# Oak Ridge National Laboratory Response to 86 FR 60021, Stewardship of Software for Scientific and High-Performance Computing


David E. Bernholdt, Joshua Brown, Franklin Curtis, Christopher Davis, Thomas Evans, Ana Gainaru, David Green, Steven Hartman, Scott Klasky, Ross Miller, Kenneth Moreland, Slaven Peles, Norbert Podhorszki, David Pugmire, Dale Stansberry, and Ruonan Wang

**December 2021**

OAK RIDGE National Laboratory

ORNL IS MANAGED BY UT-BATTELLE LLC FOR THE US DEPARTMENT OF ENERGY

# OAK RIDGE NATIONAL LABORATORY RESPONSE TO 86 FR 60021, STEWARDSHIP OF SOFTWARE FOR SCIENTIFIC AND HIGH-PERFORMANCE COMPUTING

David E. Bernholdt
Joshua Brown
Franklin Curtis
Christopher Davis
Thomas Evans
Ana Gainaru
David Green
Steven Hartman
Scott Klasky
Ross Miller
Kenneth Moreland
Slaven Peles
Norbert Podhorszki
David Pugmire
Dale Stansberry
Ruonan Wang

December 2021

Prepared by
OAK RIDGE NATIONAL LABORATORY
Oak Ridge, TN 37831-6283
managed by
UT-BATTELLE LLC
for the
US DEPARTMENT OF ENERGY
under contract DE-AC05-00OR22725

**CONTENTS**

## INTRODUCTION

Oak Ridge National Laboratory (ORNL) is managed by UT-Battelle, LLC for the Department of Energy (DOE). With more than 5,700 staff and an annual budget of $2.4 B, it is the largest multi-program science and technology laboratory in the DOE complex. ORNL is home to nine national user facilities, including the Oak Ridge Leadership Computing Facility (OLCF), the Spallation Neutron Source (SNS), and the Center for Nanophase Materials Sciences (CNMS), and a broad array of scientific activities in support of the DOE mission.

Software permeates the modern scientific enterprise, and ORNL is no exception. It is perhaps most obvious in the area of computational science, where modeling and simulation and data analytics may be deployed at massive scales on OLCF or other high-performance computing (HPC) systems, all the way down to individual workstations, in pursuit of scientific discovery. But software is also central to experimental facilities, such as SNS and CNMS, where it is used in the design and control of the facility and individual instruments, and the data collection, reduction, and analysis, increasingly in combination with modeling and simulation software. And similarly – at smaller scales – individual researchers and laboratories depend on software to collect, organize, analyze, and understand data.

Scientific software may be a tool, used in conducting research, or it may be the object of research itself; sometimes both. It may originate within ORNL or outside – ORNL staff participate in many software-focused projects which involve contributors at multiple institutions, including various national laboratories, universities, and companies. While open source licensed software is widespread, proprietary or commercial software also plays an important role in many environments.

In an organization as large and diverse as ORNL, concerns about the sustainability and stewardship of the software that's central to our scientific mission also vary widely. Historically, such concerns have been dealt with mostly at the level of the individual projects that create and use the software. However, as our dependence on software has deepened, the software itself has also grown in size, sophistication, and the level of integration – the increased reliance on specialized software packages produced by expert teams that may be distinct from the core development team for a software package. And as we have become more dependent and more interdependent, researchers at ORNL, and across the scientific computing community, have begun to recognize that more explicit attention is needed to the sustainability and stewardship of scientific and HPC software to continue our ability to leverage the capabilities that powerful computers, large and small, can bring to our science. Thus, we welcome this opportunity to provide input to the Department of Energy on this topic.

## 1. SOFTWARE DEPENDENCIES AND REQUIREMENTS FOR SCIENTIFIC APPLICATION DEVELOPMENT AND/OR RESEARCH IN COMPUTER SCIENCE AND APPLIED MATHEMATICS RELEVANT TO DOE'S MISSION PRIORITIES

For an organization of the size and breadth of ORNL, it is impossible to provide a detailed catalog of specific dependencies in need of stewardship. However, we can say a number of things about the *kinds* of scientific software needing stewardship.

First, we feel that scientific software in general would benefit from greater attention to sustainability. Although sustainability is not an immediate concern in many cases, if the software we rely upon is developed with greater attention to sustainability and best practices, the easier it is to provide the needed stewardship should the need arise. Since the sustainability of software is closely related to other "-ilities" of software, such as understandability, reproducibility, reliability, etc., an increased emphasis on sustainability will bring general benefits to users of the software even when stewardship is not a specific

concern. This requires education and training of software developers, both human and technical support and assistance to improve practices, and, probably, increased expectations and incentives from institutions and research sponsors. We will discuss these matters further below.

