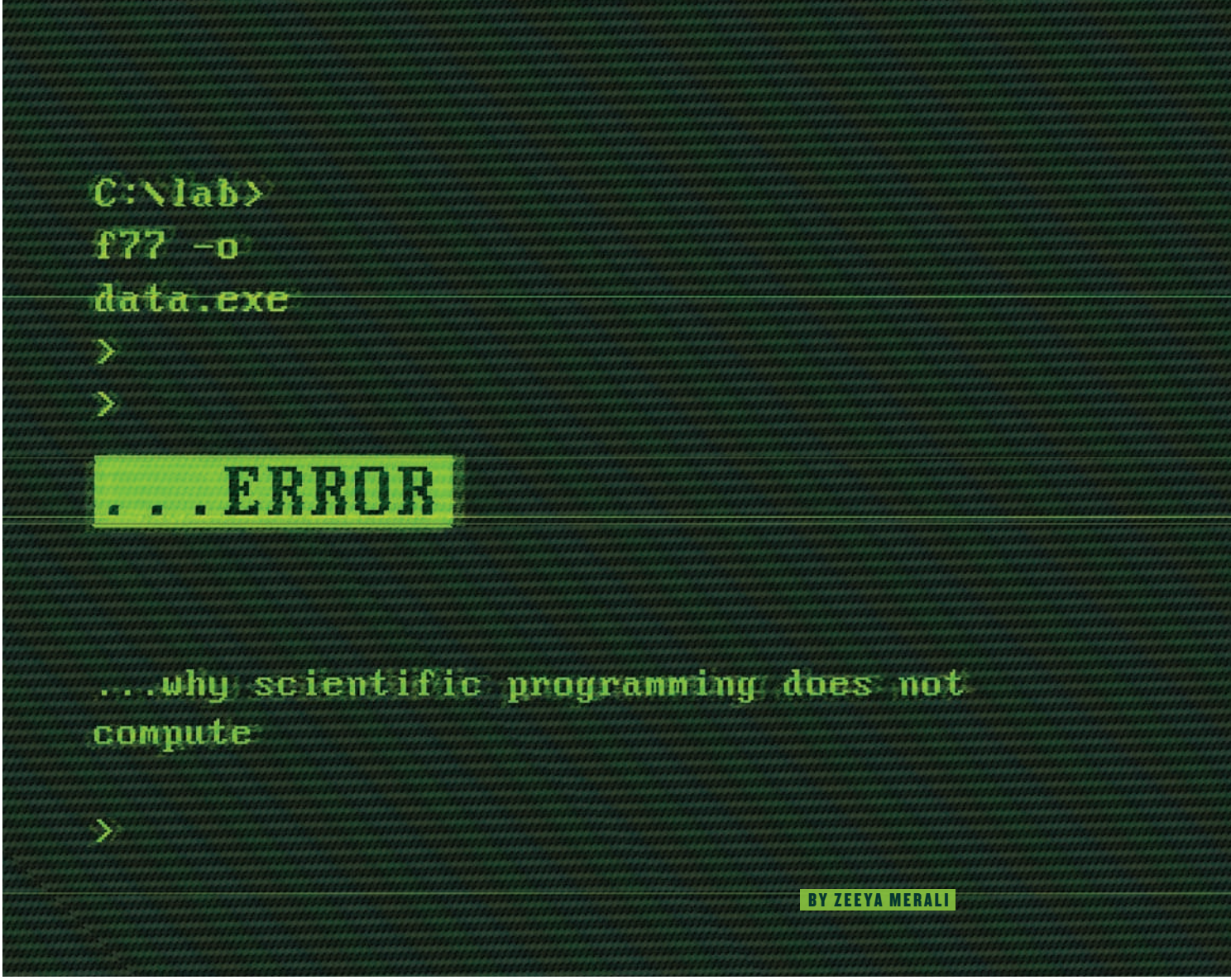


...why scientific programming does not compute

BY ZEEYA MERALI

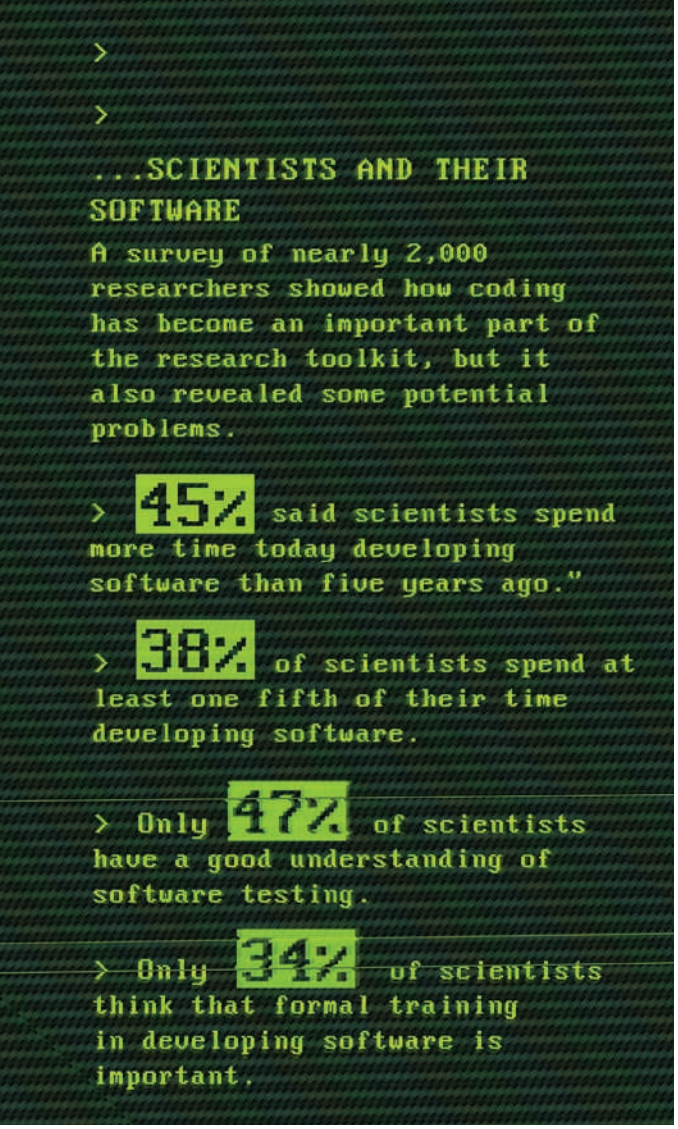
When hackers leaked thousands of e-mails from the Climatic Research Unit (CRU) at the University of East Anglia in Norwich, UK, last year, global-warming sceptics pored over the documents for signs that researchers had manipulated data. No such evidence emerged, but the e-mails did reveal another problem — one described by a CRU employee named "Harry", who often wrote of his wrestling matches with wonky computer software.

"Yup, my awful programming strikes again," Harry lamented in one of his notes, as he attempted to correct a code analysing weather-station data from Mexico.

Although Harry's frustrations did not ultimately compromise CRU's work, his difficulties will strike a chord with scientists in a wide range of disciplines who do a large amount of coding. Researchers are spending more and more time writing computer software to model biological structures, simulate the early evolution of the Universe and analyse past climate data, among other topics. But programming experts have little faith that most scientists are up to the task.

A quarter of a century ago, most of the computing work done by scientists was relatively straightforward. But as computers and programming tools have grown more complex, scientists have hit a "steep learning curve", says James Hack, director of the US National Center for Computational Sciences at Oak Ridge National Laboratory in Tennessee. "The level of effort and skills needed to keep up aren't in the wheelhouse of the average scientist."

