
EXASCALE COMPUTING PROJECT

# A Response to the "Stewardship of Software for Scientific and High-Performance Computing" Request for Information

Michael A. Heroux, Sandia National Laboratories
Lois Curfman McInnes, Argonne National Laboratory
Rajeev Thakur, Argonne National Laboratory
Jeffrey S. Vetter, Oak Ridge National Laboratory
Sherry Li, Lawrence Berkeley National Laboratory
James Ahrens, Los Alamos National Laboratory
Todd Munson, Argonne National Laboratory
Kathryn Mohror, Lawrence Livermore National Laboratory
Timothy Germann, Los Alamos National Laboratory
Ryan Adamson, Oak Ridge National Laboratory
Todd Gamblin, Lawrence Livermore National Laboratory
Sameer Shende, University of Oregon
James Willenbring, Sandia National Laboratories

December 13, 2021

U.S. DEPARTMENT OF ENERGY | Office of Science
National Nuclear Security Administration

# TABLE OF CONTENTS

# INTRODUCTION

The U.S. Department of Energy (DOE) Exascale Computing Project (ECP) is an aggressive research, development, and deployment project focused on delivery of mission-critical applications, an integrated software stack, and exascale hardware technology advances [1, 2, 3], as motivated by the needs of extreme-scale science [4, 5, 6]. This response to the Request for Information (RFI) on the *Stewardship of Software for Scientific and High-Performance Computing* represents the experience of the leadership team of the Software Technology (ST) Focus Area ECP and key colleagues. The authors are the leaders of the ECP efforts to design, develop, deliver, and deploy a reusable scientific software stack [7] as part of the ECP project mission. The ECP community has created the Extreme-scale Scientific Software Stack, E4S (`https://e4s.io`), as a response to this imperative. With the drive to exascale, supercomputer architectures have shifted from general-purpose CPUs to *accelerated* computing using GPUs. The three U.S. exascale machines (Frontier, Aurora, and El Capitan) will derive the bulk of their computational power from AMD and Intel GPUs, and the Perlmutter machine at NERSC uses a third GPU architecture (NVIDIA). While there are similarities in the design of these machines, their programming models can be very different; we cannot expect all ECP applications teams to rewrite their code in three different ways for three different machines. The ECP software stack serves to insulate applications from the hardware, to separate portability and (some) performance concerns from application developers, and to allow ECP ST teams to make improvements to widely used components that benefit *all* applications.

These goals are not new, and they have been pursued with limited success by projects throughout DOE's history. What is new is that E4S represents the first *integrated* attempt to maintain a common DOE software stack, and *sustainable* processes have been developed to ensure that the stack remains robust and reliable. Hardware and software environments, especially in the world of HPC, are constantly changing, and if the software stack cannot be continuously built, tested, and evolved on relevant target environments, then it will fall out of use and will no longer provide value for its users. We expect future HPC hardware to be even *more* heterogeneous than it is now, and the need for a common stack will only become more acute in the future.

For this reason, E4S is comprehensive in scope—it integrates not only the core DOE products that define it, but *also* their critical dependencies. E4S leverages modern tooling such as package management and continuous integration, so that testing is done in advance, by experts, before end-users touch the software. To provide value, E4S ensures that the components in the software ecosystem work seamlessly together, leaving a *minimal* set of integration activities to the end user. E4S exploits the efforts of a broader community of contributors by building on top of the Spack [8] package manager, curating changes made by a community of worldwide users, and ensuring that the software works on DOE- and industry-relevant platforms. This level of integration is not possible without a dedicated team of expert integrators and maintainers working closely with DOE software developers, computing facilities, and the broader scientific software community.

This team has collectively had decades of experience in scientific software design, development, and delivery across a broad spectrum of algorithmic and application areas, focused on developing and providing capabilities for each new generation of supercomputers. The authors and their roles are shown in Table 1. During the past five years, we have been working with a broad range of ECP software teams [7] to establish new and sustainable software products that will enable portable and high-performance execution for many new and existing applications on the U.S. exascale computers.

This response addresses elements of all ten topics listed in the RFI.

## ECP ST E4S AND SDKS

Fundamental to this RFI response is the ECP experience in establishing the three-tier software architecture shown in Figure 1. To better prepare the reader for the subsequent discussion, we briefly introduce E4S and software development kits (SDKs).

### *The Extreme-Scale Scientific Software Stack*

In November 2021, the ECP ST focus area released V21.11 of E4S, containing a collection of the software products to which ECP ST contributes. E4S is the primary conduit for providing easy access to ECP ST capabilities for the ECP and the broader community. E4S is also the ECP ST vehicle for regression and integration testing across DOE pre-exascale and exascale systems. E4S has the following key features:

- **Michael A. Heroux - ST Director:** Lead overall of Software Technology project planning, executing, tracking, and assessing.
- **Lois Curfman McInnes - ST Deputy Director:** Co-lead overall of Software Technology project planning, executing, tracking, and assessing.
- **Rajeev Thakur - Programming Models and Runtimes:** L3 lead of cross-platform, production-ready programming infrastructure to support the development and scaling of mission-critical software at the node and full-system levels.
- **Jeffrey Vetter - Development Tools:** L3 lead of tools and supporting unified infrastructure aimed at improving developer productivity across the software stack.
- **Sherry Li - Mathematical Libraries:** L3 lead of mathematical libraries and frameworks that provide scalable, resilient, and numerical algorithms that facilitate efficient simulations on exascale computers.
- **James Ahrens - Data and Visualization:** L3 lead of production infrastructure for the analysis and visualization of data.
- **Todd Munson - Software Ecosystem and Delivery:** L3 lead of development and coordination of software development kits (SDKs) and E4S across all ECP ST projects.
- **Kathryn Mohror - NNSA ST:** L3 lead of development and enhancement of open-source software capabilities that are primarily developed at Lawrence Livermore, Los Alamos, and Sandia National Laboratories.
- **Timothy Germann - Application Co-Design:** L3 lead of reusable crosscutting software components that capture the most common patterns of computation and communication in ECP applications.
- **Ryan Adamson - Software Deployment:** L3 lead of delivery of the software stack to leadership computing facilities.
- **Todd Gamblin - Software Packaging:** Technical lead of packaging technologies for delivery of the software stack to leadership computing facilities.
- **Sameer Shende - E4S Software Portfolio:** Technical lead of the Extreme-scale Scientific Software Stack (E4S).
- **James Willenbring - SDK Portfolios:** Technical lead of Scientific Software Development Kits (SDKs).

**Table 1:** Authors of the RFI response and their roles in the Exascale Computing Project (ECP). Authors are part of the ECP leadership team, representing some of the key software leadership roles on the project.

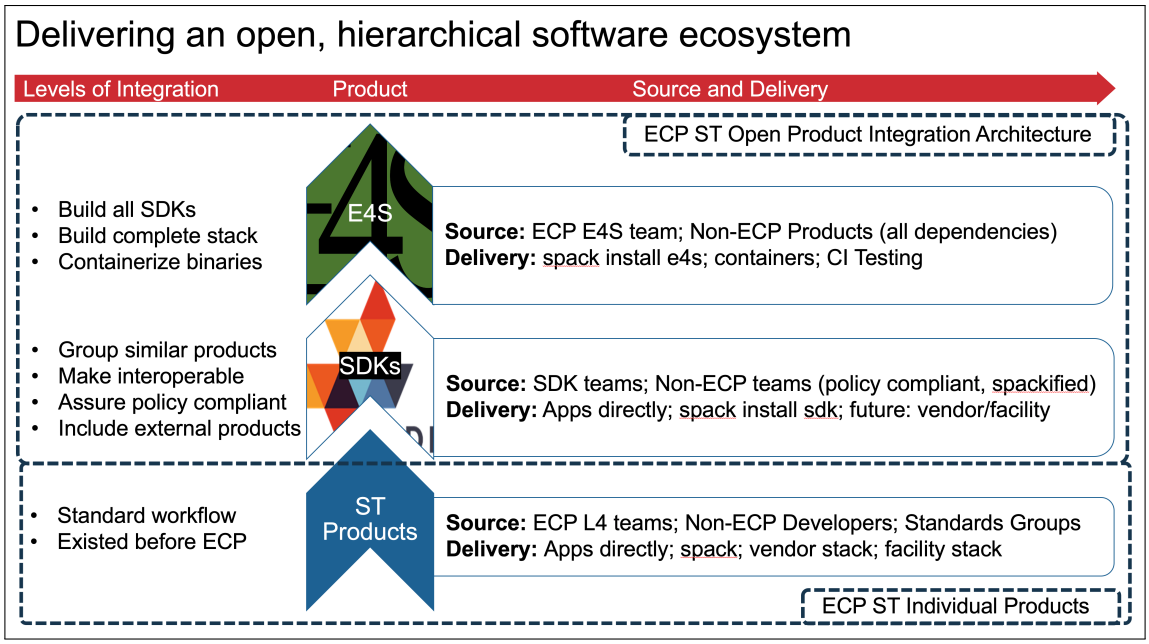



**Figure 1:** E4S and SDK hierarchical architecture. Individual ECP Software Technology (ST) products are still developed by small teams of computer science, mathematics and related experts in a particular field of expertise. Examples include math libraries like PETSc, I/O libraries like HDF5, and tools such as TAU. SDKs are aggregations (see Figure 2) of similar projects for the purpose of coordinated planning, delivery, testing, tutorials and similar cross-cutting activities. E4S is the full collection of all ECP reusable software products and all dependencies, providing a full, portable software stack containing the latest products, including the latest capabilities targeting exascale computing platforms.

- **The E4S suite is a large and growing effort to build and test a comprehensive scientific software ecosystem.** In November 2018, E4S V0.1 contained 25 ECP products. This year, E4S V21.11 (the ninth E4S release) contains 91 ECP ST products and numerous additional products needed for a complete software environment. Eventually, E4S will contain all the open-source products to which ECP contributes and all the related products needed for a holistic environment.
- **E4S is not an ECP-specific software suite.** The products in E4S represent a holistic collection of capabilities that contain the ever-growing SDK collections sponsored by the ECP and all additional underlying software required to use ECP ST capabilities. Furthermore, the E4S effort is expected to live beyond the time span of the ECP, becoming a critical element of the scientific software ecosystem.
- **E4S is partitionable.** E4S products are built and tested together by using a tree-based hierarchical build process. Because the entire E4S tree is built and tested, users can build any subtree of interest without building the whole stack.
- **E4S uses Spack.** The Spack [8] package management tool invokes the native build process of each product (or installs from a cached binary), enabling the quick integration of new products, including non-ECP products. Spack has over 6,000 builtin packages, all of which can be used together with E4S.
- **E4S is available via containers.** In addition to source and binary installations with Spack, E4S maintains several container environments (e.g., Docker, Singularity, Shifter, CharlieCloud) that provide the lowest barrier to use. Container distributions dramatically reduce installation costs and provide ready-made environments for tutorials that leverage E4S capabilities. For example, E4S containers now support custom images for ECP applications, such as WDMapp and Pantheon.
- **E4S distribution.** E4S products are available via the E4S website.[1]

[1] http://e4s.io

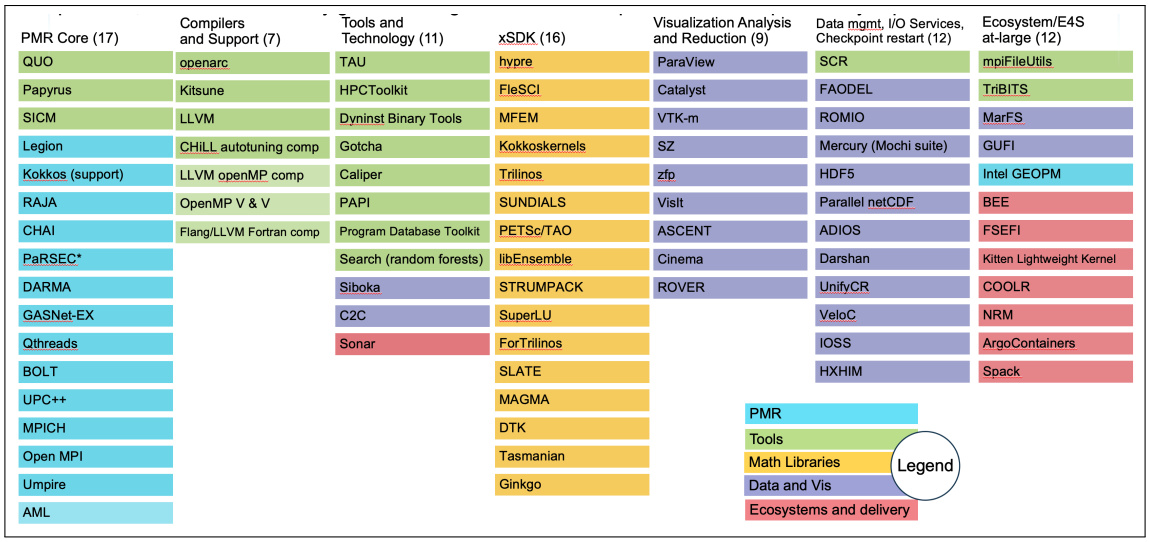



**Figure 2:** Current collection of E4S software products listed by membership in their respective SDKs. The aggregation of similar products into SDKs has tremendous benefit in creating communities of practice for sharing knowledge, technique and other cross-cutting information. Each team cooperates with other teams in exploring design spaces, delivering tutorials and more, but each team still develops and delivers its own products. This "coop-etition" model has led to improved design, a sense of camaraderie across teams and still retains the healthy sense of small-team pride in product capabilities.

- **E4S developer community resources.** Developers interested in participating in E4S can visit the E4S-Project GitHub community.[2]

The first set of E4S community policies [9] was adopted in October 2020 (Figure 9). These policies are membership criteria for a product to become an E4S member package. The purpose of the community policies is to establish baseline expectations for software quality and practices to help address sustainability and interoperability challenges for the ST software ecosystem. Although a package does not have to demonstrate compatibility with the policies as a condition of inclusion in E4S releases, compatibility is necessary for member package designation.

### *SDK Overview*

The ECP ST SDKs efforts support a set of activities focused on:

- Establishing community policies aimed at increasing the interoperability among and sustainability of ST software packages, using the xSDK [10] community package and installation policies [11] as a model.
- Coordinating the delivery of ECP ST products through the E4S [12], a comprehensive and coherent set of software tools, to all interested stakeholders on behalf of ECP ST, including ECP applications and the broader open source community.

An ECP ST software development kit (SDK) is a collection of related software products (called packages), where coordination across package teams will improve usability and practices, while fostering community growth among teams who develop similar and complementary capabilities. SDKs have the following attributes:

- **Domain scope:** Collection makes functional sense.
- **Interaction model:** How packages interact; compatible, complementary, interoperable.
- **Community policies:** Value statements; serve as criteria for membership.

[2] https://github.com/E4S-Project

- **Community interaction:** Communication between teams; bridge culture; common vocabulary.
- **Meta-infrastructure:** Encapsulates; invokes build of all packages (Spack); shared test suites.
- **Coordinated plans:** Inter-package planning – does not replace autonomous package planning.
- **Community outreach:** Coordinated, combined tutorials; documentation; best practices.

### HOW THIS DOCUMENT IS ORGANIZED

Each of the ten numbered sections in this document addresses the corresponding categories of questions in the RFI. The first subsection in each section summarizes the key points that are explained in detail in the remaining subsections. A reader who is only interested a high-level overview can focus on just these first subsections: Section 1.1 through Section 10.1.

## 1. SOFTWARE DEPENDENCIES AND REQUIREMENTS

### 1.1 DEPENDENCIES AND REQUIREMENTS: KEY POINTS

1. The ECP software technology (ST) and co-design (CD) teams provide and use a comprehensive collection of scientific software in programming models, runtimes, development tools, math libraries, data and visualization, and software packaging and delivery. We anticipate our software scope will continue to grow, soon including AI-for-science libraries and tools, toolkits for edge computing, and scientist-friendly libraries for specialized quantum, neuromorphic and other future devices.

2. Using hierarchical long-, medium- and near-term planning, along with a responsive change control process, we can effectively plan several years ahead and then adapt as needed to reduce, expand, or redirect cost, scope, and schedule.

3. We believe that scientific progress is best assured by embracing the opportunities and challenges of large ecosystems of reusable software [13]. At the same time, the quality of the individual products must be excellent, and dependencies across products must be rigorously managed. Large ecosystems like E4S, developed by ECP, provide a path forward toward a dependable, portable and ubiquitously available turnkey software stack for scientific discovery, available on laptops to supercomputers to off-premises cloud resources.

### 1.2 CURRENT SOFTWARE ECOSYSTEM

The E4S software portfolio provides reusable software tools and libraries identified as requirements for the collection of applications sponsored by ECP and other key application needs. E4S products complement vendor and broader community capabilities. E4S specifically targets the major high-performance computing (HPC) facilities, including the new exascale and pre-exascale platforms being installed at Oak Ridge Leadership Computing Facility (OLCF), Argonne Leadership Computing Facility (ALCF), the National Energy Research Supercomputing Center (NERSC), and Livermore Computing Center (LCC). E4S products are integrated using the Spack [8] package manager (Figure 3) and are available as build-from-source, pre-installed at a growing number of computing facilities, as containers such as Docker and Singularity, and on hosted cloud platforms such as Google Cloud and Amazon AWS.

In addition to the software itself, E4S provides a single point of access to product documentation via its DocPortal website. E4S establishes quality expectations on member products via community policies, which define best practices for development, integration, responses to user requests, documentation, and testing.

E4S targets leadership computing environments, but because of how scientific software is developed within the community, E4S software products also work on laptops, commodity clusters, cloud environments, and most other commonly available computer system used for scientific computation.

Finally, E4S is a community of scientific software developers, users, and integrators who strive to impact science through efforts to improve the effectiveness and efficiency of the software we provide. In this document we describe a vision for E4S from the perspective of its major stakeholders and collaborators. Each section

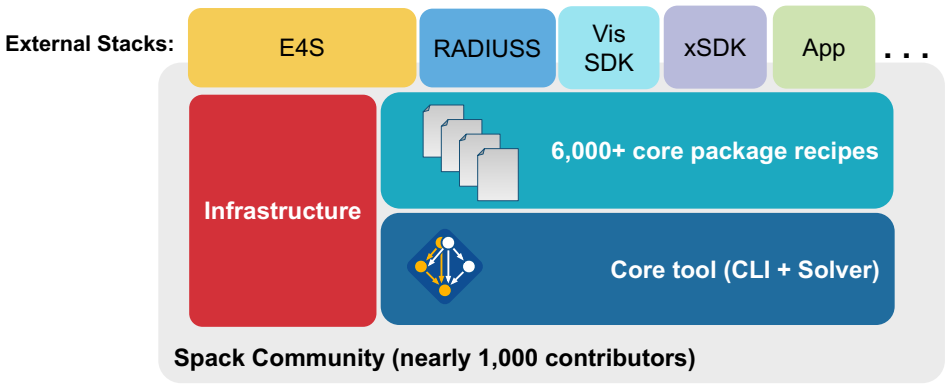



**Figure 3:** Structure of the Spack project. Spack provides the backbone of the E4S software portfolio. In addition to enabling the encoding of cross-package dependencies using portable package recipes, Spack increasingly supports build, binary packaging, testing, and integration features commonly needed for software development workflows on diverse computing platforms. E4S and its related Software Development Kits (SDKs) are Spack clients.

