A Community of Practice Around Peer-review for Long-term Research Software Sustainability

Karthik Ram
UC Berkeley / The rOpenSci Project

Carl Boettiger
UC Berkeley / The rOpenSci Project

Scott Chamberlain
UC Berkeley / The rOpenSci Project

Noam Ross
EcoHealth Alliance / The rOpenSci Project

Maëlle Salmon
The rOpenSci Project

Stefanie Butland
The rOpenSci Project

Scientific open source projects are responsible for enabling many of the major advances in modern science including recent breakthroughs such as the Laser Interferometer Gravitational-Wave Observatory (LIGO) project recognized in the 2017 Nobel prize for Physics. However, much of this software ecosystem is developed ad hoc with no regard for sustainable software development practices. This problem is further compounded by the fact that researchers who develop software have little in the way of resources or academic recognition for their efforts. The rOpenSci Project, founded in 2011 with the explicit mission of developing software to support reproducible science, has in recent years undertaken an effort to improve the long-tail of scientific software. In this paper we describe our software peer-review system which brings together the best of traditional academic review with new ideas from industry code review.

INTRODUCTION

Modern science relies very heavily on open source software to acquire, manipulate and analyze large volumes of data. One would be hard pressed to find areas of science today that do not rely on software to generate new discoveries. Such software, often referred to as research software, have become indispensable for progress in most areas of science and engineering. By some estimates, the National Science Foundation has spent close to 10 billion dollars between 1996 and 2006 in support of research proposals that described a software development component (Madhavan et al, 2011). Despite its importance, a large proportion of academic software is developed in an ad hoc manner with little regard for the high standards that are characteristic of other research activities. As a result, the research software ecosystem is fragile and the source of numerous problems that afflict modern computational science. Perhaps the most widely publicized of those problems has been the reproducibility or replication crisis (Baker 2015). Although a multitude of reasons have culminated in different forms of this crisis, some reproducibility challenges have been attributed directly to software and how it is developed in academia. When research software is inaccessible, lacks adequate usage instructions, is difficult to install, suffers from inadequate testing, or is poorly maintained, it can impede efforts to reproduce results generated using such tools. Most often, reproducibility is made impossible when one or more of these issues cause the software to break unexpectedly.

In 2014 and 2015, Christian Collberg and colleagues carried out a series of studies to quantify the state of computational reproducibility in applied computer science, an area that relies more heavily on software than other fields. Their premise was that software described in papers should be readily accessible and contain enough instructions for an informed reader to build from source code on their own systems. Shockingly, less than 20% of the 600+ papers they reviewed had software that could actually be downloaded and built with any reasonable effort. The study did not consider whether that 20% of software functioned correctly or produced valid scientific results. In many ways such a result is not entirely surprising. Scientists spend decades in training to become experts in their chosen domains. Yet critical skills necessary to engineer software are rarely taught.

Despite lack of formal training, a growing number of researchers are now spending their time writing software and code to organize and analyze their data. The free version control software Git and the associated collaboration platform GitHub have been playing an increasingly important role in scientific software over the past several years. Much of the collaboration around such software development now happens on platforms such as GitHub (http://www.nature.com/news/democraticdatabases-science-on-github-1.20719), which is now being taught both in a handful of university courses and by independent programs such as The Carpentries. Such widespread exposure has made the platform and vocabulary familiar to a large group of researchers, making it easier for new tools built in this ecosystem to thrive. Yet the lack of formal training and mentorship contributes to the growing problem of

poorly engineered software that is fragile, difficult to use, and prone to bit rot.

In 2011, a subset of us noticed the need for well developed software that scientists routinely use. These range from simple utilities to produce figures, perform conversions, to more elaborate operations such as retrieving complex data from the web in a repeatable manner. This was the original motivation for the creation of the rOpenSci project (https://ropensci.org). rOpenSci plays a critical role in the scientific software ecosystem, particularly in the R programming language. Our primary mission is to promote development and use of high-quality research software in the scientific community. We enable this transformation in two major ways: In house software development and peer-review of community contributed software.