As a general rule, researchers do not test or document their programs rigorously, and they rarely release their codes, making it almost impossible to reproduce and verify published results generated by scientific software, say computer scientists. At best, poorly written



SOURCE: G. WILSON

programs cause researchers such as Harry to waste valuable time and energy. But the coding problems can sometimes cause substantial harm, and have forced some scientists to retract papers.

As recognition of these issues has grown, software experts and scientists have started exploring ways to improve the codes used in science. Some efforts teach researchers important programming skills, whereas others encourage collaboration between scientists and software engineers, and teach researchers to be more open about their code.

A PROPER EDUCATION

Greg Wilson, a computer scientist in Toronto, Canada, who heads Software Carpentry — an online course aimed at improving the computing skills of scientists — says that he woke up to the problem in the 1980s, when he was working at a physics supercomputing facility at the University of Edinburgh, UK. After a series of small mishaps, he realized that, without formal training in programming, it was easy for scientists trying to address some of the Universe's biggest questions to inadvertently introduce errors into their codes, potentially "doing more harm than good".

After decades griping about the poor coding skills of scientists he knew, Wilson decided to see how widespread the problem was. In 2008, he and his colleagues conducted an online survey of almost 2,000 researchers, from students to senior academics, who were working with computers in a range of sciences. What he found was worse than he had anticipated[1] (see 'Scientists and their software'). "There are terrifying statistics showing that almost all of what scientists know about coding is self-taught," says Wilson. "They just don't know how bad they are."

As a result, codes may be riddled with tiny errors that do not cause the program to break down, but may drastically change the scientific results that it spits out. One such error tripped up a structural-biology group led by Geoffrey Chang of the Scripps Research Institute in La Jolla, California. In 2006, the team realized that a computer program supplied by another lab had flipped a minus sign, which in turn reversed two columns of input data, causing protein crystal structures that the group had derived to be inverted. Chang says that the other lab provided the code with the best intentions, and "you just trust the code to do the right job". His group was forced to retract five papers published in *Science*, the *Journal of Molecular Biology* and *Proceedings of the National Academy of Sciences*, and now triple checks everything, he says.

"How many fields have been held back, and how many people have had their careers disrupted, because of a buggy program?" asks Wilson.

More-rigorous testing could help. Diane Kelly, a computer scientist at the Royal Military College of Canada in Kingston, Ontario, says the problem is that scientists rely on "validation testing" — looking to see whether the answer that the code produces roughly matches what the scientists expect — and this can miss important errors[2]. The software industry relies on a different approach: breaking codes into manageable chunks and testing each piece individually, then visually inspecting the lines of code that stitch these chunks together (see 'Practicing safe software').

Many programmers in industry are also trained to annotate their code clearly, so that others can understand its function and easily build on it. But scientists often lack these communication and documentation skills. Even if researchers lift a whole working code and reuse it, rather than writing their own, they can apply the program incorrectly if it lacks clear documentation. Aaron Darling, a computational biologist at the University of California, Davis, unwittingly caused such a mistake with his own computer code for comparing genomes to reconstruct evolutionary relationships. He had designed the program to work only with closely related organisms, but discovered that an independent group had used it to look at sequences far outside the code's working range.

"It was lucky that I came across it, because their published results were totally wrong, but they couldn't know that because I hadn't clearly documented how my code worked," says Darling. "It's not something that I am proud of, but I am careful to be more clear now."

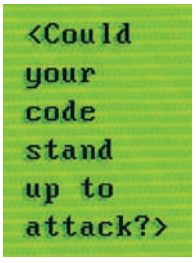
SLAYING THE MONSTER

Problems created by bad documentation are further amplified when successful codes are modified by others to fit new purposes. The result is the bane of many a graduate student or postdoc's life: the 'monster code'. Sometimes decades old, these codes are notoriously messy and become progressively more nightmarish to handle, say computer scientists.

