Building and Supporting the Human Capability to Steward the Research Software Stockpile

Kenton McHenry (University of Illinois at Urbana-Champaign), Daniel S. Katz (University of Illinois at Urbana-Champaign)

The National Center for Supercomputing Applications (NCSA) at the University of Illinois Urbana-Champaign has a long history of supporting the technological needs of scientific efforts through high performance computing (HPC) expertise and systems, as well as broadly impacting applications development both for science and engineering and beyond (e.g., Mosaic and httpd that helped establish the Internet). This continues to this day with NCSA's operation of the Blue Waters supercomputer and its subsequent Delta system, which provide resources in support of the growing needs around machine learning, along with its leading role in the Extreme Science and Engineering Discovery Environment (XSEDE), which coordinates and provides support for NSF HPC resources throughout the country. Over the past few years, NCSA has also moved more and more towards supporting the data management needs of scientific endeavors, supporting large astronomy efforts such as the Dark Energy Survey (DES) and the NSF Major Research Equipment and Facilities Construction (MREFC) Large Synoptic Survey Telescope (LSST) initiative, as well as numerous other activities around ecological forecasting, risk assessment, phenomics, genomics, crop modeling, critical zone science, materials science, and urban informatics, activities that span NSF, DOE, NIST, NIH, NEH, international partnerships, and industrial partnerships. With regards to software specifically, NCSA has two established directorates that develop, maintain, and work with research software as well as providing user support, housing roughly 70 Research Software Engineers (RSEs). Through these directorates, NCSA serves as an intersection of scientific software development activities by providing support for a broad number of efforts with its RSEs who carry experience and technology from one effort to another. This maintained body of RSEs allow projects to ramp up quickly by increasing awareness of leverageable technologies, both internally and externally developed, and amplify development outputs through leveraging synergies with other ongoing efforts, providing:

- expertise in open source practices, tools, community development, data analysis methodologies, user interface and user experience design, and efficient usage of computational/storage resources;
- training and support to students and researchers on software development best practices and software technologies; and
- institutional memory and sustainability for developed software capabilities by maintaining them for future efforts.

NCSA endeavors to build reusable and repurposable frameworks from the software activities in which it is involved, and then offers these frameworks to new efforts in order to accelerate research activities and to further develop the frameworks (a virtuous cycle). In support of scientific data needs, NCSA offers frameworks/services for data management, analysis and machine learning, visualization, simulation and modeling, and cyberprotection through software

such as Clowder, ERGO, Zeek (aka Bro), CILogon, Cactus, funcX, GeoDashboard, Parsl, XRAS, and yt to name a few.

Based on this experience, we write to provide information on what we have done in software to be successful in accomplishing our mission, as well as what we have learned and what we plan to change. Specifically, we provide input on the following topics: 5) Challenges in building a diverse workforce and maintaining an inclusive professional environment, and 6) Requirements, barriers, and challenges to technology transfer, and building communities around software projects, including forming consortia and other non-profit organizations.

(5) Challenges in building a diverse workforce and maintaining an inclusive professional environment: *What challenges do you face in recruiting and retaining talented professionals to develop software for scientific and high-performance computing? What additional challenges exist in recruiting and retaining talented professionals from groups historically underrepresented in STEM and/or individuals from underserved communities? What challenges exist in maintaining inclusivity and equity in the development community for scientific and high-performance- computing software? What successful strategies have you employed to help overcome these challenges? What opportunities for professional recognition and career advancement exist for those engaged in developing scientific and high-performance- computing software?*