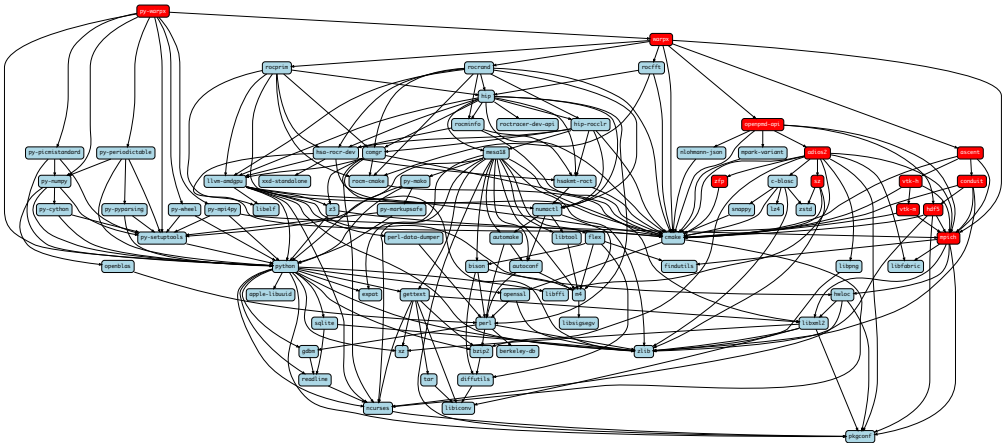


**Figure 4:** Dependencies of WarpX (built with AMD ROCm GPU support enabled). E4S packages are shown in red.

describes the possible role E4S can have as DOE and ASCR shepherd the growth and sustainability of this new software ecosystem in support of DOE's mission.

### 1.3 CURRENT SOFTWARE DEPENDENCIES AND TRENDS

Component software development is a concept that dates back to at least the late 1960's [14]. Fifty years later, modern applications make heavy use of components. The wide availability of open-source software has made it easy to find software projects, or *packages* that implement useful functionality, and in many cases application developers can save considerable time by reusing external packages. As such, packages have become the fundamental unit of software reuse in most modern software ecosystems. A single application may rely on tens or hundreds of external packages called *dependencies*, and tools called *package managers* have emerged that address and automate dependency declaration, detection, and management.

Dependencies are not without their own costs—supporting an application on a new platform requires support for all of its dependencies, and all packages integrated into a single application must remain compatible with each other and the host system. In many cases, dependency integration problems can dominate the porting time when releasing a new version or when moving an application to a new platform [15, 16].

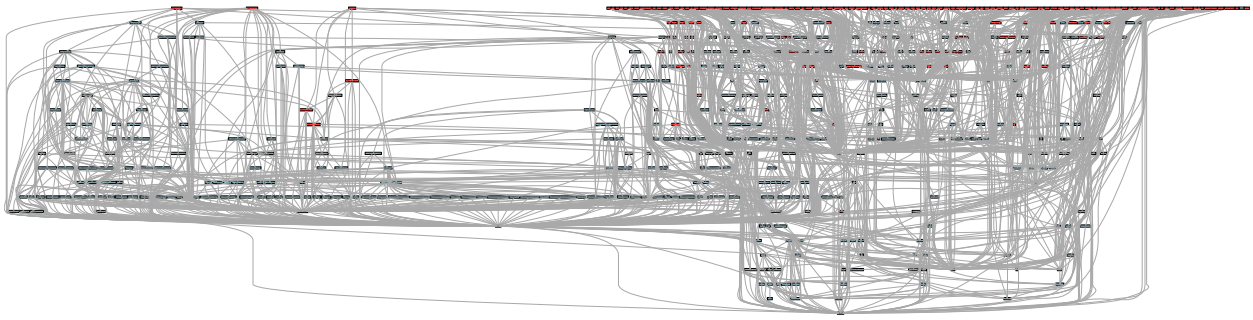


**Figure 5:** All packages in E4S 21.11 (red) and their dependencies (blue). E4S 21.11 (the current release) is deployed using a Spack environment containing roughly 100 root specs, which equates to the installation of as many as 500 Spack packages to complete the installation. The precise number of packages required for a deployment of E4S is system-specific as they need the underlying hardware (e.g., NVIDIA, AMD GPUs). The total number of packages deployed also depends on the use of facility-managed packages that are already installed on the system, as these may be re-used with Spack as externals in the custom E4S Spack environment.

### *1.3.1 Single Applications*

E4S includes a common set of dependencies on which scientific applications rely. For example, Figure 4 shows the dependency graph of the ECP WarpX application [17], which models plasma-based particle accelerators. WarpX itself is split between two packages: a set of Python language bindings for the code in `py-warpx`, and the application itself in `warpx`. The following nodes are shown in red because they are E4S core packages, and they rely on 10 other E4S packages: `openpmd-api`, `adios2`, `hdf5`, `zfp`, and `sz` for I/O and data compression; `ascent`, `vtk-h`, `vtk-m`, and `conduit` for in-situ visualization; and `mpich` (or, optionally, `openmpi`) for interprocess communication. Each of these is an ECP code project in its own right. Each package is developed, tested, released, and maintained by a separate team. Developers prepare releases for each package on their own schedule; they may or may not coordinate or test their packages together. Individually itegrating the 12 core E4S packages needed for WarpX would be its own challenge; moreover, each of these packages has its own set of external dependencies that are *not* part of ECP. Shown in blue in the diagram, they include build tools like `cmake`, the Python language and various Python packages, additional compression and imaging libraries, low-level network libraries like `libfabric`, the AMD ROCm suite of GPU runtime libraries, and more. All together, the stack of packages that must be deployed to use WarpX includes 79 different packages. This level of integration is typical for modern HPC applications.

### *1.3.2 Integrated Stacks*

To support an entire ecosystem of HPC applications, a much larger software stack is required. Figure 5 shows all of the packages in E4S version 21.11 (the latest version). There are around 100 core packages (shown again in red) with 400 dependencies (blue). Versions of all of these packages are built and tested as part of E4S integration, and a user of E4S can pick and choose from these vetted components to build their applications.

The effort that goes into maintaining E4S provides value to end users by ensuring that these key HPC software components are tested and work *together*. The task of integrating 400 to 500 different packages is not trivial, and E4S ensures that there is a known set of working versions of all components, and that the *same* stack works across different computing environments. Because E4S aims to support at least three different exascale machines, there are multiple configurations of many packages (e.g., CUDA, ROCm, and CPU versions of the same code). E4S also provides a number of key performance portability frameworks (e.g., `RAJA` and `Kokkos`) to reduce the burden of support on application developers.

At first glance, it may seem that the support for dependencies should be handled by each application team, as they are the primary stakeholders. But if we assume (accurately) that WarpX's 80 dependencies are typical of HPC applications, then with 25 application codes in ECP, we would pay the support cost of 2,000 packages. With E4S, the maintenance burden is reduced significantly—the E4S distribution is just under 500 packages, or less than 1/4 the total amount of maintenance that would be needed *without* a consolidated maintenance effort. Because teams in ECP rely on a common set of libraries, there is great

benefit to consolidating their maintenance burden.

Finally, the task of maintaining these 500 packages does not fall solely to the E4S team. As mentioned, E4S is built on top of Spack (Figure 3), and *all* packages in E4S are also in the upstream Spack repository. Each release of E4S leverages changes made by Spack's nearly 1,000 contributors—the E4S team curates and tests package versions from the Spack mainline, ensuring that they work on systems of interest to DOE. Changes needed to make a stable E4S release are upstreamed to the Spack mainline, benefitting the larger Spack project. To be sustainable, a stack of E4S's size must rely heavily on the broader open-source ecosystem, as no one party can afford to maintain all components. By using sustainable processes, E4S can provide a well tested and robust stack *without* taking on the cost of maintaining *all* of its component software.

### 1.4 THE IMPORTANCE OF APIS AND IMPLEMENTATIONS

An Application Programming Interface (API) enables the separation of the functionality a piece of software is supposed to provide from how that functionality is implemented. APIs can also present a mental model that is close or matches what a user expects the software to do, while enabling multiple adapters to provide an implement of the API. Perhaps the most successful API in HPC is MPI, the message passing interface. Nearly all HPC applications that work on distributed memory parallel computers rely upon MPI via its standard interface. Because many MPI implementations support the MPI standard, there is little hesitation in using the MPI interface. The importance of separating the interface from implementation is hard to overstate. Compilers use a similar approach, achieving the same kind of user adoption.

As part of ECP efforts, we have made some progress in the I/O software domain by requiring I/O libraries to make their functionality available via the HDF5 API. HDF5 is the most commonly used I/O interface within scientific applications. Because of the popularity of HDF5 and because there are other useful I/O libraries that provide significant feature not available in HDF5, there is value in making these libraries available to applications via the HDF5 API.

As we go forward, the scientific software community can benefit from additional efforts to standardize API's and make existing and new libraries and tools accessible through these APIs. The impact of this approach will both increase the attractiveness of using new and emerging libraries and tools and reduce the effort needed to integrate them. Applications can more readily rely upon third-party software with lower risk if more than one option exists through the same interface to provide the required functionality. The ability to switch easily between options also creates a positive competitive environment, where teams who provide similar functionality will more easily obtain performance and functionality comparisons between each other's products.

### 1.5 EMERGING SOFTWARE NEEDS

We foresee many emerging software needs; three of the most important are:

1. **Further advances to increase standardization of software practices and processes, as well as APIs for software tools and libraries:** Scientific software for leadership computing environments often co-evolves with the computing environments. This is especially true in high-performance computing. Even so, the scientific software community must increase efforts to establish standard APIs for reusable components. Lack of API standards greatly increases the risk that users will avoid or unsuccessfully attempt integration of third-party software. API standards also promote interchangeability of components and foster healthy cooperation and competition across teams who provide similar capabilities.

2. **Increased abstraction and portability across diverse parallel computing devices:** We foresee that new high-performance computing platforms will grow in diversity. At the same time, scientific software teams will not be productive if they have to provide custom code for each new computing device. The emergence of OpenMP with GPU support, along with the Kokkos and RAJA performance portability libraries, have been very important for performance portability; more efforts and innovation will be needed, especially as we need to support simultaneous execution on multiple heterogeneous devices.

3. **Sustained and stable funding for scientific software work, including recognized career paths for software professionals:** One of the most important requirements for managing software dependencies and requirements is a community of professional scientific software developers who are skilled at software architecture and design and who have a clear career path that enables their long-term participation in the scientific software community.

# 2. SECURITY AND INTEGRITY OF SOFTWARE AND DATA

## 2.1 SOFTWARE AND DATA SECURITY AND INTEGRITY: KEY POINTS

1. Software Quality Assurance (SQA) is a real concern that must be addressed at the requirements, design, development, delivery, and deployment phases of the software lifecycle [18].

2. Many SQA policies and activities are more effectively and efficiently designed and deployed at a portfolio level, where E4S and Spack can play important roles.

3. E4S, via Spack functionality, already employs checksums and cryptographic signing, beyond what most scientific application teams are capable of managing if they directly control their own third-party software dependencies.

4. E4S and Spack ensure that builds continue working using an extensive and scalable continuous integration system. Every contribution to Spack *must* ensure that no E4S builds have been broken, before it is merged into Spack.

5. At the same time, E4S and Spack can do even more to assure the integrity and quality of the software supply chain. We need technical and policy approaches to achieve high levels of trust, including comprehensive use of provenance tracking via Spack recipes, as well as training of DOE-funded development teams in basic software quality assurance concepts and techniques. Even so, E4S provides its users with better security than what was available just a few years ago.

6. SQA strategies for scientific libraries and tools need better characterization. Priorities are not necessarily in the same rank order as other kinds of better-studied software products. One of the most effective ways we have seen so far to address SQA for scientific software is a strong focus on correctness—making sure a product uses memory safely, executes computations correctly, and manages external data sources rigorously.

7. We can use the fact that E4S is a globally available and transparent software collection as a means to improve SQA. Just as GitHub uses Dependabot to provide and scan a registry of known security issues to inform repository owners of known risks, E4S can provide similar alerts about correctness and performance at scale, helping E4S users upgrade to fixed version of E4S member packages.

8. Collaborating with the leadership computing facilities as we develop and deploy E4S via Spack is essential to securing our software supply chain, even for software that we use but do not develop ourselves.

9. SQA is a DOE-complex-wide issue. We should not rely on only site-specific policies. All labs should be working together on a tiered plan. E4S and Spack can be part of the complex-wide solution.

## 2.2 CURRENT AND NEAR-TERM SOFTWARE QUALITY ASSURANCE EFFORTS

The E4S project currently works closely with the Spack team to build and test the E4S stack. E4S contributors write Spack recipes for their packages, allowing them to be built and tested in many different configurations. Spack has integrated the E4S stack description into its own continuous integration (CI) pipeline, which ensures that E4S configurations of interest continue working in the Spack mainline. Spack packages and binaries are assured via `SHA256` checksums and by cryptographic signing.

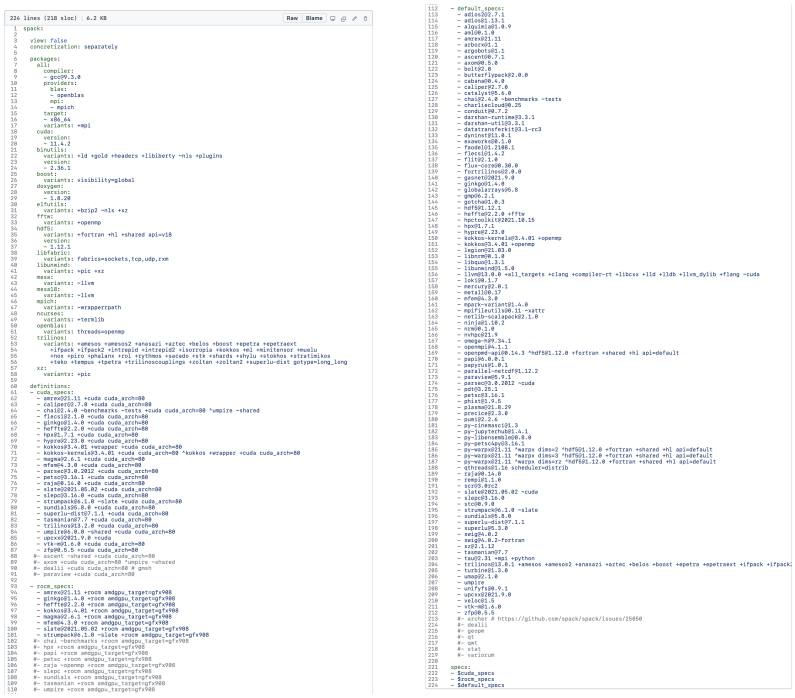


**Figure 6:** The Spack environment defining E4S 21.11. Components, versions, and build options are specified on each line. Each version of E4S has a manifest like this one that allows a user to reproduce, build, and test the stack.

### *2.2.1 Provenance and Specification as Code*

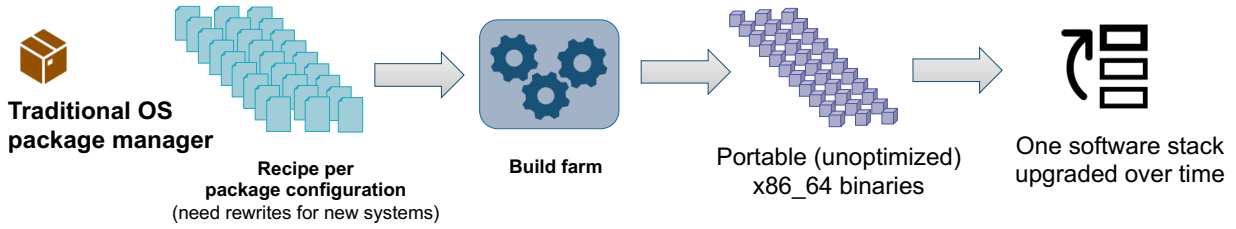
E4S requires member software projects to encode their build in a Spack package, so that they can be built easily and reproducibly on many different target platforms, using arbitrary versions and build options, as well as different compilers. Spack packages are parameterized recipes written in Python that tell Spack how to build a piece of software along with its dependencies.

E4S ties many package recipes together into Spack *environments*. Figure 6 shows the `spack.yaml` manifest used to build E4S 21.11. The manifest defines the packages and versions in the distribution unambiguously, and it allows maintainers to tailor the package configurations for site-specific deployment. Environment `spack.yaml` files are stored in the E4S git repository [19] and versioned over time. In addition to the *abstract* `spack.yaml` description, another much more detailed `spack.lock` file is written out with *all* confiugration details for building the E4S environment, including the selection of compiler, architecture, software version and featured configurations, as well as external software provided by the given facility (e.g., ROCm, Intel oneAPI, CUDA, Spectrum-MPI, Cray-mpich, etc.).

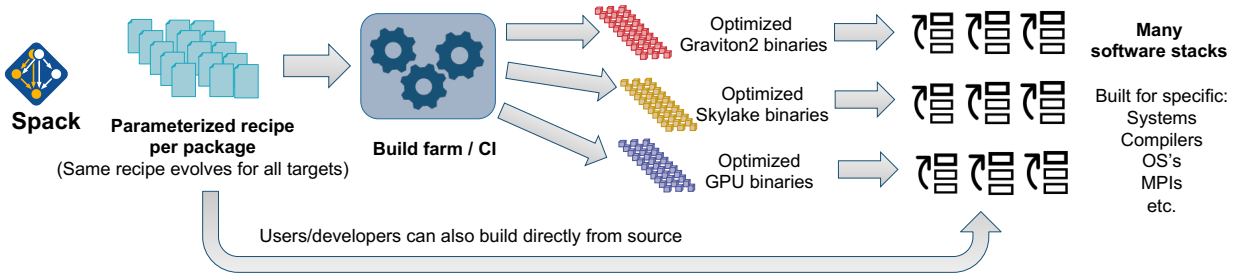
Together, the `spack.yaml` and `spack.lock` files provide sufficient provenance to reproduce an entire E4S deployment, either manually on a local machine or cluster, or in a CI pipeline.

### *2.2.2 Automated Testing*

In addition to the basic build recipes, E4S contributors can provide parameterized test recipes, which are run as part of the E4S release process. Tests can be integrated into Spack packages, in which case they evolve over time with the build recipe, or they can be written externally as part of the E4S validation test suite. This approach ensures that packages can be deployed and run on the target system and helps monitoring of failures as dependencies and other components change over time. E4S relies to a certain extent on externally installed components, where available, provided by the facility teams and vendors, such as vendor-optimized MPI implementations that may internally use GPUs and the corresponding GPU runtimes.




**(a)** Traditional software distributions manage a single stack aimed at a single target (e.g., generic x86_64), with one build recipe per package configuration. Adding new packages for new platforms means writing porting all recipes to the new platform.




**(b)** Spack's parameterized package recipes reduce the maintenance burden of maintaining a cross-platform software stack. Each package recipe is parameterized and can be used to build software for a wide range of optimized targets, across many software stacks.

### *2.2.3 Checksums and Signatures*

Spack package recipes include checksums (`SHA256`) for build artifacts, which ensure that no one has tampered with the built source code. Binaries created from checksummed source code are cryptographically signed using Spack's integration with GPG—a widely used and trusted program for cryptographic signing of data.

### *2.2.4 Multi-platform Builds*

E4S maintains builds for a core set of packages working across the wide range of target platforms of the U.S. pre-exascale and exascale supercomputers. A traditional Linux distribution (Figure 7a) typically has a build recipe per binary package produced, i.e., there is one-to-one correspondence between source code, recipes, and binaries. Binaries are portable, i.e., they are not optimized for a specific system but built with a lowest-common-denominator instruction set for maximum compatibility.