Second, it must be noted that lack of confidence in the sustainability and stewardship of a software product is often a primary impediment to its uptake by others. Projects supported by DOE, in general, and the Office of Advanced Scientific Computing Research (ASCR) particularly, produce a great deal of software as products of their research – embodying new numerical solvers, new programming environments or runtimes, or other capabilities. Often, a primary goal of such research is to address challenges faced by other researchers, and an important way of demonstrating the value of the new work is through adoption by others who find that it helps them meet their challenges. However, decisions about whether to adopt an external software dependency hinge not only on whether the tool is useful, but also the comfort level of the team with the long-term viability of the dependency. Are they comfortable with the quality of the product and that they will be able to rely on it for long enough to pay back the effort required to integrate it into their code? This gets directly to the sustainability of the software and how it will be stewarded through the coming years. Unfortunately, it is quite common for technically useful new tools to be passed over because a team does not trust that the software will be maintained or ported to new computer platforms or otherwise kept current. Which often comes back to the fact that, historically, DOE (and other U.S. research sponsors) generally fund *research and development* (R&D) rather than *stewardship*. Consequently, research teams must make decisions about the balance between producing new scientific results and maintaining the resulting software and would-be users. To one extreme, researchers may produce lots of great new scientific advances that end up having little practical impact on the larger scientific enterprise because others are not comfortable relying upon their software products. It is not uncommon for this situation to lead to others reimplementing the desired capabilities. Experience shows, however, that the result often falls short – either in terms of fidelity of the copy to the original (taking shortcuts to reduce the cost which provide reduced capabilities or require later rework) or simply in terms of ending up with the community trying to maintain two competing codebases with (roughly) equivalent capabilities. To the other extreme, researchers may provide well-supported software that many are willing to rely upon but fail to advance the state of the art and lose users and/or funding as a result. Making these considerations more transparent and expanding the discussion to include other stakeholders beyond just the research team (including users, would-be users, and sponsors and would-be sponsors) would facilitate decisions that provide greater overall benefit to the research community in the long run and make better use of the sponsor's funding.

Finally, as we observed above, not every software product created or used by an institution like ORNL needs explicit consideration of stewardship all the time. Much research software is produced for individual use or as a prototype – to test out ideas. As a software product becomes more widely used and/or progresses towards a "production" level of quality and support, concerns over its sustainability and stewardship are likely to increase. With budgetary limitations preventing investment in the stewardship of every piece of software that someone (a developer or a user) might think ought to be sustained, some concept of "impact" could be used to help make triage decisions. Our notion of impact combines the size of the user base, the "production readiness" of the software, and the importance of the software (or its uses) to DOE programs and overall mission. We offer no fixed formula for weighting these factors; in general, each case should be an individual discussion among the stakeholders (producers, consumers, and sponsors of the software product). DOE user facilities, for example, rely extensively on software to support their operations and their users' research, and some or all of that software might be high impact with respect to stewardship considerations. But by the same token, a software package that has become foundational to the research activities of a particular scientific community ("community software"), may also have a high impact for a specific program of research. Software tools, often, but not exclusively, produced by computer science and applied mathematics researchers, may reach high levels of quality and wide adoption across many different use cases – another example of high impact.

Once the decision is made that a particular software package or collection of software is high-impact *and* in need of more explicit consideration of stewardship, how to go about it is also, generally, an individual consideration. There are certainly some situations where significant collections of software can be considered together, in an "ecosystem" approach, such as that pioneered by the Extreme-Scale Scientific Software Development Kit (xSDK, https://xsdk.info/) effort, which encompasses many numerical mathematics libraries, and at even larger scale by the Extreme-Scale Scientific Software Stack (E4S, https://e4s-project.github.io/) effort, part of the DOE's Exascale Computing Project (ECP). These approaches are predicated on being able to identify collections of software that are widely used across a well-defined set of computer platforms in a limited set of use cases. Computational science (primarily modeling and simulation) on DOE high-performance computers is a good example of an ecosystem that's both large enough and sufficiently well characterized for such an approach to potentially be effective. This is the space in which both xSDK and E4S operate. However, it must be noted that the primary value of the ecosystem approach is where common platform support, and package compatibility and general interoperability are primary concerns. Each individual package in such an ecosystem will still have more individualized concerns about sustainability and stewardship that will need to be addressed. Further, across ORNL and across DOE, there are many other situations that are less amenable to an ecosystem approach. These include more specialized or unusual use cases, different computing platforms (i.e., experimental or testbed systems not in wide use, or smaller-scale systems of less relevance to the HPC computational science community), or situations where the breadth and generality of the ecosystem approach is not beneficial or may even work against a particular case.

Consequently, we believe that *in general* stewardship of scientific software needs to be considered on a case by case basis, but at the same time would benefit from increased transparency and openness about the needs and methods for software sustainability and the development of common models or rubrics for making decisions about stewardship to make it easier for producers, consumers, and sponsors of scientific software to work together to provide best value for DOE's substantial investments in software.

## 2. PRACTICES RELATED TO THE SECURITY AND INTEGRITY OF SOFTWARE AND DATA

Security and integrity concerns about both software and data have multiple facets.

