RFI Response: Stewardship of Software for Scientific and High-Performance Computing

Input from Argonne National Laboratory

December 14, 2021

Contributors

The response from Argonne National Laboratory is the summation of feedback from people with varied backgrounds including people from the Mathematics and Computer Science (MCS), Computational Science (CPS), Data Science and Learning (DSL), High Energy Physics (HEP), Biosciences (BIO), Energy Systems (ES), and Environmental Science (EVS) divisions, and from the Leadership Computing Facility (LCF) and the Advanced Photon Source (APS). Responses were contributed by the following people at Argonne through listening sessions, interviews, an email list, and an open document for direct entry of their comments.

Ahmed Attia, Ray Bair, Ramesh Balakrishnan, Prasanna Balaprakash, Anouar Benali, Franck Cappello, Kyle Chard, Mathew Cherukara, Ian Cloët, Emil Constantinescu, Mark Dewing, Sheng Di, Anshu Dubey, Jun Fang, Ian Foster, Rinku Gupta, Carlo Graziani, Iulian Grindeanu, Salman Habib, Kevin Harms, Steven Henke, Paul Hovland, Robert Jacob, Michael Kruse, Jeffrey Larson, Sven Leyffer, Nevin Liber, Zhengchun Liu, Ye Luo, Vijay Mahadevan, Todd Munson, Sri Hari Krishna Narayanan, Raymond Osborn, Michael Papka, Scott Parker, Hannah Parraga, Ken Raffenetti, Katherine Riley, Paul Romano, Rob Ross, Nicholas Schwarz, Emily Shemon, Patrick Shriwise, Rick Stevens, Rajeev Thakur, Richard Tran Mills, Thomas Uram, Alvaro Vazquez-Mayagoitia, Logan Ward, Stefan Wild, Tim Williams, Justin Wozniak, Xingfu Wu, Xingqiu Yuan, Hong Zhang, and Junchao Zhang.

Introduction

Building on the successes and momentum generated by the Exascale Computing Project requires a reinvestment in the research and development activities that drive innovation and fully supporting the operations of the leadership computing facilities to capitalize on the investment in equipment to deliver scientific impacts. Between these two elements is a software development and delivery element that is critical to the success of both. This element, termed *software stewardship*, works inside research and development efforts to transition software from research and development prototypes to hardened community software that is ready for deployment at the facilities, while providing support, testing, and packaging of that software to ease this deployment at the leadership computing facilities.

Software stewardship means that the software will deliver as expected when needed. Activities included in stewardship are: *porting and optimizing* the software for new architectures, *maintaining* software for existing architectures and updating the software as standards and dependencies change, *upgrading* the software to include new capabilities required by the scientific community, *sustaining* the community so that there is a continuity of people and expertise, and *working* with the users of the software to enable science and ensure applicability. The definition of “software” should be expansive and include not only compilers, libraries, and tools, but also scientific applications and services as well as software developed at experimental facilities. The scope should be on scientific computing with a focus on community software, and the ultimate metric of success must be the science and engineering achievements enabled by that software.

Maintainability is the capacity to keep the software working with limited resources and includes a software architecture that promotes code reuse, avoids code duplication, and is amenable to community contributions. It also includes updating third-party package interfaces as they evolve, testing and continuous integration, and scalability and performance testing.

Sustainability is the capacity to persist and flourish as the active community grows and evolves and includes scalable processes, such as for merge requests and continuous integration, and governance, developing new capabilities needed by users and making the code robust in order to grow the active user and developer base, sustained interactions between users and developers including tutorials and updated documentation, and stable APIs. It also includes the ability to adapt to new architectures and deprecate old architectures and unused capabilities.

While elements of a stewardship effort can be embedded within research and development activities, by exposing the stewardship elements as a separate activity, those highly trained stewardship specialists can focus on their craft and benefit from a well-defined career pathway, with the overall software quality improved by employing common practices. Similarly, while a stewardship effort could be included in the operations for leadership computing facilities, the stewardship of a particular software package transcends facilities and computing architectures. The software stewardship effort also benefits from working with applications to deliver capabilities required to advance their scientific missions. Therefore, we argue that software stewardship needs to be considered as a separate entity that focuses on its own unique mission and position between the research and development activities and the operations of the leadership computing facilities.

1 Scope (Q7)

The software stewardship scope should include porting and optimizing code, maintenance and core sustenance activities (especially when new architectures need to be supported), and a career path for these specialists; resources for the research and development that leads to new capabilities required by the scientific community; the equivalent of an “application user” program that supports working with applications to use the software being stewarded; defining a “software developer” program that supports transitioning software from a research and development activity to hardened community software that could be later incorporated into the software stewardship portfolio; and processes for governing the software stewardship portfolio, including the process to “adopt” and “retire” software.