In E4S and Spack (Figure 7b), there is a single recipe for each package, and it is able to *generate* build instructions for a wide range of package versions, configurations, and targets. Spack is designed to keep *many* different versions of a software stack working at once. This enables E4S to support many platforms, but it requires extensive testing to ensure that all of the different builds continue to work. The goal is to shift the burden of platform support from humans (who would otherwise need to port many package recipes from platform to platform) to machines (by automating the process of doing platform-specific builds). This broad support is not nearly as easy to do with other package managers—Spack's single-source recipes are key to ensuring that the stack builds the same way across different platforms.

### *2.2.5 Continuous Integration and Security*

Spack receives over 500 contributions per month, and many of them are changes to E4S packages and their dependencies. If not tested thoroughly, these contributions can cause E4S package builds to break, and users will no longer be able to build these packages using Spack. To ensure that the E4S package recipes continue to build in our environments of interest, the Spack team, working with E4S, has deployed a large-scale continuous integration system (Figure 7). Contributions arrive in the form of pull requests on GitHub, and if the contribution touches E4S, Spack's CI system dynamically generates a pipeline that tests all affected packages from the stack. The CI pipeline is defined in a YAML file like the one in Figure 6. We test these packages in containers in Amazon Web Services (AWS), on testbed machines at the University of Oregon,

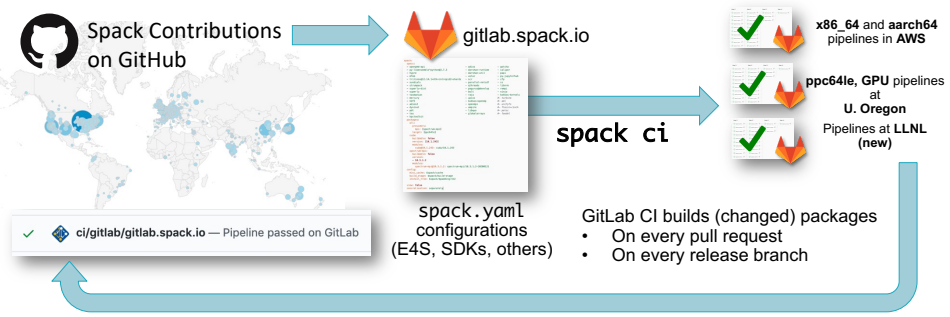



**Figure 7:** The Spack continuous integration and testing architecture provides insulation from potentially anonymous open source contributions. Contributions are first built in a sandboxed environment, and final binaries are rebuilt after approval to create official binaries.

and on clusters in a new open build farm at LLNL. If all builds and tests succeed, then the CI system marks the pull request with a green check mark showing that it has passed the CI test, and maintainers are allowed to review the code and merge the pull request.

There is a fundamental security challenge to this kind of quality assurance. Pull requests on GitHub are not fundamentally secure, as they can come from untrusted contributors who may try to contribute malicious code to Spack (and therefore to E4S). At the same time, we want *contributors* to ensure that package contributions build correctly, so we *need* to run these checks in the CI environment. Running the checks *before* merge reduces the burden on maintainers, as they do not need to fix bugs introduced by new PRs. They can ask the source of the PR—the contributor—to do so. To address the security issue, Spack CI uses a two-tier system to ensure security. PR builds are stored in their own, separate build cache that is *only* used for CI on pull requests. Binaries created from untrusted code can be reused across PRs, but contributions are only merged to mainline if they are approved by maintainers, and binaries in the public buildcache are only built from code that was approved by a human maintainer. This allows us to better maintain Spack and E4S, by pushing the burden of vetting contributions onto contributors, and by avoiding putting any untrusted, unreviewed changes on the main branch of Spack.

## 2.3 FUTURE SQA IMPROVEMENTS VIA A COORDINATED SOFTWARE PORTFOLIO

Mechanisms like `SHA256` checksums on tarballs and signatures on Spack binaries help ensure the integrity of components downloaded by Spack and E4S users, and continuous integration ensures that Spack packages continue to build without error. However, there is more we can do to ensure the integrity of the packages in Spack and E4S, and we expect that further progress will be made as we go forward.

Beyond digital signing, large open-source software platforms such as GitHub take advantage of the global view available from open-source software products hosted on their platform. GitHub maintains a registry of software product security risks, tracking the product version that has a known vulnerability and an updated version that fixes that vulnerability. These risks are also associated with particular files, identified by their `SHA256` hashes. If you keep your software on GitHub and use a version of a product that has a known vulnerability, GitHub tools will inform you of the problem, and the risk level and suggest to you a possible fix.

We anticipate that E4S and Spack can leverage mainstream tools by adding virus, malware, and spyware scanning to their pipelines. For example, Spack can leverage GitHub and other services to check that the tarballs submitted by package contributors are not known to be malicious. Because Spack allows *many* configurations to be built with the same package, we can explore providing similar capabilities for concerns not typically addressed by mainstream tools. In addition to leveraging public services, E4S and Spack can provide their own services such as detecting and reporting known performance and correctness bugs on special-purpose processors and at system scales that are peculiar to leadership computing platforms. Of course, the kind of infrastructure required to create an issue registry, scan for vulnerabilities, and report them to users, can be used to handle many kinds of security and correctness issues that are beyond the scope of mainstream tools such as those provided by GitHub, enabling us to further enhance software quality as we go forward.

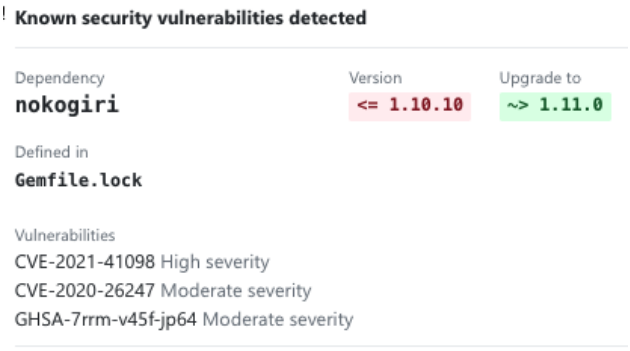



**Figure 8:** The GitHub Dependabot [20] service provides registry and reporting of known security issues along with guidance for addressing the problem. Using the portfolio approach of E4S we can enhance this kind of service to include problems specific to scientific and parallel computing, for example, performance and scaling issues on large-scale parallel computers.

The advanced software quality assurance features we mention here are made possible by managing the software products as a portfolio. Without the E4S portfolio, it quickly becomes infeasible to manage cross-product interactions like a vulnerability or performance registry for a specialized collection of software.

### 2.4 E4S COMMUNITY POLICIES AND SQA

E4S Community Policies [9], listed in Figure 9, are membership criteria for a product to become an E4S member package. The policies were arrived at by the ECP SDK team, including representation from all areas of ECP software technologies (programming models and runtimes, development tools, math libraries, data and visualization packages, and software ecosystems and delivery), and after considering multiple rounds of feedback from across the ECP ST community. The policies address several aspects of software quality and usability, including sustainability, testing, error handling, documentation, and others.

A major focus of the E4S community policy effort is quality improvement. The community policies support quality improvement in multiple ways. First, the policies provide a standard that can be measured against. As part of the recently-completed 2021 ECP ST project reviews, teams reported data concerning compatibility with the nine E4S Community Policies, as well as an analysis of any gaps in compatibility. This data can be used to understand where E4S is as a whole with respect to each policy, as well as what the specific gaps are. Depending on the particular circumstance, it may be appropriate to fund an effort to close the gaps, but it might be preferable to reconsider specific modifications to one or more policies. Over time, we will reassess policy compatibility, which will reveal trends in the level of compatibility across E4S.

Second, the policies can be used to drive effort to specific areas of concern. For example, since the beginning of the E4S effort a major focus area has been validation testing. The E4S team did a great job of assembling tests for dozens of products, but the E4S team does not have the depth of knowledge of each product necessary to select, or definitely write, all of the best tests for the test suite. When work on the E4S Validation Test Suite [21] began in early 2019, the idea was to populate it with some initial tests for some products, but also to engage teams to add tests for their products. Prior to September 2021, one product team submitted a pull request to add a new product to the E4S Validation Test Suite. Since then, nine product teams have submitted such pull requests. This timing aligns with the announcement that product teams would need to submit compatibility data for the ECP ST project reviews. It is reasonable to conclude that the adoption of E4S Community Policy P2 concerning minimal validation testing has driven behavior toward improving tests for the correctness of E4S software installations.

Third, the policies can be used to better align the efforts and goals of products across E4S. The Development Tools SDK project has been conducting an exascale readiness survey for the past few years. This year, the team was able also to summarize data with respect to community policy compatibility. The policies are designed to provide a level of specificity that allows individual teams to interpret the policy in their own context. Seeing data aggregated at the SDK level is useful in identifying trends where a particular policy

**P1** Spack-based Build and Installation Each E4S member package supports a scriptable Spack build and production-quality installation in a way that is compatible with other E4S member packages in the same environment. When E4S build, test, or installation issues arise, there is an expectation that teams will collaboratively resolve those issues.

**P2** Minimal Validation Testing Each E4S member package has at least one test that is executable through the E4S validation test suite (https://github.com/E4S-Project/testsuite). This will be a post-installation test that validates the usability of the package. The E4S validation test suite provides basic confidence that a user can compile, install and run every E4S member package. The E4S team can actively participate in the addition of new packages to the suite upon request.

**P3** Sustainability All E4S compatibility changes will be sustainable in that the changes go into the regular development and release versions of the package and should not be in a private release/branch that is provided only for E4S releases.

**P4** Documentation Each E4S member package should have sufficient documentation to support installation and use.

**P5** Product Metadata Each E4S member package team will provide key product information via metadata that is organized in the E4S DocPortal format. Depending on the filenames where the metadata is located, this may require minimal setup.

**P6** Public Repository Each E4S member package will have a public repository, for example at GitHub or Bitbucket, where the development version of the package is available and pull requests can be submitted.

**P7** Imported Software If an E4S member package imports software that is externally developed and maintained, then it must allow installing, building, and linking against a functionally equivalent outside copy of that software. Acceptable ways to accomplish this include (1) forsaking the internal copied version and using an externally-provided implementation or (2) changing the file names and namespaces of all global symbols to allow the internal copy and the external copy to coexist in the same downstream libraries and programs. This pertains primarily to third party support libraries and does not apply to key components of the package that may be independent packages but are also integral components to the package itself.

**P8** Error Handling Each E4S member package will adopt and document a consistent system for signifying error conditions as appropriate for the language and application. For e.g., returning an error condition or throwing an exception. In the case of a command line tool, it should return a sensible exit status on success/failure, so the package can be safely run from within a script.

**P9** Test Suite Each E4S member package will provide a test suite that does not require special system privileges or the purchase of commercial software. This test suite should grow in its comprehensiveness over time. That is, new and modified features should be included in the suite.

**Figure 9:** Version 1 of E4S community policies. These policies serve as membership criteria for E4S member packages. A member package's maturity and usability are in part measured by how well the package addresses these policies. The existence of E4S community policies has provided a strong incentive for quality improvement efforts in ECP. During recent project reviews, every team provided an assessment of their status against these policies. Almost all teams stated that they worked to improve their status against one or more policies as part of their status assessment. Teams also identified gaps, enabling the ECP leadership team to determine cross-cutting approaches to assist teams in reducing these gaps.

might be more or less well-aligned with the needs of an SDK.

While we have only just begun to leverage the quality improvement potential of the E4S Community Policies, the initial experience has been very encouraging.

# 3. SOFTWARE DEVELOPMENT INFRASTRUCTURE REQUIREMENTS

## 3.1 DEVELOPMENT INFRASTRUCTURE: KEY POINTS

1. Continuous Integration, specifically open CI environments matching those of the HPC facilities, are still the biggest gap to integration for Spack and E4S.

2. ECP relies on early-access systems provided by the leadership computing facilities. These systems support early testing of our software to find our bugs and to help the computer system vendors find bugs. These bugs are not just correct behavior but also correct design and usage assumptions.

3. In addition to DOE early-access systems, the E4S team has pioneered the practice of providing a multi-node open platform containing instances of all of the latest compute devices and software stacks. The system is affectionately called Frank (for Frankenstein). Frank has a shared file system that enables software teams to quickly test their software on a variety of new systems, getting assistance from the Frank support team. Access to Frank has greatly accelerated our porting progress and reduced the number of failures when we port to the early-access systems.

4. The availability of cloud resources from Amazon AWS, Google Cloud, and similar providers gives E4S and Spack teams blocks of free cycles for integration testing. These cloud providers see a benefit to having our software on their platforms and are happy to give us free time allocations so that they can have an optimized software stack for their customers.

5. Regression testing on leadership platforms requires careful selection of "cardinal" tests that provide optimal failure protection at a low execution cost. We have made some progress in designing these tests but much more work is needed and our teams need training to do it well.

## 3.2 CURRENT E4S DEVELOPMENT, TESTING, DELIVERY AND DEPLOYMENT RESOURCES AND WORKFLOWS

As described in Section 2.2, the Spack team and the E4S team work closely together to ensure that the core E4S packages continue to build and function correctly on key resources. This work has resulted in the creation of a CI system shown in Figure 7, with the goal of building optimized binary distributions of E4S and other stacks as shown in Figure 7b.

Spack testing has not always been so rigorous—these CI systems have emerged only in the past two years, and before their introduction, the likelihood that a particular Spack package would build successfully on the first try was much lower. When testing was performed at release time but we did not test each commit to the Spack mainline, updates to one package frequently ended up breaking a dependent package, as discovered either by vocal users or at the time of next release.

The only way we have found to guarantee that a particular software stack still builds successfully and works on a particular platform is to build and test it continuously. Thus, continuous integration and automation tools are *the most* critical piece of infrastructure for ensuring the reliability of E4S. In light of this circumstance, we have built extensive infrastructure to scale and automate the testing process. Today, we have a system that can run tests on every commit to the Spack GitHub repository.

### *3.2.1 Cloud Infrastructure*

The Spack project manages roughly 400-600 pull requests per month, and each pull request triggers continuous integration runs in a cloud-hosted GitLab instance at `https://gitlab.spack.io`. This GitLab instance lives in a Kubernetes cluster hosted in Amazon Web Services, and it is set up with multiple replicas to allow it to handle the load and volume of requests that come from the Spack community. This Kubernetes cluster is maintained by collaborators at Kitware, who have been instrumental in designing and automating Spack's

continuous integration system. They have also helped with scaling GitLab CI, which was originally designed for small projects, but is now able to build all of E4S.

There are 6 different stacks registered in the Spack CI system, two of which are variants of E4S. On each pull request to the Spack repository, our GitLab instance generates pipelines containing changes that need rebuilding, and it triggers builds of these stacks.

### *3.2.2 Build Resources*

GitLab drives CI build resources in three locations. First, the Frank system at the University of Oregon [22] provides an unrestrictive development environment accessible to both the internal E4S team and the E4S product teams so that they may work with NVIDIA, AMD, and Intel GPUs and architectures that are of interest to ECP efforts yet may not be as easily accessible to them otherwise. These diverse platforms support containerized workflows and GitLab integration.

Second, in addition to the high availability GitLab instance previously mentioned, the Kubernetes cluster also has several node pools that can spin up VM instances on demand. We use these to burst to the cloud when activity on the Spack repository is high. Currently, we use around 13 runners on Frank, but we still do around 60 percent of our builds in AWS during peak CI usage.

As discussed earlier, when a pull requests is submitted to Spack that, if merged, would affect the software products included in E4S, all affected packages are rebuilt. Build jobs for these packages are mapped to a combination of AWS and Frank systems. Tests are scheduled to systems with relevant hardware capacity via GitLab/GitLab runner tagging semantics (e.g., A100, MI100). These tags route the jobs to the appropriate nodes that provide the relevant GPU and CPU architectures or high memory capacity. Current pipelines are being modified to support post-build function testing. These PR merge jobs are critical to ensuring that changes that would break the integrity of Spack are not merged and rejected for authors to fix, with help from the Spack team. This gating is critical for sustainability, as this is where we catch bugs that would otherwise break the Spack mainline, and we ask contributors to fix them. Once the tests succeed, the PR is merged by a member of the Spack team.

### *3.2.3 Build Throughput*

The current CI pipeline averages around 100,000 builds per month, and the status of these jobs and errors encountered are available on the E4S job statistics webpage [23], shown in Figure 10. This page highlights failures in the last four hours and includes detailed statistics of job failures on AWS as well as Frank. The failures may be related to infrastructure issues (e.g., virtual machines not being ready in time on cloud runners); monitoring the past 20 failures listed on the webpage can provide an overview of any hardware or maintenance issues. Other GitLab runners are also deployed to build and test products (e.g., LLVM nightly tests with the QMCPack test case, building of E4S products, application testing) on Frank and on some DOE systems (e.g., Cori at NERSC).

Scaling this CI infrastructure requires an aggregate team of developers, including members from the Spack, Kitware, and E4S teams; currently we are working to reduce the number of infrastructure failures and to increase the throughput of the overall CI system. There is a pressing need to expand the list of E4S packages that are built as part of this automated testing to include more specialty variants of E4S software products (e.g., +rocm, +cuda, and

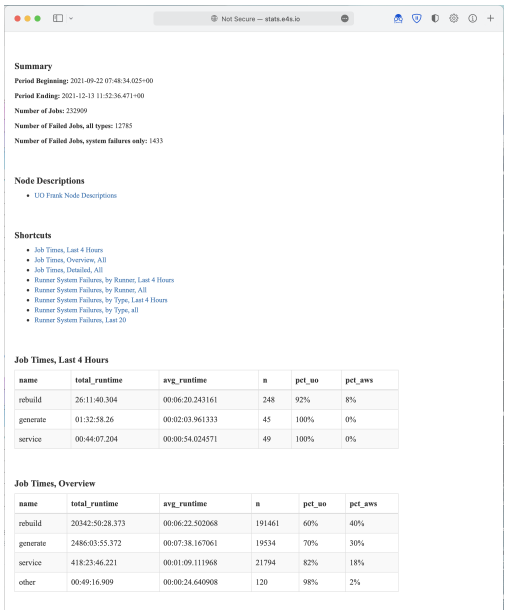



**Figure 10:** Spack CI statistics at http://stats.e4s.io. The project is averaging around 100,000 builds per month, split between Amazon and E4S.

+sycl variants for a spec). In addition, as we add more execution targets (CPU micro-architecture versions, tuned builds, and GPU compute-capability-specific builds), we will need to increase the throughput and increase load on GitLab.

#### *3.2.4 Manual Testing*

Testing is not limited to the automated testing described above. Additional manual, ad-hoc testing occurs directly on the targeted facility systems and Frank in preparation for E4S quarterly releases. Problems discovered via ad-hoc testing are described in GitHub issues posted to the Spack GitHub repository and via direct correspondence with the E4S product teams.

### 3.3 CURRENT AND NEAR-TERM GAPS IN DEVELOPMENT, TESTING, DELIVERY AND DEPLOYMENT NEEDS

The main gap in testing and delivery of E4S to facilities is the lack of dedicated continuous integration cycles available *to open projects* in the facility CI environments. Under ECP, nearly all facilities have set up GitLab CI services *inside* their security fences. Internal users can easily test changes to internal repositories on facility resources. However, for completely open source projects, it can still be very difficult to test contributions *from pull requests*, because pull requests originate *outside* the facility. Running them on DOE HPC resources currently goes against nearly all DOE security policies because pull requests may come from untrusted users who do not have accounts at facilities.