One facet is software and data to which access must be limited due to security concerns due to export control (including the International Traffic in Arms Regulations (ITAR) category), and Protected Health Information (PHI). Facilities accredited for such information are relatively few; development environments accredited for such information are even less common and rarely support the current level of capabilities that are considered "standard" in the more open world. The situation hampers collaboration, especially across institutions, and can make it harder to implement best practices for software development and sustainability.

A second facet is the increasing use of distributed computing to support scientific workflows which, for example, may connect instruments or other data sources to systems for data reduction or analysis pipelines, modeling and simulation, archival, etc. While software running on a single system is not without concern, distributing the work across multiple systems, perhaps crossing administrative boundaries, and, typically involving numerous software packages, opens a much greater surface area to potential attack. Though institutions and facilities typically have cybersecurity teams to watch over their computing environments and set requirements for the software that runs on them, developers of research software may see them as obstacles to their scientific progress as much as defenders of their working environment. We believe that this is because in many cases the developers of research software have little or no training in cybersecurity concerns and how to address them, and cybersecurity teams do little or no outreach to such developers,

focusing more on the "business" activities of researchers and other staff – detecting malicious email and web sites, and other external threats.

Third, we come to more fundamental considerations of security and integrity which apply to software more broadly. Developers, users, paper and proposal reviewers and readers, and sponsors need to have confidence in the software used to produce scientific results, which ties directly back to concerns of transparency, reproducibility, reliability – in essence, the security and integrity of the software. Though such considerations have been the subject of increasing attention throughout science (software-intensive or not) in recent years, there is still ample room for improvement. Practical experience and some research have identified a variety of software development "best practices" that can increase developer productivity, and software quality, sustainability, and reproducibility. Scientific software teams are increasingly adopting many such practices. Version control, peer code review, and automated testing are three significant examples among many. But some potentially beneficial practices remain underutilized in scientific software. For example, static code analysis can be a very effective way to detect issues in code that may result in security vulnerabilities or threats to scientific reliability. High quality static analysis tools are primarily available as commercial products, often fairly expensive for R&D projects that have to decide how best to spend limited funding and may not have been exposed to the potential benefits. As commercial products, they may also ignore small markets that are unlikely to be profitable, such as supporting software written in the Fortran language, which is still widely used in scientific software, but rarely in the broader world of software.

With the increasing interdependence of software – one package relying on others, developed and maintained separately – another basic concern about the security and integrity of software has to do with our confidence in relying on the security and integrity of those third-party dependencies. Often, such dependencies originate outside of ORNL or even outside of the DOE complex, but nevertheless, we must also be able to have confidence in them. In other words, it may be important to take a hand in the stewardship (vulnerability scanning, testing, bug fixing, and perhaps more) of software packages that we did not develop, but merely use.

Finally, the long-term stewardship of scientific data, which underpins considerations of reproducibility and the scientific method, also raises concerns about software stewardship. For data to remain accessible and usable, software that knows how to read and write it must also be maintained. Consequently, data management software may require special attention from a stewardship perspective.

## 3. INFRASTRUCTURE REQUIREMENTS FOR SOFTWARE DEVELOPMENT FOR SCIENTIFIC AND HIGH-PERFORMANCE COMPUTING

The rise of free and low-cost public platforms for software development and related services has really revolutionized the practice of collaborative scientific software development. Such platforms tend to be at the leading edge of capability and offer environments accessible to researchers at multiple institutions (a common case for software development activities at ORNL and across the DOE complex) with a minimum of friction. However, when developers need paid services, even though the cost might not be significant, institutional purchasing and sometimes security rules often kick in and the procurement of such services may incur costs in staff time and effort to jump through the hurdles that seem far out of proportion with both the cost and the risk associated with the service.

Similarly, as noted above, when software (or data) requires higher levels of protection (e.g., export controls, non-disclosure agreements, etc.), it tends to become much harder to find hosting environments that provide the same level of capability and flexibility as in the public services. Often, an institution or facility will locally install "enterprise" versions of popular products, but due to either the vendor policies or institutional

choices, or both, the local installation often significantly lags the capabilities of the public services. It may be worth exploring whether vendors can make leading-edge products available in settings that can satisfy higher levels of confidentiality, integrity, and availability. This kind of thing may not be cost-effective for an individual institution like ORNL, but at the scale of the DOE or even the whole of the U.S. government, it is likely that much better terms could be had.

For more costly commercial software development tools it may be worth encouraging institutions to explore site licensing arrangements to make the software available at a lower cost than if individual researchers or projects acted independently. This might be facilitated by creating opportunities for researchers to "test drive" tools and receive appropriate training so that they can evaluate them and understand their value in conjunction with their particular projects. Such a service might be provided by a "software excellence center", which we elaborate on below.

