Multilab Software Sustainability Working Group Response to Software Stewardship Request for Information

Anshu Dubey, Mathematics and Computer Science, Argonne National Laboratory
Katherine Riley, Argonne Leadership Computing Facility, Argonne National Laboratory
Nicholas Schwarz, Advanced Photon Source, Argonne National Laboratory
David E. Bernholdt, Computer Science and Mathematics and Oak Ridge Leadership Computing Facility, Oak Ridge National Laboratory
Bronson Messer, Oak Ridge Leadership Computing Facility, Oak Ridge National Laboratory
Reuben D. Budiardja, Oak Ridge Leadership Computing Facility, Oak Ridge National Laboratory
Mathieu Doucet, Neutron Scattering Division, Oak Ridge National Laboratory
Deborah Agarwal, Scientific Data Division, Lawrence Berkeley National Laboratory
Daniel Gunter, Scientific Data Division, Lawrence Berkeley National Laboratory
Harinarayan Krishnan, Advanced Light Source/Computing Research Division, Lawrence Berkeley National Laboratory
Stephen Leak, National Energy Research Scientific Computing Center, Lawrence Berkeley National Laboratory

December 13, 2021

Computing powers much of the mission of the Department of Energy (DOE) and software is its backbone. ASCR research programs are building codes that are beneficial broadly across the sciences, and sustainability of that software is key to its adoption. Additionally, software is essential to the operation of all the DOE experiments and scientific user facilities, and to many of the discoveries happening every day at these facilities. The sustainability of this software is as important as the maintenance of the instruments and deserves the same sort of attention.

Research software is diverse and used by SC and outside R&D programs, in experimental, observational and computational endeavors. Research software includes high-performance computing (HPC) codes running on supercomputers, data pipelines near scientific instruments, operational software for the instruments, and web-based scientific services. For the success of the DOE mission we need the sustainability of software to be a universal value among DOE scientific software stakeholders. At its most fundamental level, this means that all concerned parties should have an appreciation of what makes software sustainable and what makes it unsustainable, and there should be a general expectation that choices will be made and practices used that favor sustainability and avoid unsustainability.

We are an informal group of researchers with experience in numerous SC R&D programs as well as leading SC user facilities for computational, experimental, and observational sciences.

Our collective experience and observations about software stewardship are distilled into the following responses to the questions raised in the RFI document.

Software dependencies and requirements for scientific application development and/or research in computer science and applied mathematics relevant to DOE's mission priorities

A huge range of software contributes to meeting DOE's mission priorities. It is impossible to provide a concrete or comprehensive list of the software that DOE scientific endeavors depend on to get the job done. The software may originate within or outside the DOE complex; it may be developed within a single institution or (increasingly) by multiple institutions; and in many cases, includes contributions from people not sponsored by DOE. The software that DOE R&D relies upon may be a product of research or a tool for research, and in many cases, both. For example, a numerical library might be a vehicle for applied mathematics researchers to explore new algorithms, while a computational science application project or an experimental facility might rely on that same library to provide solvers for their scientific models or data analysis needs. We must also recognize that the software powering DOE R&D is also provided under a range of business models. Freely available or open source software is a common mode for much "research software", but plenty of important software is closed source and commercialized.

All of the software we work with could undoubtedly benefit from more attention to sustainability and perhaps more explicit stewardship efforts. However, the level of need will vary greatly both across the broad space of software dependencies and over time. Sustainability and stewardship become greater concerns as software has more "impact" on the completion of mission objectives. Impact may stem from the number of users or projects relying on it, from the importance of the use cases or the services provided, or other factors. We recognize that, given limited budgets, cases must be evaluated individually. General efforts to train developers and to set expectations for sustainability of software produced with DOE support can bring benefits to the long term software stewardship. Sustainability and stewardship are rarely explicitly discussed today, and we believe that individual producers and consumers of research software, as well as DOE collectively, would benefit from the development of a more thoughtful and consistent approach to making decisions about the stewardship of individual packages.

Practices related to the security and integrity of software and data

We take it for granted that facilities with any physical installation have appropriate safety and security measures in place. This must be made true also for software and data. Finding the right security and integrity measures for software and data requires a holistic approach which balances rigor with mission needs without overwhelming the resources, and will require a combination of hardware- and software-based approaches.

Hardware-based security and integrity measures include network access policies enforced via firewalls and routing, strategies to mitigate data corruption and/or loss and backup policies for robust data staging, storage and archival, and execution environments for internal and external facing users.