Scientific applications as well as libraries and tools need to be considered in the context of software stewardship. Scientific output from HPC computing resources needs investment in scientific applications to maximize the investment in those compute resources. In addition, the choice of what tools or libraries to support is a direct function of the scientific applications using the systems. Optimal decisions regarding tool and library stewardship should be made as part of a strategic investment in scientific applications. The scope of the application investment should be significant to ensure scientific competitiveness and effective use of the large investments in HPC resources. Therefore, we advocate for a peer reviewed “application user” program to engage with application teams and ease adoption of the software stewardship portfolio.

In any application, library, tool, or other software product, there is research expertise that should stay connected to the domain experts that developed the software. Establishing a mechanism to support porting, functionality, and performance of scientific applications of broad need will be a great aid to the science community. We also advocate for a peer reviewed “software developer” program that would help research and development efforts make the transition to hardened software tools that can be incorporated into the software stewardship portfolio. This activity would help ensure the best return on investment in the software and would be important to feeding into a governance strategy to “adopt” or “retire” software and keep the software stewardship portfolio relevant.

The deployment of a piece of software on targeted systems should also be in scope for the stewardship effort. A great deal of software development for DOE supercomputers happens on laptops and other smaller systems. The most efficient and productive software sustainability effort will ensure that application and software developers have a robust development environment that can span laptops to supercomputers. The process of porting and optimizing software should require the optimization and sustained study of performance at compute facilities. If the targeted goal is to support a supercomputing ecosystem, participating and understanding performance over time is important.

Training, outreach, community building, and workforce development are also essential parts of a software stewardship activity. These activities help prepare new users for competitive proposals for the “application user” and “software developer” programs and contribute to the continued development of a world-class software stewardship workforce.

We now elaborate on topics for software dependencies; security, integrity, and scientific reproducibility; sustainable communities and community software; workforce development; infrastructure requirements; and other considerations.

1.1 Software Dependencies (Q1)

In any software stewardship effort, the packages and dependencies will be dynamic and change over time. A broad approach for the stewardship effort is suggested that takes into account system software, programming models, math libraries, analytics libraries, applications, services, and tools. A prospective stewardship program should select software packages based on input from users, developers, and facilities and should focus on packages that are portable across platforms and architectures.

Some basic packages and languages are ubiquitous in computational science and engineering. These languages include modern C++, modern Fortran, and Python. Non-vendor, community compilers should be part of the DOE standard toolkit. The PyTorch, TensorFlow, and Horovod packages are indispensable in AI/ML for science. MPI will likely remain ubiquitous for at least another ten years. Libraries providing FFTs, and sparse and dense linear algebra (including distributed execution like ScaLAPACK) are top-priority needs across the spectrum of HPC. Community software implementing these elements should both provide community implementations and define or use interfaces that can be coordinated with vendor-specific or commercial libraries. At a higher level, there is widespread dependency on solvers and other functionality in PETSc/TAO, which is likely to persist on a long-time horizon (ten years plus). At the highest level, there are programming model abstraction layers that have risen in use over the past five years and may continue to rise in the future. Two of the current key examples for moving DOE Office of Science applications toward exascale are Kokkos and SYCL. The most important capability not already present in all these established packages, languages, and frameworks is interoperability. An application implemented with SYCL and MPI may not be able to use a package implemented with OpenMP or PyTorch. APIs enabling interoperability should be part of stewarded software. An advantage of community implementations of all these dependencies is that it might be tractable to enable interoperability—as opposed to case-by-case custom interfaces to vendor implementations. Interoperability will be needed in the next two years to support complex workflows with many components combining traditional HPC and data-driven methods, some needing tight coupling, that are currently in development targeting the exascale systems.

Dependencies within applications will be a complex topic for the stewardship process. Dependency chains within an application can cause ripple effects from changes introduced by various levels, implying two considerations: one that the stewardship effort should take a broad approach on software, and another that proper stewardship will require analyzing dependency requirements across a variety of packages and determining the need to simultaneously support multiple versions of a package. Support of packages is also complicated by the fact that development may happen on platforms or environments that are different from user facility systems. An effective stewardship program will need to bridge differences in the leading edge of software development and deployment on platforms that may have different capabilities. Stewardship may need to develop a method to incorporate feedback from current and future activities into facility system acquisitions.

For the stewardship effort to be successful, the process must have a lifespan sufficiently long to have a measurable impact on software packages. Packages evolve over time and the stewardship process will need to adapt to these changes. The process should allow the effort to “adopt” new packages and gracefully “retire” older packages that may no longer be used. The stewardship effort also needs to address changes in the wider community over time. For example, Fortran has seen less support over time as the wider software development community has shifted to more modern languages such as C++ and Python. To address this change, Fortran codes should be migrated to newer languages or an investment must be made to increase Fortran support by supporting Flang or similar projects.

1.2 Security, Integrity, and Reproducibility (Q2)

Scientific reproducibility is essential for application developers to adopt libraries and other tools for their applications. Scientific reproducibility can take many forms. Some research communities demand bit-for-bit reproducibility before any changes can be committed to software repositories. Other communities argue that in the presence of roundoff, discretization, and other numerical errors, demanding bit-for-bit reproducibility is unrealistic and that probabilistic approaches, symbolic execution, roundoff error estimation, uncertainty quantification, or other verification strategies should be used to assess the significance of numerical differences due to software changes. It is further noted that demanding bit-for-bit reproducibility can impede or prevent performance optimizations including the use of nondeterministic execution schedules.