An important part of the leading-edge science that ORNL and other researchers pursue entails the use of leading-edge computational facilities. Scientific software must be ported to new platforms and tested across all relevant platforms. Though DOE funds a great deal of computational infrastructure already, such as the Oak Ridge Leadership Computing Facilities, access to them is often highly competitive and significantly oversubscribed – much like funding for smaller scale R&D projects. And as with research funding, the focus is on the scientific results from the computer allocation, while needs for software maintenance and testing tend to be hidden. In some cases, software products that may be important to others may have a hard time obtaining allocations on leading-edge systems because the development team may not have a scientific agenda that would be considered competitive for allocations, and therefore have a hard time supporting their software on a platform where others might be highly dependent upon it. As with research funding for software-intensive projects, we advocate for greater transparency and perhaps greater flexibility in some cases, to ensure that scientific expectations are appropriately balanced with adequate levels of testing to ensure quality and reliability of the software behind those results.

Other leading-edge computer systems may be available as testbed platforms. These are often early demonstrations of new technologies or approaches. And here too, it is useful to be able to have access in order to port software and help evaluate the platform. However, it is often harder for researchers to obtain access to testbed systems, especially if they're at other institutions. It may be useful to consider treating testbed systems as (part of) user facilities in order to simplify and expand access.

## 4. DEVELOPING AND MAINTAINING COMMUNITY SOFTWARE

The "community" associated with a software product can grow in essentially two dimensions. One is the users of the software; the other is those who contribute to the development of it. Growth in either or both dimensions will benefit from clear articulation of a governance model for development, and for the overall software project, and from increasing the level of "professionalism" in the management of the project. By professionalism, in this context, we mean more emphasis on the disciplines of software engineering and project management and correspondingly reduced emphasis on the "research" aspect within the core project team.

As the user base of a software product grows, the core project team will need to spend increasing portions of their time on software maintenance (stewardship) and user support activities. Quality software with good documentation can be easier to support. In many cases, it is possible to engage members of the user community as active participants in user support (e.g., a mailing list for users of the software where questions posed by one user are often answered by other users, rather than only members of the core project team responding). But overall, more users tend to mean more and different use cases, and platforms leading to more issues with the software. Users may request features that are not on the core project team's roadmap

and the team must decide whether, and how, to satisfy those requests. As the user base grows, the software becomes more of a product, and needs a support team that is willing and able to treat it as a product. This can be hard to reconcile with the expectations typically placed on an R&D project. Others may be using the software as a tool in their research. But for the core project team, supporting those users effectively is very different from exploratory and innovative science being carried out within the software. It might be helpful to such software teams if sponsors would consider funding the maintenance and support for the community separately from R&D within the software – once again, a principle of transparency that allows a recognition of the costs and expectations for the stewardship efforts distinct from new science within the software. And it may be useful to encourage the employment of (research) software engineers and software project managers rather than researchers to staff the stewardship component.

As the contributors to a software product, including more and more people who are outside of the core project team, the core team needs to spend an increasing portion of their time supporting these contributors (assuming they want them to continue to contribute). Contributors may need assistance to become effective contributors of quality code. Their code may require more work to integrate into the code base, including testing on platforms that the contributor may not have access to but are considered important for the product to support. These activities take time away from the core team's own development efforts. And although outside contributions can significantly enhance the software package, contributors develop for their own ends, which are not necessarily aligned with the R&D roadmap of the core team. Once again, there is a decision to be made as to whether to invest the time and effort in welcoming and supporting outside contributors that we believe benefits from transparency about the costs of this kind of stewardship.

## 5. CHALLENGES IN BUILDING A DIVERSE WORKFORCE AND MAINTAINING AN INCLUSIVE PROFESSIONAL ENVIRONMENT

Recruitment itself can be a challenge in computing-related areas, where the extensive "tech economy" provides opportunities for people with computing skills and can often pay substantially more than a national laboratory. Building a diverse workforce in this environment multiplies the challenge. However, we believe that embracing new ways of staffing software-intensive projects, with increased involvement of research software engineers, may offer new opportunities to build a more diverse and inclusive workforce.

Historically, R&D projects at national laboratories like ORNL have been dominated by *researchers* who hold (most commonly) PhDs in a relevant technical area – computer science, mathematics, or the physical sciences. Researchers conceive and drive the research projects, as they should, but typically, they are also writing the bulk of the software. However, most researchers have little or no training in software engineering in their formal background, and even those who have (for example in undergraduate computer science programs) find that their coursework in software engineering bears little resemblance to the practical world of leading-edge scientific computing, especially when targeting high-end HPC environments.

The rise of the research software engineering (RSE) movement provides a means to help professionalize software development in research settings, while at the same time providing workforce-building opportunities by creating openings for people with different backgrounds than institutions like ORNL have traditionally targeted and the potential to reach further back into the educational pipeline, where diversity is greater. Research software engineering is a term emerging from the United Kingdom in the last ten years or so, but which has been quickly embraced around the world, including many institutions in the United States. RSEs are, briefly, professionals who focus on the engineering of research (or scientific) software. In the kind of career tracks that are being established for RSEs, they are not expected to lead research, but rather support it through software development, software engineering, and related activities. Institutions that have RSE groups, like ORNL, set expectations that RSEs are software professionals, who gain and

utilize knowledge and experience with software engineering practices and help to convey that knowledge to others as part of their contributions to R&D software projects. Many who identify as RSEs hold PhDs in a technical area, but have found that they're more interested in the software side of science than they are in leading the research side per se. But there are plenty of RSEs with Master's or Bachelor's degrees. Some technical background can be useful to help understand the models and algorithms they are helping to implement, but RSEs are expected to be generalists in the technical sense, working across projects and across domains.

