**Data-driven software sustainability**

(a white paper for the 2019 Collegeville Workshop on Sustainable Scientific Software (CW3S19), also posted as https://danielskatzblog.wordpress.com/2019/05/28/data-driven-software-sustainability/)

Daniel S. Katz, University of Illinois at Urbana-Champaign
d.katz@ieee.org, @danielskatz

While it's difficult to define or measure software sustainability as a future property of software, we can define it in hindsight as "the software has continued to exist, been supported, and been used over some period of time." When this is the case, we can say that sufficient work was done to make this happen, which we can simplistically view as:

resources $\geq$ needed work

Thinking about the future then, we can use the same "equation" but without completely knowing either the future resources or needs. Given this uncertainty, we can focus on the things we can control, increasing the resources and/or decreasing the needs as much as possible. Given some experience with research software, we can often say whether we think we will have sufficient resources to meet the expected needs, recognizing that this is really a guess.

Thus, to increase the likelihood that any particular software package will be sustained, the developers of that project need to increase the resources they have, or decrease the needed work.

Resources are money and effort, and a software project can use money to purchase effort. There are a number of general sources of money, including public and private funding agencies, and institutional support, and Nadia Eghbal maintains a list of options. We, as a community, can also try to influence public and private funding agencies to increase and sustain their funding of software projects.

In addition, effort can be provided by volunteers. Such volunteers can be motivated to contribute to software projects because of intrinsic and/or extrinsic motivation. Intrinsic motivation factors include self-fulfillment, altruism, satisfaction, accomplishment, pleasure of sharing, curiosity, and real contribution to science. Extrinsic motivation factors include job, rewards, recognition, influence, knowledge, relationships, and community membership.

Some of these are specific to the software and the project, and the project can work to improve them. It can to build a community around the project, including having an identity via stickers, prizes, regular communications, etc. It can highlight how the software is used, and what discoveries it enables. It can create roles for contributors (AstroPy provides a great example) and ways that contributors can govern parts or all of the project.

Some of the motivation factors are related to the ecosystem, and we as a community can also work to improve them. For example, we can create jobs for research software developers, such as Research Software Engineers (RSE) positions. We can also measure the impact of software by using software citations. We can encourage hiring and promotion processes to take software work into account. We can create prizes for open source development, maintenance, and contributions. And we can build a general research software community that underpins and enables the community (e.g., via SSI, URSSI, BSSw).

To decrease work, we can consider best practices – ways to improve the processes by which the work is performed. These processes can be studied by researchers interested in the intersection between software engineering and research software and communities, including those in the SE4Science and CSCW communities. Once we understand good and best practices, we can use them to provide guidance and training, which can be provided and disseminated through training-focuses communities such as the Carpentries, institutional groups such as at the German Aerospace Center (DLR), disciplinary groups such as ELIXIR, and more general research software communities such as SSI, URSSI, BSSw.

Once we've done all these general things, a project can also start to remove sentiment from project decisions. If an essential goal is to make the project's software sustainable, the project needs to consider this in all of its decisions.

For example, for each pull request (PR) we get, the project should consider if accepting it will decrease the future work, or increase the future resources. Will it make the software easier to maintain in the future? Will it increase the chance of getting future funding? Will it increase the developer and maintainer communities? If none of the answers to these questions are yes, the project should not accept the PR. And the project can prioritize the PRs they do accept based on these factors.

An example of a PR not to accept might be one that provides a working solution to a single user's problem, but adds a large amount of technical debt to the project. Another might be a PR from a user whose attitude and comments will poison the project's environment and reduce the future contributor community.

Of course, these examples ignore the fact that research software, in many cases, is being developed to solve a problem that the researcher-developers have, and the choices are not as black and white as I've described.

In addition, we might think about target audiences not just based on the size of the potential user community, but also (or even primarily) on the size of the potential developer community. We might try to balance two future states – one where there are 500 users of our software, but only a couple of them are likely to contribute to future development and maintenance, and the other where there are 20 users, but all are likely to contribute to future development and maintenance. The first case might lead to more funding, if we think a funding agency will be

supportive based on the number of users, while the second might lead to a larger community of developers. Evaluating which case is more sustainable is difficult.

And of course, how to measure any of these things is part of the difficulty. We first need to decide how to measure the health of a project today, perhaps by the CHAOSS metrics, or perhaps by other means, and then we need to understand how these measures might change in the future.

Determining how to do this is a community challenge. Some ideas include:

- Gather any data tying such decisions in past projects to their impact, and use these to project the possible impact of future decisions in new projects
- Run role-playing exercises with real developers and real users
- Perform A/B testing with real projects
- Gather data from successful and unsuccessful projects and tie anecdotes about these projects to their outcomes
- Survey the leaders of successful projects to understand what choices they would make in a particular situation

---

Thanks to Katy Huff for helpful feedback on an earlier draft of this white paper.