Our core development team primarily builds robust software to help researchers access, discover, publish and work with disparate types of scientific data. Over the years we have created 287 high quality software packages to support the scientific community (192 of them published on R's primary package manager, CRAN, 2 on Bioconductor https://www.bioconductor.org/, and 93 on GitHub https://github.com/ropensci). Compared to core infrastructure projects that are the poster children of scientific open source such as Jupyter and NumPy, rOpenSci serves to support packages that make up the 'long tail' of science, where software packages have been designed for specialized applications rather than general purpose needs. A complete list of software packages that rOpenSci has reviewed or accepted is available at https://ropensci.org/packages/.

Perhaps rOpenSci's biggest contribution to improving the state of research software is not just the development and maintenance of critical software tools in house, but in mentoring domain scientists in good software development practices and fostering a peer-review culture for research software. In this article we briefly describe our efforts to improve the state of research software by creating a peer-review system that shares many similarities with the publishing system but also addresses challenges that are unique to software development in research.

CHALLENGES WITH RESEARCH SOFTWARE

Documentation: Over the past decade scientists have been embracing various open source programming languages for scientific computing. Python, R, and Julia in particular have fueled the meteoric rise in popularity of data science as a new field. As of this writing there are 147,000 packages in the Python package manager (PyPi), 13,800 packages on CRAN, the Central R Archive Network, and 2400 packages in the Julia language (via libraries.io). Sorting through such large collections of software packaging and finding the right tools presents a serious challenge for most researchers. Beyond the software functionality itself, one of the biggest hurdles to using software is lack of good documentation. Given the tedious nature of the process and the need for different skills than for programming, most packages are documentation poor (Geiger et al, 2018).

Testing: Software plays a very important role in research and is used for critical reasons such as formulating policy, supporting drug discovery and mitigating the effects of climate change. Mistakes in implementation can have catastrophic consequences (Hatton 1997). Scientific software often lacks tests, and some surveys show that ~66% of research software contain unit tests (Kanewala, 2014).

Lack of community: Nearly two thirds of the open source projects on GitHub have only 1-2 maintainers, a very low number that increases the likelihood that projects will go stale (Eghbal 2016). Most scientific software is developed by scientists who rarely possess software engineering skills. The most successful projects are the ones that turn one-person projects into thriving communities.

Long-term archiving: Although software collaboration platforms such as GitHub bring visibility to projects, they are not a solution for long-term archiving. Nearly 30% of the papers surveyed in Collberg 2014 could not be located. Without permanent archiving of source code in persistent repositories such as Zenodo, one cannot ensure availability of software over time.

Software design and code smells: Poorly designed software with inconsistent and unintuitive user interfaces is a problem that cannot be easily surfaced without a detailed review. Deeper design issues include issues such as attention to software dependencies, good error messages and handling of unexpected inputs, and following conventions and coding styles that are characteristic of a community. There are also various other code smells (Fowler 2002) that indicate deeper problems with the software. These include lack of a clear README with installation instructions and basic examples, no contributed code of conduct or clear contribution guidelines, lack of continuous integration (services that allow for the software to be tested any time a change is made to the code) for testing, and large gaps in documentation.

Maintainability: Design considerations that make it easy for future developers to understand the software, extend functionality, and fix bugs are critical to prevent software from becoming stale before it reaches a natural end of cycle.

SOFTWARE REVIEW AS A SERVICE

To combat these issues, we created a peer-review system for software analogous to that for scientific publications (Ross et al, 2017). Since 2015, rOpenSci has been piloting a system of peer code review (called onboarding) for software submissions. This approach brings together best practices for publication peer-review along with new practices that are unique to reviewing software. This system deliberately combines elements of traditional academic peer review (external peers), with practices from open-source software review. Commonalities with traditional publishing workflow include a full editorial board with handling editors, two reviewers per submission and revisions before acceptance. The process differs in a few key ways. The review process is fully open and anyone is welcome to weigh in with constructive feedback. Unlike traditional peer-review where there are only one to two exchanges between authors and reviewers over months, all exchanges in our review happens in real time and dozens of interactions are not uncommon. The forced transparency ensures that interactions are non-

adversarial (see Salmon 2018 for a sentiment analysis of review threads). It is important to emphasize that research software is almost never reviewed and our process not just serves as a quality filter, but also works to elevate and standardize development practices within the research community.

