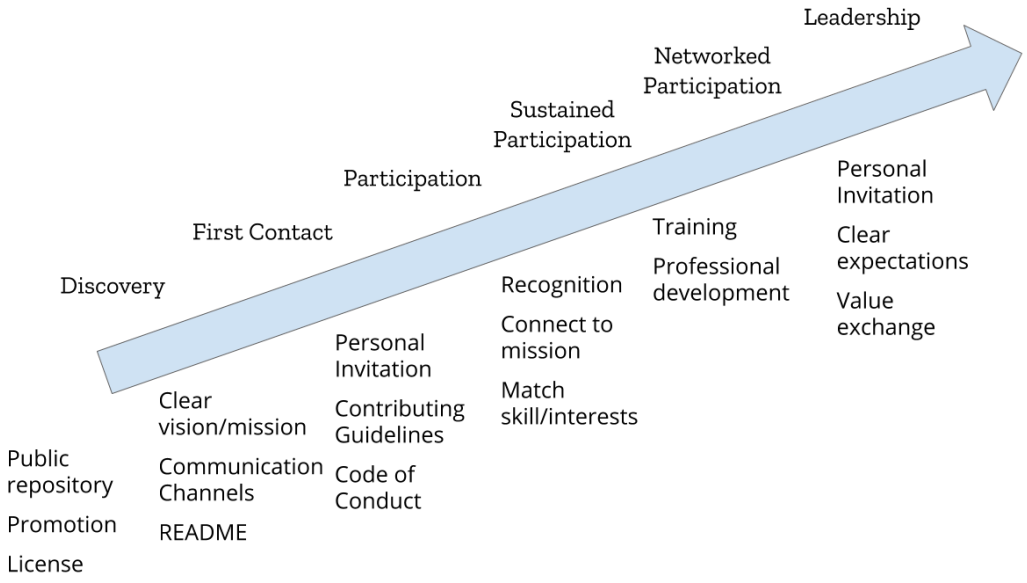

Figure 1. Community member engagement. For each level of engagement (above the arrow), methods to promote a community member to reach that level are shown below and to the left of the arrow [12].

To encourage engagement, we can impact many of these factors. For example, we can use GitHub for development (reduce friction by using a known technology, at least known to developers though potentially not to users), provide templates for issues and guidelines for good pull requests (reduce friction by providing knowledge of how to work with our project and simultaneously increase support by easing the means of doing so), provide a code of conduct and a welcoming and encouraging environment (increase extrinsic motivation by helping develop relationships and sense of community), add contributors to a list of authors who are cited when the software is used (increase both intrinsic motivation and extrinsic motivation through recognizing accomplishments), and highlight examples of how the software is used (increase intrinsic motivation by demonstrating the contribution to science).

Additionally, we can consider a progression of types of engagements, as defined by Abigail Cabunoc Mayes [12], as shown in Figure 1. Here, a potential contributor initially discovers the software project, has a first contact with the project, participates in it once, continues to participate, becomes part of the project, and eventually takes on a leadership role. The project has a number of things it can do before each level to encourage the potential contributor to move to that level, from promotion to encourage discovery to defining a value exchange and using a personal invitation to encourage a contributor to become a leader.

4. NCSA experiences

We describe our experiences at NCSA in two categories, software and people, though there clearly is overlap between them.

4.1. Software

Projects that develop software in academia can be divided into two categories, those that intend to develop and share software as a primary goal of the project, and those where the software is developed as a means to accomplish a different primary goal, often a research investigation. In the first category, the project likely has planned for the software to be released, but may not have firm plans to sustain it after the project funding ends, and in the second category, it's very unlikely there are any plans to sustain the software after the project ends, other than perhaps assuming that follow-on projects by the same team for which the software is needed will continue to maintain it. The software we are discussing here falls into the first category.

