COMPUTATIONAL SCIENCE

Troubling Trends in Scientific Software Use

"Blind trust" is dangerous when choosing software to support research.

Lucas N. Joppa,[1]* Greg McInerny,[12] Richard Harper,[1] Lara Salido,[3] Kenji Takeda,[1] Kenton O'Hara,[1] David Gavaghan,[2] Stephen Emmott[1]

Software pervades every domain of science (*1*–*3*), perhaps nowhere more decisively than in modeling. In key scientific areas of great societal importance, models and the software that implement them define both how science is done and what science is done (*4*, *5*). Across all science, this dependence has led to concerns around the need for open access to software (*6*, *7*), centered on the reproducibility of research (*1*, *8*–*10*). From fields such as high-performance computing, we learn key insights and best practices for how to develop, standardize, and implement software (*11*). Open and systematic approaches to the development of software are essential for all sciences. But for many scientists this is not sufficient. We describe problems with the adoption and use of scientific software.

One might assume that two principal scientific considerations drive the adoption of modeling software: the ability to enable the user to ask and answer new scientific questions (*12*) and the ability of others to reproduce the science (*13*). Alas, diffusion innovation theory defies such objectivity, indicating the importance of communication channels, time, and a social system, as well as the innovation itself (*14*). An individual's adoption of an innovation relies on awareness, opinion leaders, early adopters, and subjective perceptions. Scientific considerations of the consequences of adoption generally occur late in the process, if at all.

This may be appropriate when deciding which smartphone application one uses. But we must hold scientific inquiry and adoption of scientific software to higher standards. Does use of modeling software conform to basic tenets of scientific methods? We describe survey findings suggesting that many scientists adopt and use software critical to their research for nonscientific reasons. These reasons are scientifically limiting. This result has potentially wider implications across all disciplines that are dependent upon a computational approach.

Surveying Species Distribution Modelers

We surveyed scientists across a single domain, species distribution modeling (SDM) (*15*) [see supplementary materials for details]. This strategic targeting separates our analysis from previous efforts in important ways, allowing an analysis spanning computational skill sets, while addressing the interplay between models and computation. Our ~400 respondents ranged from those who "find it difficult to use software" to those "very experienced and very technical." Asking people to first identify with a scientific domain and addressing models and software through that lens extended the diversity of respondents and minimized self-selection bias that potentially plagued previous efforts (*16*).

We studied modelers of complex interactions between species and their environment because previous studies have typically chosen atypical groups predefined by computational, and not scientific, expertise [e.g., high-performance computing (*11*)], where the domain leads to self-selection by those comfortable with computational methods. We flip that approach and investigate the diversity of computational expertise within a domain defined by scientific, rather than computational, problems. Although there are limits to the extent that one can generalize from a domain-targeted study [but also the value of case studies (*17*)], SDM modelers likely represent a wider ecological, biological, and/or environmental science community: (i) a range of software packages for modeling is available, (ii) scientists are principally educated as biologists and ecologists, and (iii) scientific challenges are broadly the same. Nonetheless, extending the line of inquiry, we present here more widely through further studies, is clearly desirable [such as in (*18*)].

Our results are intuitive but provide troubling insight at the intersection of scientific pursuits and the adoption of computational methods. Nearly 30% reported that they used particular software because it had been "validated against other methods in peer-review publications." This rose to 57% for those who used "click-and-run" software with easy-to-manipulate user interfaces and dropped to 11% for those who used "syntax-driven" platforms. Further, 7, 9, and 18% of scientists cited "the developer is well-respected," "personal recommendation," and "recommendation from a close colleague," respectively, as reasons for using software. Only 8% claimed they had validated software against other methods as a primary reason for choice; 79% expressed a desire to learn additional software and programming skills.

Many of these scientists rely on the fact that the software has appeared in a peer-reviewed article, recommendations, and personal opinion, as their reason for adopting software. This is scientifically misplaced, as the software code used to conduct the science is not formally peer-reviewed (*6*). This is especially important when a disconnect occurs between equations and algorithms published in peer-reviewed literature and how those are actually implemented in software reportedly used in those papers (*6*, *19*, *20*).

