IDEAS IN ECOLOGY AND EVOLUTION **8: 55–57, 2015**
doi:10.4033/iee.2015.8.9.c

Received 13 May 2015; Accepted 8 June 2015

Commentary

Some thoughts on best publishing practices for scientific software

Ethan P. White

Ethan P. White (ethan@weecology.org), Department of Wildlife Ecology and Conservation, University of Florida, Gainesville, FL 32611-0430 and *Department of Biology, Utah State University, Logan, UT 84322*

It is increasingly recognized that software is central to much of science, and that rigorous approaches to software development are important for making sure that science is based on a solid foundation (Wilson et al. 2014). While there has been increasing discussion of the software development practices that lead to robust scientific software (e.g., Jackson et al. 2011, Osborne et al. 2014, Wilson et al. 2014), figuring out how to actively encourage the use of these practices can be challenging.

Poisot (2015) proposes a set of best practices to be included as part of the review process for software papers. These include automated testing, public test coverage statistics, continuous integration, release of code in citeable ways using DOIs, and documentation (Poisot 2015). These are all important recommendations that will help encourage the use of good practice in the development of scientific software (Jackson et al. 2011, Osborne et al. 2014, Wilson et al. 2014). Requiring these approaches for publication of an associated software paper should help improve the robustness of published software (automated testing, continuous integration), its ease of use (documentation, continuous integration), and the potential for the scientific community to build on and contribute to existing efforts.

As part of thinking about these best practices, Poisot (2015) grapples with one of the fundamental challenges of scientific software publication: how do we review scientific software? Most scientists are not trained in how to conduct code reviews (Petre and Wilson 2014) and the time commitment to do a full review of a moderately sized piece of software is substantial. In combination, this would make it very difficult to find reviewers for software papers if reviewers were expected to perform a thorough code review. Poisot joins Mills (2015) in suggesting that this task could be made more manageable by requiring all software submitted for publication to have automated testing with reasonably high coverage. While Mills (2015) suggests that this will "encourage researchers to use this fundamental technique for ensuring code quality", Poisot takes the idea a step further by suggesting that reviewers could then focus on reviewing the tests to determine if the software does what it is intended to do when provided with known inputs. This approach isn't perfect. Tests are necessarily limited in the inputs that are evaluated and mistakes can occur in tests as well as in the code itself. However, reviewing tests to determine whether they are sufficient and whether the code produces correct outcomes in at least some cases is, I think, much more tenable than reviewing an entire codebase line by line. It is one of the most reasonable solutions I have seen to the challenge of reviewing software.

While I agree with all of the major recommendations made in Poisot (2015), I think the ideas related to making software citeable will benefit from further discussion. While the benefits of citeability for scientific software are clear, it is less clear whether this citation should necessarily occur through the use of DOIs. As Poisot (2015) points out, there are advantages to using DOIs for scientific software. Many journals accept scholarly products with DOIs for inclusion in reference lists, which means that software gets acknowledged in the same way as papers. This helps provide credit to scientists developing software in a manner that is easily understood by academic reward structures. It also has the potential to make bibliometric analyses of software use more straightforward. However, many major scientific software products do not use DOIs for the software itself, preferring generic citations to the software (e.g., SymPy: SymPy Development Team

2014), citation of the associated software paper (e.g., IPython: Pérez and Granger 2007), or citation of the package repository for the language in which the code is written (e.g., rplos: Chamberlain et al. 2015). These practices are justified by the idea that stable URLs or other identifiers can serve the same purpose as DOIs (Boettiger 2013). This is not to say that I disagree with the DOI-based approach; the question is whether a DOI should be considered part of the minimally sufficient set of practices. In general, providing a clear citation for the software itself is important, but I think that the form of that citation should be flexible so that it can best fit the needs and process of the developers.

More importantly, I think the majority of the value in the approach that Poisot (2015) outlines is actually separate from citeability per se. This is because Poisot (2015) suggests using an archiving service (e.g., Zenodo; http://zenodo.org/) for the purpose of making the code citable. Using of an archiving service addresses two additional practices that can help improve the reuse of scientific software: archiving and versioning.