To date we have reviewed 121 software packages, engaged 149 reviewers and our reviews have fast tracked 42 publications into the Journal of Open Source Software (a free, open journal) with 1 publication at Methods in Ecology and Evolution (a journal of Wiley publishing). The rOpenSci onboarding process has also inspired an entirely new journal around research open source: The Journal of Open Source Software (JOSS, http://joss.theoj.org/). The aim of JOSS is to provide recognition for academic open source in the form of brief papers that can be cited and tracked much like academic papers. JOSS also relies on the GitHub issue tracker to operate as a full journal, with some additional automation around the editorial process.

Advantages of rOpenSci's software review process

- Provides opportunities for collaboration and is a rewarding experience for both authors (Albers 2018) and reviewers (McBain 2018, Gray 2018)
- The rOpenSci process provides detailed feedback on software design and implementation, from big-picture best practices (unit testing, continuous integration) to specific line-by-line feedback on code readability and organization.
- rOpenSci helps package authors plan for sustainability by focusing many of our package standards on maintainability. rOpenSci packages are required to have comprehensive testing, continuous integration, community contribution guidelines and are reviewed for features such as code readability and maintenance challenges so as to make ongoing maintenance and contribution for a community easier. By participating in the process, we also introduce authors to a community of practice that often leads to additional programmers contributing to the packages in large and small ways. We also act as a maintainer of last resort. rOpenSci monitors the status of packages, reaching out to authors if they do not respond to failed package tests due to dependency changes and making changes ourselves if necessary.
- rOpenSci combines an open peer-review model (open identities, open reports, open participation, open interaction, open peer-review manuscripts, and open platform as defined by Ross-Hellaeur 2017) with a review transfer model. Although rOpenSci curates packages, it is not a publisher. The editors help authors smoothly navigate submission to various package managers such as CRAN and Bioconductor but also to two journals as part of a pilot. Given that rOpenSci inspired JOSS, since the inception of JOSS it has been possible to have a rOpenSci submission fast tracked through JOSS for a rapid acceptance without further review. We extended this model in November 2017 and established a partnership with the journal Methods in Ecology and Evolution (MEE), a member of Wiley publishing.

MEE accepts software with ecology and evolution applications as 'applications papers'. When the topic of such software overlaps with rOpenSci's interests, authors have the option to submit their software to us for review before submitting a full paper to MEE. rOpenSci's software review transfers over to MEE, and their reviewers are free to focus on the science and rely on our review for the software. This model shows a lot of promise and has potential to extend to various other journals.

- Our standards and expectations also evolve based on feedback from reviewers and authors, such as the adoption of additional guidelines on what makes software most user-friendly (Ross et al, 2018). Current recommendations and review practices are described in a living developer guide (https://ropensci.github.io/dev_guide/). rOpenSci additionally recognizes all reviewers in the developer guide book and also encourages software authors to acknowledge reviewers in the software metadata (Ross 2018).

- rOpenSci has been steadily working towards automating as many software checks as possible, freeing up the reviewer use their limited time on assessing other aspects that cannot be easily automated. One particular software package we rely on heavily is the `goodpractice` package (Csardi and Frick, 2018). The package currently performs 230 checks that cover test suites, source code linting, cyclomatic complexity, and a range of other potential issues.

LIMITATIONS AND CHALLENGES OF SOFTWARE ONBOARDING

- Currently rOpenSci only accepts packages that fit a narrow scope of topics - those that support the data management lifecycle and facilitate computational reproducibility for scientific research. This scope is based on our organization's mission and the expertise of the editor and reviewer base we have developed, and such a scope is necessary due to a finite (though growing) number of volunteers. This leaves a large fraction of packages and authors without a similar peer-review forum. Other forums, such as JOSS, Bioconductor, and the Journal of Statistical Software, have similarly limited scope or less full-featured review processes. In particular, authors of small packages that implement statistical methods lack a place for comprehensive feedback.