Software-based measures for security and integrity include the use of version control (Git has become the de-facto standard), quality control via test-driven development and Continuous-Integration/Continuous-Deployment (CI/CD), and well-defined contribution policies. Version tagging, digital signatures for software and data artifacts, and containerization of software stacks for reproducibility also support software and data security and integrity.

FAIR data stewardship principles support the integrity and reproducibility of research that uses software and data: these state that data should be Findable (rich metadata and globally unique and persistent identifiers), Accessible (access protocols, including authentication and authorization where appropriate, are public), Interoperable (via use of metadata) and Reusable (clear licensing, detailed provenance and metadata that meets domain-relevant community standards).

Policies and plans for software and data sustainability, based on the measures described above, should be defined and training in these practices should be made available to developers and researchers. Providers and users of data should have an understanding of which data cannot be released into the public domain, leaders should have a socio-technical understanding of the effects of different incentives and practices (eg, what conditions lead to information leaks), and the project onboarding process should ensure participants have the training and knowledge their role requires.

Infrastructure requirements for software development for scientific and high-performance computing

Github, and similar offerings, are highly functional and effective for collaborative software development on teams of varying sizes. Github is widely used because it has a broad array of features for collaborative software development such as issue tracking, agile project boards, integrations for testing and documentation, and software review and discussion workflows. As a platform used by millions of developers world-wide, the quality of these interfaces and features on Github (and Gitlab) are a welcome resource for scientific computing. These platforms do not provide, however, built-in support for larger scale applications or those requiring specialized resources. Infrastructure could be created to bridge this gap and leverage the workflows and execution capabilities of DOE facilities and projects as a back-end to provide software assurance, while using the industry-standard and sophisticated interfaces to perform the software development workflow.

Another form of infrastructure is building up knowledge, within the organization, of how to develop and sustain scientific software, particularly in cross-functional and distributed teams. A successful approach in some larger projects (e.g., NAWI, KBase) has been to create a specific role of software release and software project management, distinct from software development or project PI responsibilities. This role promotes the importance of coordinating software development, documentation, testing, and maintenance across a diverse and cross-functional team. Personnel developing these skills can serve as a knowledge resource across many projects.

Developing and maintaining community software

We interpret community software to be a critical resource for its community that is freely available, or available under terms compatible with public science funding, that is not meant for use by its developers exclusively, and for which ongoing development and/or maintenance is supported in part or entirely by public science funding. This software is often, but not always, open-source; and not all open source software is community software. Such software may have a core set of developers that are responsible for its continued viability, while accepting contributions from others in the community after appropriate review and testing. The software will also have users (possibly indirectly via inclusion in other software) who do not make any direct contributions to the software base. For example, the free and open-source Python Material Genomics (*pymatgen*) code that the Materials Project provides for analysis would be considered community software, whereas the open-source but copyrighted and non-free Vienna Ab-initio Simulation Package (VASP) that the Materials Project uses to perform material calculations at NERSC would not.

Many challenges are unique to community software. Legal issues such as copyrights and ownership etc. may come up when the software in question is developed and distributed by one institution. The equation becomes more complex when software becomes community owned and maintained. In addition to the perennial challenge of funding to support development and maintenance, added concerns about distribution of responsibility and authority come into play. Questions may arise about what, if any, legal framework should exist for the group that takes on such responsibilities and what authority should be allotted to this group. Software that has taken many person-years to develop and reach a level of maturity where science communities have confidence in its usefulness is an invaluable asset for the concerned communities. Prevailing incentive structures promote re-inventing the wheel rather than making the invented wheel better, therefore, obtaining funding for continued incremental development and maintenance of software has traditionally been a challenge. Obtaining funding for the governance of such software is even harder, both these issues must be addressed as a part of any software stewardship initiative.

A class of scientific applications are more amenable for adoption by some research communities because of their need for diverse models and methods to solve even the simplest problems in their domain (e.g. thermonuclear explosions in astrophysics, or laser plasma experiment modeling). Such codes require a huge amount of upfront investment, and they usually require a team effort to build. Once produced these community codes reduce the time to obtain research results for the communities they serve. They provide further benefits; the results they produce are more reproducible due to greater scrutiny, and a well-designed software can expand to serve communities beyond its original target. Research communities that have similar computational requirements can symbiotically improve the process of computation-based research for each other. However, the development and maintenance of such codes has its own set of technical and sociological challenges.