The recognized RSE role is relatively new, and organizations like ORNL are grappling with how to incorporate them into the organizational structure and culture. But we believe this is already a step forward. Even before there was a term to associate with the role, it was common to find people scattered throughout an organization like ORNL, who focused more on software development and software engineering than on research leadership. In a culture that emphasizes research, where the default is to evaluate staff based on typical research metrics, like papers published, talks given, and proposals funded, it is easy for the software developer to be marginalized and not receive due recognition for their contributions. Creating the RSE role and forming RSE groups is an important step to including and recognizing these contributors to the scientific mission. But more is needed, both within the organization and externally, to fully include RSEs as a critical part of the workforce supporting software-intensive science in the DOE. Research sponsors could facilitate this by helping to normalize the presence of RSEs on R&D projects, and perhaps even setting expectations for their involvement as part of setting expectations for higher levels of software quality and sustainability, as appropriate to the scale and scope of the project.

Because we can be more flexible about the background of RSEs, we also believe that increasing the role that RSEs play in scientific software can also help to address issues of diversity in the workforce. It is well recognized that the educational pipeline is leaky – as the level of education increases, under-represented minorities drop out of the process at higher rates. The ability to recruit RSEs from the Master's and Bachelor's level rather than just the PhD level will, therefore, provide a more diverse pool to draw from.

But focusing on Master's and Bachelor's level candidates also poses a challenge. As noted above, few students at these levels have useful experience with software engineering, especially as practiced in scientific computing. It is at the PhD level where students will often gain at least a degree of relevant experience. We believe that national laboratories and DOE as a whole should partner with interested universities to enrich the pipeline through a combination of internship programs, on-the-job training, curriculum revisions or the creation of focused sub-programs that bring students at the Bachelor's and Master's level more practical experience of scientific software projects as embodied at the national laboratories. At the laboratories, a typical summer internship is only 10-12 weeks long. For an advanced graduate student (i.e., a PhD candidate) with significant experience, this can be enough time to accomplish a focused research project which can be published as a paper. But for a less experienced student interested in gaining experience in the practice side of scientific software engineering, it may be challenging to bring them to the stage of usefully contributing to the project in such a short time, with suboptimal results for both the intern and the hosting project. A partial solution may be to provide incentives to encourage staff to take on such students, perhaps through a separately sponsored internship program so that the R&D projects do not bear the costs directly. It may also be useful to look for ways to expand the time frame for such internships. For example, focusing on an academic semester, which is more typically around 15 weeks. Or, with the recognition that remote work can be very effective, we can imagine a student "on-boarding" with a team by spending a summer in an on-site internship, followed by one or more semesters of part-time remote participation, for which they might receive course credit at their universities, and perhaps payment as a part-time employee from the laboratory.

Engaging with students in a serious way earlier in their education may also have implicit benefits for recruitment and retention. Historically, we have found prior interns to be one of the best sources of

candidates for postdocs and junior staff positions. They already have experience with what the lab has to offer and understand the unique opportunity to contribute to the advancement of science. In disciplines where we are in direct competition with technology companies offering high salaries to people with the same skills we need at ORNL and other national laboratories, the draw of impacting science, and the culture of the labs, are often intangible benefits that can offset the lure of higher salaries for some candidates. Bringing these experiences to more and younger students will help our recruiting in the long run.

Efforts of the sort proposed here may benefit from connection and perhaps integration with other developing workforce diversity, equity, and inclusion initiatives, such as the Exascale Computing Project's Broader Engagement initiative.

## 6. REQUIREMENTS, BARRIERS, AND CHALLENGES TO TECHNOLOGY TRANSFER, AND BUILDING COMMUNITIES AROUND SOFTWARE PROJECTS, INCLUDING FORMING CONSORTIA AND OTHER NON-PROFIT ORGANIZATIONS

Communities and outside organizations are powerful tools in the pursuit of more sustainable scientific software and stewardship activities, starting at the basic level of awareness of these issues and opportunities for common cause in working to address them. There are a variety of communities already existing across the US and around the world who have interests in various aspects of scientific software sustainability and stewardship, and more are formed as awareness increases. Relevant examples include, but are not limited to, the UK's Software Sustainability Institute; the Research Software Alliance (ReSA); FORCE11, which focuses on scholarly communications, including software; The Carpentries, which teaches foundational coding and data science skills; and the United States Research Software Engineer Association (US-RSE). It is important to realize that many software packages that are important for ORNL and other DOE scientific projects originate outside of ORNL or the DOE complex, or have significant contributions from outside organizations, so DOE science benefits from broader connectivity. Engagement with, and where appropriate, launching new communities can facilitate broader awareness and action on relevant issues and should be encouraged. Fortunately, we have not so far observed significant barriers to participation in such communities.

