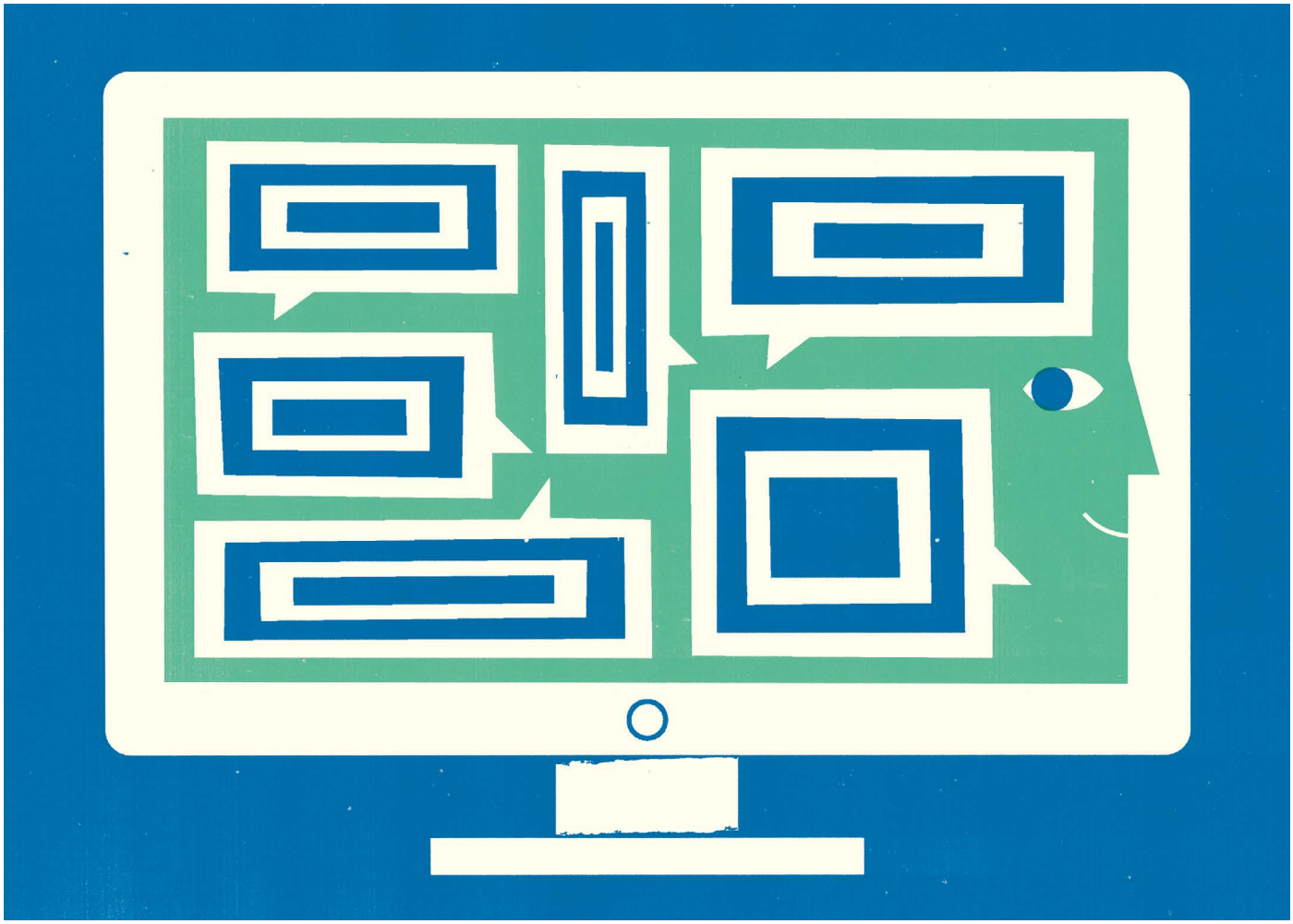

ILLUSTRATION: THE PROJECT TWINS

SIX TIPS FOR GOING PUBLIC WITH YOUR LAB'S SOFTWARE

It's not enough to write high-quality programs. If you want to make your apps widely available, you should also follow these steps. **By Julian Nowogrodzki**

When computational neurologists Carsen Stringer and Marius Pachitariu built Cellpose, a software that automatically identifies and outlines cells in microscopy images, their goal was to release it publicly for anyone to use. They did so in 2020 (see go.nature.com/3bbeey3). But that doesn't mean it was smooth sailing.