Recruiting and maintaining software development professionals in support of research is challenging in general. A key aspect of this is the contrast between working on the development of software in an academic environment versus in industry. The first and foremost issue is the motivation of the developer. Developers who tend to pursue positions within academia, and stay in them, are typically less motivated by monetary compensation than by recognition and impact (on human knowledge), with the ability to contribute to research that leads to a new discovery, changes society, and even simply leading to publication with the developers as fellow co-authors, and establishing their own CV. Another motivation that typically accompanies developers in academia is that of a more balanced work-life experience as well as the ability (and encouragement) to always learn new skills. These motivations often drive decision making with regards to accepting a position in academia, where typical salaries are lower than those found in industry. A second aspect is that of engagement and outreach. While not always a necessary skill for those developing software in industry, this is often a must within academia in terms of engaging scientists and their students, with regards to timelines and continuously evolving requirements, engaging in dissemination (a key aspect of sustainability of the software), and supporting users of the software. A final motivation is the opportunity to work collaboratively in a research environment, where searching and questioning together are valued as methods to determine the best software to build and maintain. Software developers aligning with these areas have been emerging globally over the past decade in support of scientific software, and realizing the commonalities among them in contrast to software development within industry have begun referring to themselves as Research Software Engineers (RSEs) - http://researchsoftware.org.

There are however a number of challenges in the recruiting of RSEs. Even with the above stated differences, there is still a core requirement for software development knowledge and best practices which entails that those hiring these individuals must often contend with a very competitive job market, one where the temptation of higher paying salaries within industry is a factor in decision making. Additionally, because of the above stated characteristics of an RSE, the number of hireable applicants tends to be even further reduced. Lastly, another typical and significant challenge is that of stability, given that most projects are grant-based with relatively short durations in contrast to say one's career. This aspect can make it not only difficult to recruit but also retain staff and preserve the core institutional knowledge that is vital to the continuation of developed software.

A number of models have nonetheless been piloted in regards to recruiting and retaining groups of RSEs (see Katz, https://arxiv.org/abs/1903.00732). A foundational aspect of many of these groups is that RSEs are hired as part of the group/organization and then allocated to different projects rather than the RSEs being hired directly by a particular project. At NCSA, for example, roughly 40 RSEs are housed within the software directorate and are assigned to work on about as many projects, with the specific projects changing over time, and staff members typically and intentionally being split across multiple projects. Leadership, made up of a number of senior RSEs, serves to identify and bring in new efforts with researchers at the university and elsewhere so that staff expertise and developed software can be leveraged and maintained in a win-win situation for 1) the project, which ideally benefits from experienced staff and existing customizable frameworks being developed, and 2) the body of RSEs who in turn are able to continue on in a more sustainable manner. Another set of key roles of the software directorate are recruiting RSEs, training them, and at the end of the day, retaining them and their institutional knowledge. The core part of these activities is that of training, with senior RSEs serving as advisors to newer RSEs who might come from non-engineering domains or industry, or who are perhaps recent graduates. Doing this allows the applicant pool to be greatly increased so that applicants who may not have every RSE skill needed for a given project are potentially still hireable, as the mechanisms to effectively develop those needed skills are provided through this organizational setup. Finally in regards to retention, because the RSE group has a consistent influx of projects over time, the establishment of a career path that allows staff to advance regularly and grow within an organization is possible. The skills needed by the group as a whole can be developed in individual staff as they advance through the years via training and careful project assignments. At NCSA, for example, there is a 5-level career path that starts with the associate RSE who is largely just involved in software development, but moving up to Senior and above, this involves significant aspects of fostering collaborations, community engagement, architecture, and overseeing teams.

Overall, challenges in building a diverse workforce and maintaining an inclusive professional environment within scientific software development are deeply tied to the constrained needs of research software positions along with the often inherent instability of those positions over the long term. By addressing these aspects, along with the growing recognition within science of the importance of the RSE role, we greatly increase the opportunity for new RSEs and a more diverse workforce.

(6) Requirements, barriers, and challenges to technology transfer, and building communities around software projects, including forming consortia and other non-profit organizations: *ASCR recognizes that successful software for scientific and high- performance computing often has many stakeholders, including academic research activities, research laboratories, and industry. Moreover, while DOE has provided funding for the development of a significant number of foundational software packages within the modern software ecosystem for scientific and high-performance computing, as the complexity of the software ecosystem continues to increase, and number of stakeholders has grown, ASCR seeks to understand how it might encourage sustainable, resilient, and diversified funding and development models for the already-successful software within the ecosystem. Such models include, depending on circumstances that ASCR seeks to better understand, both the private sector and non-profit organizations. Non-profit organizations include both charitable organizations (e.g., those with 501(c)(3) status) and R&D consortia (e.g., those with 501(c)(6) status). What are the important characteristics and components of sustainable models for software for scientific and high-performance computing? What are key obstacles, impediments, or bottlenecks to the establishment and success of these models? What development practices and other factors tend to facilitate successful establishment of these models?*