There are a wide range of "business models" that might be useful to manage and steward different scientific software projects, some of which building communities around software products, and others which involve outside organizations. Communities may develop naturally around software packages which become more widely known and widely used, facets of which we discussed in response to question 4, above. In many cases, such communities can remain informal, though they may benefit from sponsor recognition of their nature and the need to support the community beyond specific R&D activities, as discussed above. However, in some situations, it may be useful to establish an independent organization to take charge of the community in some way. One example is where a single institution or small group may be perceived by others as having too strong a hold on the management and direction of a software project or not being sufficiently responsive to the needs of other users. Shifting "ownership" of a project in this situation to a separate organization which is constructed to include more voices can be useful to help address such concerns. Such organizations may take many different forms depending on the specific needs. It might need to hold the intellectual property (i.e., software copyrights, trademarks, etc.) for the project. It might need to provide services based on the software product or commission development and maintenance work in the software. Or it may be more focused on *guiding* the development and maintenance of the product, relying on other ways to get the actual work done (e.g., in-kind contributions of developer effort). An outside organization, especially one organized as a non-profit organization (i.e., 501(c)(3) or 501(c)(6) status) may also be able to obtain and deploy software and hardware infrastructure which can be shared across project participants in ways that would be challenging, or at least entail more friction, if the infrastructure were hosted at one of the participating institutions. In some cases, a non-profit may also be

able to obtain more favorable pricing for services or tools than a national laboratory, which some vendors treat as a commercial entity. While becoming members of existing technical organizations is not a high bar at ORNL, we are generally not familiar with how to *establish* such organizations or how we can leverage organizations that specialize in hosting software projects and related initiatives, such as the Linux Foundation, NUMFocus, or Software in the Public Interest. Consequently, we don't know what is feasible, the levels of effort and time involved, etc. and so we are unable to appropriately evaluate the costs and benefits of creating new external organizations as a possible means to help address software stewardship concerns. Anecdotally, we believe staff at other laboratories are in a similar situation. We believe it would be useful, therefore, to examine and document this information and make it available to laboratory staff to better inform their thinking about software stewardship. Depending on the findings, such an activity may also lead to recommendations to our institutions and DOE that might facilitate or accelerate the establishment of such organizations.

Commercial solutions are less common in the DOE scientific software environment, but not unknown. Certainly, commercial software products play a role, but open source software dominates the space. Traditional commercial software products offer a straightforward approach to stewardship: customers buy the software (or buy a license), and perhaps also an on-going maintenance agreement, which provides updates as the product evolves for an annual fee. There is an expectation of some level of support for the product by the vendor. Commercialization may be a useful strategy to consider for some open source software as well – particularly in cases where the user base is substantial and generates a significant burden to support well. There are a variety of commercial business models that make sense around open source software. Since the software itself is freely available, most revolve around adding value in one way or another. Perhaps most straightforward is offering commercial support contracts. Customers pay a set fee for the ability to file bug reports and get them addressed, perhaps also for "help desk" style support with installation and use. Companies may also offer consulting services and bespoke development around an open source software product, or "freemium" solutions in which they develop and deliver enhanced capabilities for the product which some users are willing to pay for. Often, such enhancements, in time, migrate into the free version of the product as new enhancements take their place in the paid "premium" product. Commercialization as a solution for software stewardship concerns should be used with care. The company needs to have sufficient understanding of the product, its applications, and its user base that they can be successful in delivering support, enhancements, or consulting services. And pricing of the services needs to be in line with the willingness of users to pay. Frankly, our experience is that in the academic and national lab R&D community, many project leaders tend to be rather parsimonious, and we expect maintenance contracts to be a tough sell in many cases, especially when the underlying software is freely available.

The federal Small Business Innovation Research (SBIR) and Small Business Technology Transfer (STTR) programs may be particularly useful vehicles to promote commercialization of important research software products. Indeed, for several years now the SBIR/STTR calls from the DOE Office of Science have included a variety of topics aimed at hardening, packaging, and broader deployment of Office of Science software products. As laboratory staff are often engaged as peer reviewers in the SBIR/STTR process, we have had the opportunity to review some of the proposals DOE receives on these topics. Based on this experience, we see two challenges. One is that many of the applicants have no demonstrable experience with the DOE scientific software environment. This means that proposers may not appreciate how the culture of this community works or have a real understanding of its needs. Further, the high-pressure structure and timing of Phase I awards makes it hard for companies who do win awards through the programs to engage effectively with the community. From this perspective, it may be useful for DOE and/or the laboratories to consider how they might mentor or otherwise engage small businesses interested in participating in the DOE software community, ideally in advance of their developing proposals. The other challenge is that, given the emphasis in these programs on commercialization, applicants are often looking to open up new markets to DOE software products. While perfectly reasonable, given the nature

and goals of the SBIR and STTR programs, such endeavors, even when successful, are likely to have limited benefits to the established user base for the software. From this perspective, we suggest that it may be possible to slightly reframe some of the topics in the calls to encourage consideration of the existing user base in addition to attempting to expand it. Finally, although we are often called upon as reviewers of SBIR and STTR proposals, we rarely learn about the outcomes and successes of the programs. It may be useful for the DOE SBIR/STTR Program Office to engage with leaders of DOE software projects to explore what program managers consider to be success stories in these topic areas and perhaps provide additional feedback that might help increase the effectiveness of this approach to commercialization of DOE scientific software.

