Lawrence Berkeley National Laboratory

LBL Publications

Title

Lessons learned working with protected assets in an open-source collaborative scientific software project

Permalink

<https://escholarship.org/uc/item/0kn8s03p>

ISBN

9781665444606

Authors

Beattie, K
Gunter, D

Publication Date

2021-06-01

DOI

10.1109/BoKSS52540.2021.00014

Peer reviewed

Lessons learned working with protected assets in an open-source collaborative scientific software project

Keith Beattie

Computational Research Division
Lawrence Berkeley National Lab
Berkeley, CA
ksbeattie@lbl.gov

Dan Gunter

Computational Research Division
Lawrence Berkeley National Lab
Berkeley, CA
dkgunter@lbl.gov

Abstract— We describe a use-case for balancing the demands of commercial intellectual property and open-source in a collaborative scientific software project.

Keywords—software engineering, release engineering, collaborative development, software sustainability

I. CONTEXT

The promise of open-source software is an enticing one. From the user's perspective, it is "free", with the code open and available to be examined, documented, tested and extended by an unlimited number of people. From the developer's perspective there is at least the possibility that your code contributions will survive the ending of a project's funding, changing jobs, or even a company's bankruptcy. This corresponds to *technical sustainability* in the Karlskrona Manifesto (<https://www.sustainabilitydesign.org/>). However, the *economic sustainability* of an applied scientific project can be greatly increased by successful collaboration with commercial entities, e.g., by increasing the applicability and robustness of the scientific tools. Commercial partners often have protected intellectual property (IP) that they do not want to release to the public, which demands that we somehow use a combination of both closed and open software development practices.

Here we describe one such situation with a specific example of how we came to balance these, at times competing, interests between commercial and open-source approaches to developing software.

The Institute for the Design of Advanced Energy Systems (IDAES) was formed in 2016 to develop new Process Systems Engineering (PSE) capabilities for the identification, synthesis, optimization, and analysis of innovative advanced energy systems at scales ranging from materials, to process, to system and market. The IDAES team spans three national laboratories (NETL, SNL, LBNL) and three Universities (Carnegie-Mellon University, West Virginia University, University of Notre Dame) and has active partnerships with several energy commercial partners.

This research was conducted as part of the Institute for the Design for Advanced Energy Systems (IDAES) with funding from the Simulation-Based Engineering, Crosscutting Research Program of the U.S. Department of Energy's Office of Fossil Energy. This manuscript has been authored with the support of Lawrence Berkeley National Laboratory under Contract No. DE-AC02-05CH11231 with the U.S. Department of Energy

One of the primary results of this project is the IDAES open-source software platform, used by both commercial and research interest alike in formulating, manipulating, and solving large, complex, structured optimization problems. The commercial partnerships within IDAES are managed as CRADAs (Cooperative Research and Development Agreements) which allows government agencies, universities and private companies to work together to help speed the development of new technologies while protecting the private company's intellectual property.

The goal and benefit of having commercial partners is to gain real-world insights into the specific issues faced and use those insights to improve the software framework in a general way to better address not just those specific issues but entire classes of similar problems.

II. CHALLENGE

We wish to effectively collaborate with commercial partners such that their IP is protected while simultaneously developing an open source software framework that is capable of solving the classes of problems faced by these users. One of the primary forms of collaboration is to apply the IDAES optimization tools on an existing process for which the commercial partners have data, specifications, and sometimes existing process models. For example, we have a small subset of developers who work closely with our commercial partners, under an NDA (Non-Disclosure Agreement) using the IDAES framework to understand and frame their computational problems around an operating power plant. The IP that this involves is a combination of raw data, algorithms, and code. The goal is for the knowledge gained to be added back in a more general form to the open-source framework.

We struggled with how best to manage this process, trying to ensure that no IP was leaked while also preserving the transparency, ease of accepting contributions and attributions essential to running an open-source project. What follows is a description of the evolution from our initial approach to what, we believe, is a better approach, along with a description of the warts of both approaches.

A. Dual-Repository Approach

Our first solution to this problem was to break our development into two separate source code repositories: a private one and a public one, as shown in Figure 1(a).