We have *mostly* addressed this problem by using resources on the Frank cluster at the University of Oregon and sandboxed AWS resources in the cloud. These environments are open, and we can run untrusted code with the mitigations described in Section 2.2. However, there are still proprietary software packages, including MPI implementations and the bulk of the Cray Programming Environment (PE) that is used on all of the U.S. exascale systems, that we *cannot* test in the cloud or on the Frank machine. Only the facilities have licenses for this software.

Recall from Figure 4 that most modern HPC applications are composed *mostly* from open-source components, and that most of them are hosted on GitHub. There is a crucial need to test applications and other critical DOE software against new changes to their dependencies, but there currently is not an environment where we can test PR contributions *and* the Cray PE. We are negotiating with Hewlett Packard Enterprise (HPE) on an agreement that might let us use a containerized version of the PE on Frank or in AWS.

Until the access issue is resolved, our testing will not *perfectly* represent the same stack that facilities deploy, and there will still be a mismatch between the testing environment and the stack that facilities that need to run our codes. Easy-to-access, publicly available CI for supercomputers is the largest and most important remaining obstacle to sustainability in the E4S ecosystem.

#### *3.3.1 Availability of Flagship Scientific Computing Resources For Software Testing*

There exists a tradeoff regarding the value of software testing and cost of HPC cycles required to perform such testing. This value per cycle proposition is extremely important to HPC facilities that seek to maximize scientific output. By definition, ASCR's leadership computing facilities seek to prioritize workloads that are very difficult or impossible to perform elsewhere. As such, software tests that can be run on commodity testbeds or in the cloud may be unsuitable for HPC systems, but many tests can be performed only on HPC systems. Unfortunately, HPC systems have a long lead time to procurement, and current platforms have not factored in scientific software testing workloads, such as:

- Modest additions to existing HPC resources to allow for additional software testing;
- Support for continuous integration infrastructure within existing facilities; and
- Flagship HPC system-specific test maintenance and team-facility collaboration by software development teams.


**Figure 11:** End-to-end software build and test infrastructure.

Potential research activities in this space include algorithms to detect redundant testing to prevent wasted cycles on HPC resources. For example, if a test passes on a commodity cluster or Spack testing pipeline, can a facility testsuite determine whether there is value to re-running the test in the facility environment? A descriptive composition of tests, test results, and environment may need to be created for this purpose. In addtion, ways to ascribe a value metric to a test and collection of tests may help software teams and facility staff develop policies around appropriate use of testing to align with value per cycle thinking.

Other research activities include integrating performance testing for scientific software at the right granularity on flagship resources; such functionality does not exist today within this ecosystem. Because current facility acceptance testing is heavyweight, it is typically performed infrequently. Research into lightweight performance tests, whether they be runtime, bandwidth, space, or energy related, will help to detect performance regressions.

### 3.4 THE EMERGING ROLE OF CLOUD AND CONTAINERIZED ENVIRONMENTS

The E4S team collaborates directly with ECP ST teams to support reproducible, containerized deployments of scientific software on HPC facilities. Examples of such collaboration includes the development and deployment of custom E4S containers on Summit at ORNL for the ExaWind application [24] that includes the Nalu-Wind and AMR-Wind codes. The workflow was tested on 1024 nodes with 6144 NVIDIA GPUs at scale using Singularity. The E4S collaboration with the Pantheon [25] team allowed consistent reduction of build times by 10x by leveraging a custom E4S Spack build cache hosted on AWS S3 cloud infrastructure [26]. E4S provides full-featured container images for Docker and Singularity that allow users to explore the vast array of ECP E4S software products on their local systems as well as to deploy these images at scale on supercomputers. Similarly, commercial cloud environments such as AWS are supported by provisioning custom Amazon Machine Images (AMIs) that are deployed on AWS EC2 instances.

### 3.5 EMERGING INFRASTRUCTURE RISKS AND MITIGATIONS

The ECP software development, testing, delivery, and deployment infrastucture consists of three major parts. Build specification and test recipes are contributed to Spack by software development teams. These Spack packages are built, tested, and curated by E4S according to community needs. Facility and cloud CI infrastructure is used by E4S to automate the most valuable tests while also being available for individual software teams to use at their discretion. This general workflow is shown in in Figure 11.

HPC Facility site-local testing by E4S and ECP ST teams has steadily increased as early-access systems such as Spock at the OLCF and Arcticus at the ALCF have been made available. Facilities with ECP CI platforms average about 2 CI-related user tickets per week, and are running between 2500 to 4000 CI pipelines per month with job times ranging from 1 to 120 minutes [27]. Facilities are reporting that their CI resources are adequate to satisfy user needs at this point in time, but there are many more tests yet to be written as the ecosystem grows. The following are key emerging risks to the sustainability of scientific CI infrastructure:

1. **The end of linear scaling.** Spack and E4S have enabled scalable software testing by enumerating and reducing the complexity of the ECP ecosystem. Additional software products need only adhere to the E4S community best practices, and additional deployment destinations need only interface with E4S for support. If this scaling were to end, complexity could render the software ecosystem unsustainable. One such threat to this scaling could be continuing to support deprecated software, E4S deployments at facilities, or aging hardware and system environments long after they are useful. Research opportunities

exist regarding the quantification of value in supporting older software, platforms, and architectures based on cost benefit analysis.

2. **Failure of a single common depenency: a reverse iceberg Problem.** Spack makes building sets of packages from dependencies easier by automatically determining an appropriate set of dependencies and variants thereof. An inverse problem exists where a key failure in a *superdependency* that is common to many Spack packages may be disproportionately impacting to the software ecosystem.

3. **Loss of support or talent from key facility stakeholders.** HPC facility support is crucial to the continuity of the scientific software ecosystem. Many tests must be performed on the bespoke systems found at HPC centers. Software deployment engineers from facilities provide feedback to guide new Spack and E4S features, maintain facility CI testing platforms, and play a liaison role to help integrate software developers into the HPC environment.

4. **CI platform infrastructure inadequacy.** GitLab CI infrastructure has proven to be sufficient so far during the lifetime of ECP for the workloads targeted there. If future automation needs require strong levels of trust between platform components or significantly more pipeline tasks to be executed, the security model and operational scaling of the current platform may need to be re-examined to discover and solve fundamental issues. Research into securing scientific workflows will transfer directly to the GitLab CI infrastructure; at its core, the CI server / distributed runner model is similar to directed scientific workflows.

5. **Undersizing of bespoke testing resources.** Finally, world-class HPC systems are a combination of unique, cutting edge, and expensive components. These properties tend to mean that resource allocations are extremely valuable, and a fundamental tension may exist between allocating cycles to scientific outcomes versus towards software assurance. Research to discover methods of quantifying the value of dedicated allocations to software assurance may serve to support the appropriate sizing of these world-class systems so that there are adequate resources for both science output and software assurance.

# 4. DEVELOPING AND MAINTAINING COMMUNITY SOFTWARE

## 4.1 COMMUNITY SOFTWARE: KEY POINTS

1. Current funding models make it difficult for a scientific software team to increase the sustainability of their software products. Short-term funding for projects and the lack of sustainable software career paths are two major concerns.

2. DOE has established a rich business model with computer system vendors, providing highly-leveraged funding to create new technology that can be cost-shared with other customers as it goes into production. We need a similar model to share the support costs for DOE-developed software products. Establishing a relationship with a software support company that can provide DOE and other users (industry and other agencies) with software support for E4S provides a similar kind of win-win-win outcome: DOE reduces its support costs, external users get the kind of direct support that enables them to invest in E4S use, and the whole scientific computing community gets a higher-quality scientific software stack in E4S.

3. Package management is critical for leveraging work from the broader open source community. E4S relies critically on the broader Spack community for E4S's dependencies and for the tooling that allows scientists to use them.

4. ECP-funded software efforts are deeply engaged in the broader scientific computing community. Our staff are members and leaders on software standards committees such as C++ standards, LLVM, OpenMP, MPI Forum, and more.

5. DOE math, I/O, and visualization libraries and tools are widely used within DOE and beyond, but there is much latent potential to increase impact. The use of capability integrations (our KPP metric) promises to accelerate the expansion of our impact.

6. ECP Co-design centers have demonstrated an effective model for bootstrapping new reusable software products. The co-design model is essential as an approach for new software domains in the future.

7. Some of the most impactful open-source scientific products are incubated in the NNSA labs, where software development funding has historically been provided over a longer time arc. At the same time, a relatively modest investment from DOE ASCR to these same NNSA software teams has provided tremendous value to everyone. As ECP ramps down, continued ASCR funding is required to enable NNSA teams to interact with the broader community.

### 4.2 PROGRESS AND IMPEDIMENTS IN DEVELOPING AND MAINTAINING DOE LIBRARIES AND TOOLS

It is difficult to accurately separate the cost of developing code for a small group of users versus providing it to a larger community. There are certainly explicit costs, such as careful interface design for users who did not write the software, documentation of the software for third-party use, setting up user support mechanisms such as support portals, issue tracking, tutorials, and more. At the same time, if libraries and tools are considered substitutes for what an application team would have to otherwise provide for themselves, then reusable libraries and tools can often be better, faster, and cheaper than what any team could develop independently.

The ECP sponsors approximately 25 application projects, 35 software technology projects, and 7 co-design projects. Assuming similar budgets [28], some simple arithmetic would imply that an emphasis on reusable software libraries and tools is not substantially more expensive than the development of individual applications that are not targeted for broad reuse. A few reasons why this could be true is that reusable community software provides some of the most important functionality within the scientific software ecosystem, in a way that is accessible to many clients. The cost of designing, implementing, documenting, and delivering these reusable products is roughly similar to creating unique application codes, at least to a first-order approximation. In addition, each time a reusable product is in fact reused, the proportional cost of that use is relatively small, given that the product was created for reuse. However, the long-term costs to develop and maintain community software are difficult to predict and are a function of several variables. The book *Working in Public: The making and maintenance of open-source software* [29] by Nadia Eghbal describes this complicated situation in some detail.

One attribute of publicly funded research software is the challenge of providing explicit funding for software maintenance. In most communities, maintenance funding is provided as an implicit tax on incoming research funds. Because this maintenance cost can apparently be kept modest, sponsors do not seem to object to this kind of overhead. The problem of this model emerges when a product remains useful to the community but is no longer perceived as a valuable research funding target. In this situation, the widely used but not funded product often decays and becomes unusable.

The lack of an explicit support model for community software is also an issue when commercial organizations consider the use of the software. While the software itself could be very useful, the lack of a supporting entity that could take in commercial funds leads to a reluctance from industry to use community developed libraries and tools. Establishing a business model that enables industry contributions for maintenance and support would be attractive.

### 4.3 EMERGING BUSINESS MODELS FOR IMPROVING COMMUNITY ADOPTION OF DOE SOFTWARE

The emergence of the E4S software portfolio has enabled a new business model. The E4S support team can engage with DOE leadership computing facilities at a peer level not possible for any individual E4S member product. The aggregation of products into the E4S portfolio means that we can support DOE facility users via a hierarchical support model. The first level of support is typically a facilities staff member who is contacted by an application team to help address an issue they are seeing in compiling or executing their code. Sometimes the staff member can identify and fix an issue directly. However, sometimes the fix is not possible. In this situation the E4S support team takes responsibility for identifying the software issue, fixing it themselves, or identifying which E4S member product is causing the issue. This tiered support model

has become very attractive and is improving the willingness of application teams to use E4S products when compared to the past.

In the future, we can further leverage this business model by outsourcing E4S support to a private software company that specializes in knowledge of E4S and how to support it (there is already one such company). Then, in addition to supporting DOE facilities users, the private software company can also accept funding from E4S users who are outside DOE, including other US agencies and private industry. This kind of distributed support model of E4S increases the user base beyond DOE and distributes the cost of support. Furthermore, it increases the value of the software to US industry, a long-desired goal of our DOE software efforts. Specifically, because E4S includes many new capabilities for emerging GPU platforms, other agencies and industrial users can make a more rapid transition to new hardware platforms that are presently too difficult for their user and developer communities to adopt. The transition to GPU-based systems is essential for anyone who wants to obtain high performance computing capabilities in the future.

### 4.4 ENGAGEMENT WITH PACKAGING TECHNOLOGIES

Packaging technologies and the role of Spack in E4S are discussed in detail in Sections 1.3 and 2.2. Package management has been instrumental in automating the process of software reuse, and it is the glue that holds modern software ecosystems like E4S together. As shown in Figure 5, the E4S stack comprises around 100 software products, but it relies on another 400-500 from the broader open-source ecosystem. This is made possible through packaging, specifically by tight integration with the 6,000 packages in the mainline Spack repository.

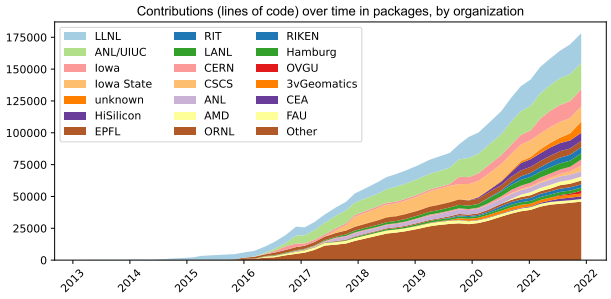



**Figure 12:** Contributions to Spack's 6,000 packages over time, by organization. Most contributions to the Spack repository come from outside of DOE, but DOE maintainers ensure that these contributions are vetted and integrated into the Spack mainline.

Figure 12 shows all contributions, in lines of code, to Spack packages over time. There are over 175,000 lines of package recipe code in Spack, and less than 20% of them come from DOE national laboratories. Despite this, there is still considerable effort required to sustain these contributions. A group of 5-6 core developers at DOE labs manage the core Spack tool, and package contributions are managed by a team of 30-40 trusted maintainers (including some of the core developers). In addition to these core members, there are over 150 "package maintainers" who provide input and reviews on contributions to specific packages.

The E4S team already has a close relationship with the Spack project, and moving E4S forward over time will require continued improvements to Spack to manage the increasing complexity of the software ecosystem. Core development is required to add features to Spack's core so that it can continue to handle more complex relationships between packages. Specifically, compiler interoperability, increasing hardware heterogeneity, and the likely need for better cross-compilation and bootstrapping support in the future will likely require deep changes to Spack's package DSL and the solvers that it implements to understand dependency relationships. The Spack package ecosystem will also continue to grow, and maintainers will need to manage and review contributions from an even larger community of contributors. If the Spack community were to languish or to collapse under its own weight without sufficient core development, much of the Spack ecosystem would need to be reinvented within the E4S project to sustain E4S.

### 4.5 COMMUNITY ENGAGEMENT FOR DOE PROGRAMMING MODELS

The ECP ST Programming Models and Runtime portfolio comprises several major software development efforts that have large communities of users (e.g., MPICH, Open MPI, Kokkos, RAJA).

MPICH is primarily developed at Argonne National Laboratory, with contributions from vendor partners and external users. The project has a long and successful history of developing software that is widely used and forms the basis of MPI implementations running on many of the largest supercomputers in the world. The project has historically been funded through ASCR Research and most recently through ECP. The funding has historically been for research into efficient and scalable MPI implementation and high-performance

communication in general for large supercomputers. The additional effort needed for developing sustainable software that is usable by others, documentation, outreach and training, responding to user questions and bug reports has been done by the developers in their spare time and out of their own interest because they recognized the value in it. The project has also participated over the years in the MPI standardization effort in the MPI Forum, contributing to successive generations of the MPI standard. The standard communication interface provided by MPI, along with the availability of high-performance portable implementations of MPI, have enabled the development of portable parallel applications and libraries and led to the widespread use of parallel computing for scientific applications. Open MPI is another popular MPI implementation with a distributed development effort spanning multiple institutions, including vendors, and has been funded through ECP ST.

Kokkos and RAJA are more recent efforts that provide a C++-based programming abstraction for node-level performance portability across multiple heterogeneous node types (multicore CPUs and GPUs from different vendors). They have gained significant adoption in the ECP application community as well as other applications because they provide an effective solution to the performance portability challenge posed by modern architectures. The core technology has been developed under NNSA funding in support of NNSA applications, and ECP ST has provided additional funding to enable them to reach out to the broader set of ECP applications and for efforts related to sustainability and outreach/training, which are essential to increasing the adoption of these products. When ECP ends, funding will be needed to sustain these efforts for the wider DOE community beyond the NNSA applications that their primary funding supports. Both projects also work with the ISO C++ standards organization to get necessary features added to future versions of the C++ standard, so that concepts from these efforts are available as standard C++. Kokkos and RAJA support programming at the node level, and applications use MPI + Kokkos/RAJA for portable applications that scale to the full machine.

### 4.6 DOE ENGAGEMENT WITH THE LLVM COMMUNITY

Over the past decade, DOE and ECP staff have developed rich engagements with the international, open source community that develops and supports the LLVM compiler ecosystem (e.g., see llvm.org). Currently, our ECP strategy focuses on improving the open source LLVM compiler and runtime ecosystem; LLVM has gained considerable traction in the vendor software community, and it is the core of many existing heterogeneous compiler systems from NVIDIA, AMD, Intel, ARM, IBM, and others. Specifically, ECP funds three projects where the main deployment strategy is that of upstreaming changes to the public LLVM software. They are 1) SOLLVE - Improving OpenMP for LLVM, 2) Flang - Developing Fortran for LLVM, and 3) PROTEAS-TUNE - Developing OpenACC for LLVM. Additionally, all three projects contribute expertise and improvements to the broader LLVM ecosystem as expected in an open source project of this significance. These activities include reviewing source code changes proposed by other authors and improving infrastructure such as the LLVM testing framework. Many of our LLVM contributions address these trends for the entire community and will persist long after ECP ends. For example, our contributions for directive-based features for heterogeneous computing (e.g., OpenMP, OpenACC) will not only provide direct capabilities to ECP applications, but will also impact the redesign and optimization of the LLVM infrastructure to support heterogeneous computing. In a second example, Flang (open source Fortran compiler for LLVM) will become increasingly important to the worldwide Fortran application base, as vendors find it easier to maintain and deploy to their own Fortran frontend (based on Flang). Furthermore, as Flang becomes increasingly robust, researchers and vendors developing new architectures will have immediate access to Flang, making initial Fortran support straightforward in ways similar to what we are seeing in Clang as the community C/C++ frontend.

In terms of sustainability, DOE supports LLVM by participating in the LLVM community; this mode of sustainability has several differences. First, the LLVM community has to accept changes offered by DOE; they can always decide certain packages are not important enough to include in the public LLVM releases. Second, the community could decide to focus on other design priorities, driving LLVM in a direction that is less attractive for HPC. Third, DOE does not have a long-term model for funding continued support for these types of community software packages. That is, after ECP ends, it is unclear how the ECP contributions will be maintained in LLVM. This fact alone may lead to the rejection of ECP requests by the LLVM community.

### 4.7 DOE-FUNDED PERFORMANCE TOOLS AND LIBRARIES

DOE has a strong history in creating performance tools that specifically support HPC platforms and programming models (e.g., MPI). These tools are open-source, long-lived (20+ years in two cases), and widely deployed across systems in DOE and around the world. ECP currently supports three specific tools projects: 1) ExaPAPI - library and tools for standard access to performance and power analysis using hardware counters, 2) HPCToolkit - performance analysis tool framework that uses instruction sampling to reduce the perturbation and data volume produced by performance analysis of complex systems, and 3) TAU - performance analysis framework that provides comprehensive analysis of node and system-wide performance metrics and is compatible with nearly all HPC programming models. It should be noted that these tools were some of the original open source software sponsored by DOE ASCR.