From the outset, a good understanding of more than one discipline is necessary in order to produce such multidisciplinary codes. Such efforts need to bring together a multidisciplinary knowledge base, and for members of the team from different disciplines to be able to converse with one another meaningfully. The knowledge base can be in the form of individuals with good working knowledge of many fields, a team composed of experts from many different fields, or a combination of the two. A team composed of developers from different disciplines may find that technical terms have unique meaning for each community which might differ between communities. Perception of creativity and complexity of work involved may also cause friction in multidisciplinary teams. However, the complexity and sophistication needed to get meaningful results from a wide range of software has reached a point where formation of such teams is unavoidable. Therefore, we as a community must develop norms and training standards that address these very legitimate concerns.

At a minimum the question of support for the governance and maintenance of such codes must be resolved where the survival of software useful for many users is not contingent upon continued research funding of any one group or institution. Additionally, the effort of scientists and developers who take on the responsibility of stewardship and quality assurance of these and other community codes must be recognized as being valuable to the community and should be rewarded as such.

Challenges in building a diverse workforce and maintaining an inclusive professional environment

Career paths to software engineering (software development, etc.) can include several education and professional backgrounds. Scientific software is developed by staff with a science background, CS background, math/engineering backgrounds, social science and design backgrounds. Developers may come from diverse professional backgrounds, from the commercial sector or the public sector, both nationally and internationally. Some of these paths are more widely accepted and better regarded than others, despite no strong evidence of their superiority in practice. While hiring practices attempt to be inclusive, important factors still introduce biases: who sees the job posting and thinks they might be qualified to apply to it; how the "screening" questions might discourage some applicants, or emphasize in-group traits that don't actually affect job performance. Because the job interview is very imperfect, personal recommendations and contacts are still practically important, and this of course reinforces some existing privilege networks.

Research software engineering groups often are cross-functional teams. Because of the complexity of the requirements, and because of the research environment itself, effective teams are often a mix of software engineers, computer scientists, and subject matter experts. There is a need to better understand the different perspectives and incentive systems associated with the development of scientific software.

Building a diverse and inclusive workforce - Diversity and inclusion tend to reinforce each other. As an individual, it may be hard to feel included if you are in the minority. As stated earlier, the hiring practice of reaching out to those you know reinforce biases and makes progress towards diversity more difficult. One approach that has worked is to have HR representation at venues like the Tapia Conference and the Grace Hopper Conference. Another approach is to invest in community outreach to universities with diverse populations. Internship programs could focus on bringing in diverse students and teaching software engineering and sustainability skills during the internship. In addition, there is a need to encourage students from diverse backgrounds earlier in their education especially at the K-12 level. Finally, it is important to make the hiring committee itself as diverse as possible (in every way), and encourage people in that group to speak up when they feel like there are biases that are not being addressed and considered. Attention to putting inclusive language in the job postings is also critical to encouraging diverse candidates.

Identifying and hiring the right talents- As HPC and more complex algorithms become necessary for scientific discovery, the ideal profile of a research team becomes broader. In particular, there is a wide range of needs between small software projects to support a single research group and production code to support an experimental facility. For scientific domains outside of CS, finding the right balance of talent with limited resources is difficult. There is often an effort to find candidates with both domain expertise and software development expertise, but these candidates can be difficult to find. Help can come from multidisciplinary masters programs that have been developed by some institutions. More such programs and other informal training venues are needed to mitigate this critical challenge.

Requirements, barriers, and challenges to technology transfer, and building communities around software projects, including forming consortia and other non-profit organizations

The ability to easily adopt *open-source software licenses*, and *uniform intellectual property policies and processes* at the Laboratories are required to better enable formation of communities around software projects. Each Laboratory has its own process for open-sourcing software. These processes can be complex and time-consuming to navigate. The objectives and motivations for open-sourcing software are inconsistent across the Laboratories. That is, some Laboratories readily embrace open-source software while others attempt to protect intellectual property rights at the expense of openness. These varying processes and motivations greatly hinder the ability to collaborate and build communities across Laboratories, within the DOE, and among government organizations, universities, and other non-profit organizations.

Programmatic goals targeted at building software communities are required in order to successfully promote collaboration outside of the original development group. Currently, much funding is allocated for R&D activities where software and community building around software may be an artifact instead of the main objective. Support is required for collaboration and community building that includes related activities such as software support, maintenance, and education/training.