Research software activities supported by NCSA staff have spanned the spectrum described above. In general, however, NCSA has aimed to leverage and sustain past developments wherever possible, with the aim of benefiting not only current users of the software but also new users, particularly as part of

new efforts where similar functionality is needed, if perhaps adapted with additional features. When this is possible, larger frameworks have emerged with a wide array of features to support needs across a number of scientific disciplines. The broader user base in turn helps sustain activities around the software and through that the software itself.

Of course, this must be done carefully and considerately, as there are some potential downsides. The code can expand in size as additional functionality is added, and can become brittle and hard to maintain. The developers need to watch for this, aiming to stay true to the core intention of the software, as well as potentially rearchitect and refactor the code, particularly if the code was originally designed for a limited purpose. Often times, however, extensions to support additional data types, interfaces, or types of analytics are all that is needed in order to support a new community's needs.

An example of successful leveraging and sustaining software is the Clowder framework [13]. Started in 2009 as Medici [14, 15], initially a visual frontend for the Tupleo [16] RDF triple store for the purpose of an image database, the software has lived on for over a decade supporting efforts with overlapping needs across a broad spectrum of disciplines with funded efforts from agencies such as the National Science Foundation (NSF), the National Institutes of Health (NIH), the National Endowment for the Humanities (NEH), the Environmental Protection Agency (EPA), and the European Union (EU). Though it started with an initial focus on image data, the framework developed into its current incarnation, Clowder, as a much broader data management framework supporting many types of data, its curation, publication, as well as analysis. As data management was found to be a common need across many domains requiring similar core functionality but on different types of data, Clowder was developed as a major refactoring of the original Medici. In addition to supporting scalable analytics, this refactoring made nearly every aspect of the architecture customizable and so that it became fairly simple to add support for a new data type or visualization. Currently Clowder is leveraged by over a dozen active activities, as shown in Figure 2, supporting data well beyond just image data such as depth, numerical, and streaming data in support of geoscience, video data in support of social science, and 3D data in support of cultural heritage.

While some other sustained software activities rigorously promote their software and largely only collaborate on efforts that utilize their software, NCSA does not, at least not in the same sense. NCSA focuses on providing access to Research Software Engineers (RSEs) and supporting projects and scientific software development broadly. Software, such as Clowder tend to emerge organically and comes into new efforts not as "NCSA Clowder" or an "NCSA" thing, but as a readily leverageable tool in a toolbox that can be brought to bear on a new effort should it fit. If there is such a fit, perhaps with some modification, gains can be obtained within a project in terms of leveraging all the past developments and capabilities, leveraging the RSEs' already existing familiarity with the tool, and providing another option in terms of sustainability of new developments, since the new modifications that are created will go back into the toolbox in support of future endeavors.

Another good example is the Ergo framework [17]. Ergo has its origins with MAEviz [18], which was developed as part of the NSF Mid-America EarthQuake Center from 1997 to 2008 for risk assessment of city infrastructure during an earthquake for city planning and potential first responder deployment. After the project ended, NCSA continued to maintain MAEviz as a tool in its toolbox. Over seven years, a broad international open source community was built around the framework, and this community then re-branded the software as Ergo. NCSA held a user community meeting every two months to discuss Ergo and related topic activities, to discuss grant proposal developments, to support and share scientific research, and to plan participation in international conferences. Five to 10 partners (leads of their group) regularly attended the meeting. The framework proved to be a particular good fit in terms of ramping up a new effort in 2015 through the NIST (National Institute for Standards and Technology) Center for Risk-Based Community Resilience Planning as the IN-CORE (Interdependent Networked Community Resilience Modeling Environment) [19] framework, where the software continues to be developed to this day in support of not only earthquakes but also many other types of modelling on hazards such as fires, tornadoes, and tsunamis.