An important point is that performance tools are very sensitive to the underlying hardware and low-level system software (e.g., operating systems, drivers, hardware performance counters). As such, their sustainability requirements can be considerably different from other software packages. First, unanticipated changes in hardware, system software, or deployment can inhibit the tool's functionality to the point of making it unusable, so the teams maintaining the tools must be readily capable of maintaining this software to keep pace with these changes. Moreover, these teams must be in close contact with other tools teams and vendors in order to identify potential problems and provide fixes prior to deployment. Second, package managers like Spack and Conda typically cannot control the configuration of the Linux operating system or its drivers. Third, automation of sustainability best practices like continuous integration is more complex because the package management, build, and execution of tests is substantially limited by these two prior constraints.

### 4.8 DOE MATH LIBRARIES

Decades of DOE investment have led to a diverse and complementary collection of mathematical software, including AMReX, Chombo, Dakota, DTK, hypre, MAGMA, MFEM, PETSc/TAO, PLASMA, ScaLAPACK, SUNDIALS, STRUMPACK, SuperLU, and Trilinos. DOE-supported libraries encapsulate the latest results from mathematics and computer science R&D; many DOE mission-critical applications rely on these numerical libraries and frameworks to incorporate the most advanced technologies available.

As of a few years ago, if an end user needed to employ multiple packages aforementioned, a productivity bottleneck resulted due to the challenges of handling each package's installation idiosyncrasies and overcoming incompatibilities among different packages. Recent work in the community xSDK project [10] mitigates these issues in several ways. The project focuses on community development and a commitment to combined success via *xSDK community package policies*[3], which prescribe a set of minimum requirements (e.g., configuring, installing, testing, MPI usage, portability, contact and version information, open-source licensing, namespacing, and repository access) that a software package must satisfy to be included in the xSDK. In addition, the xSDK community has been developing interoperability layers among numerical libraries in order to improve code quality, access, usability, interoperability, and sustainability. The first release of xSDK in 2016 contained only four math libraries (hypre, PETSc, SuperLU and Trilinos). The latest xSDK release in November 2021 contains 24 packages, which are all compatible with the community policies and can be installed via Spack package manager. Many of these packages are also included in the latest E4S release.

Going forward, the following challenges and opportunities arise, which require sustaining resources to support these activities. For wider community use, especially for non-HPC-savvy users, the sheer size of the xSDK release can be intimidating. Most of our recent R&D in math libraries aims at exascale computers and applications. Therefore, the software toolchain is deep, depending on many HPC system software libraries, such as MPI, OpenMP, CUDA, ROCM, and so on. These are well tested on several DOE flagship machines. For many academic or commercial users who want only to solve one single mathematical problem, we can provide "xSDK-lite" versions with gradual ramping up of the library components and functionality. We also need to develop tutorials with varying levels of use scenarios and sophistication to help lower the entry-level threshold. These tutorials can be given at various training venues, such as ATPESC, SC conferences, and SIAM conferences. The training materials and videos will be accessible on the xSDK website.

Further investment is needed to develop community standards for code documentation. At present, many

[3] https://xsdk.info/policies

packages use standard tools, such as Doxygen, for generating documentation from annotated source code. However, the level of detail in annotation is highly non-uniform. Sometimes the user-callable routines do not clearly specify the user's responsibility of allocating/deallocating certain data structures, which may cause memory violation deep in the software stack. We need to develop some standards with regard to how to annotate input arguments, output arguments, and error conditions for each user-callable API.

The computational reproducibility is critical for the credibility of scientific results. At present, most of the libraries do not guarantee reproducibility. Traditionally, reproducibility means getting the bitwise identical answer when running the same program repeatedly. Bitwise reproducibility requires imposing severe constraints on the order of execution for operations on a parallel machine, which is often too costly. When relaxing the bitwise reproducibility requirement, as long as the key math kernels (e.g., linear solvers) guarantee certain error bounds on the solutions, the correctness of scientific simulations are still ensured. Therefore, it would be beneficial to invest in error analysis work in various math libraries in order to understand error bounds and deliver them to the users. In addition, we may consider having a debugging mode in the code to enforce the same order of operations albeit at a lower speed, very useful for validation purposes. For performance reproducibility, we need to ensure that the program is run in the same computing environment with the same system toolchains. Having a history database to record the software versions and configurations for each execution can achieve performance reproducibility.

Currently, most math libraries do not have explicit funding dedicated for software maintenance and user support. These activities are mostly carried out in an ad-hoc fashion using developers' spare time. For a long time, computer vendors have adopted some commonly-used math libraries developed by academia and labs in their core software stack. The most notable examples are BLAS and LAPACK. In recent years, we have seen more of the DOE's high-end parallel math libraries (such PETSc and Trilinos) being added in the vendors math library distributions, e.g., Cray Scientific Library and Intel MKL. Through ECP, we have made good connections with several GPU vendors to develop standard interfaces for key linear algebra computations, such as batched dense and sparse matrix operations and linear solvers. These types of industry engagement could generate broad impact on commercial applications, and the efforts need to last far beyond the ECP project's lifetime. Hence, sustaining funding is highly desirable.

Looking beyond ECP and the post-Moore's Law era, future supercomputers will consist of many forms of heterogeneous accelerators, not only GPU accelerators. Examples include FPGA and various AI architectures like Google TPU. It will become inevitable to invest in refactoring the algorithms in the math libraries for these extreme heterogeneous architectures and for problems coming from AI and machine learning domains.

### 4.9 DATA AND VISUALIZATION

DOE supports most of the research, development, and deployment activities associated with large-scale scientific data and visualization packages. For data, these packages include HDF5, PnetCDF, MPIIO and ADIOS. Development occurs at both Office of Science and NNSA laboratories, while deployment is primarily supported by the HDFGroup Inc. and DOE facilities staff. For visualization, these packages include ParaView and its insitu mode, Catalyst, Visit and its insitu mode, LIBSYM, Ascent (a lightweight in situ library), Cinema, VTK, and VTK-m. Development occurs at both Office of Science and NNSA laboratories, while deployment is primarily supported by Kitware Inc. and DOE facilities staff. From our experience in ECP, the additional effort to develop and maintain packages for use by the wider community (above the effort needed to develop and maintain software packages solely for use in specific research projects or for internal use) is likely three quarters of the ECP software support that is currently received. The other quarter is focused on applied research to solve specific technical challenges as they arise. ECP software support consists of high-quality software engineering, including requirement gathering, design, implementation, user support, bug fixes, testing, and user training. Tasks that are the largest contributors include application outreach, functionality improvements, and porting to new architectures.

Before the ECP project, it was difficult to sustain data and visualization software packages due to a lack of steady long-term funding commitments to existing data and visualization software packages, and a lack of external community support from industry and other agency users for these packages. A good example is the H5Py library. This library is used extensively by the industry ML community for I/O, but there is a lack of industry support to the HDFGroup to maintain, update, and modernize this library. Another challenge before ECP was the lack of coordination between laboratories. For example, some laboratories provided

small contracts for specific features to industry partners such as HDFGroup or Kitware. These independent contracts, however, did not fully fund the long-term cost of maintaining the collection of features developed and the unintended feature interactions that occurred as more features were added over time.

The largest non-monetary impediments to this additional work is community commitment to a more centralized approach. A centralized approach offers cost savings due to resource sharing. For example, ECP is currently experimenting with contracting direct DOE user support for building and testing via a commercial company. However, from each individual project's perspective, a community approach may not be considered in its best interest. For example, in a distributed approach each project may receive a small amount for such user support. Centralizing this support, while a globally good idea, might look like a loss in support to each project.

One advantage of a project like ECP has been the ability to provide higher-level direction to the data and visualization communities that in turn benefit the scientific community. In the data space, ECP has directed each data project to support a HDF5 I/O interface (via providing an HDF "implementation" via the HDF "VOL" abstraction layer). ECP's officially supported data API is the HDF5 interface. This approach is beginning to show significant benefits, including faster I/O time for scientific applications via the HDF implementations by the ADIOS, ExaIO and DataLib project teams. Visualization projects are arranged globally in a pipelined fashion, with the ALPINE project scope focusing on providing in situ algorithms and infrastructure as well as outreach to scientific applications. The ALPINE project achieves high performance through its dependency on VTK-m, an ECP project that focuses on the multi-threading implementation of visualization algorithms. Before ECP, two visualization teams might compete to try to provide all this functionality, leading to duplication of efforts and the development of inferior packages.

In summary, investing in developing and maintaining data and visualization community software supports the scientific application community with stable, evolving, full-featured data and visualization software they can count on. In addition, centralized support of building, testing, and outreach enables economies of scale and tighter integration across the software stack. Directed management of multiple data and visualization projects reduces duplicated efforts between projects and provides a more unified vision of where the community is going collectively.

### 4.10 BOOTSTRAPPING NEW REUSABLE SOFTWARE FRAMEWORKS VIA CO-DESIGN

One of the core activities of the ECP has been identifying a creating a collection of new software libraries and frameworks through application-library co-design activity. These frameworks represent relatively new code bases started in anticipation of common requirements in preparation for exascale platforms. Co-design activities focus on adaptive mesh refinement, efficient discretizations, online data analysis and reduction, particle-based application technologies, graph algorithms, machine learning, and proxy applications [30].

While all ECP libraries and tools teams work with their users to design and develop new capabilities, the co-design centers place a stronger emphasis on co-evolving with their target application codes (the user codes), requiring an informed exchange for the new algorithms and functionality they must provide. One area targeted by the co-design effort is adaptive mesh refinement (AMR). Prior to ECP, existing AMR codes did not provide robust GPU support. Designing and implementing GPU-scalable AMR functionality is essential to many scientific applications and difficult to accomplish. The AMReX (AMR for Exascale) co-design center is establishing a framework and collection of libraries that provide the critical functionality and GPU performance in a single code base that can be used by many application codes.

The co-design model provides a way to bootstrap new and widely needed functionality, providing these capabilities to the open-source scientific software community. Any future organization for sustainable scientific software needs to include a co-design pathway that would provide this bootstrapping process. Then, as the co-design phase completes, the resulting products would transition into the broader collection of software products that, while still evolving, have a stable core design and user base.

### 4.11 NNSA OPEN SOURCE PRODUCTS

The ECP NNSA Software Technology portfolio represents a broad range of software products spanning the scope of the rest of ST: Programming models and runtimes, development tools, math libraries, data analysis and visualization, and software ecosystem and delivery. The goal of the NNSA ST portfolio is to support NNSA national security applications on associated exascale platforms. All of the products in this portfolio

are open source, important or critical to the success of the NNSA national security applications, and are used (or potentially used) in the broader ECP community. More than half of the products in this portfolio also have complementary funding from the Office of Science in ECP, which enables the product teams to broaden their scope and additionally support ECP applications and platforms.

The benefit of having NNSA projects as part of ECP is that the additional funding from ECP has enabled interaction and outreach by the teams to the wider community. The benefit is bidirectional, in that both NNSA and the wider ECP have benefited. There are numerous examples of this bidirectional benefit, but in the interest of brevity we highlight only two examples. The first is the Spack package manager from LLNL, which began with funding by the ASC program and was intended to support the build complexity of NNSA multiphysics applications. In ECP, Spack is currently funded through both the NNSA ST and software ecosystem and delivery areas. The increased funding and interactions from ECP participation have led to Spack becoming a key dependency in the success of E4S. Nearly all ST projects now depend on Spack for easing build complexities. The second example is Cinema from LANL, which is a lightweight, interactive framework for analyzing and visualizing scientific data. In ECP, the Cinema team has tight collaborations with ALPINE, which is part of the data analysis and visualization area. Together, these teams have produced end-to-end workflows for ECP applications, showcasing how to use in situ analytics techniques with Cinema to provide deep understanding of simulation results.

Overall, the interactions between NNSA ST and the wider ECP community highlight the need for funding that encourages interaction between software development groups that might otherwise be siloed. The interaction provides opportunities for teams to improve their software by adapting it for a wider array of customers, making their software more robust overall. In addition, outreach activities encourage customers to adopt and support software that already exists instead of inadvertently re-inventing the wheel because they were unaware of the existing software.

# 5. BUILDING A DIVERSE WORKFORCE AND INCLUSIVE ENVIRONMENT

## 5.1 DIVERSITY AND INCLUSIVITY: KEY POINTS

1. DOE scientific computing staff have very desirable skills in the broader computing community. While we do not have summary data, the ECP leadership has seen attrition, especially among mid-career emerging and established leaders.

2. Embracing the likelihood of a steady outflow of staff, we can benefit from a more comprehensive recruitment program that reaches out to non-traditional communities. These recruitment efforts need to take into account cultural barriers as we try to engage the next generation of scientists.

3. ECP has provided its funded staff an opportunity to work with scientists from many DOE, university, and industry groups. Many ECP staff consider themselves to be members of the DOE complex as much as they are members of their home institution. DOE program management can continue this broader community membership strategy, enabling a richer cultural and geographic landscape for staff in the future, providing people with career options across the DOE laboratory complex.

4. Social and cognitive sciences are an important and increasingly necessary skill set for scientific software teams. Increasing the diversity of our teams to include members with these skills will further improve our ability to incorporate new staff with diverse backgrounds and perspectives.

## 5.2 CHALLENGES OF RETAINING TALENT WITH HIGH-DEMAND SKILLS

To attract and retain top talent, we need to ensure that staff are recognized as a critical element of the DOE mission and its success, that they are part of a larger community of software stewardship professionals, and that there are long-term career paths and development opportunities. The camaraderie of groups and teams (sometimes co-located, sometimes distributed) can also contribute to retention. In terms of development opportunities, we also must recognize that there are multiple tracks for career advancement, including developing and applying new technical skills, as well as developing leadership skills and opportunities to apply

those skills by becoming leaders in teams and organizations. A premier leadership course that draws from staff across the effort into a cohort could be a useful addition for staff wanting to develop leadership skills.

In addition to attracting and retaining staff with traditional skill sets, we find that expanding the range of skills we can effectively use on a software team goes beyond typical software development tasks. Specifically, people with good design and analysis skills can make large contributions to upstream activities, before implementation starts. The broader software community is also recognizing these opportunities [31].

### 5.3 CHALLENGES AND OPPORTUNITY FOR DIVERSE WORKFORCE DEVELOPMENT

Addressing workforce challenges in high-performance computing requires broad community collaboration to change the culture and demographic profile of computational science. Impactful DOE-wide programs, lab-specific initiatives, and activities in the wider computing community are making headway. The DOE HPC community has a compelling opportunity to address workforce challenges through a complementary lens that focuses on the distinct needs and culture of DOE high-performance computing. Consequently, in August 2021 ECP established a task force on broader engagement, whose members represent ANL, BNL, LBL, LLNL, LANL, ORNL, PNNL, and SNL, as well as DOE Office of Science Computing Facilities ALCF, NERSC, and OLCF. The task force, building on the special collaborative character of ECP, has begun a broader engagement initiative[4] to expand the pipeline and workforce for DOE high-performance computing through three complementary thrusts:

- Establishing an HPC Workforce Development and Retention Action Group: fostering a supportive and inclusive culture in ECP, DOE labs, and communities, to promote the workforce pipeline for, and the retention of, a diverse DOE lab HPC workforce;
- Building an "Introduction to HPC" Training and Workforce Pipeline Program: developing and teaching accessible beginner material; and
- Creating Sustainable Research Pathways for HPC (SRP-HPC): faculty/student internships with community/mentoring.

The ECP broader engagement initiative is a partnership among the ECP and facilities communities, as well as with lab staff who focus on education and workforce issues.

An overview of initial work by the task force, including responses to RFI questions related to workforce issues, is available [32]. These preliminary steps are just the starting point of long-term work to change the culture and demographic profile of HPC computational science. We face two urgent challenges: (1) the ECP project will conclude in 2023, and (2) we need to engage the entire DOE HPC community in this initiative, not just people funded through ECP. A critical challenge is identifying a path toward long-term sustainability of work initiated by the task force to broaden participation in DOE high-performance computing, including next phases beyond initial pilot projects.

### 5.4 THE EXPANSION OF SCIENTIFIC SKILLS: RESEARCH SOFTWARE SCIENCE

Scientific software teams have become larger and more diverse as our attempts to model and simulate phenomena and analyze data have become more sophisticated and diverse. Diversity arises from incorporating multiple scientific domains, such as when exploring multiphysics and multiscale concerns, and also by recognizing the value that disciplines such as computer science and software engineering can bring to scientific software efforts. Computer scientists can help scientific software teams by improving algorithm and data structure techniques that improve execution time and programmability. Software engineers can help teams by introducing effective and efficient processes and tools for software development, delivery, and sustainability. The introduction of software engineers into research software teams has become so successful that the job category of Research Software Engineer (RSE, see `https://society-rse.org` and `https://us-rse.org`) has emerged as an increasingly recognizable career track, helping to build community, mentoring, and professional recognition for scientific software specialists.

[4] https://exascaleproject.org/hpc-workforce

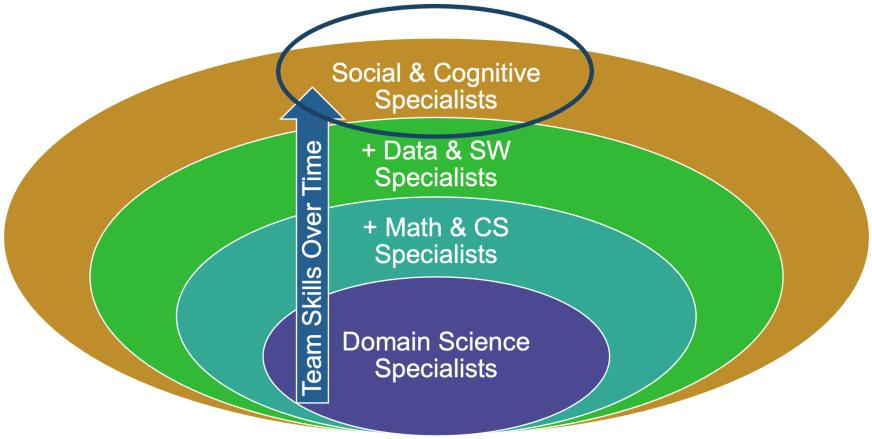



**Figure 13:** The addition of social and cognitive sciences to the collection of skills on a scientific software team will enable the understanding and improvement of how individuals and teams engage in the development and use of software to conduct scientific research. The use of the scientific method to understand and improve how research software is developed and used is called Research Software Science (RSS).

Presently, the size and diversity of scientific software teams can benefit from introducing additional scientific disciplines, specifically knowledge from the cognitive and social sciences (Figure 13). Adding scientists with skills in these areas enables the study and improvement of the efforts of individuals and teams who develop and use software for research. This use of the scientific method to understand and improve how research software is developed and used is called Research Software Science (RSS).

# 6. TECHNOLOGY TRANSFER AND BUILDING SOFTWARE COMMUNITIES

## 6.1 TECH TRANSFER AND SOFTWARE COMMUNITIES: KEY POINTS

1. The ECP focus on capability integration as the core metric for success forces software teams to plan for where their software capabilities will be sustainably integrated. These plans are also assessed for credibility, leading to corrections early in the project. Regularly failing to integrate capabilities into the client environment provides the ability to descope projects that are not providing sufficient value.

2. Interactions with the ECP Industry and Agency Council have enriched every aspect of our work and provided us with insight into how we can more effectively provide E4S products to industry and agency partners.

3. The Leadership Scientific Software (LSSw, `https://lssw.io`) Town Hall meetings are providing a forum for engaging with the broader scientific software developer and user communities, enabling new opportunities to collaborate and broaden the impact of DOE scientific software work.