Reliance on personal recommendations and trust is a strategy with risks to science and scientist. "End-user developers" commonly create scientific software (*17*, *21*, *22*), but they are often unaware of or ignore traditional software engineering standards, leaving trust in their coding expertise potentially misplaced (*1*, *2*, *9*). A "well-respected" end-user developer will almost certainly have earned that respect through scientific breakthroughs, perhaps not for their software engineering skills (although agreement on what constitutes "appropriate" scientific software engineering standards is still under debate).

Most people, in some form, "trust" software without knowing everything about how it works. More complex modeling software is a special case, particularly when the answer cannot be checked without the software, and there is thus no ability to validate its output. We have reason for approaching scientific software with healthy circumspection, rather than blind trust.

Given that scientists in general want to learn "enough" to do their science, our finding that an overwhelming majority of scientists wanted increased computational skills suggests something more. Perceived insuf-

[1]Microsoft Research, Cambridge CB1 2FB, UK. [2]Department of Computer Science, University of Oxford, Oxford OX1 3QD, UK. [3]Centre for Ecology and Hydrology, Penicuik, EH26 0QB, UK.

*Corresponding author. E-mail: lujoppa@microsoft.com

Published by AAAS

Selected quotes from survey respondents

"The research question and the data should be king, with an approach being selected on the basis that it is appropriate to both the research question and the data rather than the research question and the data being selected to fit the approach which a person knows how to use."

"I regularly see peer-reviewed articles that apply SDM incorrectly from either a statistical or inferential perspective. This is largely a user problem rather than a software problem as some people treat [SDM Software] as black boxes rather than inferential tools, and thus do not put in the intellectual effort required to do good work."

"We don't need fancier software, we need people who understand ecology and the importance of multiple types of data … The key is the ability to think in ecological terms."

ficient understanding of what the software is doing suggests that users fret over whether it is indeed doing what is expected.

The most popular click-and-run software in our study was specifically designed for SDM, released in 2006 (*23*), and has been cited >1800 times. Confusion around the implementation of the software's algorithms is common (*24*), even though the algorithms have been published in peer-reviewed literature (*23*). An explanation was published aiming to describe the methods from a "viewpoint likely to be more accessible (to ecologists) … than previous ones" (*25*). Clearly, there were many in the SDM domain unable to interpret the original algorithms, much less understand how they were implemented in the distributed code.

Recommendations, Moving Forward

Education: Universities should produce scientists capable of instantiating science in code such that other scientists are able to peer-review code as they would other aspects of science. Formal training in statistics, computational methods, mathematics, and software engineering should be a core part of the science curriculum at undergraduate and research student levels.

The UK Research Council–funded Doctoral Training Centres (DTCs) were designed to provide such a contextualized curriculum. The Life Sciences Interface DTC at the University of Oxford takes students from both physical science (e.g., mathematics, computer science, and engineering) and life science (e.g., biologists, biochemists, and zoologists) backgrounds to produce multidisciplinary natural scientists. Graduates are fluent in biological, mathematical, computer science, and statistical methods and are capable of conversing and collaborating across these disciplines.

Scientific publication: Scientific software code needs to be not only published and made available (*6*, *7*) but also peer-reviewed. That this is not part of the current peer-review model means that papers of which science is primarily software-based (i.e., most modeling papers) are not currently fully or properly peer-reviewed. It also means peer-reviewers need to be able to peer-review the code (i.e., be highly computationally literate). Scientific software code should meet a baseline standard of intelligibility in how it is written (and commented on) (*1*, *2*, *9*). This requirement is analogous to the widely used standard of English in peer-reviewed publications in order to ensure general accessibility of articles. A standard of transparency and intelligibility of code that affords precise, formal replication of an experiment, model simulation, or data analysis, as well as peer-review of scientific software, needs to be a condition of acceptance of any paper using such software.

There are journals providing examples of how this might be done. The journal *IPOL: Image Processing on Line* requires authors to submit source code for peer-review. *Insight Journal* has an emphasis on automated code compilation and testing. The *Journal of Open Research Software* peer-reviews code and publishes concise descriptions of the software. Dealing with citable, peer-reviewed software in this way would relieve some of the burden on the peer-review process at more general journals.

Journals can also educate. The *British Medical Journal*, with many submissions deficient in statistical implementation, initiated a series of tutorial articles (*26*). These cover a wide range of statistical concepts clearly and concisely, giving detailed worked examples and explaining how to describe the results of such studies in a manner that makes it easy for the reader to validate for him- or herself the statistical calculations. Journals publishing research relying on computational science software might publish tutorial papers covering the mathematical and computational underpinnings of key software in their domains, authored by leading authorities.