An example of software where NCSA plays a collaborating (rather than leading) role is the family of software that includes Swift [20], Parsl [21], and funcX [22], all of which are systems for programming applications by assembling a mix of existing and new components, including components that are also standalone applications. Swift is a C-like language for distributed parallel computing that was created about 15 years ago, and which was used by application developers and users in a variety of disciplines for applications that involved ensembles, optimization, workflows, and other types of programming programs. Swift was developed by a team primarily located at the University of Chicago and Argonne National

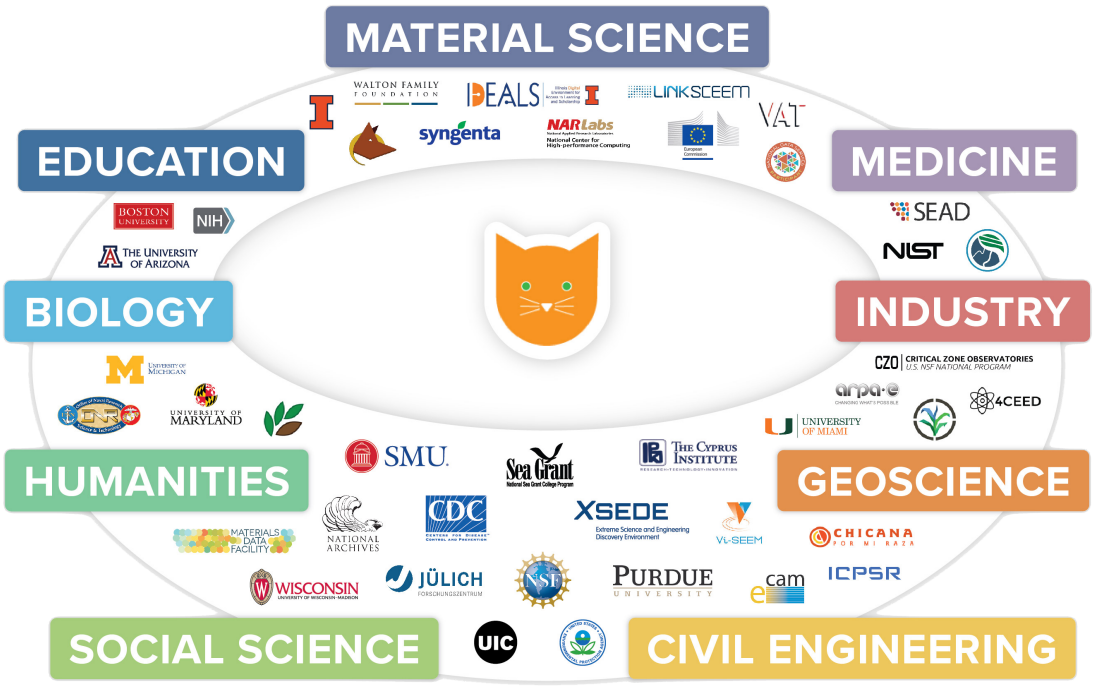

Figure 2. Clowder collaborators, users, and uses. The accumulation of these groups over time across these different fields and the contributions and use cases they provided enabled Clowder to be sustained.

Laboratory.

About 4 years ago, the Swift team (which had added a co-PI at NCSA) began thinking about the decisions that had been made at the start of Swift, and what similar decisions would be now (4 years ago), and decided that rather than being both a language and a system, the ideas of Swift could be implemented in pure Python to ease the learning curve for new users who would then not have to learn a new language. This led to the development of Parsl under a new NSF award. Parsl also was gained from the experiences of Swift, as it was written from scratch in a much more clean and more simple style, and with newer collaborative (community/git-based) software practices built-in. We believe these changes have led to Parsl being easier to contribute to and more welcoming to contributors. Parsl now has over 40 contributors, most of whom are not funded by Parsl and are not at any of the three institutions that are funded by NSF specifically for Parsl development and maintenance. This is a large number for a typical academic project, in line with projects of a similar age and similarly sized focus that are supported by open source organizations such as NumFOCUS [23].