Scientific reproducibility hinges on program correctness. Consequently, effective methods for program debugging, testing, verification, and validation are needed. As important, or perhaps more so, is that these methods are used. Good software stewardship requires an incentive structure that rewards efforts to ensure program correctness in the form of funding for such efforts, expectations of such efforts as part of the peer review process, and consideration of such activities as part of career advancement. Program correctness is also a concern for software that is generated automatically, through artificial intelligence or other techniques. Mechanisms for verifying the correctness and tracking the provenance of such software are imperative.

Research software must, by its very nature, evolve and change as it is used to answer new scientific questions. Community software must, therefore, employ good software engineering principles from the initial design and subsequent refactoring through the entire software lifetime. It must not be considered in isolation, as some artifact to be locked away and preserved, but in the context of the scientific workflows to which it is applied. Effective and correct use of scientific software requires good communication. Software used outside its domain of applicability could yield erroneous scientific conclusions. Developers of community software should clearly communicate its capabilities and limitations. Users of community software should engage the software developers whenever its fitness for purpose is unclear. Good communication is also key to good software maintenance, since users of the software may expose bugs or defects not detected by the software developers. Working together ensures that the problem is found and corrected and that future testing is expanded to prevent similar defects from emerging.

Scientific reproducibility in the context of scientific software is inextricably linked to data: input data, output data, and, particularly in the case of machine learning, data used to generate a model. As such, science based on software should employ the same sort of mature data lifecycle assurance practices as other scientific undertakings. This assurance goes beyond the development of minimalist data management plans and includes the development of community standards for data preservation and FAIR (findable, accessible, interoperable, and reusable) data, expectations of and incentives for robust data management practices, and sponsor-funded resources for maintaining data integrity and security and tracking data provenance.

1.3 Sustainable Communities (Q6)

Successful communities around software have been built over time by focusing on building out a “global” community, which defines clear paths for contribution and participation. Communities should start by defining guidelines for contribution and governance of the software over time. The community can be expanded if the software is modular or component based such that users can pick and choose parts to use as well as contribute new parts. Flexibility of the software will encourage contributions and discourage fragmentation. At the same time, clear guidelines for new functionality or specifications must require example code and demonstrated value before acceptance to prevent expanding APIs with no clear path. A documented plan for release schedules and strategies improve the user/community engagement by providing transparency and setting expectations.

The software stewardship effort should also engage with existing communities that are important to HPC, such as standards bodies and specifications (e.g., ISO languages, Khronos specifications). These existing communities have a meaningful impact on the HPC community, and the stewardship process should help guide them toward capabilities important to DOE and away from features that would be detrimental. The software stewardship effort should encourage active participation in these organizations and fund effort to participate in them.

There are several barriers that make it challenging to build strong communities. There is lack of a path for funding to develop the communities from the original research funding for the project. A method to identify promising packages and provide funding to transition the package to a sustainable model and grow the community is needed. Sustaining software projects requires a clear career path defined for those who are contributors. These current issues raise a challenge that it is often more beneficial to create a new software product than contribute new capabilities to existing products. The stewardship process must incentivize and recognize developers for contributing to existing projects the same way existing programs have done for creating new software packages. The many existing redundant projects fracture the existing larger community into smaller ones, likely decreasing the value of the community. Stewardship will need to find a way to unify redundant packages to bring these smaller communities together while still fostering innovation. Sometimes these redundant projects form because applications cannot find or do not know that the capabilities needed already exist in another package. A strong documentation and outreach system is needed to allow existing developers to find and use already established packages as well as easily join those existing communities. The stewardship process may also drive unification by establishing more specifications or standards that would allow distinct software packages to interoperate and potentially merge communities.

Finally, the stewardship process must understand how the DOE communities can fit into the broader software communities outside of HPC. These communities often have different goals, but would be valuable to be part of because of the larger community investment.

1.4 Community Software (Q4)

When software packages transition from being used solely for research projects/internal use to being developed and used by a wider community, a significant burden is placed on the original developers to ensure the quality and maintainability of the software going forward. The level of effort needed to support community software depends on many factors—for example, what language it is written in, what platforms it is used on, how modular it is, the pace/frequency of external contributions, the environments in which it is used, and the dependency stack—but tends to be proportional to the size and complexity of the codebase itself. For large, widely used community software packages, developers may spend a significant fraction of their time on support activities.

There are a wide variety of tasks that contribute to the additional effort needed to sustain community software. These include but are not limited to: building and automating test suites; refactoring code; fixing bugs; implementing new features; improving performance; performing validation and verification; writing documentation; supporting new/additional compilers, platforms, and operating systems; dealing with API changes in dependencies; adopting newer language standards; reviewing external contributions; responding to messages on forums or mailing lists; training other developers; and running workshops, classes, seminars, and other events.