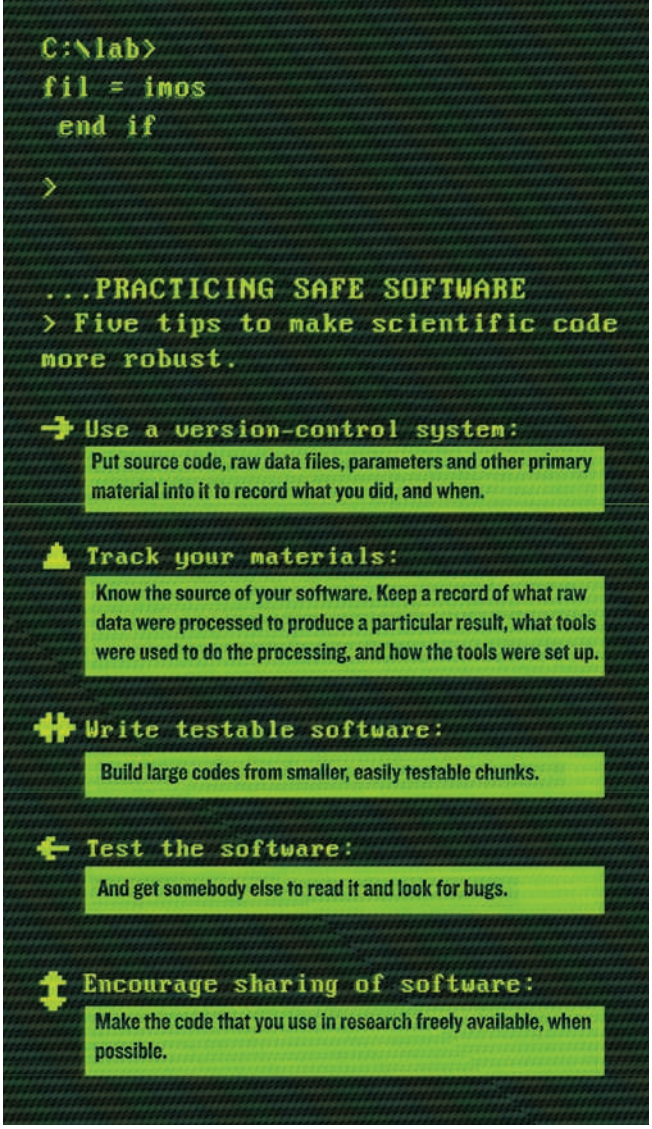
"You do have some successes, but you also end up with a huge stinking heap of software that doesn't work very well," says Darling.

The mangled coding of these monsters can sometimes make it difficult to check for errors. One example is a piece of code written to analyse the products of high-energy collisions at the Large Hadron Collider particle accelerator at CERN, Europe's particle-physics laboratory near Geneva, Switzerland. The code had been developed over more than a decade by 600 people, "some of whom are excellent programmers and others who do not really know how to code very well", says David Rousseau, software coordinator for the ATLAS experiment at CERN. Wilson and his students tried to test the program, but they could not get very far: the code would not even run on their machines.

Rousseau says that the ATLAS group can test the software only on the Linux operating system at the moment, but is striving to make the code compatible with Mac computers. This is important, he says, "because different platforms expose different types of errors that may otherwise be overlooked".



Some software developers have found ways to combat the growth of monster code. One example is the Visualization Toolkit, an open-source, freely available software system for three-dimensional



computer graphics. People can modify the software as they wish, and it is rerun each night on every computing platform that supports it, with the results published on the web. The process ensures that the software will work the same way on different systems.

That kind of openness has yet to infiltrate the scientific research world, where many leading science journals, including *Nature*, *Science* and *Proceedings of the National Academy of Sciences*, do not insist that authors make their code available. Rather, they require that authors provide enough information for results to be reproduced.