People often underestimate how difficult it is to release a tool publicly, says Stringer, who, with Pachitariu, works at the Howard Hughes Medical Institute's Janelia Research Campus in Ashburn, Virginia. There's a big difference between a program that technically works and one that is robust, is well documented and has a polished user interface. Stringer and Pachitariu's experience building two earlier neuroscience tools, Suite2p and Kilosort, meant that they knew what they were getting into. But Cellpose was more popular than they expected, and they were caught off guard by the volume of user queries, she says.

From an engagement point of view, that's a good problem to have. Still, researchers can make their lives easier. *Nature* spoke to Stringer and other scientific-software developers to learn their tips for smoothly releasing a laboratory tool for public use. Here's what they said.

Anticipate maintenance time

Software isn't static; it needs to be maintained. That means addressing bugs and requests for extra features, working with collaborators and answering user questions. It's a lot of work, but that's how tools get better, says computer scientist Anna Kreshuk at the European Molecular Biology Laboratory in Heidelberg, Germany.

Kreshuk has been working since 2012 on Ilastik (www.ilastik.org), a program that analyses microscopy images using interactive machine learning. (The project was initiated by another researcher a year earlier, she says.) The difference between a tool that is useful mainly to you and a public one, she says, is maintenance: "just iterating over the feedback of the people who try to use it and then find that it's actually very inconvenient, and then fixing that and then iterating again".

Stringer estimates that it takes between 5 and 15 hours a week to maintain most software tools — less if you're focused on maintenance, more if you're trying to improve the

program — and Cellpose, which requires about 10 hours a week of a PhD student's time, falls in that range, she says. (The student is paid as a consultant.) Bioinformatician Titus Brown at the University of California, Davis, devotes 20–30% of his time to maintaining Sourmash (N. T. Pierce *et al. F1000 Res.* **8**, 1006; 2019), a tool for genome comparison and metagenomic analysis. By contrast, Ilastik has a dedicated, full-time developer, Kreshuk says.

Indeed, scientific software is often built by PhD students or postdocs to address a pressing problem, but if you plan on making a tool public, you might need to think bigger. "If you want to stay in the game seriously," Kreshuk says, "it's good to think about when you're going to hire the first professional software developer to work on this and how you're going to pay for it, because they don't work for academic wages."

Simplify installation

Good software should be easy to use, and nothing sours the experience for potential users like a difficult installation process. "Install issues are the number one reason people will be deterred," Stringer says. Brown echoes that sentiment: "If I'm interested in your tool and it takes me more than a minute to install, I'm no longer interested in your tool."

Instead of simply posting code to the online code-sharing service GitHub, create ready-to-install binary packages for different operating systems. Working with existing package repositories, such as PyPI for programs written in the programming language Python or Bioconductor for code written in R, can provide quality control and a central location for downloading.

Minimizing dependencies, or other software tools that users need to have installed for your tool to work, can also simplify installation. Cellpose has 11 dependencies, including the machine-learning library PyTorch and numerical-analysis tool NumPy, but it automatically installs those that are missing. Having fewer dependencies also cuts down on your future work, because you might need to update your program as dependencies change. "The hardest thing about releasing software is making sure the dependencies are up to date," says Stringer.

Alternatively, allow users to try out the software without installation. Researchers can run Cellpose on the tool's website to see whether it works with their data before they decide to install it. They can also run it on Google's cloud-based computational notebook system, Colab, which enables access to remote graphical processing units that users might not have locally, Stringer says.