Even with funding for general support and maintenance of scientific software, there are many non-monetary impediments to performing these activities. One large non-monetary impediment for researchers at Argonne and other national laboratories is that often researchers do not feel like they will be given credit and recognition for activities that do not directly contribute toward research activities. In addition, the time commitment required to carry out the above activities can detract from other activities that are incentivized (e.g., publishing, proposal writing).

For software packages that are not yet open-source community packages, but where the authors may be considering going down that path, there are additional impediments. Some researchers believe that the software they develop gives them a competitive advantage in research and may be reluctant to “give away” their intellectual property. Some researchers may also prefer not to take on the additional work needed to support a community around their software, instead opting to focus on science and research activities.

Unfortunately, it is very common that the additional effort for these activities is performed in researchers’ spare time. In certain cases, researchers may build in “overhead” in a proposal to account for time needed for software stewardship activities. Targeted development activities are sometimes supported by non-DOE sponsors, but are often limited in scope and do not address the full spectrum of activities needed for responsibly stewarding a software package.

There is wide agreement among Argonne community members that additional resources are needed to support the above activities and that such resources should be separate from funding to support basic research and development activities focused on advancing science. Software packages are an important artifact in and of themselves, and without dedicated funding, ensuring that a particular package is robust and stable can be very challenging. Funding should also support interactions between domain and computational scientists.

1.5 Workforce Development (Q5)

Maintaining DOE's leadership in software innovation and stewardship requires a talented and diverse workforce. Fortunately, the broad societal impact of DOE science applications combined with the unique opportunities of DOE science and engineering research serve as strong incentives to attract and retain talented research software engineers.

Now more than ever before, we are struggling to recruit and retain the research software engineers necessary to build robust, performant, and scalable software. Even before the COVID-19 pandemic, talented staff were increasingly lured to industry for various reasons, such as financial benefits, career growth, and other opportunities. The size, complexity, and impact of DOE projects is becoming less unique, as industry positions can provide similar opportunities, such as deploying applications at enormous scale on in-house HPC systems and contributing to cutting-edge AI/ML research. The benefits of working in industry remain attractive, with opportunities for greater compensation, better benefits, and even more flexible work environments. The changes in work environment, in part expedited by the pandemic, have also leveled the playing field as we move to an international work environment in which developers can work around their schedules from their preferred locations. As such, there is no need to relocate and Bay Area salaries are accessible to those in locations with significantly lower costs of living.

The range of specific challenges for recruiting and retaining talented software professionals include non-competitive pay, lack of yearly financial incentives, ambiguity in terms of career progression and promotions, lack of recognition for software developers (as opposed to scientists) both internally and externally, lack of continuing education opportunities, difficulty including software engineers in proposals due to roles and salaries, and the current funding cadence that provides short-duration financial support and periods without funding, which in turn leads to a lack of certainty.

The national laboratories provide a mission and being recognized as an essential part of that mission can overcome some of these challenges. To attract and retain talented professionals, we must ensure that these professionals are recognized as a critical element of the DOE mission and its success, that they are part of a larger community of software professionals, and that there are established career paths and development opportunities. There is a growing movement in the U.S. and around the world to recognize a new role, Research Software Engineer (RSE), as an important contributor to research. There are now RSE conferences and communities that are working together to address these issues. Efforts in this area should consider partnerships with these wider efforts and enactment of the policies and processes coming from the community.

Focused development activities centered on growing the technical and/or leadership skills of the workforce in the software stewardship effort can also improve retention. In terms of development opportunities, we must recognize that there are multiple tracks for development, including developing and applying new technical skills, but also developing leadership skills and opportunities to apply those skills by becoming leaders in their organization. Certification as a Research Software Engineer might be an element of recognition and retention, while collocation of research software engineers could help establish a community of like-minded professionals, raise the level of recognition for these efforts, and help establish attractive career pathways.

We must also look beyond traditional science backgrounds, and instead engage talented individuals with diverse education and work backgrounds. However, attracting these individuals requires changes throughout the employment pipeline, from ensuring that advertisements are general and cast a wide net through policies for hiring even when qualifications do not necessarily match. We must also be willing to hire those that are "over-qualified" or that have skills in adjacent fields. A software stewardship effort should provide resources to support the development of such pipelines, for educating those within the labs responsible for hiring and interviewing, and to provide targeted on-ramp training for new employees focusing on those without traditional software engineering backgrounds.

The challenges associated with recruiting and retaining talented professionals from underrepresented groups is a broad topic facing the entire STEM community. Improvements must be made at all levels: from early education at schools and universities to inclusive hiring processes and work environments in the laboratories. Unfortunately, there are relatively few dedicated programs that provide pathways for underrepresented professionals. Studies have shown that unknown biases in job postings, candidate selection, interviewing, and even making job offers significantly affect hiring decisions. There is an opportunity for this software stewardship effort to foster growth in these areas by building pipelines to engage with individuals at schools and early in their careers. Such efforts will require investments in training and dedicated personnel to address these challenges broadly.