Changing the status quo will not be easy. Despite the promise of these early efforts, it remains to be seen if they are effective, scalable, and, most important, will be adopted by the broader scientific community. Most scientists, despite an increasing number of programming skills and practices initiatives aimed at scientists (e.g., Software Carpentry, Software Sustainability Institute), continue to emerge from natural science research training without formal training in computational methods and software development and/or engineering. A 2010 survey showed that only 3 of the 20 most highly cited journals required even the most basic step of making source code available upon publication (*7*). Current models for how scientists and journals are rewarded must change, as the would-be editors of the *Open Research Computation* journal (now a series of the journal *Source Code for Biology and Medicine*) discovered during efforts to establish a journal for publishing peer-reviewed software (*27*).

Societally important science relies on models and the software implementing them. The scientific community must ensure that the findings and recommendations put forth based on those models conform to the highest scientific expectation. Learning from efforts such as those noted here, and acting upon their findings, may help transform scientific peer-review and training.

References and Notes

1. S. M. Baxter *et al.*, *PLOS Comput. Biol.* **2**, e87 (2006).
2. G. Wilson, *Am. Sci.* **97**, 360 (2009).
3. Advisory Committee for CyberInfrastructure, National Science Foundation, *Task Force on Software for Science and Engineering Final Report* (NSF, Arlington, VA, 2011).
4. S. Emmott *et al.*, *Towards 2020 Science* (Microsoft Research, Cambridge, 2006).
5. T. Hey *et al.*, *The Fourth Paradigm: Data-Intensive Scientific Discovery* (Microsoft Press, Redmond, WA, 2009).
6. D. C. Ince *et al.*, *Nature* **482**, 485 (2012).
7. A. Morin *et al.*, *Science* **336**, 159 (2012).
8. L. Hatton, A. Roberts, *IEEE Trans. Softw. Eng.* **20**, 785 (1994).
9. D. A. Aruliah *et al.*, arXiv.org, arXiv:1210.0530v1 (2012).
10. C. Drummond, in *ICML '09: Proceedings of the Evaluation Methods for Machine Learning Workshop* (Association for Computing Machinery, New York, 2009), article no. 7.
11. B. R. Basili *et al.*, *IEEE Softw.* **25**, 29 (2008).
12. G. Wilson, *Am. Sci.* **94**, 5 (2006).
13. G. Wilson, *Comput. Sci. Eng.* **10**, 5 (2008).
14. E. M. Rogers, *Diffusion of Innovation* (Free Press, New York, ed. 5, 2003).
15. J. Elith, J. R. Leathwick, *Annu. Rev. Ecol. Evol. Syst.* **40**, 677 (2009).
16. J. E. Hannay *et al.*, in *SECSE '09: Proceedings of the Second International Workshop on Software Engineering for Computational Science and Engineering* (IEEE Computer Society Washington, DC, 2009), pp. 1–8.
17. J. Segal, C. Morris, *J. Organ. End User Comput.* **23**, 51 (2011).
18. R. Sanders, thesis, Queen's University (2008); http://hdl.handle.net/1974/1188.
19. G. Miller, *Science* **314**, 1856 (2006).
20. Z. Merali, *Nature* **467**, 775 (2010).
21. J. Segal, *VL-HCC '07 IEEE Symposium on Visual Languages and Human-Centric Computing* (IEEE Computer Society, Washington, DC, 2007), pp. 111–118.
22. D. F. Kelly, *IEEE Softw.* **24**, 120 (2007).
23. S. J. Phillips *et al.*, *Ecol. Modell.* **190**, 231 (2006).
24. I. W. Renner, D. I. Warton, *Biometrics* **69**, 274 (2013).
25. J. Elith *et al.*, *Divers. Distrib.* **17**, 43 (2011).
26. S. Mallett *et al.*, *BMJ* **345**, (jul02 1), e3999 (2012).
27. C. Neylon *et al.*, *Source Code Biol. Med.* **7**, 2 (2012).

Acknowledgments: The authors thank scientists who participated in the survey on SDM and S. Pimm for comments.

Supplementary Materials

www.sciencemag.org/cgi/content/full/340/6134/814/DC1

10.1126/science.1231535