These contributors include Parsl users who find bugs and fix them, and who suggest new features via pull requests. An example is a staff member at a company who contributed improved build technology while working to incorporate Parsl into an internal application. Parsl has also worked to integrate multiple cyberinfrastructure tools and systems. For example, Parsl can use the Work Queue [24] executor developed at Notre Dame, and the work to do this integration was mostly performed by the Work Queue developers. A final example is a student at a liberal arts college who developed a method to use Parsl as a uniform interface to his college's cluster, so that users don't need to have expertise in HPC job submission and scheduling [25]. This work led to 57 commits to the Parsl repository, affecting over 7,000 lines of code, and has made the student the 7th largest contributor to Parsl.

An even newer development in this set of related software projects is funcX, a function-as-a-service platform for research applications on academic (high-performance computing, or HPC) and commercial infrastructure (clouds) being developed collaboratively between University of Chicago, Argonne National Laboratory, and NCSA. funcX, which was recently funded by NSF through awards to the Universities of Chicago and Illinois, relies on Parsl to provide some of its functionality.

Both Parsl and funcX are projects where NCSA is part of the team, involved in the architecture, development, documentation, and outreach, but does not lead the project. While the overall style of these projects are similar to that of Clowder and Ergo at the level of the team and its goals of building products that are useful to end users and building a community of developers and end users to contribute to the software, it is also somewhat different in that decisions must be collaborative across institutions. Clowder and Ergo have also developed in this same direction over time, because asking people to make significant contributions to a software package is not usually successful without also giving them a stake in the governance of the software. While this lesson is more obvious in projects that are funded across institutions, we have also found it to be true in projects where the development awards are to a single institution (e.g., NCSA).

4.2. People

To better enable the ability to support and sustain scientific software activities in this model, NCSA has fleshed out and put into place aspects for the improvement of recruitment, training, and career growth of Research Software Engineers (RSEs) [27]. As described in Katz et al. [26], teams are organized less by specific areas and more along the lines of senior RSEs guiding newer RSEs, who can either be new software developers, developers from various scientific domains, or software engineers from industry backgrounds. Mentorship by senior RSEs revolves around not only technology and best practices but also interacting with collaborators, being part of teams made up of researchers and students, writing/publishing/presenting their work, over time overseeing efforts and possibly serving as the point or even Co-PI for projects, and being taught to think beyond the short term towards addressing longer-term software aspects such as reusability, maintainability, and overall sustainability.

As depicted in Figure 3, the RSE career path rewards the development of these aspects within the staff, specifically overseeing collaborations as well as software, by including these skills as prerequisites to moving to the next level, meaning that staff members can see what is needed to move to the next level and work on those aspects of their skills. We note that while this career advancement that has been put into practice is based on nearly a decade of experience, it still continues to evolve. Aspects that continue to need to be addressed include the sharing of knowledge across a large (approx. 40 member) overarching team with regards to the collective "tools within our toolbox" as well as caveats to matrixing projects across teams based on this non-speciality but instead mentorship-oriented team model.

Specifically, leveraging existing software, which can include extending its capabilities, and creating new software that could become a new reusable framework, is encouraged. The RSEs who can successfully do this become a resource within the group, because they can support these frameworks within new projects and train others as part of teams that use the frameworks. Developers with experience in building and maintaining one or more frameworks typically find themselves with more projects to choose from than they can do, requiring them to train and mentor newer research programmers to support those projects. This choice of projects around software they support provides job security, since the software without the people who know it is not very useful in the long term, while simultaneously preserving institutional knowledge of those frameworks.

The NCSA Software Directorate relies on a group of senior developers overseeing the group's software frameworks and serving as PI/Co-PI/lead of the projects the directorate works on. The group contributes to decisions about projects for the directorate to take on, staffing assignments across the directorate, project priorities (e.g., staffing increases, reallocation of senior developers when needed for upcoming milestones), as well as other financial/strategic decisions relevant to the group and the frameworks it oversees. To grow the number of developers in this more strategic category, and to foster the ability to grow activities, the directorate also considers technical strategy aspects, finding and encouraging senior people to further mentor research programmers who have demonstrated an aptitude for advancing along the career path. These research programmers then begin to take a larger part in strategic decisions across the projects served by the directorate (e.g., which underlying technologies to use, codifying best practices) and to begin leading aspects of the projects. In general, each project staffed by the directorate has one month of a senior person in addition to the development staff, with the role of the senior person being to oversee, guide, and align efforts and to serve as a backup if additional development is needed or as the project memory in the event of the loss of a newer staff member.