There is also a need to foster a culture of inclusivity and flexibility that considers work environment, location, work hours, communication, career pathways and support, mentorship, and many others. Such challenges often arise quickly as new employees face a steep learning curve. There is a clear link between learning environments and creating an inclusive environment for a diverse workforce. The non-technical barriers faced by new employees can be the deciding factor in whether or not they will continue to engage with the project (see "Social Barriers Faced by Newcomers Placing Their First Contribution in Open Source Software Projects"). Providing a positive and inclusive learning space for newcomers along with useful training material is important, not only for the software development itself, but also for the required software engineering tools and practices necessary to be an effective developer.

1.6 Infrastructure Requirements (Q3)

Maintenance, deployment, and support of software demands infrastructure including both hardware and software. Hardware needs appropriate access policies to be useful. Software takes the form of development environments and associated tools.

Hardware infrastructure. Developers need test and debug (T&D) systems, including parallel systems, aligned with facility architectures. They also need testbed systems for new architectures. Facilities can play a role in providing both T&D and testbed hardware. Policies needed include highly interactive access to T&D systems for debugging and modernized access to facility systems beyond ssh and the command line (e.g., single authentication, use of Jupyter notebooks to run jobs and examine results). Any convergence/interoperability of high-level access software with equivalents in high-level software for managing experiments and their data would be a step forward in data-centric computing.

Software infrastructure. Developers need productive laptop/desktop environments aligned with cluster and supercomputer environments, including support for relevant programming languages (modern C++, modern Fortran, Python) and the ability to build library/framework dependencies. As tools of the trade, they need fully-funded support for at least one source repository system (e.g., GitHub) and support for a continuous integration (CI) system that spans relevant software and hardware types. Facilities are necessary partners in defining and maintaining the developer software environment. Developers need functional debuggers available on all types of systems. The starting point needs to be laptop/desktop MPI and a GPU-aware debugger for a few MPI ranks that is also available on facility systems. Bugs and problems at scale are common in large-scale computational science. Hardware vendors, however, do not seem to care about parallel debugging. DOE has tried to support it via TotalView and DDT, but those still seldom work at scale.

Key capabilities. Scientific software developers need to be *productive*. The most productive development environment is an unshared laptop/desktop whose software environment supports the languages, compilers, libraries, and tools available on DOE facility systems. Simple parallel GPU-accelerated testing and debugging on that unshared system reduces unexpected bugs and concurrency issues that will show up on larger T&D systems. Similar development infrastructure on highly-interactive T&D systems reduces the likelihood of issues that show up only at leadership scale, where debugging is the most difficult. Robustness, functionality, and error handling of non-application software at leadership scale are nonetheless essential to ensure unimpeded and believable scientific results from applications.

Data-centric computing infrastructure. Increasing use of data and ML models changes software from code and executables to data and metadata—training data needing curation and provenance, and model data including DNN topologies and parameters also need curation and provenance. Reproducibility and community usability of the data and models are the science drivers. Where it is insufficient to manage data-centric computing software, developer infrastructure software needs to be modified or augmented to support it. Machine learning is an exploratory engineering technique. The ability to track numerous AI/ML training experiments will be needed to accelerate the development and testing process. Hardware requirements in the data-centric domain include a balance of high-bandwidth data transfer access and sufficiently-large distributed repositories. Required hardware policies include data access to distributed repositories for direct use and transfer out and/or in.

Infrastructure needs for Software Stewardship itself. The stewardship effort needs fully automated infrastructure to report on use of software (e.g., spack builds of scientific codes, dependencies, patterns of use, repository clones/commits/bug reports). Laptop/desktop users could opt-in to reporting, which would help prioritize support resources and would also help determine what software should be phased-out (and maybe phased-in) to the stewardship effort.

The needs for data-centric and AI-based computing are evolving, so the confidence in requirements is less than for traditional simulation-centric computing. Needs for access and curation of data are clear on a multi-year horizon; this is the common denominator of data-centric computing. The form of the models—neural networks of perhaps unexpectedly increasing depths or changing topologies or paramterized physics models informed by data-driven analysis—is less clear as research works toward interpretability and evaluating the effectiveness of different approaches in moving science forward.

Risks that the infrastructure addresses include correctness and reproducibility. These are addressed by CI testing and validation, version control of source code, and data and metadata versioning and provenance. Correctness is especially important at the largest and most expensive scales (LCF computing and APS Upgrade data volumes, for example).

1.7 Other Considerations (Q10)

The stewardship effort should be viewed as a holistic activity within the wider Exascale science and engineering effort. The stewardship component exists to facilitate the use of the scientific software developed for exascale systems and beyond. A key component will be the continued funding of scientific research that uses the software that has been developed, reinforcing the investments that have been made. The holistic approach should link the research and development activities, the sustaining activities, and the scientific usage together. A process that involves the full stack from end to end will maximize the return on investment.