THE SEARCH FOR SOLUTIONS

In November 2009, a group of scientists, lawyers, journal editors, and funding representatives gathered for the Yale Law School Data and Code Sharing Roundtable in New Haven, Connecticut, where they recommended that scientists go further by providing links to the source-code and the data used to generate results when publishing. Although a step in the right direction, such requirements don't always solve the problem. Since 1996, The *Journal of Money, Credit and Banking* has required researchers to upload their codes and data to an archive. But a 2006 study revealed that of 150 papers submitted to the journal over the preceding decade that fell under this requirement, results could be independently replicated with the materials provided for fewer than 15 (ref. 3).

Proponents of openness argue that researchers seeking to replicate published results need access to the original software, but others say that more transparency may not help much. Martin Rees, president of the Royal Society in London, says it would be too much to ask reviewers to check code line by line. And in his own field of astrophysics, results can really be trusted only in cases in which a number of different groups have written independent codes to perform the same task and found similar results. Still, he acknowledges that "how to trust unique codes remains an issue".

There are signs that scientific leaders are now taking notice of these concerns. In 2009, the UK Engineering and Physical Sciences Research Council put out a call for help for scientists trying to create usable software, which led to the formation of the Software Sustainability Institute (SSI) at the University of Edinburgh. The SSI unites trained software developers with scientists to help them add new lines to existing codes, allowing them to tackle extra tasks without the programs turning into monsters. They also try to share their products across disciplines, says Neil Chue Hong, the SSI's director. For instance, they recently helped build a code to query clinical records and help monitor the spread of disease. They are now sharing the structure of that code with researchers who are trying to use police records to identify crime hot spots. "It stops researchers wasting time reinventing the wheel for each new application," says Chue Hong.

➲ NATURE.COM
To discuss programming in research, visit:
go.nature.com/ed3hsl

Another solution is to bring trained computer scientists into research groups, either permanently or as part of temporary alliances. Software developer Nick Barnes has set up the Climate Code Foundation, based in Sheffield, UK, to help climate researchers. He was motivated by problems with NASA's Surface Temperature Analysis software, which was released to the public in 2007. Critics complained that the program, written in the scientific programming language Fortran, would not work on their machines and they could therefore not trust what it said about global warming. In consultation with NASA researchers, Barnes rewrote the code in a newer, more transparent programming language — Python — reducing its length and making it easier for people who aren't software experts to understand how it functions. "Because of the immense public interest and the important policy issues at stake, it was worth taking the time to do that," says Barnes. His new code shows the same general warming trend as the original program.

In the long term, though, Barnes says that there needs to be a change in the way that science students are trained. He cites Wilson's online Software Carpentry course as a good model for how this can be done, to equip students with coding skills. Wilson developed the week-long course to introduce science graduate students to tools that have been software-industry standards for 30 years — such as 'version control', which allows multiple programmers to make changes to the same code, while keeping track of all changes.

Science administrators also need to value programming skills more highly, says David Gavaghan, a computational biologist at the University of Oxford, UK. "There needs to be a real shift in mindset away from worrying about how to get published in *Nature* and towards thinking about how to reward work that will be useful to the wider community."

Gavaghan now uses the software industry's 'master–apprentice' approach to train graduate students in his lab. New software projects are split up into bite-sized chunks, with each segment assigned to a pair of programmers — one experienced and one novice — who work together on it. "It forces students to become consistent code-builders," says Gavaghan.

Bringing industrial software-development practices into the lab cannot come too soon, says Wilson. The CRU e-mail affair was a warning to scientists to get their houses in order, he says. "To all scientists out there, ask yourselves what you would do if, tomorrow, some Republican senator trains the spotlight on you and decides to turn you into a political football. Could your code stand up to attack?" ■ SEE WORLD VIEW, P.753

Zeeya Merali *is a freelance writer in London.*

1. Hannay, J. E. *et al. Proc. 2nd Int. Workshop on Software Engineering for Computational Science and Engineering* (2009).
2. Kelly, D. *IEEE Software* **24,** 119–120 (2007).
3. McCullough, B. D., McGeary, K. A. & Harrison, T. D, *J. Money Credit Banking* **38,** 1093–1107 (2006).