Open-source licenses and outside engagement do not themselves guarantee the ability to transfer the technology to new developers (whether in the same institution or across institutions). Like many things, the best time to plan for transferring the technology is at the start. This is commonly accepted wisdom for data, and one of the goals of a data management plan; it should become part of the software development plan as well.

Overall scope of the stewardship effort

The list below enumerates what we consider the scope of a stewardship effort should be.

- Large spectrum of capabilities and services, and guidance and/or education available for product development and maintenance.
- Definition & promotion of best practices for sustainable software within scientific software context. These would range from standard software engineering best practices to those aspects unique to our stakeholder communities.
- Guidance on all aspects of sustainable software, including education and sustainable software engineering.
- Promotion, facilitation, and sustainment of long-lasting partnerships among and between stakeholder communities. Formation of partnerships between computer scientists, computational scientists, and domain scientists and engineers.
- Formation of communities e.g. for HEP specific needs.
- Help scale software on current and future computing landscapes.
- Help shape the future computing and software landscape.
- Provide up-to-date, cutting-edge knowledge base. Become the go to place for keeping code running, e.g. on DOE's next supercomputing platform or on the latest edge devices.
- Center point for sustaining software across all of SC, including ASCR, BER, BES, HEP, NP, etc.
- Institute "Sustainable Software Seal of Approval"

Management and oversight structure of the stewardship effort

Software projects may have a diverse range of governance structures. Some, especially smaller or newer projects, come from a single institution or have an obvious governance structure such as a single Principle Investigator (PI). Others are decentralized and community-driven with multiple sub-projects, with perhaps some de-facto leads.

Governance structures may - and should - evolve over time. While a project is small or new, overly-rigorous governance acts as a barrier to development. But an established software that is critical to other projects needs the resilience and reliability that rigorous governance provides.

To be sustainable a software project should be governed at an institutional or community level, as individual PIs and developers may change focus or move to different institutions: depending on an individual source of either funding or development work is a risk for sustainability.

Software stewardship is an ongoing process: although the greatest resource needs will occur during key phases such as initial development and while preparing for new hardware; without ongoing maintenance, debugging and user support, software becomes incompatible with its supporting ecosystem and ceases to be useful.

While the DOE is a significant stakeholder, software stewardship beyond the timeframe of a given DOE program (such as ECP or SciDAC) requires that other stakeholders be motivated and empowered to contribute to the stewardship of software projects. Other stakeholders in this context would include users of the software, user facilities dependent on the software, and developers of other software depending on it. Therefore efforts such as SciDAC and ECP should also foster a sustainable support community wherein other stakeholders can and do contribute significantly to the stewardship of the software. The DOE should also be prepared to participate in these support communities beyond the timeframe of specific programs, in preparation for when DOE next needs specific development of the software.
User facilities tend to be long-lived and have knowledge of software projects required for their facility and used by their users. User facilities are, therefore, in a good position for management and oversight of software stewardship. To enable this, user facilities should prioritize critical software according to the needs of their users; identify and support principal developers of critical software; and facilitate access to relevant testbed systems.

Assessment and criteria for success for the stewardship effort

The impact of stewardship can only be demonstrated if we devise ways of quantifying it.
This would require selecting a set of metrics from the outset for which initial measurements are made before the stewardship effort gets going for a selected set of projects. Along with that one could design studies that can evaluate the progression through periodic repeat measurements of selected metrics. Some of the possible metrics and questions for before and after could be:

- Increase in number of projects with rigorous verification program
- Projects that adopt on-boarding process and measure improvement in how quickly do new people become integrated into the project (i.e. start contribution non-trivial code)
- Length and age of feature-request / bug backlogs
- Success in helping projects transition to distribution and production phase
- For community oriented software, steady versus new users and also users switching to other software
- Whether a community software can expand its reach to communities with similar modeling and algorithmic requirements
- The ability of projects to attract diversity in team members
- Perceived value of various team members with different kinds of expertise and different career choices
- Correlate adoption of practices to change in outcomes (could be positive or negative)

In summary, the role of software in scientific research and discovery has evolved to the point where it should be treated as a scientific instrument, and given the same amount of care and scrutiny. Huge investments have been made in developing software for various aspects of research ranging from operations and supporting to simulations and analysis. To derive commensurate benefit from this investment we, as a community, must bring the issue of software stewardship at the forefront of research concerns, which is where it should always have been.