Consider your interface

Software can have an unintuitive, messy interface — and that's OK, if you're the only one who uses it. If not, you might need to put some energy into 'user experience'.

It takes a lot of work to design a graphical user interface, says Stringer, but having one makes it a lot easier for people to get started. A command-line interface is less work but requires a more tech-savvy user.

Cellpose has both a graphical user interface and a command-line one — the former allows users to interact with their images directly, the latter enables automation. "Data visualization is so important to see if there's a bug or a bias in the output of the software," Stringer explains. "Having everything abstract and without visualization can make it very difficult to see if you have data-quality issues."

Document

Commercial software used to come with printed manuals. Those days are long gone, but documentation remains a must for usable software — even if it isn't fun to write. "I have a love/hate relationship with documentation," Brown admits. "It's a constant struggle" to properly document software. But it's essential.

"If I'm interested in your tool and it takes me more than a minute to install, I'm no longer interested in your tool."

At the most basic level, developers should state their tool's computational requirements (such as memory and dependencies), its licence and to what extent it is being supported, says Brown. Open-source programs could also include whether developers are interested in contributions or not.

For writing documentation, Brown and Stringer use Sphinx and Read the Docs. These tools gather the software's 'docstrings' — bits of code that act as comments to document the rest of the code — and "convert them to a nice readable format", Stringer says.

There are other ways to teach users about your tool, too. Brown advises writing a quick-start guide, which is a condensed, essentials-only version of a manual. Or, you could create a video tutorial, as Stringer did for Cellpose. "I didn't realize how useful video tutorials can be," she says of her 45-minute presentation, which has racked up more than 27,000 views. According to Stringer, her video follows the formula she uses for in-person talks: "Introduce some background about how the tool works, a use case and then show how it works in practice."

Use GitHub

Version control is crucial in software development because it tracks how and when the code was altered (and why and by whom), maintains a clear current version of the software and allows developers to easily revert unwanted changes if, for example, a collaborator alters the code in a way that breaks the tool.

One of the most popular version-control systems is Git. Git is a command-line tool built into most modern development environments, and many developers use it in conjunction with GitHub. But GitHub is more than just a repository: it also allows users to submit code changes and issues, which "makes it easier to see how people are struggling with the software and what you need to improve on", Stringer says. It also connects developers with one another.

For those thinking about career transitions, there's another benefit: GitHub provides a record of your work. Employers often scrutinize your GitHub history when considering job applicants, says Stringer.

Automate your testing

Once your software works, you want to make sure it's doing so correctly. That's the job of software testing — a task that is often automated using 'continuous integration' services. With each code change, these services, which include Travis CI and GitHub Actions, run a series of user-defined tests and flag when those fail.

Cellpose initially used Travis CI but now uses GitHub Actions, because the software is free for open-source tools and built into GitHub. It has everything the Cellpose team needs, Stringer says: testing across different operating systems and Python versions, as well as checking for installation problems.

Long story short: making software ready for public use takes work, but building public tools furthers transparency, community building and, ultimately, science. "The present and the future of biological understanding depend on software," says Brown. "We're not going to make progress in biology if we can't analyse our data."

And there are other advantages. Through developing Sourmash and adding features requested by users, many of whom use the software in ways he never predicted, Brown says he and his team have expanded their ideas of what kinds of scientific problem they can investigate. For example, it turns out that Sourmash is surprisingly good at the taxonomic classification of species in metagenomic-sequencing data from microorganisms. When Brown investigated, he found that the features of microbial genomes that the tool focuses on seem to be linked to the way in which such genomes evolve.

Brown says that it's worth it for him to keep developing software — and devoting the time it takes to maintain public tools, "because I keep on learning new things that I don't think I could have learned any other way".

Julian Nowogrodzki is a science writer and editor in Boston, Massachusetts.