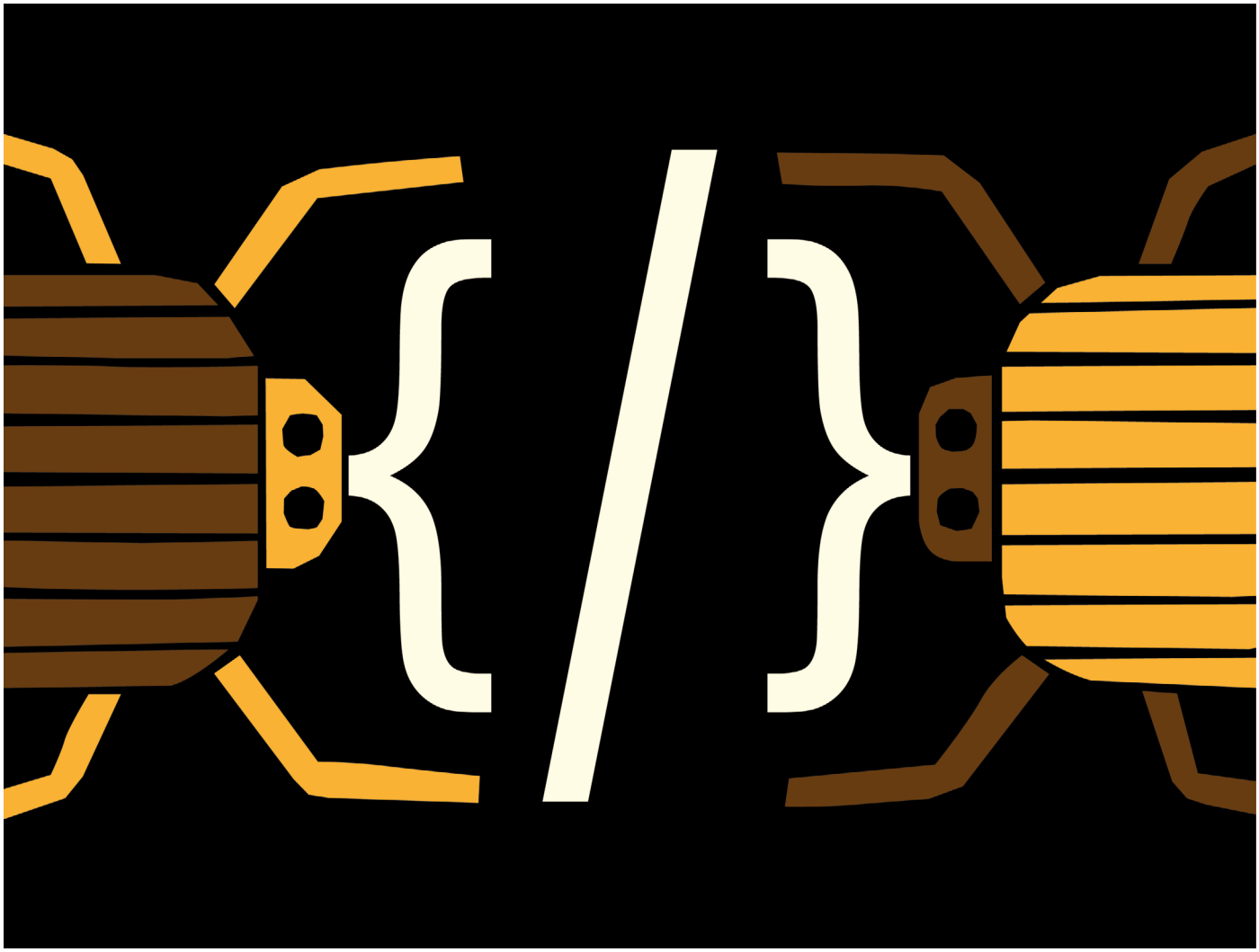

ILLUSTRATION BY THE PROJECT TWINS

HOW TO FIX YOUR SCIENTIFIC CODING ERRORS

Software bugs are frustrating. Adopting some simple strategies can help you to avoid them, and fix them when they occur. **By Jeffrey M. Perkel**

As a graduate student, Steven Weisberg helped to develop a university campus – albeit, a virtual one. Called Virtual Silcton, the software tests spatial navigation skills, teaching people the layout of a virtual campus and then challenging them to point in the direction of specific landmarks[1]. It has been used by more than a dozen laboratories, says Weisberg, who is now a cognitive neuroscientist at the University of Florida in Gainesville.