Software sustainability has been a key challenge within science for some years now. A number of sustainability approaches and models have been documented in regards to this (see https://doi.org/10.1787/302b12bb-en) along with programs emphasizing the development of software with sustainability in mind from the start (e.g., the NSF CSSI program). From the perspective of work supported by NCSA, we see two key challenges in regards to the increased sustainability of scientific software.

The first revolves around the generality of developed software, specifically, how readily can others with similar needs leverage existing software, and on top of that, customize it to meet needs that are more specific to a given effort. Far too often we see communities developing software from scratch, with the mindset that the given community's needs are unique, and thus that a custom solution is required. The end result, however, is that dozens of applications are built where 80-90% of its capabilities overlap, and where if a solution had been architected with flexibility in mind, it could have been leveraged by the rest. This has been witnessed within science numerous times with regards to scientific workflow systems, captured in the adage of “yet another workflow system”, as well as more recently with data management systems. Duplicating anything has obvious drawbacks in terms of cost and expended effort, but it additionally has a drawback in terms of sustainability, specifically, in that the community that would otherwise be able to leverage and contribute to such developments is instead carved into smaller parts. While the individual contributions and resources could have been combined to develop a common tool, instead each struggles to continue on and competes with each either over smaller amounts of resources.

The second, ironically, is that regardless of the duplication in terms of the overall applications built, scientific software at its roots are typically built in a complicated stack, with layers that range from custom scripts and notebooks to domain-specific tools to scientific software infrastructure to non-scientific (general) software infrastructure to operating systems (see

Hinsen, 10.1109/MCSE.2019.2900945). While for any given scientific project, some of the software, particularly at the top layer(s) of this stack, is built by scientists, at DOE labs or elsewhere under DOE funding, the software stack (at lower layers) includes elements that are not developed for science, and are not (and cannot be) controlled by DOE, though DOE might be able to encourage/guide some of them. In many cases, these complicated software stacks are assembled over time, and the developer at one level might only make choices one or two levels below the level at which they are working, without fully understanding the choices that were previously made by those who developed the software at those levels. In particularly, developers make choices about how sustainable the software they choose to depend on is, but because they don't always fully understand the choices made by others at lower levels, the choices they make may lead to dependencies on software that is ill-supported and potentially dangerous (e.g., heartbleed, Log4j, https://xkcd.com/2347/).

As almost all DOE-developed (and DOE-mission-critical) scientific software (along with all other scientific software) has these dependencies, two clear choices beyond the status quo (ignore this until there's a disaster then wait for the overall software community to fix it) exist: 1) understand specific instances of the problem in context and address them; and 2) understand the problem in general and prevent it from occurring.

To understand specific instances and address them, DOE would need to audit all of its software, understand its dependencies, and traverse this structure, continuing downward until all indirect dependencies are also known. All of these dependencies could then be cataloged, and analyzed in terms of potential problems (lack of ongoing support, security audits, etc.) and any software with such problems could be addressed on a case–by-case basis, potentially by funding DOE staff or others to do this work. However, others who also depend on the same software may attempt to do the same thing, which could lead to splits of commonly used packages if not done carefully in a coordinated manner, and additionally is not cost-efficient for any such organization.

To understand this problem in general and address it more systematically and more cost-effectively, a similar software audit could be done by DOE and other research software funders (government agencies, private foundations, industry, etc.), with information about which software needs to be supported merged and shared across such funders, which could then come together to decide on how to support these software packages, with this being done in a coordinated and shared manner. This would require a group such as the Research Software Alliance (ReSA) has proposed as the Research Software Funders Forum, though since the Funders Forum's initial meeting is in January 2022, it has not discussed this specific problem.