- Our approach requires expert reviewers who not only understand algorithms implemented by the software but also details of software engineering. Despite this, the average time required for reviewing is not different from a traditional paper (https://ropensci.org/blog/2018/05/03/onboarding-is-work/)

- As open-source software is typically in a constant state of development, it is challenging to define an appropriate stage for external review. Reviewers are not available to return to

evaluate every change or release. We advise submission when software is fully functional, documented, and tested, but before release to a broad audience through distribution hubs such as CRAN that would likely result in extensive user feedback and additional features. Packages early in development are difficult to review thoroughly but allow reviewers to shape the design and structure of the software. More mature packages are easier to review but addressing design flaws at this stage requires considerable refactoring effort and can come at a cost of breaking APIs for existing users. A survey of our authors suggests only a thin majority of authors prefer submitted fully developed packages over one in earlier development (https://ropensci.org/blog/2018/04/17/author-survey/). We have also experimented with re-reviews of software following major refactoring and re-release.

CHANGING ACADEMIA'S INCENTIVE STRUCTURES

The results of scientific endeavors are primarily communicated as peer-reviewed scientific publications. The citation impact of such products is captured using metrics such as the h-index and form the basis for hiring, promotion, and tenure. The changing nature of science means that researchers now produce many more valuable outputs than just papers, one of which is software. While software research products often have a greater overall impact than publications, both the reward and review systems have failed to keep up. Scientists have little incentive to release high quality software, and although more and more researchers are releasing their software in some form, the lack of formal review mechanisms is contributing to the fragility of the system.

More recently several journals have emerged as venues for software papers. These include JOSS, Journal of Open Research Software (JORS) and Software X. By providing traditional citations, these venues allow researchers to obtain credit in more traditional forms. For more on various publishing options, please see Smith et al in this special issue.

LOOKING TOWARDS A BRIGHTER FUTURE FOR ALL SCHOLARLY PRODUCTS, ESPECIALLY SOFTWARE

Scientists routinely produce a wide range of highly valuable outputs as part of their research activities. These often include datasets and software, both of which get far less recognition than peer-reviewed publications. Often these side effects can have a disproportionately greater impact on the advancement of science than the publications themselves. The open science movement gained momentum around 2010 and has been successful in drawing attention to overall dysfunction of the academic credit system. Even federal funding agencies like the National Science Foundation announced in 2012 that they allow users to include not just publications but other products that emerge from research endeavors

such as data and code ("US NSF - Dear Colleague Letter - Issuance of a new NSF Proposal & Award Policies and Procedures Guide (NSF13004)" 2012). Despite these positive changes, software development practices in academia still lag far behind that of industry. There have been various attempts at capturing software metrics (e.g. Depsy, Priem 2016). Although these have not panned out, there are various new initiatives aimed at solving the software citation problem. How should software be cited? When should software be cited? What infrastructure is necessary to make this possible? A FORCE11 task force laid out a consensus on these basic questions (Smith et al, 2016), emphasizing the importance of citing specific versions, the role of software papers, and the need for permanent identifiers such as DOIs to persistently identify software products. Subsequent initiatives such as the CodeMeta project (Boettiger et al, 2017) focus on tooling required to make software metadata operate within the existing publication metadata ecosystem. Another recent effort that is seeking to improve credit around software is the US Research Software Sustainability Institute (http://urssi.us/) which is in early stages of planning. Efforts like the rOpenSci software onboarding review not only foster community discussion about these practices but put these ideas to the test through an active and evolving process across a broad range of software applications. Despite the challenges, these examples are steadily changing the culture around software. It remains to be seen if these practices can gain more adoption across more of science.

Acknowledgements

This publication was supported in part by The Leona M. and Harry B. Helmsley Charitable Trust under award number 2016PG-BRI004 and by NSF award #1743188.

REFERENCES

Albers, S. 2018. 5 Things I Learned Making a Package to Work with Hydrometric Data in R. https://ropensci.org/blog/2018/01/16/tidyhydat/

Boettiger, C. Matthew B. Jones, Abby Cabunoc Mayes, Arfon Smith, Peter Slaughter, Kyle Niemeyer, Yolanda Gil, Martin Fenner, Krzysztof Nowak, Mark Hahnel, Luke Coy, Alice Allen, Mercè Crosas, Ashley Sands, Neil Chue Hong, Patricia Cruse, Daniel S. Katz, Carole Goble. 2017 The CodeMeta Project, https://codemeta.github.io