5. NCSA lessons learned

Software doesn't maintain itself – without active maintenance, it will stop working. And for research software, particularly for software that is designed to be shared to solve the problems the communities have, we

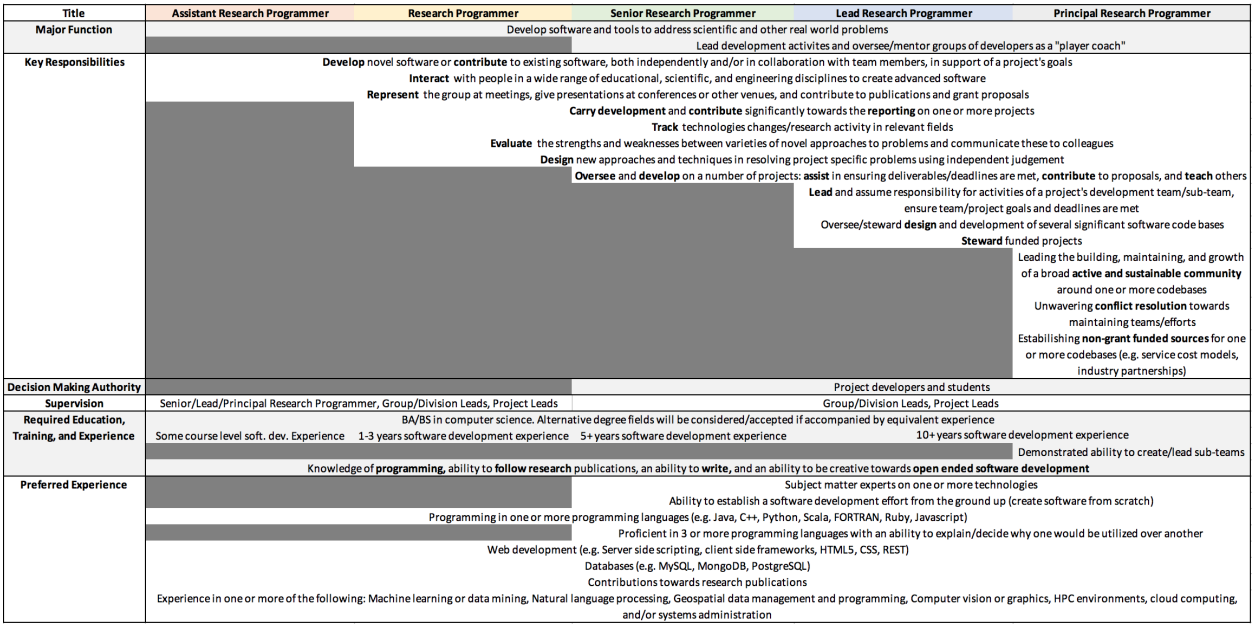

Figure 3. NCSA career path for RSEs (currently research programmers) [26].

believe we have demonstrated a set of methods to obtain the resources to perform this maintenance that can be combined. These include

- Making projects collaborative by design, including opening up governance of the projects to contributors
- Designing and applying mechanisms to move users to developer to maintainers
- Offering career paths that allow full-time research software developers to act as institutional memory and continuing expertise
- Considering senior developers a key part of the overall Software Directorate, with specific roles in mentoring staff, leading projects, and providing strategic guidance to the directorate
- Designing software to be as flexible and adaptable as possible, and working to combine software needs into a small set of well-architected packages
- Providing institutional support to bridge software packages between projects

We continue looking for additional methods to bring in, understanding how best to combine the methods we have, and working on institutional issues such as trying to make the case for increased institutional support and career paths that enable research software engineers to be as productive as possible.