## 6.2 DRIVING TECHNOLOGY TRANSFER BY PLANNING AND EXECUTING CAPABILITY INTEGRATIONS

In Section 9.2 we describe the ECP capability integration framework and scoring system. Capability integrations are the fundamental metric of reusable software libraries and tools in ECP. There are other meaningful attributes such as good interfaces and documentation, good user support and more, that can indicate if a product is successful. However, these other attributes tend to be strongly correlated to whether or not a product is sustainably integrated into the larger ecosystem that will enable long term availability, use and support.

Capability integrations are important for technology transfer because many scientific libraries and tools must eventually find their home in software ecosystems that DOE does not directly fund. Examples include

computer system vendor software stacks, community ecosystem such as LLVM, and community programming models and standards such as MPI, OpenMP, C++, and Fortran.

As part of planning a software development capability, ECP requires teams to identify one or more integration targets for the planned capability. Then we track the progress of integration from beginning to end. By focusing on integration from the beginning we can adapt plans and better assure the successful transfer of products belonging in ecosystems outside of DOE.

### 6.3 LESSONS LEARNED FROM ECP INDUSTRY AND AGENCY ENGAGEMENT

Industry is focused on solutions for specific problems. E4S products need to fill a targeted need so it is easy for the users to choose between alternatives with overlapping functionality. The products should be available in a binary form (in E4S build caches) that support rapid assembly of related products on a target platform and for creating custom images starting with a base image for a specific GPU architecture.

E4S community policies ensure that software included in E4S releases adhere to certain standards and contain necessary building blocks that help support continual quality assurance. Good packaging can help kickstart collaborations, and it is important for E4S products to be installable via the Spack package manager. The software should be appropriately versioned and these versions should describe the code at a fixed point in time precisely, i.e., they should not track branches of the code in a source code repository. It should be possible to run the tests provided with the software on the target systems and it should be possible to monitor the tests for failure as other components change. All of these basic quality requirements make it easy for a new user, e.g., from industry, to pick up DOE products and begin using them rapidly. Without such standards, a collaborative effort may be stifled early on as team members struggle to learn each others' conventions and to adapt software products to new environments.

Collaboration with industry partners is key to improving software quality. The collaboration on Spack with Kitware (discussed in Section 3.2.1) was essential for standing up pull request (PR) testing on the Spack repository. PR build testing - where E4S products are built whenever changes are made to Spack's installation recipes - serves as an essential element of the software quality assurance process. An even higher level of quality assurance will be provided when existing capacity is extended to support post-build functional testing on target GPU platforms.

Collaboration between Kitware, Spack developers, and the E4S team at the University of Oregon allowed Continuous Integration (CI) support to be directly integrated into Spack (via the 'spack ci' command). Native support for CI in Spack opened the door for software and facility teams to more easily stand up custom CI processes, using GitLab. This allowed E4S to target specific systems and products of interest via Spack environments.

Building in elements of reliability, reproducibility of results, continuous integration, verification and validation are key to improving software quality, and documentation are key to fruitful collaboration with stakeholders. Finally, as discussed in Section 4.3, establishing third-party commercial support for E4S enables shared support costs among all E4S users and provides industry partners the kind of software support they need in order to adopt externally-developed software products.

### 6.4 LEADERSHIP SCIENTIFIC SOFTWARE (LSSW) TOWN HALL SERIES

DOE software teams contribute to many community organization such as the MPI forum, the C++ standards committee, the OpenMP standard, and the LLVM ecosystem. Even so, we believe that engaging additional entities can provide more value.

One effort organized by ECP Software Technology leads is the Leadership Scientific Software (LSSw) Town Hall series [33]. LSSw sponsors a series of community-building conversations centered around themed panel discussions intended to explore and build common understandings of current and future activities in the leadership scientific computing communities.

One desired outcome from LSSw discussions is an understanding of the kinds of organizations needed to address the sustainability of our scientific software products. A particular overarching theme of the conversations is to explore and expand the definition of leadership scientific software to include other scientific software development and user communities that have not been as visible in the high-performance computing (HPC) community before. Examples include scientific instrument software, edge computing environments, and users of previous-generation HPC platforms.

We believe that these inclusive conversations will serve to increase cross-community awareness and provide incentives to collaborate and form the new organizations needed for assuring the sustainability of our scientific software.

# 7. THE SCOPE OF SOFTWARE STEWARDSHIP

## 7.1 SOFTWARE STEWARDSHIP: KEY POINTS

1. Training in software best practices, use of new tools and products, and related content as part of the IDEAS-sponsored webinar series on *Best Practices for HPC Software Developers* [34] and the panel series on *Strategies for Working Remotely* [35] have regularly attracted hundreds of attendees who participate regularly, demonstrating the sustained value of these events.

2. The cross-cutting collaboration of planning and hosting training events has contributed to the sense of complex-wide community and a deeper awareness and appreciation for how software, applications, and facilities staff can collaborate to provide high-value training content for high-performance computational science.

3. Community-building activities, such as the BSSw Fellowship Program (https://bssw.io/fellowship), Sustainable Research Pathways for HPC (SRP-HPC) Program of the ECP Broader Engagement Initiative (https://exascaleproject.org/hpc-workforce), and the Collegeville Workshops on Scientific Software (https://collegeville.github.io/CW21) are essential for recruiting and training new and current workforce members.

4. The E4S support team provides essential value to software teams preparing their software to run on new platforms, in combination with access to the Frank multi-node, multi-device platform.

5. The E4S DocPortal (https://e4s-project.github.io/DocPortal.html) was developed as a result of briefing facilities and requesting from them their unmet requirements. The DocPortal provides a single searchable and sortable source for E4S product information, raked daily from the product teams' repositories.

6. The ECP ST Capability Assessment Report (CAR) [7] provides value as a driver for identifying gaps, communicating progress and informing key stakeholders about our progress and plans as input to their own activities.

## 7.2 HPC SOFTWARE TRAINING AND OUTREACH

The ECP Training thrust partners with the ECP community to provide a wide variety of high-impact training and outreach on high-performance scientific software, with much content developed and delivered by ECP ST teams (https://www.exascaleproject.org/training-events). In addition, the IDEAS Productivity Project (https://ideas-productivity.org) [36] collaborates with the ECP community to overcome challenges in software development for emerging exascale architectures and to ensure that investment in the exascale software ecosystem is as productive and sustainable as possible. Work spans four synergistic areas: (1) curating methodologies to improve software practices of individuals and teams, (2) incrementally and iteratively upgrading software practices, (3) establishing software communities, and (4) engaging in community outreach.

Impactful initiatives include the IDEAS-sponsored webinar series on *Best Practices for HPC Software Developers* [34], the panel series on *Strategies for Working Remotely* [35], tutorials on various aspects of software engineering for science (https://github.com/bssw-tutorial), and the Better Scientific Software site (https://bssw.io)—a central hub for sharing information about practices, processes and tools to improve software productivity and sustainability. Moreover, the BSSw Fellowship Program (https://bssw.io/fellowship), the Collegeville Workshops on Scientific Software (https://collegeville.github.io/CW21), and other IDEAS-organized events provide training and build community around topics related to high-quality scientific software. For example, the HDF Group recently partnered with members of the IDEAS project to use Productivity and Sustainability Improvement Planning (PSIP) to improve practices in documentation,

revision control, and coding standards for the development of HDF5 [37], thereby helping advance HDF5's functionality in the ECP software ecosystem.

### 7.3 FOSTERING HPC SOFTWARE COMMUNITY GROWTH

Numerous studies have shown that diverse organizations, teams, and communities perform more creatively and effectively—and thus are more productive. While some efforts are already under way to broaden participation in computational science and engineering, our communities could benefit by increasing emphasis on sustainable strategies to advance diversity and inclusion. The ECP broader engagement initiative [32] is beginning to address these issues. Work is needed to understand how to improve teaming skills and culture, including leadership of remote, distributed, and hybrid teams, while fully leveraging tools to facilitate distributed work. Equally important is research on building a growth culture in scientific communities, with recognition of the benefits of rich engagement across diverse demographics and areas of expertise.

An additional opportunity for expanding the HPC software community is to broaden the scope of people who can effectively use sophisticated HPC libraries and tools. This requires the development of higher levels of abstraction and interfaces for non-expert users, while concurrently enabling specialists to customize lower-level choices. Such advances would offer enormous potential to expand the use of HPC in industry and decision making, enabling effective use of HPC software technologies, even for people whose primary focus is domain-specific science and engineering, rather than computer architectures, programming models, and lower-level algorithms and data structures.

### 7.4 LEADERSHIP SOFTWARE TESTING AND USER SUPPORT

The E4S team provides support for packaging, testing, and deploying ECP ST and related products. The team is uniquely positioned to bridge gaps between ECP Software Technology (ST), Hardware Integration (HI), and Facility teams. The E4S team helps to identify particular software developers who help resolve any issues that are raised on the E4S issue tracker hosted on GitHub. The E4S team has experience with HPC environments and Spack and thus can curate the issues with the developers and help resolve issues expeditiously. The E4S DocPortal is an online portal for all E4S products with high-level information (e.g., License, Changelog, README) raked nightly from the GitHub product repositories. The E4S DocPortal makes it easy to find the primary source documentation for all E4S products and raise issues through other portals. These portals include:

1. Spack GitHub repository: Tracks issues with Spack packaging, recipes, and related documentation.
2. E4S project GitHub repository: Tracks issues with the E4S website and related documentation, including the E4S DocPortal.
3. E4S Release GitHub Repository: Tracks issues with E4S Spack environments and related documentation.
4. ST product source code GitHub repositories: Track issues related to their respective products.
5. Direct communication with facility teams (OLCF, NERSC, ALCF, Frank)

In addition, E4S support is available in the form of bi-weekly office hours held on Zoom and through outreach activities such as the E4S Hackathon and workshops where developers have direct access to the relevant GPU platforms on Frank and have E4S team members available to answer questions. Additional industry support for E4S deployment is being explored.

### 7.5 GATHERING CLIENT REQUIREMENTS AND COMMUNICATING PROGRESS

Development of new reusable scientific software capabilities requires regular assessment of progress in current efforts and monitoring of new scientific software needs and opportunities. ECP drives this process through the periodic creation and publishing of the ECP Software Technology Capability Assessment Report (CAR) [7]. The CAR is a comprehensive versioned document (Version 3.0 is in draft form at this time) describing the current ECP activities on reusable software, an analysis of identified gaps since the previous CAR version, and intended activities in the coming months.

The CAR is a fundamental planning and communication document that provides summary information for every product in the E4S portfolio, a capability and gap assessment for each technical domain, and an overarching description of the portfolio as a whole. The document is used by our DOE stakeholders, our industry partners, and the broader scientific community as input into their own planning activities. Requirements for future software capabilities are obtained primarily by:

- Briefing our stakeholders on our efforts and plans and then identifying potential gaps that they foresee. For example, in previous briefings with DOE Facilities staff, we identified the need for a single location where all E4S product documentation could be discovered. We addressed that requirement by creating the E4S DocPortal [38].
- Tracking new science and technology requirements and opportunities. For example, the widespread availability of hardware-supported low-precision arithmetic represents an opportunity to provide performance improvements for numerical software. A natural area to focus investment is math libraries. As part of the xSDK (math libraries SDK) project, we have provided funding for collaborative design space exploration around the effective use of mixed and multi-precision arithmetic in ECP math libraries. The effort involves a dozen or more math library teams who, because they are working together in a discovery phase, can rapidly determine and share ideas that could be applied across multiple libraries.

# 8. STEWARDSHIP MANAGEMENT AND OVERSIGHT

## 8.1 STEWARDSHIP MANAGEMENT AND OVERSIGHT: KEY POINTS

1. ECP ST uses a hierarchical and cyclical planning and change management process that enables multi-year planning with refinement and correction. The key motto is, "Always do the most important work. If it's not in the plan, fix the plan."
2. E4S as a portfolio and the people and processes around it provide many opportunities to improve scientific progress and enhance collaboration within DOE, with other US agencies, industry, universities, and the international community.
3. The notional Leadership Software Center (LSC) builds from the ECP ST leadership teams experience. The goal of LSC discussions is to establish a credible framework that could be the starting point for a sustainable software organization as the DOE community prepares for the completion of ECP.

## 8.2 THE ECP ST SOFTWARE MANAGEMENT STRATEGY

ECP Software Technology (ST) uses a three-phase hierarchical planning process with change control (Figure 14). Phase one is the initial whole-project plan, the equivalent of saying we will reach the moon by the end of the decade. While this plan is not specific, it has both the direction and destination in mind. Phase two planning is an annual refinement of activities prior to the start of each year. We break the progress-to-the-moon plan of that year into specific goals that are actionable and measurable. The third phase of planning provides the full level of planning detail for each of the activities for a given year about eight weeks before starting the activity, refining the description and adding completion criteria, an execution plan and a list of who will do the work.

In addition to a forward planning process, ECP also has a change control process to assist in adapting to unforeseen requirement changes and changes in priorities, estimation errors, and unforeseen staffing changes. The change request process enables us to adapt our activities in a way that sheds light on the reasons plans change. It also allows the leadership team to participate in addressing challenges, often leading to solutions that are more holistic, relative to the rest of the project and more effective and efficient. Finally, the change control process naturally leads to communication across the project, and with project stakeholders, leading to transparency of project challenges and opportunities. The overarching message to ECP project teams is that we want them to be doing the most important work at any given time. If the project plan does not reflect what is most important, we use the change control process to modify the plan.

The ECP planning and change control processes are supported by a collection of project management tools. Because we are a federally-funded 413.3b construction project [39], we are required to use the Primavera

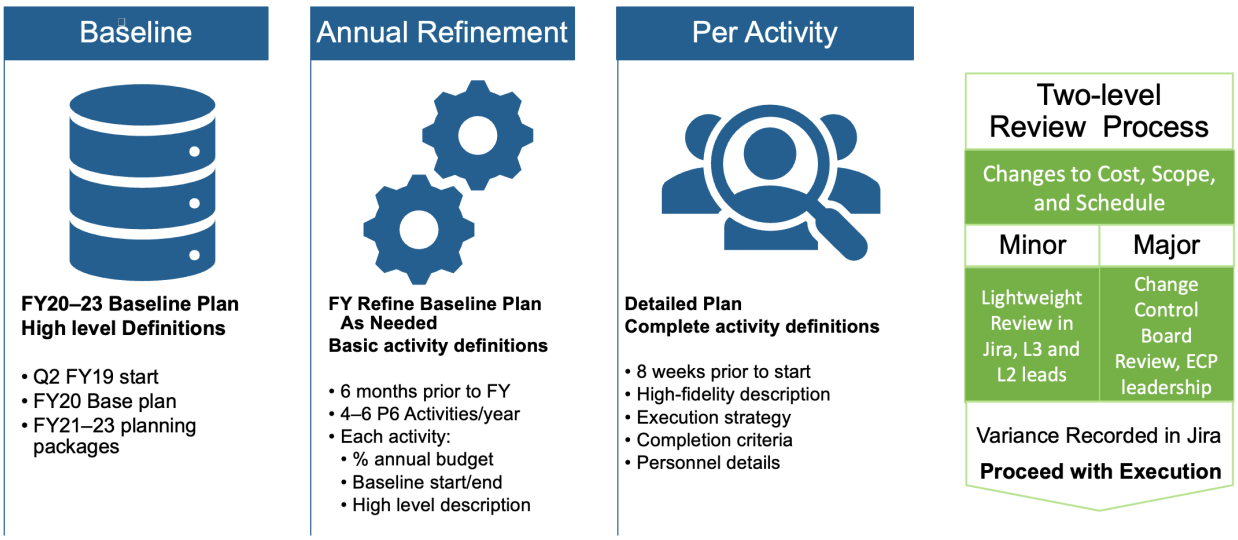



**Figure 14:** ECP uses a three-phase cyclical planning approach enabling an upfront description of the direction and final required deliverables and a refinement of plans as the project proceeds. The change management management process supports review of major and minor changes as appropriate and fosters transparency and risk management.

toolset. However, Primavera tools require a sophisticated project management background typically not available to a scientific software development team. Therefore, we also use the Jira and Confluence tools from Atlassian and synchronize data with Primavera.

Jira and Confluence are powerful yet usable project management tools for most scientific software development teams, given a modest amount of training. Jira also supports collaborative tracking of artifacts, timelines, user comments, and process control. Jira and Confluence are excellent tools for the kind of planning, executing, tracking and assessment work that ECP does. In addition, these tools support specific issue types that enable process customizations and the retention of content we must preserve for assessments and audits. We have defined a custom Jira issuetype called *P6 Activity*, named after the equivalent issue type used in the formal Primavera process. P6 Acitivity issues are what we use for the planning activities outlined in Figure 14.

ECP's use of Atlassian tools does not mean individual software product teams must also use Atlassian tools for their detailed project management activities. In fact, ECP expects each product development team to have its own planning and execution tools and processes. A team could use Atlassian products, but many teams use more commonly available open-source platforms such as GitHub and its associated GitHub Issues. GitHub Issues provide a lighter weight, broadly understood collection of tools and workflows that enable many people to contribute and benefit from a shared software development and use environment.

Our use of Atlassian tools is meant to capture coarse grain plans and deliverables that are consistent with the individual product team plans, representing the specific cost, scope and schedule that ECP sponsors. Many software teams have more than one funding source, so their use of GitHub issues or a similar integrated issue tracking platform enables them to conduct detailed planning and execution activities that were started before the beginning of ECP and will continue past the end of ECP.

### 8.3 SOFTWARE ECOSYSTEM STAKEHOLDERS

The Extreme-scale Scientific Software Stack, E4S (https://e4s.io), is a curated collection of reusable scientific software products targeting leadership computing platforms. E4S products are available as build-from-source, pre-installed at a growing number of computing facilities, as containers such as Docker and Singularity, and on hosted cloud platforms such as Google Cloud and Amazon AWS.

In addition to the software itself, E4S provides a single point of access to product documentation via its DocPortal. E4S also establishes quality expectations on member products via its community policies. These policies define expectations for product teams use of good practices, response to user requests, documentation,

and testing.

E4S targets leadership computing environments, but because of how scientific software is developed within the community, E4S software products also work on laptops, commodity clusters and cloud environments, and most other commonly available computer system used for scientific computation.

Finally, E4S is a community of scientific software developers, users and integrators who strive to impact science through efforts to improve the effectiveness and efficiency of the software we provide.

In this section we describe the potential impact of E4S from the perspective of its major stakeholders and collaborators. In each case, we describe the possible impact E4S can have as DOE and ASCR shepherd the growth and sustainability of this new software ecosystem in support of DOE's mission.

#### *8.3.1 Science Impact*

E4S efforts align very well with some of the most important challenges and opportunities that the scientific community faces. The importance of software continues to increase in almost all areas of science. The role of leadership computing plays a particularly important role in US competitiveness, science breakthroughs and national security. Since its inception in October 2018, E4S has shown the promise of what a community effort to provide a high-quality, curated scientific software stack can mean to DOE's science mission.

Going forward, E4S will provide the latest leadership computing software to scientific application teams with higher quality, as quickly as possible, in a portable software stack that teams can count on having available on any platform they use. They will not need to install it or tune it. It will be as though it were installed by the vendor.

As new algorithmic approaches emerge to address scientific problems (for example machine learning for science), E4S will provide support for algorithm developers who are creating new libraries for use in scientific applications. Because we foresee that the scientific use of machine learning and similar new computational tools will be combined with other established scientific computing techniques, the availability of new libraries and tools in E4S will allow application developers to take advantage of the full range of emerging and established software capabilities.

E4S will enable application development teams to rely on a much larger set of reusable software tools than ever before, allowing them to focus more of their time on new methods, algorithms, and science results. The overall impact on science will be that new computational results will be generated more quickly, be more trustworthy, and allow application teams to focus more on their science and less on their software.