The stewardship effort should look at the current and past efforts both inside and outside the DOE. The Apache Foundation, Linux Foundation, and GNU are successful programs for sustaining open source software projects from a wide set of domains. Within the DOE, the National Energy Software Center was established in the 1970s and continues today as the CODE project. The history of this project should be carefully examined to understand its successes and failures. The stewardship effort also needs to understand how it interacts with the DOE CODE project. (https://www.osti.gov/doecode/)

Another aspect that the stewardship process needs to contend with is the current expectations and assumptions around software that is developed. Currently, many researchers and developers see a model that prefers creating a new project over contributing to an existing project. If the stewardship process expects developers to “hand over” code to another organization for maintenance work, then this may create friction with the original developers. The stewardship process will have to consider the social engineering aspects related to the stewardship process and resolve these issues through communication and outreach.

A key obstacle is sustained, long-term funding for the effort. Just as DOE views its HPC facilities as resources that need to be maintained and updated long term, it needs to view the critical software developed under DOE funding as a resource that must similarly be maintained and updated long term.

2 Management and Oversight (Q8)

A successful stewardship effort will have several components that influence the management and oversight structure, including the main software stewardship activities, robust application user and software developer programs, and education, workforce development, and outreach activities for those inside and outside of the effort. The organizational model must support common services without duplication and include a leadership council, while allowing appropriate decisions to be made locally and provide organizational agility.

Moreover, the effort must represent the interests of key stakeholders including: the software developers, the application users, the leadership computing facilities, the laboratories and universities, and the DOE. DOE-developed scientific software has many users outside of DOE, such as the academic community, NSF supercomputing centers, NSF applications, users of scientific computing in industry (such as those represented on the ECP Industry Council), users across DoD, and the international user community. A DOE program manager for the software stewardship effort may be the most effective mechanism for interacting with the DOE. Engagement with other stakeholders would be

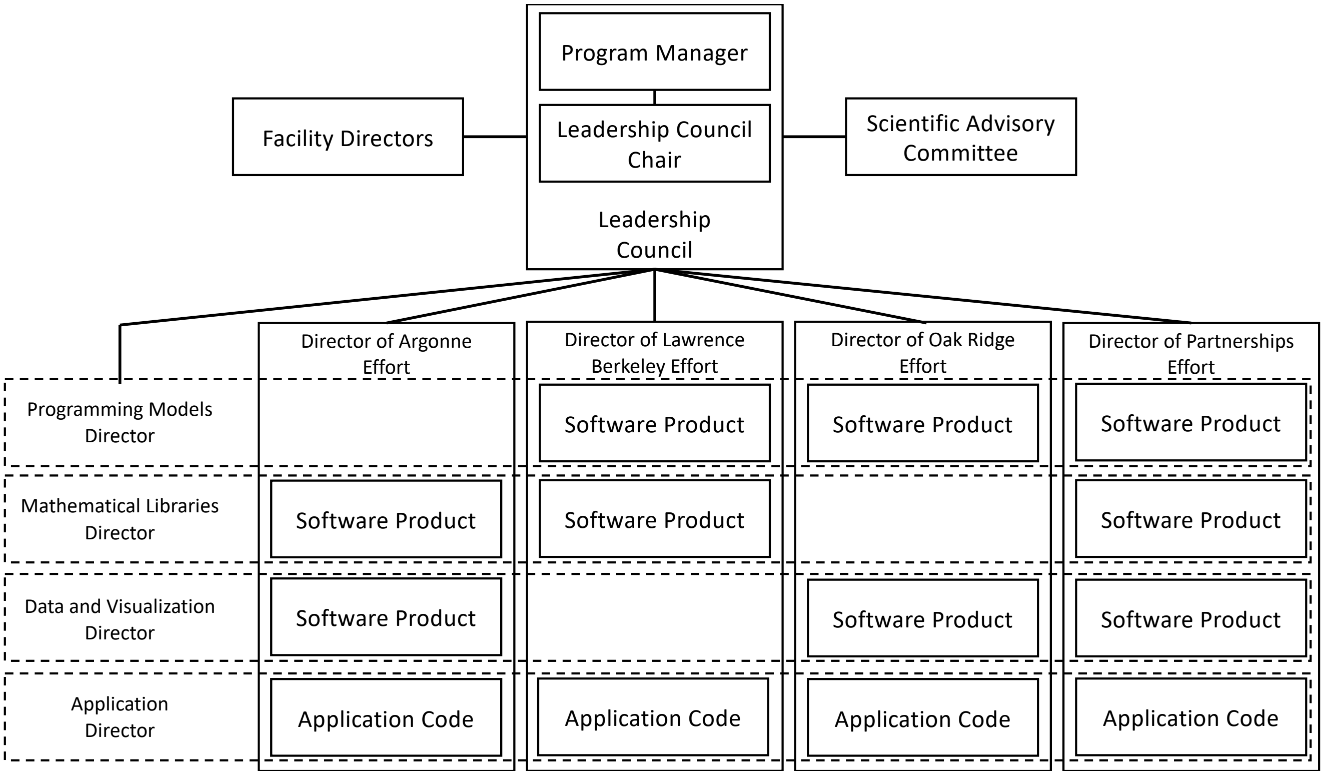
Program Manager
Leadership Council Chair
Leadership Council
Facility Directors
Scientific Advisory Committee
Director of Argonne Effort
Director of Lawrence Berkeley Effort
Director of Oak Ridge Effort
Director of Partnerships Effort
Programming Models Director
Mathematical Libraries Director
Data and Visualization Director
Application Director
Software Product
Application Code