Another type of community that can be very effective for the stewardship of certain kinds of software products or technologies are voluntary consensus standards organizations. Examples of standard that are important to the DOE scientific software community include, but are not limited to: programming language standards, such as C, C++, Fortran, OpenACC, and OpenMP; application program interfaces (APIs) for communications, such as the Message Passing Interface (MPI) and libfabric; and benchmarks, such as the Standard Performance Evaluation Corporation (SPEC). Standards such as these come into being because there is a desire to have specifications of certain functionality or capabilities well defined so that multiple organizations can implement them consistently. Most standards evolve over time, adding new functionality that the community deems important. Where R&D projects produce results that can appropriately be added to such a standard it is then stewarded by the standards organization, and multiple software products may be expected to embody the new functionality, making it available for widespread use. ORNL and other DOE laboratories are members and active participants in the aforementioned standards organizations, and have been successful in bringing new ideas into the standards. Participation in standards organizations should be encouraged. New standards organizations can also be created where there is a need unmet by existing groups. However, standards bodies have a long-term perspective, and effective (active) participation also needs to be a long-term investment which may transcend specific R&D projects that might generate new contributions to the standard. This may warrant special consideration by institutions (in most cases, the individual institutions are considered the members of the standards body) and by DOE program managers to ensure that appropriate support is available to participate in the long-term stewardship of standards that are important to DOE science.

## 7. OVERALL SCOPE OF THE STEWARDSHIP EFFORT

The Request for Information (RFI) provides an extensive list of activities that might be considered as part of the potential scope of future funded programs in scientific software stewardship. We consider this list to be quite thorough. There is nothing we would recommend removing from the list. We would, however, recommend adding *advocacy* to the list activities.

Part of what is needed is to help push forward the culture change that is beginning to take place throughout the scientific and HPC software communities, worldwide, that recognizes the importance of software and those who create and maintain it to the scientific enterprise – that software is a tool for science that must be honed and improved, not just a byproduct. Activities might include outreach, “inreach”, and engagement with sponsors and other stakeholders. Outreach activities would primarily face outside of the DOE complex, including engagement with potential partners in helping to change the culture or in specific stewardship concerns as described in our response to question 6, above. They may also include informing others about DOE efforts on software sustainability and stewardship in the spirit of encouraging other organizations to undertake similar initiatives. As we noted above, a significant portion of software that DOE relies on originates or includes substantial contributions from organizations outside of the DOE complex. Additionally, external engagement is important for exchanging knowledge and experience with other groups and bringing it back to DOE stewardship efforts. Inreach activities would be primarily focused

within the DOE scientific software community, with the goal of increasing support for and participation in stewardship activities and in improving sustainability across the board.

Finally, we believe that effective stewardship of DOE scientific software will require extensive awareness and participation by DOE program managers and other stakeholders in addition to the producers and consumers of the software. Because software-intensive science is so pervasive, communications with stakeholders are likely to be very diffuse. We believe that it will be important at times for the participants in the DOE software community to be able to speak with a common voice to program managers, and it makes sense to us that projects funded to support stewardship could naturally provide that voice.

## 8. MANAGEMENT AND OVERSIGHT STRUCTURE OF THE STEWARDSHIP EFFORT

As we suggested in our response to question 1, we believe that stewardship is, to a significant extent, individual to each software product and its situation. Even in cases where a software package can be effectively treated as a part of a larger ecosystem, that does not necessarily address all concerns about it's sustainability and stewardship. Consequently, responsibility for sustainability and stewardship of scientific software products must reside with the project leader or principal investigator, and oversight must come from the DOE program manager(s) sponsoring the work.

But we believe that the seeming chaos of such a situation can be addressed by creating what, for the purposes of this document, we will call a Center for Software Excellence (CSX) to support, guide, and help implement software sustainability and stewardship initiatives throughout the DOE complex. The CSX would be based within the DOE laboratory system (perhaps at a single lab, more likely as a multi-institutional distributed center) and may include academic or industrial partners, if appropriate. It would undertake the wide range of activities discussed in the Potential Scope section of the RFI and our response to question 7 in support of both specific project needs and, importantly, general goals and needs to improve the sustainability and stewardship of DOE scientific software.