Acknowledgements

We thank members of the current and past NCSA software development community, as well as our external collaborators in the projects mentioned in this paper. We also thank the anonymous reviewers for their helpful comments and suggestions for future work.

References

[1] National Science Foundation, "Software Infrastructure for Sustained Innovation - S2I2 (SI2-S2I2)," 2013. Program Solicitation NSF 13-511, https://www.nsf.gov/pubs/2013/nsf13511/nsf13511.htm.

[2] R. Van Noorden, B. Maher, and R. Nuzzo, "The top 100 papers: Nature explores the most-cited research of all time," *Nature*, 2014. https://doi.org/10.1038/514550a.

[3] S. Hettrick, M. Antonioletti, L. Carr, N. Chue Hong, S. Crouch, D. De Roure, I. Emsley, C. Goble, A. Hay, D. Inupakutika, M. Jackson, A. Nenadic, T. Parkinson, M. I. Parsons, A. Pawlik, G. Peru, A. Proeme, J. Robinson, and S. Sufi, "Uk research software survey 2014," Dec. 2014. https://doi.org/10.5281/zenodo.608046.

[4] U. Nangia and D. S. Katz, "Understanding software in research: Initial results from examining nature and a call for collaboration," in *13th International Conference on e-Science (e-Science)*, pp. 486–487, 2017. https://doi.org/10.1109/eScience.2017.78.

[5] D. S. Katz, L. C. McInnes, D. E. Bernholdt, A. Cabunoc Mayes, N. P. Chue Hong, J. Duckles, S. Gesing, M. A. Heroux, S. Hettrick, R. C. Jimenez, M. Pierce, B. Weaver, and N. Wilkins-Diehr, "Community organizations: Changing the culture in

which research software is developed and sustained," *Computing in Science & Engineering*, vol. 21, no. 2, pp. 8–24, 2019. https://doi.org/10.1109/MCSE.2018.2883051.

[6] M. Barker, D. S. Katz, and A. Gonzalez-Beltran, "Evidence for the importance of research software," June 2020. https://doi.org/10.5281/zenodo.3873832.

[7] K. Hinsen, "Dealing with software collapse," *Computing in Science & Engineering*, vol. 21, no. 3, pp. 104–108, 2019. https://doi.org/10.1109/MCSE.2019.2900945.

[8] M. D. Wilkinson, M. Dumontier, I. J. Aalbersberg, G. Appleton, M. Axton, A. Baak, N. Blomberg, J.-W. Boiten, L. B. da Silva Santos, P. E. Bourne, and et al., "The FAIR guiding principles for scientific data management and stewardship," *Scientific Data*, vol. 3, Mar. 2016. https://doi.org/10.1038/sdata.2016.18.

[9] "FAIR 4 Research Software (FAIR4RS) WG." https://www.rd-alliance.org/groups/fair-4-research-software-fair4rs-wg.

[10] A.-L. Lamprecht, L. Garcia, M. Kuzak, C. Martinez, R. Arcila, E. Martin Del Pico, V. Dominguez Del Angel, S. van de Sandt, J. Ison, P. A. Martinez, and et al., "Towards FAIR principles for research software," *Data Science*, vol. 3, p. 37–59, June 2020. https://10.3233/DS-190026.

[11] J. Porcelli, "How to grow users into active community members and get your community more engaged," in *2013 Open Source Software Summit*, 2013. https://web.archive.org/web/20160310200155/http://ossummit.org/.

[12] A. Cabunoc Mayes, "Work open, lead open," in *Chan-Zuckerberg Initiative (CZI) Essential Open Source Software (EOSS) Kickoff Meeting*, 2020. https://docs.google.com/presentation/d/13NAGmqP_Fb2qjBKePGVZe7awtwnFKZUHtPz8mqbYtsE/present.

