Snowmass 2021 Letter of Interest: Software Sustainability and HEP

Daniel S. Katz[1], Sudhir Malik[2], Mark S. Neubauer[1], and Graeme A. Stewart[3]

[1]University of Illinois at Urbana-Champaign
[2]University of Puerto Rico Mayaguez
[3]CERN

Thematic Areas:

■ Community Engagement Frontier
■ CommF1: Applications & Industry
■ CommF2: Career Pipeline & Development
■ CommF3: Diversity & Inclusion
■ CommF4: Physics Education
■ Computing Frontier
■ CompF7: Reinterpretation and long-term preservation of data and code

Contact Information:

Daniel S. Katz: *d.katz@ieee.org*

Abstract: Because new facilities of the 2020s, such as the High Luminosity Large Hadron Collider (HL-LHC), will be relevant through at least the 2030s, sustainability of both the facility software as well as the related data analysis software needs to be considered to enable their adaptability to new challenges, longevity, and efficiency, over at least this period, as well as being as easy as possible to develop and maintain, and that it is as reusable as possible. In return, HEP software developers will gain an important skill that is essential to careers in the realm of software, inside or outside HEP. Software sustainability practices could lead to new collaborations, including elements of HEP software being directly used outside the field, and as has happened more frequently in recent years, to HEP developers contributing to software developed outside the field rather than reinventing it.

New and being-developed facilities of the 2020s, such as the High Luminosity Large Hadron Collider (HL-LHC), will be relevant through at least the 2030s. This means that their software efforts and those that are used to analyze their data need to consider sustainability to enable their adaptability to new challenges, longevity, and efficiency, over at least this period. This will helps ensure that this software will be easier to develop and maintain, that it remains available in the future on new platforms, that it meets new needs, and that it is as reusable as possible. Software sustainability practices could lead to new collaborations, including elements of HEP software being directly used outside the field, and as has happened more frequently in recent years, to HEP developers contributing to software developed outside the field rather than reinventing it. A focus on and skills related to sustainable software will give HEP software developers an important skill that is essential to careers in the realm of software, inside or outside HEP.

To address this challenge, the authors of this LOI organized a virtual half-day workshop on "Software Sustainability and High Energy Physics" (https://indico.cern.ch/event/930127/), with about 80 attendees. This workshop had two complementary goals:

1. To bring together experts from HEP as well as those from outside to share their experiences and practices, and
2. To articulate a vision that helps the Institute for Research and Innovation in Software for High Energy Physics (IRIS-HEP) in creating a work plan to implement elements of software sustainability, as part of IRIS-HEP's Blueprint activity (https://iris-hep.org/blueprint.html)

Introduction to software sustainability

The reason software sustainability is important is that software stops working eventually if is not actively maintained.

Generally, the structure of computational science software stacks[1] is complex, ranging from project-specific software to discipline-specific tools and libraries to multidisciplinary scientific infrastructure to the general non-scientific infrastructure. Software builds and depends on software in all layers below it. Any change in an underlying layer may cause the software to collapse.

Given this, we can defined research software sustainability as the process of developing and maintaining software that continues to meet its purpose over time, which includes that the software adds new capabilities as needed by its users, responds to bugs and other problems that are discovered, and is ported to work with new versions of the underlying layers, including software as well as new hardware.

In order to sustain research software, we can

- do things that reduce the amount of work needed,
- do things that increase the available resources, or
- do things that both reduce the amount of work needed and increase the available resources

To reduce the amount of work needed, we can train its developers, which involves finding or developing training material. We can also use best practices, which involves finding or developing best practices.