#### *8.3.2 DOE and ASCR Impact*

The emergence of E4S has already elevated the awareness in the broader HPC community to what DOE/ASCR provides for software capabilities. E4S builds on ASCR's long history of contributing to the HPC software stack. Going forward, as E4S continues to expand its functionality, ease of use and installation, and availability across many platforms, we believe that DOE/ASCR will increasingly be seen as a national resource for future scientific software, especially for leadership platforms and emerging scientific applications.

The impact of E4S on ASCR will include more rapid, robust, and sustainable delivery of ASCR-funded research to Facilities, vendors, the open-source software community, and ecosystems such as LLVM, increasing ASCR's already-strong reputation for delivering software. In fact, ASCR could become known for its specific impact on scientific computing research by its ability to develop and deliver leadership computing software for the Department of Energy, other federal agencies, industry, and international partners.

**New Software Development Kits: AI, Edge, Quantum** E4S contains another layer of aggregation that combines products with similar scope and similar domain expertise. We refer to these aggregations as software development kits, or SDKs (see Figure 2). Presently, we have SDKs for Math libraries, IO libraries, visualization tools, performance tools, and performance portability layers. We anticipate, as DOE invests in other algorithmic and knowledge domains, that we will add new SDKs to E4S. Some anticipated SDKs are for data science (in particular, artificial intelligence), edge computing, and quantum computing.

Presently in AI, the scientific community uses a variety of community and industry provided software capabilities. For example, PyTorch and TensorFlow are widely used in scientific computing. Other tools such as Horovod provide a distributed computing layer that makes possible the solution of larger problem.

E4S already contains these three software products as part of its distribution. At the same time, science applications not only create and use inference engine that are central to deep neural networks, but also need to augment algorithmic and software efforts to include our knowledge of physics-based approaches that can anchor machine learning techniques to better assure correct results. We will also want to include algorithmic enhancements for explainability and interpretability.

We anticipate that the additional requirements for scientific use of AI will lead to new DOE algorithmic efforts and software products, and that this work will be done across multiple teams and institutions. Establishing an AI SDK will promote interaction across teams, the sharing of best practices, and healthy cooperative competition among peers that benefits the development teams themselves, their users, and the stakeholders who sponsor this work. We have seen all these positive outcomes from the SDK efforts that already exist. We expect that E4S will foster the establishment and growth of new SDKs, leveraging the knowledge and infrastructure that we have in place for existing SDKs. The availability of a broad collection of SDKs from within the E4S software ecosystem will be very important. We anticipate that most scientific applications will make use of more than one SDK as a part of their comprehensive application portfolio. Putting these SDKs under the E4S umbrella makes scientific sense, in addition to the expected benefits of cross-community collaboration and sharing of best practices.

#### *8.3.3 User Impact*

Users of scientific software will benefit substantially from E4S. As E4S matures and applications rely on it for many of their third-party dependencies, we expect a variety of positive outcomes.

**Laptop to Leadership Support** While E4S targets emerging leadership computing platforms, much of the product development is done on laptop and desktop computers using commodity processors and software development tools. Furthermore, basic functionality testing also occurs on these platforms or on commodity compute clusters for moderate scalability and correctness testing. As a result, users of E4S have access to the same software environment on their laptop all the way up to the leadership platforms. The common availability of E4S across all these platforms will improve productivity for all users.

**Composition and Advanced Workflows** While some applications have continued need for higher fidelity simulation and can use ever larger computers by simply refining the scale at which their current computations are done, many users are composing scales, physics, and data driven approaches into a single application, providing the next level of tightly integrated and comprehensive insight for scientific discovery. Above and beyond that, advanced workflows integrate leadership computational capabilities into an even larger and more comprehensive workflow where execution on a leadership system is one node of many from the beginning to the end of the work to be done. E4S supports the compositional enhancements for applications by providing a large suite of tools and libraries that can be integrated into one composed application and provides a workflow SDK that supports emerging simulation and data science.

**Integration into Vendor Environments** Another important role for E4S is the consolidation of key functionality into libraries and tools that get used by many applications. This phenomenon has existed for many years. For example, the basic linear algebra subprograms or BLAS have been used by many applications and computer system vendors can justify the effort to optimize these libraries. E4S will increase the number of these libraries and tools that can be used by applications, further providing incentives for system vendors to optimize implementations for their given platform. The existence of these libraries and tools in E4S will make it easier for vendors to identify where to focus their efforts and will increase the impact of their investments as applications continue to adopt E4S.

#### *8.3.4 Facilities Impact*

ECP has enabled comprehensive integration of activities across application, software technology, Facilities and vendor organizations. The availability of E4S products as a portfolio has been particularly impactful. Rather than each product team working one-on-one with Facilities staff, E4S staff can coordinate planning and activities across teams, and build, test and integrate all E4S products. The combined effort amortizes the

costs and enables sharing of information across E4S member teams as we learn how to develop, optimize and provide our software on early access systems on our way to providing E4S for the coming Exascale platforms.

Going forward, as more special-purpose hardware becomes part of our computing ecosystem, nodes that contain neuromorphic, FPGA, quantum, or other future devices, along with specialized software interfaces that drive these devices, can become part of E4S as we evolve our ability to manage multiple discrete devices within our existing applications. There is no faster, easier, or robust way to develop a new DOE software capability than to leverage the growing expertise and software delivery and deployment skills that are part of the E4S effort.

In collaboration with Facilities staff, with whom we have growing relationships, the E4S team will be able to assist any DOE-funded software team that wants to create and provide their capabilities to DOE application teams on future leadership platforms.

#### *8.3.5 Developer Impact*

For scientific software developers who contribute to the leadership software community, E4S will provide resources for improving software quality, through its R&D efforts in software best practices. It will provide the ability for users to quickly learn about, try out and rely upon new capabilities that developers provide. E4S will also provide critical testing capabilities that would otherwise be difficult for developers to access. Specialized testing platforms, new software environments, early access systems for new leadership systems, and more will allow developers to debug their newly developed software much sooner than ever before. The availability of Frank, the multinode, multidevice software testing and packaging platform, along with support from system experts, will enable software teams to port their software more rapidly to new hardware and software environments.

Once a piece of software is ready, users will be able to find it by searching the DocPortal, which provides a summary of the product on the E4S website and a direct link to the full product documentation.

#### *8.3.6 Research Impact*

While E4S will have the largest direct impact on DOE developers, it will also have an impact on DOE research efforts. As developers improve their software processes, practices, and skills, the DOE research and development community will be able to identify practices, processes and, skills that can be translated to earlier stages of the R&D pipeline so that even in the earliest stages of algorithm exploration, design, and implementation some of the fundamental good practices for any kind of scientific software can be instilled into DOE research scientists as they conduct all their R&D efforts. This transfer of knowledge to the early phases of the R&D pipeline will further improve and accelerate the quality and dissemination of DOE R&D.

#### *8.3.7 Software Quality Impact*

The scientific software development community has had mixed success in adapting and adopting practices, processes, and tools from other software domains. Most software literature and studies do not consider the specific workflows, team composition and purpose of scientific software product development. As a result, many of the common and well-known approaches to software development require significant adaptations to be useful in a scientific software development environment. In addition, because scientific software needs are not specifically considered, gaps in the literature leave us with significant uncertainty about how to develop effective and efficient software for science.

One significant element of our E4S efforts is to establish a team with expertise in studying how research software can be best developed and used. This area of research, called Research Software Science (RSS) [40] involves applying the scientific method to understand and improve how we develop and use software to do research. An RSS effort does require deep understanding of the software and algorithmic domains of research software. In addition, we also need people with skills in organizational psychology, and the social and cognitive sciences. See Section 5.4 for details.

### *8.3.8 Outreach Impact*

DOE has a long history of impact beyond the specific needs of its own community. E4S will extend this impact in multiple ways.

**US Agency impact** DOE-funded scientific software efforts have impacted US agencies for many years. DOE-funded capabilities in programming models, runtimes, performance tools, compilers, math libraries, I/O and visualization are widely used by scientists at agencies such as NASA, DoD, NSF, NOAA, and NIH.

This legacy will continue, especially as these agencies put their focus on the use of accelerator (GPU) and other compute devices. Specifically, the emphasis of the Exascale Computing Project on providing software capabilities for GPUs from multiple vendors, will provide an important starting point for applications from these other agencies that need to migrate to GPUs. Very few agencies outside of DOE have the resources to rewrite their own software stack without leveraging investments made by ECP and DOE.

Furthermore, an overall trend in high performance computing is the requirement to use more advanced software stacks. Computer architectures have become far more complicated than in the past, making it difficult to port an application to a new architecture without performance portability support. In addition, scientific applications are becoming more integrated by tightly coupling multiple scales and physics and incorporating data-driven approaches such as machine learning via deep neural networks.

These two broad trends require leveraging software ecosystems that includes many third software products. High-quality software ecosystems such as E4S become essential for our ability to use computation for scientific advances.

**Industry impact** Except for widely used tools and libraries such as the message passing interface, DOE reusable software products have not had broad adoption by industry application teams because the barrier to use was too high. The cost of installation, experimenting with new capabilities, and adopting new products was costly and risky.

E4S has significantly reduced and even eliminated some of the barriers that make it difficult for industrial applications to use DOE software. Specifically, the E4S portfolio provides a curated, documented, ready-to-use stack that is available not only from source code, but also in containers such as Docker, and Singularity, and in cloud environments such as Amazon AWS and Google Cloud.

Because users who want to evaluate E4S can access it in several pre-built environments, they only need to worry about creating the interfaces to the libraries and tools one time and use them across many different platforms. Even so, there are still significant impediments to the optimal use of E4S in industrial environments. While DOE software provides very powerful capabilities, we typically do not have enough documentation that signals to industrial users which software product or combination of products can be used to solve a particular industrial problem. We provide good product-by-product documentation but not guidance on what collection of DOE software product would be best suited to addressing that application need. More work needs to be done.

**International collaborations** E4S and its related software development kits (SDKs) provide open architectures. Full membership in an SDK or E4S is accomplished by satisfying the requirements of the posted community policies. None of these policies dictate funding source or institutional affiliation but are directly related to the quality of the software product itself and its availability to and support of the scientific community.

Already, E4S contains many notable products that are not funded by DOE. Some products come from industry, or from open-source communities that have diverse funding sources. Similarly, our existing SDKs have numerous products that have diverse funding from organizations such as NSF, the German Aerospace Research Center (DLR), and other scientific communities across a variety of agencies and international organizations.

We anticipate that this form of contribution from our international partners will continue to be attractive going forward. The existing functionality and set of policies for E4S and the SDKs are driven by pragmatic concerns that are relevant to all scientists who want to provide and use software in the high-performance computing community. Therefore, E4S efforts will continue to make sure our policies, processes, tools,

and interactions are attractive and compatible with the goals and efforts of the broader international HPC community.

## 8.4 LOOKING FORWARD: TOWARD A SUSTAINABLE SCIENTIFIC SOFTWARE ECOSYSTEM

U.S. Department of Energy laboratories have reliably provided reusable software components to application scientists and computing facilities for many years, enabling more rapid, robust and impactful scientific results for these stakeholders. The Exascale Computing Project continues this effort and also offers these capabilities as a curated portfolio (E4S). This portfolio is needed by ECP in order to successfully complete its project goals but also has emerged as a new first class entity in the scientific computing ecosystem.

ECP's management of these capabilities as a portfolio enables new peer conversations between software development teams and their stakeholders. The curated portfolio enables coordinated integration of software into the computing facilities, into applications, in coordination with commercial vendors and the open source software community.

The ECP's curated portfolio approach has been in existence for about three years but we are already seeing its impact. When combined with advances in software tools and processes, we are observing application teams relying upon this curated stack as a portable, easily installed software ecosystem they can trust for performance, portability, and new emerging functionality. In addition, the overall complexity of using products from the curated stack has diminished greatly, when compared to independent management of individual products.

By the end of ECP, the software stack will include scalable solvers, IO, visualization, tools, and portability layers that enable the use of high performance GPU devices from all major computing vendors. In combination with the applications built on top of this stack, the scientific community will see tremendous advances in high fidelity modeling and simulation, data science, and more.

At the same time, the long term viability and usability of the ECP software stack relies upon its sustainability, continued growth, and continued integration into application codes, computing facilities software environments, vendor software stacks, and the broader open source ecosystems. This software sustainability plan lays out the mission, vision, values, goals, strategy and implementation framework for assuring the sustainable growth and integration of DOE reusable software into the future.

### *8.4.1 A Sketch of a Future Leadership Software Center*

To better ensure continued growth and sustainability beyond ECP, we are exploring ideas now to better orient E4S efforts toward the post-ECP era. Here we provide a sketch of a DOE ASCR Leadership[5] Software Center (LSC).

**Mission** Deliver a robust, reliable, high-quality and sustainable software stack that enables the rapid development of DOE scientific applications for the pursuit of scientific discovery in leading edge environments. Our mission is to deliver a curated collection of high-quality, reliable, performance, and reusable scientific software products addressing the needs of DOE application developers, helping them to achieve their scientific goals. Our efforts will complement and integrate with capabilities developed in the broader software ecosystem. We will develop capabilities that are both essential for DOE's scientific mission and not already part of the scientific software ecosystem. To assure sustainability, we will deliver our capabilities to their long-term destination in vendor stacks, community ecosystems, facilities environment or our own directly-managed stack.

**Vision** Build the best leadership computing scientific software stack in the world to enable innovation in computational and data science solutions to global challenges and breakthrough problems. A sustainable, high-quality software ecosystem that is continuously improved by a robust research and development effort, is deployed on advanced computing platforms, and is broadly adopted by application teams and software

[5]We intend leadership in our setting to mean emerging and leading-edge software for emerging and leading-edge scientific computing environments, including HPC, AI/ML for science, large-scale edge computing for science, quantum, and other scientific computing software products that complement industry efforts and facilitate scientific progress.

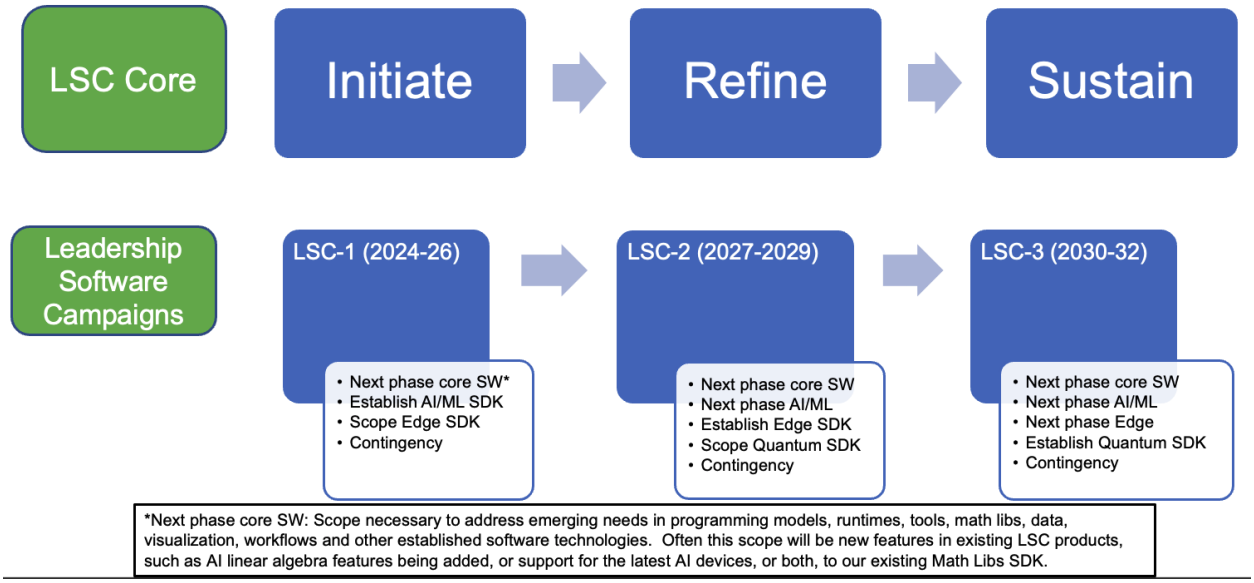



**Figure 15:** The notional organization of the LSC is inspired from our experiences in ECP.

developers to accelerate their science. We will create, deliver and deploy a software stack that provides capabilities needed for the leading edge of scientific computing. Our software efforts will be known for performance, portable execution and high quality. The capabilities we provide will be unique in how they are among the first to address emerging architectures; how they anticipate and address the needs of new scientific software requirements; and how they accelerate the advance of scientific discovery by providing a curated stack of portable high performance software libraries to users, vendors and the broader scientific community. We will work in collaboration with facilities, vendors, community ecosystems and the broader HPC community to assure that our products are sustainable. Each product will have a home in our curated stack, or vendor and community stacks as appropriate.

**Values** We value software that addresses leading edge and emerging application needs, on leading edge and emerging computing platforms, and unambiguously adds value to the software ecosystem because of the usefulness, quality and sustainability of our work. We value working software that addresses user requirements; has a clear path from development to sustainable delivery; addresses leading edge and emerging application needs on leading edge and emerging computing platforms, and unambiguously adds value vs independently developed software capabilities. We value delivery of capabilities as early as possible through frequent releases, and use of agile software development methodologies that enable response to changing and emerging requirements. We value continual improvement of our software development methodologies and tools, adapted and adopted from the broader software community and effectively applied to the scientific computing domain. We value collaboration with the broader software community, assuring that our efforts deliver unique and complementary capabilities unavailable from other sources, directly focused on leading edge and emerging needs and opportunities.

**Goals** LSC goals include:

1. **ECP Sustainability:** The Leadership Software Center (LSC) will enable the sustainability of ECP contributions, and development and delivery of future capabilities, including new domains like AI/ML, Edge and Quantum.
2. **Tailored Agile:** The LSC will use tailored project management practices, processes, tools, and a distributed multi-institutional organization to enable effective and efficient delivery of ASCR software investments.

3. **New Ecosystem Entity:** The LSC will establish an essential and new ecosystem entity to complement Facilities, ASCR Research, vendors, industry and other entities.

4. **Workforce Development:** Establishing the LSC assures the creation of a scientific software workforce for sustainable leadership scientific software development and delivery.

**Execution and Management** We anticipate that the LSC can bring forward the planning, executing, tracking and assessment processes and tools described in Sections 8.2 and 9.2 that we developed to manage the E4S portfolio. One modification required to address the need for both a stable funding and staffing core and the need to be mission driven for identifying, developing and delivering new capabilities. For the LSC, we propose exploring a two-tier funding model as shown in Figure 15. By incorporating fixed-time campaigns, with appropriate change control processes similar to what we have in ECP, we can address the long-term need for stability and career growth and the specific software requirements of a particular phase of scientific software requirements.

**Connections to the Broader Scientific Community** Efforts to establish a Leadership Scientific Software Center are consistent with broader scientific community efforts. For example, UNESCO recently released the report "UNESCO Recommendation on Open Science" [41], in which they point out the large and growing role that software has in scientific research. This acknowledgment comes from an organization that pays attention to large scope topics. Their recognition of software is a strong indicator of the importance of software now and going forward. The LSC activities are timely in this respect.

# 9. ASSESSMENT FOR SUCCESS

## 9.1 ASSESSMENT FOR SUCCESS: KEY POINTS

1. The ECP Key Performance Parameter KPP-3 provides an effective framework for defining and assessing success for a re-usable software product using capability integrations as the key metric.