But in February 2020, a colleague who was testing the software identified a problem: it couldn't compute your direction accurately if you were pointing more than 90 degrees from the site. "The first thing I thought was, 'oh, that's weird'," Weisberg recalls. But it was true: his software was generating errors that could alter its calculations and conclusions.

"We have to retract everything," he thought.

When it comes to software, bugs are inevitable – especially in academia, where code tends to be written by graduate students and postdocs who were never trained in software development. But simple strategies can minimize the likelihood of a bug, and ease the process of recovering from them.

Avoidance

Julia Strand, a psychologist at Carleton College in Northfield, Minnesota, investigates strategies to help people to engage in conversation in, for example, a noisy, crowded restaurant. In 2018, she reported that a visual cue, such as a blinking dot on a computer screen that coincided with speech, reduced the cognitive effort required to understand what was being said[2]. That suggested that a simple smartphone app could reduce the mental fatigue that sometimes arises in such situations.

But it wasn't true. Strand had inadvertently programmed the testing software to start timing one condition earlier than the other, which, as she wrote in 2020, "is akin to starting a stopwatch before a runner gets to the line".

"I felt physically ill," she wrote – the mistake

could have negatively affected her students, her collaborators, her funding and her job. It didn't – she corrected her article, kept her grants and received tenure. But to help others avoid a similar experience, she has created a teaching resource called Error Tight[3].

Error Tight provides practical tips that echo computational reproducibility checklists, such as; use version control; document code and workflows; and adopt standardized file naming and organizational strategies.

Its other recommendations are more philosophical. An 'error tight' laboratory, Strand says, recognizes that even careful researchers make mistakes. As a result, her team adopted a strategy that is common in professional software development: code review. The team proactively looks for bugs by having two people review their work, rather than assuming those bugs don't exist.

Joana Grave, a psychology PhD student at the University of Aveiro, Portugal, also uses code review. In 2021, Grave retracted a study when she discovered that the tests she had programmed had been miscoded to show the wrong images. Now, experienced programmers on the team double-check her work, she says, and Grave repeats coding tasks to ensure she gets the same answer.

Scientific software can be difficult to review, warns C. Titus Brown, a bioinformatician at the University of California, Davis. "If we're operating at the ragged edge of novelty, there may only be one person that understands the code, and it may take a lot of time for another person to understand it. And even then, they may not be asking the right questions."

Weisberg shared other helpful practices in a Twitter thread about his experience. These include sharing code, data and computational environments on sites such as GitHub and Binder; ensuring computational results dovetail with evidence collected using different methods; and adopting widely used software libraries in lieu of custom algorithms when possible, as these are often extensively tested by the scientific community.

Whatever the origin of your code, validate it before using it – and then again periodically, for instance after upgrading your operating system, advises Philip Williams, a natural-products chemist at the University of Hawaii at Manoa in Honolulu. "If anything changes, the best practice is to go back and just make sure everything's OK, rather than just assume that these black boxes will always turn out the correct answer," he says.

Williams and his colleagues identified what they called a 'glitch' in another researcher's published code for interpreting nuclear magnetic resonance data[4], which resulted in data sets being sorted differently depending on the user's operating system. Checking their numbers against a model data set with known 'correct' answers, could have alerted them that the code wasn't working as expected, he says.

Recovery

If code cannot be bug-free, it can at least be developed so that any bugs are relatively easy to find. Lorena Barba, a mechanical and aerospace engineer at George Washington University in Washington DC, says that when she and her then graduate student Natalia Clementi discovered a mistake in code underlying a study[5] they had published in 2019, "there were some poop emojis being sent by Slack and all sorts of scream emojis and things for a few hours". But the pair were able to quickly resolve their problem, thanks to the reproducibility packages (known as repro-packs) that Barba's lab makes for all their published work.