Figure 1: Example organizational structure with geographic and functional unit directors forming a leadership council with a program manager that interacts with a scientific advisory committee and the facility directors. Note that not every geographic unit needs to have a software product in every functional category.

managed by the leadership council in coordination with the DOE program manager and a scientific advisory committee.

Close coordination between this scientific software stewardship effort and the Facilities Division at ASCR that provide relevant testbeds and other HPC resources is also required to make those resources available to projects funded under software stewardship. Individual projects should not need to independently negotiate with the facilities or apply to a program where only some projects get selected.

Therefore, we advocate for a hierarchical model as illustrated in Figure 1. At the top of the structure is the leadership council consisting of the geographical and functional unit leads and the DOE program manager. The leadership council interacts with the key stakeholders through a scientific advisory committee and coordinates with the experimental and Leadership Computing Facilities. The leadership council also establishes common policies and procedures, arranges annual peer reviews using a format similar to the reviews of the Leadership Computing Facilities, and provides for common support services. A primary institution would be responsible for: cross-cutting elements to avoid replication, such as training and workforce development events and user surveys; defining the equivalent of a peer-reviewed “application user” program that provides resources to work with applications that apply for support in using the software being stewarded; defining a peer-reviewed “software developer” program that provides resources to support transitioning software from a research and development activity to hardened community software that could be incorporated into the software stewardship portfolio; and a process for retiring software from the software stewardship portfolio. We envision that the resources for the application user and software developer programs would consist of tiger teams drawn from the subordinate units and include some resources for developing new capabilities to support the application needs. The impact of the user program could be measured by the new science enabled, and coordination with the Leadership Computing Facilities would be required for time on HPC machines and testbeds.

The subordinate units can be organized by function, such as math libraries, programming models, and applications, or by geography, such as the stewardship effort at each laboratory. From a logistics perspective, we believe the primary organization of the subordinate units should be by geography, as larger blocks of funds can be sent to a geographic unit with decisions made locally on the software to include in the stewardship effort in coordination with the leadership council. Moreover, the people involved in the stewardship effort in the geographic unit are collocated, which provides greater opportunities for workforce development and movement among individual software efforts. Within each geographic unit will be individual stewardship efforts associated with particular software products. These individual stewardship efforts could subcontract with university and industry partners as needed. We would define a secondary organization of the individual stewardship efforts by functional units. These functional units, such as math libraries and programming models, would span the geographical units and mimic the organization of the software technologies in the Exascale Computing Project via Software Development Kits. These functional units coordinate the synergistic activities and community policies common to that function. The leadership council consisting of the geographic and functional unit leads and the DOE program manager would ensure that the necessary coordination occurs across geographies and functions. The decision-making authority of the geographic and functional unit leads would be determined in coordination with the leadership council. University and industry partners that are directly funded by the stewardship program would have a primary home in a partnerships unit and be included in the functional units. We believe that oversight of the individual software efforts should focus on results and not be burdensome, but at least include the equivalent of annual field work proposals and a peer review of the individual software efforts on a regular cadence that includes the geographic and functional unit directors in the reviews.

This organization provides the necessary communication channels for an effective stewardship effort, but is not prescriptive of where the geographic units reside at a particular institution or the funding model. The geographic units might be their own separate entity or be integrated as a subunit of a division or leadership class facility at a national laboratory to take advantage of existing management structures and resources. The majority of personnel within each geographic unit should be research software engineers, augmented by performance engineers, software engineering researchers, and domain experts when necessary. We expect each geographic unit to be unique, driven by the evolving needs of the scientific community supported. With regard to funding, all funding could be routed through a single institution with subcontracts to the subordinate units or funds could be provided directly to each geographic unit for distribution. While any of these choices of location and funding model are viable, the most important organization aspects need to remain intact: the clear lines of communication and coordination and the decision making processes provided via the leadership council.

3 Assessment Criteria (Q9)

To be successful, software must be deployed, used, and supported to deliver results that advance science. Software success metrics must quantify these four qualities. There is no single metric that applies equally to all software or fully describes any software. We must use multiple metrics in a rational and flexible way. Metrics for success of a software stewardship effort would naturally derive from metrics measured on the stewarded software, together with evaluating the value of science results—software case by case and in aggregate. Metrics should be data-driven, with automated data collection wherever possible. Compute facilities can help with collecting and providing data in that context.