2. The combination of P6 Activities (defined in Section 8.2) and capability integrations provides a beginning-to-end lifecycle model for the macro-engineering of reusable scientific software products 16.

## 9.2 THE CENTRAL ROLE OF CAPABILITY INTEGRATION

Measuring the impact of re-usable scientific libraries and tools has typically been very challenging. Any individual software product team has difficulty determining if and how their work is impactful. It is even more challenging for funding agencies to assess whether their software funds are having the desired impact on the research community.

Because ECP is a formal project, it must achieve threshold values for its key performance parameters (KPPs). In the ECP Software Technology focus area, we have defined a KPP (KPP-3, the third of four KPPs for the project) around the concept of capability integration. Because our work focuses on delivering reusable software libraries and tools, the fundamental value of our work must be the integration of our capabilities into client environments. Perhaps the most obvious integration is an application's use of a library, for example, the use of a linear solver library to solve a set of algebraic equations.

In ECP we have established a formal framework for defining and tracking capability integrations. A given software product contributes to the overall success of ECP by achieving integration of a substantial capability into a target client environment. While linking and using a library from an application code is one approach to capability integration, our definition is more varied. For tools, a successful integration occurs when a team uses that tool for its intended purpose with success. For example, a performance analysis tool such as TAU is intended to give an application team insight into the performance of their code on a target computing system. If an application developer uses TAU and gains insight for performance improvement, then TAU has successfully integrated its capabilities into the client environment. Table 2 lists the types of possible integrations.

ECP has developed a point system for capability integrations. A member product in the ECP software technology portfolio achieves a point when a capability that is worth approximately one staff member's effort

- **Tool:** Utility or library used in client workflow in preparation for or execution in exascale environment
  - Performance Tools (TAU, HPCToolkit, . . . )
  - Prototype libraries (pre-exascale math libraries – even if not ultimately used on exascale platforms)
  - Viz (Paraview, Cinema, . . . )
  - Ecosystem (Spack, DocPortal, . . . )
- **Library:** Software integrated into client environment and executed in exascale environment
  - Parallel Programming (MPICH, Kokkos, . . . )
  - Math (hypre, PETSc, . . . )
  - I/O (HDF5, ADIOS, . . . )
  - In Situ (ALPINE, SZ, . . . )
- **Upstream:** Tool or library integrated into a sustainable community software ecosystem
  - Ecosystems (LLVM)
  - Standards (OpenMP, MPI Forum, C++, . . . )
- **Prototype:** Working prototype products that enables commercial software vendors to improve their products
  - Vendor math libraries (BLAS, LAPACK)
  - Parallel programming (MPICH)

**Table 2:** These are the basic categories for what it means to integrate a capability. One point is achieved if a capability worth approximately one year of staff effort is successfully integrated into one client environment on one exascale platform. For most products, sustainability is achieved by being part of the E4S portfolio and satisfying the E4S community policies [9].

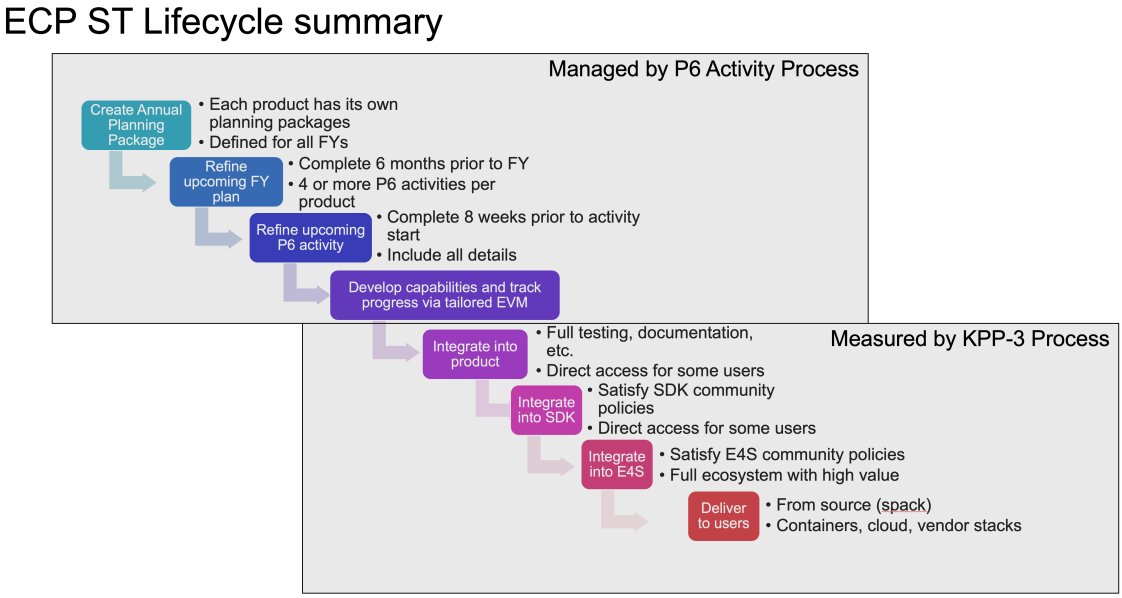



**Figure 16:** ECP ST product planning, execution, testing, and assessment are governed by the combination of P6 Activities for hierarchical planning (Figure 14) and KPP-3 for measuring capability integrations (Table 2). This figure shows how these two elements capture the entire lifecycle of ECP ST feature development.

for one year (one FTE) is successfully integrated into one client member environment and is successfully demonstrated on one exascale platform. An additional point can be achieved by integrating a new capability into the same client environment on the same platform or by integrating the same capability into a different client environment on the same platform or by integrating the same capability into the same client environment onto a different exascale platform. Overall product success is determined by whether the product achieve a passing score or higher. The value of the passing score is product dependent and predefined for each product. A typical passing score is between 4 and 8. These are modest passing scores but reflect the philosophy of KPPs to be a sign of modest success. Most ECP products will achieve many more successful capability integrations by the end of the project.

It is worth noting that the integration of a capability must be shown to be sustainable. In other words, if the demonstration of the integration used a version of the product that was not eventually integrated into the main branch of the software repository or did not have sufficient documentation, testing, or other important sustainability attributes, then the integration is itself not achieved. The emphasis on sustainable integration has had a profound effect on software quality in ECP. First, we have been able to identify products that have not achieved a sustainable integration and have then worked with the product teams to come up with a new plan that will succeed or, in rare situations, have determined that there is no sustainability path for a product, in which case the product efforts have been descoped and the funds returned to ECP.

### 9.3 THE E4S SOFTWARE LIFECYCLE MODEL

Combining the hierarchical and cyclical planning activities described in Section 8.2 and the capability integration scoring system described in Section 9.2, we obtain the E4S software lifecycle, as shown in Figure 16. From their inception as P6 Activity planning packages, which are refined annually and given detailed information before starting the activity, all the way to the successful integration of a capability into the client environment, E4S features are governed by this lifecycle. Each product team conducts its own integration planning that incorporates other funding sources and stakeholders, but the E4S lifecycle intersects the product lifecycle for capabilities funded by ECP.

### 9.4 ASSESSING USER IMPACT

The return on investment of the software included in the stewardship effort should include new science performed at scale. This can be measured in terms of papers and prize submissions, such as the Gordon Bell submissions, and potentially other means of tracking meaningful software contributions to science. The quality of the software should steadily improve as a result of the stewardship effort and can be quantified in the coverage of the tests, history of performance on benchmarks, documentation that is consistent with the code, and the turnaround time for fixing bugs. Community involvement in the software can be determined by contributions of new capabilities, performance improvements, and bug fixes through merge requests. The evolution of the stewardship activity can be measured by the number of new software products included and the number of older software products that have been retired.

## 10. ADDITIONAL TOPICS

### 10.1 ADDITIONAL TOPICS: KEY POINTS

1. Export control challenges appear to be missing from the RFI topics. We think they are important enough to be considered in scope for the RFI response.

2. Stable funding and careers for scientific software professionals is certainly an implicit topic in several RFI questions, but we think it can be made an explicit topic for discussion.

### 10.2 EXPORT CONTROL

Several software packages and libraries have components that are publicly available, while other components are subject to export controls. As part of the security posture, the stewardship effort needs to include best practices for limiting access to the export controlled components and being able to provide information to people who have permission to access such components.

### 10.3 STABLE FUNDING AND CAREERS FOR SCIENTIFIC SOFTWARE WORK

Developing high-performance scientific software requires the combined contributions of many people, whose skills span applied mathematics, computer science, core disciplines of science and engineering, software engineering, scientific programming, performance analysis, team collaboration, and more. Due to traditional metrics for career advancement focusing on publications rather than software contributions, people who focus on the important work of software development often face challenges in professional recognition and career advancement. Addressing these scientific software challenges requires broad community collaboration to change the culture of computational science, increasing the emphasis on high-quality software itself and the people who create it. Responding to these challenges, various grass-roots community groups have arisen in recent years to nurture "communities of practice" [42] in their respective spheres of influence, where like-minded people share information and experiences on effective approaches for creating, sustaining, and collaborating via scientific research software. These groups articulate key issues and needs to stakeholders, agencies, and the broader research community to effect changes in policies, funding, and reward structure, while advancing understanding of the importance of high-quality software in multidisciplinary CSE collaboration and the integrity of computational research [43].

For example, the Research Software Engineering (RSE, https://society-rse.org and https://us-rse.org) movement is helping to build community, mentoring, and professional recognition for scientific software specialists. Also, the Better Scientific Software Fellowship Program (https://bssw.io/fellowship), launched in 2018 as an initiative of the IDEAS productivity project [36], provides recognition and funding to leaders and advocates of high-quality scientific software.

## CONCLUSION

The leadership team of the Software Technology (ST) Focus Area of the U.S. Department of Energy Exascale Computing Project (ECP) is leading the development and delivery of a collection of essential software

capabilities necessary for successful results from exascale computing platforms, while also delivering a suite of products that can be sustained into the future. This response to the Request for Information (RFI) on the *Stewardship of Software for Scientific and High-Performance Computing* represents the experience of the authors as part of our work to design, develop, deliver, and deploy a reusable scientific software stack as part of the ECP project mission.

## ACKNOWLEDGMENTS

This research was supported by the Exascale Computing Project (ECP), Project Number: 17-SC-20-SC, a collaborative effort of two DOE organizations—the Office of Science and the National Nuclear Security Administration—responsible for the planning and preparation of a capable Exascale ecosystem—including software, applications, hardware, advanced system engineering, and early testbed platforms—to support the nation's Exascale computing imperative.

# REFERENCES


[1] D. Kothe, S. Lee, and I. Qualters. Exascale computing in the United States. *Computing in Science and Engineering*, 21(1):17 – 29, 2019. https://doi.org/10.1109/MCSE.2018.2875366.

[2] DOE Exascale Computing Project (ECP). https://www.exascaleproject.org.

[3] Francis Alexander, Ann Almgren, John Bell, Amitava Bhattacharjee, Jacqueline Chen, Phil Colella, David Daniel, Jack DeSlippe, Lori Diachin, Erik Draeger, Anshu Dubey, Thom Dunning, Thomas Evans, Ian Foster, Marianne Francois, Tim Germann, Mark Gordon, Salman Habib, Mahantesh Halappanavar, Steven Hamilton, William Hart, Zhenyu (Henry) Huang, Aimee Hungerford, Daniel Kasen, Paul R. C. Kent, Tzanio Kolev, Douglas B. Kothe, Andreas Kronfeld, Ye Luo, Paul Mackenzie, David McCallen, Bronson Messer, Sue Mniszewski, Chris Oehmen, Amedeo Perazzo, Danny Perez, David Richards, William J. Rider, Rob Rieben, Kenneth Roche, Andrew Siegel, Michael Sprague, Carl Steefel, Rick Stevens, Madhava Syamlal, Mark Taylor, John Turner, Jean-Luc Vay, Artur F. Voter, Theresa L. Windus, and Katherine Yelick. Exascale applications: skin in the game. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 378(2166):20190056, March 2020. https://doi.org/10.1098/rsta.2019.0056.

[4] J. Hack, K. Riley, R. Gerber, K. Antypas, R. Coffey, E. Dart, T. Straatsma, J. Wells, D. Bard, S. Dosanj, I. Monga, M. Papka, and L. Rotman. Crosscut Report: Exascale Requirements Reviews, 2017. https://www.osti.gov/servlets/purl/1417653.

[5] Scientific Grand Challenges Workshop Series. Office of Science, U.S. Department of Energy, see http://extremecomputing.labworks.org/workshops.stm.

[6] Robert Rosner (Chair). The Opportunities and Challenges of Exascale Computing. Office of Science, U.S. Department of Energy, 2010.

[7] Michael A. Heroux, Lois Curfman McInnes, Rajeev Thakur, Jeffrey S. Vetter, Xiaoye Sherry Li, James Ahrens, Todd Munson, Kathryn Mohror, and ECP Software Technology teams. ECP Software Technology Capability Assessment Report. November 2020. https://www.osti.gov/biblio/1760096.

[8] Todd Gamblin, Matthew P. LeGendre, Michael R. Collette, Gregory L. Lee, Adam Moody, Bronis R. de Supinski, and W. Scott Futral. The Spack Package Manager: Bringing order to HPC software chaos. In *Supercomputing 2015 (SC'15)*, Austin, Texas, November 15-20 2015. LLNL-CONF-669890.

[9] E4S Community Policies. https://e4s-project.github.io/policies.html.

[10] xSDK Web page. http://xsdk.info.

[11] xSDK Community Package Policies. https://github.com/xsdk-project/xsdk-community-policies.

[12] E4S Web page. http://e4s.io.

[13] Lois Curfman McInnes, Michael A. Heroux, Erik W. Draeger, Andrew Siegel, Susan Coghlan, and Katie Antypas. How community software ecosystems can unlock the potential of exascale computing. *Nature Computational Science*, 1:92–94, 2021. https://doi.org/10.1038/s43588-021-00033-y.

[14] Peter Naur and Brian Randell. Software Engineering: Report of a conference sponsored by the NATO Science Committee, Garmisch, Germany, 7th-11th October 1968. 1969.

[15] Hyunmin Seo, Caitlin Sadowski, Sebastian Elbaum, Edward Aftandilian, and Robert Bowdidge. Programmers' build errors: A case study (at Google). In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, pages 724–734, New York, NY, USA, 2014. Association for Computing Machinery.

[16] Elvan Kula, Ayushi Rastogi, Hennie Huijgens, Arie van Deursen, and Georgios Gousios. Releasing fast and slow: an exploratory case study at ING. In Marlon Dumas, Dietmar Pfahl, Sven Apel, and Alessandra Russo, editors, *Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2019, Tallinn, Estonia, August 26-30, 2019*, pages 785–795. ACM, 2019.

[17] WarpX application. https://www.exascaleproject.org/research-project/warpx.

[18] Ashwin Ramaswami. Securing Open Source Software at the Source. https://www.plaintextgroup.com/reports/securing-open-source-software-at-the-source.

[19] E4S Geb repository. https://github.com/E4S-Project/e4s.

[20] GitHub.com. Dependabot. https://github.com/dependabot.

[21] E4S Validation Test Suite. https://github.com/E4S-Project/testsuite.

[22] Frank: OACISS Systems Wiki. https://oaciss.uoregon.edu/frank.

[23] E4S Job Statistics. http://stats.e4s.io.

[24] ExaWind application. https://www.exascaleproject.org/research-project/exawind.

[25] Pantheon Science. The Pantheon Project: Reproducible Workflows for Extreme Scale Science. https://pantheonscience.org.

[26] E4S Spack Build Cache on AWS. http://cache.e4s.io.

[27] Shahzeb Siddiqui, Sameer Shende, Ryan Adamson, and Mike Heroux. E4S at DOE Facilities with Deep Dive at NERSC . https://www.exascaleproject.org/event/e4s_at_doe_100421/.

[28] Doug Kothe. "The Exascale Computing Project and the Future of HPC " with Doug Kothe - YouTube. https://www.youtube.com/watch?v=KRIn0w7CLL0.

[29] N. Eghbal. *Working in Public: The Making and Maintenance of Open Source Software*. Stripe Press, 2020.

[30] ECP Co-Design Centers. https://www.exascaleproject.org/research-group/co-design-centers.

[31] Sanjiv Khosla. Why I Killed The Coding Test. https://medium.com/motivation-u/why-i-killed-the-coding-test-ceedc45fdb62.

[32] Ashley Barker, Dan Martin, Mary Ann Leung, Lois Curfman McInnes, Jini Ramprakash, Julia White, Jim Ahrens, Erik Draeger, Shaun Fomby, Yasaman Ghadar, Rinku Gupta, Mahantesh Halappanavar, Mike Heroux, Christopher Kelly, Mark Miller, Hai Ah Nam, Slaven Peles, Damian Rouson, Dan Turner, Terry Turton, David Brown, and Valerie Taylor. A multipronged approach to building a diverse workforce and cultivating an inclusive professional environment for DOE high-performance computing. https://doi.org/10.6084/m9.figshare.17192492.

[33] Leadership Scientific Software (LSSw) Portal. https://lssw.io.

[34] Osni Marques and David E. Bernholdt. Best Practices for HPC Software Developers: The first five years of the webinar series. https://bssw.io/blog_posts/best-practices-for-hpc-software-developers-the-first-five-years-of-the-webinar-series.

[35] Elaine Raybourn, Reed Milewicz, David Rogers, Elsa Gonsiorowski, Ben Sims, and Greg Watson. Working remotely: The Exascale Computing Project (ECP) panel series. https://bssw.io/blog_posts/working-remotely-the-exascale-computing-project-ecp-panel-series.

[36] M. A. Heroux, L. C. McInnes, D. E. Bernholdt, A. Dubey, E. Gonsiorowski, O. Marques, J. D. Moulton, B. Norris, and E. M. Raybourn et al. Advancing Scientific Productivity through Better Scientific Software: Developer Productivity and Software Sustainability Report. ECP Technical Report ECP-U-RPT-2020-0001, 2020, http://doi.org/10.2172/1606662.

[37] Mark Miller, Elena Pourmal, and Elsa Gonsiorowski. Recent successes with PSIP on HDF5. https://bssw.io/blog_posts/recent-successes-with-psip-on-hdf5.

[38] E4S DocPortal. https://e4s-project.github.io/DocPortal.html.

[39] Paul Basco. DOE Order 413.3B: Program and Project Management (PM) for the Acquisition of Capital Assets, Significant Changes to the Order. https://www.energy.gov/sites/prod/files/maprod/documents/15-1025_Bosco.pdf.

[40] Michael Heroux. Research Software Science: A Scientific Approach to Understanding and Improving How We Develop and Use Software for Research. https://bssw.io/blog_posts/research-software-science-a-scientific-approach-to-understanding-and-improving-how-we-develop-and-use-software-for-research.

[41] UNESCO. UNESCO Recommendation on Open Science. https://unesdoc.unesco.org/ark:/48223/pf0000379949.locale=en.

[42] E. Wenger. *Communities of Practice: Learning, Meaning, and Identity*. 1998. ISBN:978-0-521-66363-2.

[43] D. S. Katz, L. C. McInnes, D. E. Bernholdt, A. C. Mayers, N. P. Chue Hong, J. Duckles, S. Gesing, M. A. Heroux, S. Hettrick, R. C. Jimenez, M. Pierce, B. Weaver, and N. Wilkins-Diehr. Community organizations: Changing the culture in which research software is developed and sustained. *IEEE Computing in Science and Engineering*, 21:8–24, 2019. special issue on Accelerating Scientific Discovery with Reusable Software, https://dx.doi.org/10.1109/MCSE.2018.2883051.