[13] L. Marini, I. Gutierrez-Polo, R. Kooper, S. P. Satheesan, M. Burnette, J. Lee, T. Nicholson, Y. Zhao, and K. McHenry, "Clowder: Open source data management for long tail data," in *Proceedings of the Practice and Experience on Advanced Research Computing (PEARC '18)*, ACM, 2018. https://doi.org/10.1145/3219104.3219159.

[14] L. Marini, R. Kooper, J. Futrelle, J. Plutchak, A. Craig, T. McLaren, and J. Myers, "Medici: A scalable multimedia environment for research," in *Microsoft Research eScience Workshop*, (Berkeley, CA), 2010.

[15] C. Sophocleous, L. Marini, R. Georgiou, M. Elfarargy, and K. McHenry, "Medici 2: A scalable content management system for cultural heritage datasets," *code{4}lib*, 2017. https://journal.code4lib.org/articles/12317.

[16] K. McHenry, M. Ondrejcek, L. Marini, R. Kooper, and P. Bajcsy, "Towards a universal viewer for digital content," *Procedia Computer Science*, vol. 4, pp. 732 – 739, 2011. Proceedings of the 2011 International Conference on Computational Science (ICCS), https://doi.org/10.1016/j.procs.2011.04.077.

[17] N. Makhoul, C. Navarro, and J. S. Lee, "Seismic estimation of casualties and direct economic loss to byblos city: a contribution to the '100 resilient cities' strategy," *Sustainable and Resilient Infrastructure*, pp. 1–21, 2020. https://doi.org/10.1080/23789689.2020.1745531.

[18] T. M. McLaren, J. D. Myers, J. S. Lee, N. L. Tolbert, S. D. Hampton, and C. M. Navarro, "Maeviz: An earthquake risk assessment system," in *Proceedings of the 16th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, GIS '08, Association for Computing Machinery, 2008. https://doi.org/10.1145/1463434.1463534.

[19] N. Attary, J. W. van de Lindt, H. Mahmoud, S. Smith, C. M. Navarro, Y. W. Kim, and J. S. Lee, "Hindcasting community-level building damage for the 2011 Joplin EF5 tornado," *Natural Hazards*, vol. 93, pp. 1295–1316, 2018. https://doi.org/10.1007/s11069-018-3353-5.

[20] 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, https://doi.org/10.1016/j.parco.2011.05.005.

[21] 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*, HPDC '19, pp. 25—36, Association for Computing Machinery, 2019. https://doi.org/10.1145/3307681.3325400.

[22] R. Chard, Y. Babuji, Z. Li, T. Skluzacek, A. Woodard, B. Blaiszik, I. Foster, and K. Chard, "Funcx: A federated function serving fabric for science," in *Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '20, pp. 65—76, Association for Computing Machinery, 2020. https://doi.org/10.1145/3369583.3392683.

[23] NumFOCUS. https://numfocus.org.

[24] P. Bui, D. Rajan, B. Abdul-Wahid, J. Izaguirre, and D. Thain, "Work queue + python: A framework for scalable scientific ensemble applications," in *Workshop on Python for High Performance and Scientific Computing (PyHPC) at the ACM/IEEE International Conference for High Performance Computing, Networking, Storage, and Analysis (Supercomputing)*, 2011. http://ccl.cse.nd.edu/research/papers/wq-python-pyhpc2011.pdf.

[25] B. Glick, "Parsl for research computing at a liberal arts college," in *ParslFest*, 2019. https://drive.google.com/file/d/1fSQ1j_2rl8-IoQLoItyi8jhOopY3xpK9/view.

[26] 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 *ACM/IEEE International Conference on Software Engineering (ICSE), International Workshop on Software Engineering for Science (SE4Science)*, 2019. https://doi.org/10.1109/SE4Science.2019.00009.

[27] J. Cohen, D. S. Katz, M. Barker, N. P. Chue Hong, R. Haines, and C. Jay, "The four pillars of research software engineering," *IEEE Software*, 2020. https://doi.org/10.1109/MS.2020.2973362.