TOOLBOX

TIPS FOR OPEN-SOURCE SOFTWARE SUPPORT

Releasing lab-built open-source software often involves a mountain of unforeseen work for the developers.

ILLUSTRATION BY THE PROJECT TWINS

BY JULIAN NOWOGRODZKI

On 10 April, astrophysicists announced that they had captured the first ever image of a black hole. This was exhilarating news, but none of the giddy headlines mentioned that the image would have been impossible without open-source software. The image was created using Matplotlib, a Python library for graphing data, as well as other components of the open-source Python ecosystem. Just five days later, the US National Science Foundation (NSF) rejected a grant proposal to support that ecosystem, saying that the software lacked sufficient impact.

It's a familiar problem: open-source software is widely acknowledged as crucially important in science, yet it is funded non-sustainably. Support work is often handled ad hoc by overworked graduate students and postdocs, and can lead to burnout. "It's sort of the difference between having insurance and having a GoFundMe when their grandma goes to the hospital," says Anne Carpenter, a computational biologist at the Broad Institute of Harvard and MIT in Cambridge, Massachusetts, whose lab developed the image-analysis tool CellProfiler. "It's just not a nice way to live."

Scientists writing open-source software often lack formal training in software engineering, which means that they might never have learnt best practices for code documentation and testing. But poorly maintained software can waste time and effort, and hinder reproducibility. Biologists who use computational tools routinely spend "hours and hours" trying to ▶

▶ get other researchers' code to run, says Adam Siepel, a computational biologist at Cold Spring Harbor Laboratory in New York, and a maintainer of PHAST, a tool used for comparative and evolutionary genomics. "They try to find it and there's no website, or the link is broken, or it no longer compiles, or crashes when they've tried to run it on their data."

But there are resources that can help, and models to emulate. If your research group is planning to release open-source software, you can prepare for the support work and the questions that will arise as others begin to use it. It isn't easy, but the effort can yield citations and name recognition for the developers, and improve efficiency in the field, says Wolfgang Huber, a computational biologist at the European Molecular Biology Laboratory in Heidelberg, Germany. Plus, he adds, "I think it's fun."

HAVE A PLAN

For developers of scientific software, release day isn't the end of the labour, but often the beginning. Tim Hopper, a data scientist at Cylance in Raleigh, North Carolina, says on Twitter, "Give a man a fish and you feed him for a day. Write a program to fish for him and you maintain it for a lifetime." Carpenter hired a full-time software engineer to handle maintenance for CellProfiler, which logs about 700 questions and 100 bug reports or feature requests per year, or about 15 per week. But most open-source software maintenance is done on a volunteer basis. "I did this myself, like after midnight," says Siepel of his tech-support efforts on PHAST.

To prepare for what's coming, it helps to have an idea of what you're getting into. Some software will just need short-term support, whereas other programs might be used for decades. Nelle Varoquaux says that, in her field of machine learning in biology, software tools quickly become obsolete because the size of the data sets is changing so rapidly. Varoquaux is a computational biologist at the University of California, Berkeley, and co-developer of scikit-learn, a machine-learning package for Python. "When I started my PhD, everything I worked on fitted into RAM, and I never had a memory problem," she says. But today, memory is a huge challenge. She estimates that she will need to maintain two tools she built for analysing DNA and chromosome conformation — iced and pastis — for only another five years before they become obsolete.

Obsolescence isn't bad, she adds: knowing when to stop supporting software is an important skill. "Let a tool die when it has reached the end of its usefulness or, when a maintainer wants to quit, orphan it and search for a foster parent," Huber advises.

However long your software will be used for, good software-engineering practices and documentation are essential, says Andreas Mueller, a machine-learning scientist at Columbia University in New York City. These include continuous integration systems (such as TravisCI), version control (Git) and unit testing. "Continuous integration tells you, every time you change your code, if it still works or if you broke it," as long as you write the correct tests for it to run, says Mueller; version control is a system of recording changes to source code so that you can revert to any previous version if necessary; and unit testing tests each individual component of the software to ensure that it is sound. The combination, he says, "will 100% save you time". Organizations such as volunteer-run Software Carpentry and the eScience Institute at the University of Washington, Seattle, host bootcamps on software development, and make tutorials available on GitHub. The Netherlands eScience Center in Amsterdam provides a guide to software-development best practices at https://guide.esciencecenter.nl

To facilitate maintenance, Varoquaux recommends focusing on code readability over peak performance. "I always try to make it readable and well-documented and tested, so if something breaks I can fix it quickly," she says.