The private repository is where all the active development took place. Only internal members of the IDAES development team had read or write permission to this repository. This included members working with our commercial partners. Working within this repository, we used a common Github workflow where developers added codes through pull-requests (PRs) from their forks to bring in their changes to the upstream (main) repository. Every PR had to pass automated tests and receive two approving code reviews before it could be merged.

The public repository was only changed by manual pushes of (merged) changes from the private repository. These usually occurred weekly and were performed by the release manager. Software releases were “cut” from the public repository on a roughly quarterly basis, also by the release manager.

The advantage of this process is that individual developers could never accidentally leak IP by pushing code to their fork. A clear separation between what was public and what was private was considered important in a project where there were a number of chemical engineering graduate students contributing code. For example, when working with a public repository, even an un-reviewed change to your personal fork (of that public repository) is instantly public. If that change accidentally contains a partner’s IP, this could be devastating to the project, as the proverbial horse has already left the barn.

Even though we had no IP leaks, and this process involved three separate places where IP could be detected – code reviews, migration to public repository, and release – this process had some critical weaknesses. First, it put the burden of identifying protected IP on the people least likely to be able to identify it. Given that the number of people working closely with our commercial partners is quite small, and that this software, like many scientific software packages, depended on highly specialized knowledge, often there were not two reviewers that could easily identify protected IP in a PR. The release manager is likewise not capable of reliably reviewing all PRs for protected IP. Second, this solution defeated many of the advantages of a public repository: it hid the true pace of work from the public; weekly pushes obscured the attributions of the individual contributors (to both funding agencies and future employers); and complicated the process of accepting external contributions via the public repository. Working primarily in a private repository also made most cloud-hosted tools, such as continuous integration, harder and/or more expensive to use.

B. Single-Repository Approach

As a result of these weaknesses, we chose to simply eliminate the private repository: all non-IP development, which is the majority, now takes place in the public repository. The resulting workflow is shown in Figure 1(b).

To address the concerns of protected assets, we identify very early in the partnership and development process any work which has the possibility of working with commercial IP. Then, any team member working with our commercial partners works in a separate development environment (which may be anything that is mutually acceptable to the team and the commercial partner) and only contributes to a fork or clone of the public repository after a relatively heavyweight internal review of their proposed changes. The reviewers of those changes are

specifically identified as those able to identify IP information from that partner, and frankly are only concerned in that regard for this level of review. This review chain includes an explicit sign-off by at least one person who owns the IP, i.e. one commercial partner. Once that IP review is complete, the normal public PR submission and review process is followed (2 reviewers, documentation, automated tests, etc.) where those changes are vetted safe from any IP concerns.

To know when you should be developing in the open-source repositories, when you should not be, and how to keep these things firmly separated on the same developer laptop, we realized we needed better training for *all* team members, as soon as they join the team, on our software development practices. It is our hope that the focus on training will enrich everyone who comes through the IDAES project, providing an element of *individual sustainability*. We know that developing the training materials increased our knowledge and awareness about IP and copyrighted material in general.

Thus, to implement our new approach, we developed a set of materials to use as an onboarding training session for all new developers. Here, we explain that some people will be working with sensitive information from our commercial partners and it is both our collective and their individual responsibility to ensure that this information is protected. We note during this training that this is not fundamentally different from other instances of keeping unwanted material out of a public source tree, for example, copyrighted material from other publicly available sources. The onboarding process even includes a quiz at the end; only after new developers pass the quiz are their names added to list of members, and they are given full developer permissions for the GitHub organization.

Figure 1: Software development workflows

III. CONCLUSION

While in retrospect we see clearly that the early separation of IP work from the public repository is a better process to follow, there was understandably strong support for our initial “split repository” approach early in the project. Having gone through this experience of using one software development process, learning its strengths and weaknesses, then moving to a newer more appropriate one allowed us to more deeply understand how to approach the challenge of IP mixed with open source development in a way that promotes *technical sustainability* and *economic sustainability* of the project, while contributing to the *individual sustainability* by introducing many students and young developers to thinking more deeply about this process.