Noam Ross Scott Chamberlain Karthik Ram Maëlle Salmon. How rOpenSci uses Code Review to Promote Reproducible Science https://ropensci.org/blog/2017/09/01/nf-softwarereview/

Baker M. 2016. 1, 500 scientists lift the lid on reproducibility. Nature 533(7604):452-454

Eghbal, N. (2016). Roads and Bridges: The Unseen Labor Behind Our Digital Infrastructure. Retrieved from the Ford Foundation website:

http://www.fordfoundation.org/library/reports-and-studies/roads-and-bridges-the-unseen-labor-behind-our-digital-infrastructure

Geiger, R. S., Varoquaux, N., Mazel-Cabasse, C., & Holdgraf, C. (2018). The Types, Roles, and Practices of Documentation in Data Analytics Open Source Software Libraries. Computer Supported Cooperative Work (CSCW), 27(3–6), 767–802. https://doi.org/10.1007/s10606-018-9333-1

Gabor Csardi and Hannah Frick (2018). goodpractice: Advice on R Package Building. R package version 1.0.2. https://CRAN.R-project.org/package=goodpractice

Gray, C. 2018 An Ode to Testing, my first review. https://ropensci.org/blog/2018/03/13/ode-to-testing/

Hatton L. The T experiments: errors in scientific software. Computational Science & Engineering, IEEE. 1997 Apr-Jun;4(2):27–38

Kanewala, U., & Bieman, J. M. (2014). Testing scientific software: A systematic literature review. Information and Software Technology, 56(10), 1219–1232. https://doi.org/10.1016/j.infsof.2014.05.006

Fowler, M. (2002). Refactoring: Improving the Design of Existing Code. In Extreme Programming and Agile Methods — XP/Agile Universe 2002 (pp. 256–256). Springer Berlin Heidelberg. https://doi.org/10.1007/3-540-45672-4_31

McBain, M. Where is the value in package peer review? https://ropensci.org/blog/2018/04/06/peer-review-value/

Madhavan, K., McKenna, A.F., Johri, A., Sheppard, S.: Collaborative Research: Deep Insights Anytime, Anywhere (DIA2)-Central Resource for Characterizing the TUES Portfolio through Interactive Knowledge Mining and Visualizations. Proposal awarded by the National Science Foundation (2011)

Priem, J. 2016. Collaborating on a $635k grant to improve credit for research software. http://blog.impactstory.org/collaborating-635k-grant-improve-credit-research-software/

Ross, N. 2018. Thanking Your Reviewers: Gratitude through Semantic Metadata. https://ropensci.org/blog/2018/03/16/thanking-reviewers-in-metadata/

Ross-Hellauer T. What is open peer review? A systematic review [version 2; referees: 4 approved]. F1000Research 2017, 6:588 (doi: 10.12688/f1000research.11369.2)

Salmon, M. 2018 The social weather of rOpenSci onboarding system https://ropensci.org/blog/2018/05/10/onboarding-social-weather/

Smith AM, Katz DS, Niemeyer KE, FORCE11 Software Citation Working Group. (2016) Software Citation Principles. PeerJ Computer Science 2:e86. DOI: 10.7717/peerj-cs.86

US NSF - Dear Colleague Letter - Issuance of a new NSF Proposal & Award Policies and Procedures Guide (NSF13004). (2012). US NSF - Dear Colleague Letter - Issuance of a new NSF Proposal & Award Policies and Procedures Guide (NSF13004).

ABOUT THE AUTHORS

Karthik Ram is a data science fellow at the Berkeley Institute for Data Science, research faculty at the Berkeley Museum of Paleontology at UC Berkeley and co-founder of the rOpenSci project. Contact him at karthik.ram@berkeley.edu

Carl Boettiger is an assistant professor in the department of Environmental Science, Policy, and Management at UC Berkeley and a member of the rOpenSci leadership team. Contact him at cboettig@berkeley.edu

Scott Chamberlain is a co-founder of and software developer with rOpenSci.

Maëlle Salmon is a volunteer editor, and a part-time software engineer at rOpenSci. Contact her at https://masalmon.eu/

Noam Ross is a senior scientist at EcoHealth Alliance and an editor at rOpenSci.

Stefanie Butland is the community manager at rOpenSci. Contact her at stefanie@ropensci.org