And that's inevitable when it comes to software: "As soon as you have users, they're going to find bugs," Varoquaux says. Huber recommends fielding user questions through a public forum, such as Stack Overflow, where users can tag their question with the software name. "Do not respond to private mails for support from users," he says.

"As soon as you have users, they're going to find bugs."

Public forums offer three advantages. First, they reach many more users than do individual e-mails. "For everybody who writes an e-mail, there's probably 100 people who are too shy to ask," says Huber. Second, they tend to encourage more focused and thoughtful questions. Third, they dissuade users from the time-wasting strategy of e-mailing multiple software maintainers separately with the same question.

Huber also recommends releasing your software to a repository such as the Comprehensive R Archive Network (CRAN) or Bioconductor, an umbrella archive for biological software written in R, instead of to your personal home page or GitHub. Such repositories are curated, and have submission guidelines for naming conventions and required components, much as scientific journals do. And both CRAN and Bioconductor "offer testing and continuous integration on several platforms, and robust, easy-to-use installers", says Huber.

A MATTER OF FUNDING

Software support requires both time and money. But funding can be hard to come by. In the United States, the National Institutes of Health (NIH) and the NSF focus on new research, and the maintenance of open-source software often doesn't fit well into their requirements. "That's really the tragedy of the funding agencies in general," says Carpenter. "They'll fund 50 different groups to make 50 different algorithms, but they won't pay for one software engineer."

But some funding does exist from these and other organizations. One Twitter thread (see go.nature.com/2yekao5) documents grants from the NSF's Division of Biological Infrastructure, the NIH's National Human Genome Research Institute and the National Cancer Institute, and a joint programme from the NSF and the UK Biotechnology and Biological Sciences Research Council (now part of UK Research and Innovation). Private US foundations such as the Gordon and Betty Moore Foundation, the Alfred P. Sloan Foundation and the Chan Zuckerberg Initiative (CZI) also fund open-source software support. The CZI provides support for the Python-based image-processing software scikit-image, the ImageJ and Fiji platforms, and also funds the software engineer on Carpenter's team.

In the United Kingdom, the Software Sustainability Institute, based at the University of Edinburgh, provides free, short, online evaluations of software sustainability, and fellowships of £3,000 ($US3,800) for researchers based in Britain or their collaborators. The institute periodically makes slots available for people to work with their experts for up to six months to develop new software or sharpen existing software and maintenance practices. In Germany, Huber recommends the European Commission's network grants and the German ministry of science's deNBI initiative, both of which provide funding for Bioconductor.

The general problem of digital-infrastructure maintenance is gaining more attention. Varoquaux and her colleagues have received $138,000 from the Alfred P. Sloan and Ford foundations to study "the visible and invisible work of maintaining open-source software", she says, including burnout in researchers who devote their time to this work — part of a portfolio of 13 digital-infrastructure research projects funded to the tune of $1.3 million. In May, the CZI announced three requests for proposals to fund open-source biomedical software, the first of which opened in June. Siepel has a review article in the press in *Genome Biology* on the challenge of funding open-source software support.

And funding is needed: writing software that is easy for others to use on a wide range of data takes much more effort than software that works only for you. "The difference is at least as large as between the polished paper published in *Nature* and the first stack of slides for a lab meeting with the underlying results," Huber says.

Still, there's real value in the exercise. Siepel's team sometimes responds to user queries by pointing out that they're applying the software to the wrong data, a subtlety that an evolutionary biologist would notice but a software engineer might not. "There's a sort of idiom: eat your own dog food," Huber says: "If you use your own software for real questions, then you realize where it's bad, where it's lacking. Having a domain expert write the software tends to make the software more valuable." ■

Julian Nowogrodzki *is a science writer based in Boston, Massachusetts.*