"I started to think, 'oh dear, this maybe calls into question the original publication'."

A repro-pack is an open-access archive of all the scripts, data sets and configuration files required to perform an analysis and reproduce the results published in a paper, which Barba's team uploads to open-access repositories such as Zenodo and Figshare. Once they realized that their code contained an error – they had accidentally omitted a mathematical term in one of their equations – Clementi retrieved the relevant repro-pack, fixed the code, reran her computations and compared the results. Without a repro-pack, she would have had to remember exactly how those data were processed. "It probably would have taken me months to try to see if this [code] was correct or not," she says. Instead, it took just two days.

Brown needed significantly more time to resolve a bug he discovered in 2020 when attempting to apply his lab's metagenome-search tool, called spacegraphcats, towards a new question. The software contained a bad filtering step, which removed some data from consideration. "I started to think, 'oh dear, this maybe calls into question the original publication'," he deadpans. Brown fixed the software in less than two weeks. But re-running the computations set the project back by several months.

To minimize delays, good documentation is crucial. Milan Curcic, an oceanographer at the University of Miami, Florida, co-authored a 2020 study[6] that investigated the impact of hurricane wind speed on ocean waves. As part of that work, Curcic and his colleagues repeated calculations that had been conducted in the same lab in 2004, only to discover that the original code was using the wrong data file to perform some of its calculations, producing an "offset" of about 30%.

According to Google Scholar, the 2004 study[7] has been cited more than 800 times, and its predictions inform hurricane forecasts today, Curcic says. Yet its code, written in the programming language MATLAB, was never placed online. And it was so poorly documented that Curcic had to work through it line by line to understand how it worked. When he found the error, he says, "The question was, am I not understanding this correctly, or is this indeed incorrect?"

Strand has team members read each others' code to familiarize them with programming and encourage good documentation. "Code should be clearly commented enough that even someone who doesn't know how to code can understand what's happening and how the data are changing at each step," she says.

And she encourages students to view errors as part of science rather than personal failings. "Labs that have a culture of 'people who are smart and careful don't make mistakes', are setting themselves up for being a lab that doesn't admit their mistakes," she says.

Bugs don't necessarily mean retraction in any event. Barba, Brown and Weisberg's errors had only minor impacts on their results, and none required changes to their publications. In 2016, Marcos Gallego Llorente, then a genetics graduate student at the University of Cambridge, UK, identified an error in the code he wrote to study human migratory patterns in Africa 4,500 years ago. When he reanalysed the data, the overall conclusion was unchanged, although the extent of its geographic impact was, and a correction sufficed.

Thomas Hoye, an organic chemist at the University of Minnesota at Minneapolis, co-authored a study that used the software in which Williams discovered a bug. When Williams contacted him, Hoye says, he didn't have "any particular strong reaction". He and his colleagues fixed their code, updated their online protocols, and moved on.

"I couldn't help but at the end think, 'this is the way science should work'," he says. "You find a mistake, you go back, you improve, you correct, you advance."

Jeffrey M. Perkel is Technology Editor for *Nature*.

1. Weisberg, S. M., Schinazi, V. R., Newcombe, N. S., Shipley, T. F. & Epstein, R. A. *J. Exp. Psychol. Learn. Mem. Cogn.* **40**, 669–682 (2014).
2. Strand, J. F., Brown, V. A. & Barbour, D. L. *Psychon. Bull. Rev.* **26**, 291–297 (2019).
3. Strand, J. F. Preprint at PsyArXiv https://doi.org/10.31234/osf.io/rsn5y (2021).
4. Neupane, J. B. *et al. Org. Lett.* **21**, 8449–8453 (2019).
5. Clementi, N. C., Cooper, C. D. & Barba, L. A. *Phys. Rev. E* **100**, 063305 (2019).
6. Curcic, M. & Haus, B. K. *Geophys. Res. Lett.* **47**, e2020GL087647 (2020).
7. Donelan, M. A. *et al. Geophys. Res. Lett.* **31**, L18306 (2004).