The stewardship program itself should be evaluated first and foremost by scientific returns of applications supported directly through the program or indirectly via library usage. Scientific returns must be aligned with either DOE research priorities or with the goals of non-DOE research executed on the LCFs. Aggregating the metrics across the set of stewarded software provides the needed information. Of secondary importance is how well the stewardship program maintains high metrics across the other software qualities of deployment, usage, and support. Admitting and maintaining stewardship of high-performing software and removal from stewardship of low-performing software are two measures of this success.

Metrics for deployment. There are two approaches to deploying software on systems: (1) developer installation centrally at a facility; and (2) developer documentation for the scientist/user to install individually at a facility or personal/local system. Established metrics such as downloads from GitHub play a role. The number of software releases per year is one measure of vigor and progress, as are counts of pull requests. A lack of deployment on large-scale facilities or T&D systems should count against software that emphasizes HPC. In the current and likely near-future environment of increasing heterogeneity in computing hardware, portability measured by deployment across testbed systems is important.

Metrics for usage. Established metrics such as citations for software in publications are useful, even though users of software who are not the developers of the software do not always include proper attributions. There is a push in the community toward reproducibility and data provenance in publications; we should leverage what we can from that effort, especially if this data is available from publishers in machine-readable format. Software that takes the form of data-derived models rather than applications or libraries must be subject to the same usage metrics and should be citable.

Metrics for support. Established metrics such as outstanding and retired reported bugs play an important role and are amenable to automated collection. These fail to capture, for example, direct email Q&A that helps users debug usage issues and the existence of unresolved bug reports with serious scientific consequences or that are show-stoppers for certain users or systems. One way to approach these is through surveying users. While not an automated metric collected passively, surveys can be improved with proper standards for design and statistical evaluation. Good documentation and error handling can reduce support requests through ticketing systems. Tracking changes to documentation must go hand-in-hand with tracking changes in source code or data-driven models.

Metrics for results. The numbers of citations in domain-science publications, weighted by quantitative measures of journal impact, play a role as semi-automated metrics for software contributing to scientific results. For software that primarily targets leadership-class computing (for example, numerical methods only tractable on leadership-scale systems), the number of publications will naturally be lower and the expected impact will be higher. Metrics must account for these differences. One metric could be successful multi-year renewals and repeat successes with INCITE proposals. Assuming that awarded production projects on the LCFs are scientifically valuable, the enormous hardware, power, and staff costs of running the project's calculations there must be a heavy weighting factor on the value of the scientific results.

All software packages (applications, libraries, data-derived models) under the stewardship program should be evaluated by these metrics. Transparency in what the metrics are, so that the same metrics may be collected for non-stewarded software, will drive evaluation of new software that should be added to the stewardship program. Poor performance on the metrics by software under stewardship will drive decisions to gracefully remove it from stewardship.

Conclusion

Realizing DOE's mission to advance science and engineering, while maximizing investments in research and facilities, requires an equally significant investment in the software. The ASCR software portfolio is broad, encompassing systems software, programming models, math libraries, analytics libraries, applications, services, and tools. Addressing stewardship and sustainability concerns requires engagement with an equally diverse range of stakeholders, including the researchers and engineers who develop the software, the administrators of hardware on which that software is deployed and used, and the enormously diverse group of users for whom the software is a critical tool for their success. Ideally, the software stewardship activities would be a component of a much larger ($700M–$1.2B) computational science program, aimed at conducting cutting-edge science on the exascale computing facilities and in support of experimental facilities. This scientific research would use software developed by ECP, SciDAC, and other DOE programs and with a requirement that the software continue to evolve and expand, as driven by research requirements.

We aim to think beyond what has been achieved with prior efforts to redefine the way that ASCR supports and sustains software. This approach must be driven by data to quantify the costs and impact of software, the success of the stewardship effort, and the benefits to the DOE mission. The effort must also be broadly defined to consider software holistically, for example by defining policies and processes that directly support software stakeholders and the individuals primarily responsible for creating software—from ideation through research and development and on to the sustainability phase that follows. Such initiatives can build on prior efforts both inside and outside DOE to foster software sustainability. However, they must also consider the need to grow and retain the talented individuals that make up the ASCR software community, diversify the workforce and develop an inclusive and productive environment, provide training and support to democratize best practices, and provide accessible and representative infrastructure to reduce development overheads and increase the robustness of software, among others. Software stewardship activities should also embrace the first tenet from the Manifesto for Agile Software Development that one should "value individuals and interactions over processes and tools."

Overcoming existing challenges must consider new ways of supporting software, putting in place incentives for building and supporting crucial software that is relied upon by others, providing certainty around the software lifecycle and avoiding funding-to-funding discontinuity, and reducing incentives to build new software, while ensuring that new ideas are encouraged and fostered. Finally, it is important that this process be tightly coupled with those that use software to ensure that funding is mirrored to support those applications and foster close engagement between these groups.