Archiving is crucial to ensuring that the software continues to be available in such a way that other scientific products can rely on its existence (Bergman 2012). Software hosted on personal websites or on code repositories like GitHub, can easily become unavailable, through either neglect or active removal of the software (Schultheiss et al. 2011, Bergman 2012). In contrast, archiving services make a strong commitment to long-term preservation of the software and other research products they store and typically do not allow deletion. This helps to guarantee that the software is available for an extended period of time. In addition to Zenodo, other good options for archiving software include figshare (http://figshare.com/) and Dryad (http://datadryad.org/). Dryad is a particularly interesting example for software papers because it archives the data and software associated with a particular publication.

An additional benefit of using an archiving service that Poisot (2015) mentions is versioning. Most major archiving services include the ability to update the archived object while still keeping the previous versions. Each version has its own unique identifier. Versioning is important for scientific software because it simultaneously allows the software to advance and improve, while also providing a clear record of the exact software used for a particular analysis. This makes science more reproducible, and allows for errors in software and their consequences to be more readily addressed. Versioning can also be handled separately from archiving services. The primary approach to versioning in the software community is through the use tags and/or branches in version control systems that track the entire history of the project. A tag can be applied to this history at any point in time and that tag then clearly refers to that particular state of the code. Code hosting sites also typically support the concept of a “release,” which is a new version of the software. Combining the release of each version with updating the archive, as suggested by Poisot (2015), is an excellent way to support the continued use of the software and the reproducibility of the analyses conducted with it.

The idea of coming up with a set of criteria to use for evaluating software papers is an important step towards valuing good practice in scientific computing and improving the rigor and reproducibility of science. In doing so, it will be important to provide sufficient resources and training to scientific software developers in the use of these best practices. Many individuals writing valuable scientific software are researchers and not professional developers, and it will be important that care is taken to avoid excluding these valuable contributions by setting the bar too high, too fast. By providing training in the necessary skills through efforts like Software Carpentry (http://software-carpentry.org/), and gradually increasing the requirements for publishing scientific software, it should be possible to improve the quality of the code used in science while maintaining contributions from across a broad array of developers and scientists.

Acknowledgements

This publication is funded by the Gordon and Betty Moore Foundation’s Data-Driven Discovery Initiative bbthrough Grant GBMF4563 to Ethan White. My thinking on these matters has been strongly influenced by Titus Brown, Greg Wilson, and many other folks in the Software Carpentry and Data Carpentry communities.

References

Bergman, C. 2012. On the preservation of published bioinformatics code on github. https://caseybergman.wordpress.com/2012/11/08/on-the-preservation-of-published-bioinformatics-code-on-github/

Boettiger, C. 2013. DOI != citable. http://www.carlboettiger.info/2013/06/03/DOI-citable.html

Chamberlain, S., Boettiger, C., and K. Ram. 2015. rplos: Interface to the search aPI for the ’pLoS’ journals. Available online: http://cran.r-project.org/web/packages/rplos/index.html

Jackson, M., S. Crouch, and R. Baxter. 2011. Software evaluation: criteria-based assessment. Software Sustainability Institute, The University of Edinburgh, Available at: http://software.ac.uk/sites/default/files/SSI-SoftwareEvaluationCriteria.pdf.

Mills, B. 2015. Effective code review for journals. http://www.mozillascience.org/effective-code-review-for-journals.

Osborne, J.M., Bernabeu, M.O., Bruna, M., Calderhead, B., Cooper, J., Dalchau, N., et al. 2014. Ten simple rules for effective computational research. PLoS Computational Biology 10:e1003506. CrossRef

Pérez, F., and B.E. Granger. 2007. IPython: a system for interactive scientific computing. Computing in Science and Engineering 9:21–29. CrossRef

Petre, M., and G. Wilson. 2014. Code review for and by scientists. CoRR abs/1407.5648.

Poisot, T. 2015. Best publishing practices to improve user confidence in scientific software. Ideas in Ecology and Evolution 8: 50–54. CrossRef

Schultheiss, S.J., Münch, M-C., Andreeva, G.D., and G. Rätsch. 2011. Persistence and availability of web services in computational biology. PloS One 6: e24914.

SymPy Development Team. 2014. SymPy: Python library for symbolic mathematics.

Wilson, G., Aruliah, D., Brown, C.T., Hong, N.P.C., Davis, M., Guy, R.T., et al. 2014. Best practices for scientific computing. PLoS Biology 12:e1001745. CrossRef