There are a number of potential things we can do to increase the available resources. We can create incentives so that the reward people who contribute to the software. One specific example is to make the software citable, to make contributors authors, and to encourage users to cite the software, so that for developers in research institutions, they are rewarded through citations, which usually match the existing metrics on which they are hired and promoted. We can also attempt to change career paths and associated metrics, by either adjusting existing career paths so that they reward software work, perhaps by adding new metrics, and by developing new career paths that focus on and reward software work, such as for Research Software Engineers. Though it is a long-term activity, we can try to increase the funding available for software work by first making the role of software in research clear to research funders, and then by clearly making the case for them to increase funding for new software, and to provide funding for software maintenance. Finally, we can seek institutional resources for software that is considered sufficiently important to the institution, either operationally or for its reputation, and we can demonstrate to our own institutions when other institutions do this.

To both reduce work and bring in new resources, we can encourage collaboration. For example, using the work of others rather than reimplementing a function or package reduces what a software team (or its developers) needs

[1] Hinsen, K. (2019). Dealing with software collapse. Computing in Science & Engineering, 21(3):104—-108. https://doi.org/10.1109/MCSE.2019.2900945

to do themselves, even without assuming that the collaborators contribute to the software, which also may happen. Similarly, if others use a team's software and contribute to maintaining it, the team has less they need to do. To make this work, the software has to be designed from the start to be modular and reusable, and it must also be clearly documented and explained to potential users, even those in fields other than the developer's. And the team has to put effort into engaging and working with the potential user and contributor community

Given that volunteers and collaborators are important to a number of sustainability paths, it's worth considered why volunteers or collaborators choose to put effort into a software project, and thinking about how we can engage them. In the context of community activities and organizing, Porcelli[2] defines:

Engagement = intrinsic motivation + extrinsic motivation + support - friction

Where intrinsic motivation includes self-fulfillment, altruism, satisfaction, accomplishment, pleasure of sharing, curiosity, and making a real contribution to science; extrinsic motivation includes things like a job, rewards, recognition, influence, knowledge, relationships, and community membership; support means ease, relevance, timeliness, and value; and friction is technology, time, access, and knowledge.

Some examples of things we can do to increase engagement include the following. Use GitHub (or GitLab) for development; this reduces friction by using a technology known to most developers. Provide templates for issues and guidelines for good pull requests; this reduces friction by providing knowledge of how to work with our project, and increase support by easing the means of doing so. Provide a code of conduct and a welcoming and encouraging environment; this increases extrinsic motivation by helping develop relationships and a sense of community. Add contributors to a list of authors who are cited when the software is used; this increases both intrinsic motivation and extrinsic motivation through recognizing accomplishments. Highlight examples of how the software is used; this increases intrinsic motivation by demonstrating the contribution to science.

In addition to the general activities, we can also plan for a progression of types of engagements, as proposed by Cabunoc Mayes[3], with the goal of engaging a potential contributor in their first interaction with the software, and then moving their interaction to a higher and more significant level over time through intentional activities.

We should also remember that the challenge of software sustainability is not unique to HEP, and there are a number of other groups also working on this problem. In software sustainability generally, these include the Software Sustainability Institute (SSI, http://software.ac.uk/) and US Research Software Sustainability Institute (URSSI) Conceptualization (http://urssi.us/). The Research Software Alliance (ReSA, https://www.researchsoft.org) is working to coordinate and align these organization with an interest in research software. There are a number of different groups working on Research Software Engineering (RSE), including the Society of Research Software Engineering (https://society-rse.org/) and groups in various countries/regions, e.g., AUS/NZ, BE, DE, NL, Nordic, US (https://researchsoftware.org/assoc.html). Finally, working on training and education is The Carpentries (https://carpentries.org).

In addition to creating awareness of these issues via the workshop and this LOI, the authors of this LOI along with the speakers and attendees of the workshop are currently drafting a report on the workshop that will include specific actions to improve sustainability.

[2]Porcelli, J. (2013). How to grow users into active community members and get your community more engaged. 2013 Open Source Software Summit, Washington, DC, USA

[3]Cabunoc Mayes, A. (2020). Work open, lead open. Chan-Zuckerberg Initiative (CZI) Essential Open Source Software (EOSS) Kickoff Meeting, Berkeley, California, USA