To begin with, the CSX would work to better understand the current state of sustainability, interdependencies, and related stewardship concerns. While each package is, at some level, distinct, we believe that it will be possible to develop a modest number of categories capable of characterizing sustainability status and needs, and a similarly modest number of "business models" to support the needed levels of stewardship. Additionally, CSX should develop governance models, best practices, and other types of recommendations, such as when a software product should be discontinued, perhaps in favor of developing a new "replacement" or adopting an alternative package. CSX might also be responsible for establishing metrics and protocols by which to assess the sustainability and quality of software as part of tracking the progress of the work, discussed further in our response to question 9, below.

A significant portion of the activities of the CSX would be in supporting sustainability improvement and stewardship activities by developing and providing training, and subject matter expertise. The CSX might also include RSEs who could be tasked to assist particular projects with development, maintenance, and stewardship activities as needed. For the sustainability of the individual software projects, it might be preferable to first attempt to staff such needs from within the participating organizations before tapping CSX resources. But CSX might be able to provide special expertise not available to the team (in which case it might be expected that the CSX RSE would help bootstrap the development of the needed expertise within the project team and associated institutions) or to address special needs. Special cases might include helping to support a key dependency that originates outside the DOE complex or that has been "orphaned" by its developers.

In conjunction with these activities intended to directly support stewardship and sustainability improvement, the CSX would engage with institutions across the DOE complex in related workforce development efforts. CSX could help to design, and perhaps coordinate or administer the kinds of extended internship opportunities described in our response to question 5, as well as leading related engagement with universities. Additionally, because we envision research software engineers playing a prominent role in addressing the needed improvements in software sustainability and stewardship, we also see the CSX as a resource and advocate for the development of the RSE community across the DOE complex, together with outside organizations such as US-RSE.

The CSX could provide shared hardware, software, and service infrastructure to support software development and stewardship. Although in our response to question 3 we discussed the need for additional resources for software testing, we don't believe that the CSX would be the most appropriate place to house such resources. We would focus the CSX on emerging and exploratory capabilities (for example, providing opportunities to "test drive" commercial tools or services) and capabilities that might be fairly specific to the DOE context (for example, higher security software development hosting platforms).

The CSX would also serve to broker and support individual stewardship conversations and decisions, helping to bring together the stakeholders and provide object assessments and advice to help obtain the most effective resolution to each case.

Much of what we have described may sound like a service organization or a resource and that is an important component of what the CSX must provide. But it must also, simultaneously and continuously, operate as a research project. In this case the research question is, in effect, what is the most effective way to steward the scientific software that the DOE's scientific mission depends upon? CSX must use its unique position, to gain insight into the wide variety of scientific software projects and products important to DOE to improve and systematize our understanding of what "works" and what doesn't. It must work to improve our ability to meaningfully assess the quality and sustainability of software products. It must refine our understanding of best practices and other guidelines. In addition to using the Center's experience to improve DOE's own software stewardship, CSX must participate in the software engineering research community to share the insights and experience more broadly and to benefit from research on the engineering of scientific software by others.

## 9. ASSESSMENT AND CRITERIA FOR SUCCESS FOR THE STEWARDSHIP EFFORT

Ultimately, the goal of improving the stewardship of scientific software is to improve the ability of DOE to execute on its scientific mission which might be evidenced through things like increased scientific output, more impactful scientific discoveries, or faster "time to science". Of course, this can be hard to measure because "science" doesn't come in specific quantities on a particular schedule – there can be many reasons for variations. Digging down a level, one might ask the leaders and participants of software-intensive R&D activities whether they are experiencing less "friction" that they would attribute to software sustainability concerns. Again, this can be hard to quantify a priori, but where stewardship efforts have been successful, we would expect project participants to at least notice qualitative differences. Practically speaking, the development of quantitative measures probably needs to be individualized to each project and its particular situation – in other words, each project would be responsible for making its own best case for how to evaluate the benefits of stewardship activities. (This is similar to the way in which each application team in the Exascale Computing Project was asked to propose its own metric for how to evaluate the performance of their application to establish the baseline and, later, gauge the success of their efforts.)

The software engineering community has proposed many metrics that can be applied to individual software packages to assess their quality, which can be loosely equated to sustainability. Software metrics are often

contentious because there is often little empirical evidence to support their value, particularly in the context of *scientific* software, which is far less studied by software engineering researchers than non-scientific "commercial" software. Consequently, such metrics, if they are used, must be used with extreme care. But, with appropriate caveats, they might be used in a research study of the broad range of DOE scientific software in order to try to develop evidence, as to the utility of various metrics. Subsequently, appropriate metrics can be used to help guide software projects in improving their quality and sustainability.

Because of the importance of communities around many scientific software projects, we should similarly consider the emerging concept of community health metrics, which seek to gauge the way software-focused communities operate. For example, the Community Health Analytics Open Source Software (https://chaoss.community/) has working groups on Common Metrics, Diversity and Inclusion, Evolution, Risk, Value, and App Ecosystem, all of which might be of some interest to our efforts. Other metrics, perhaps yet to be defined, may also be useful.