Lightning talk: A model for peer review and onboarding research software

Karthik Ram
Berkeley Institute of Data Science
UC Berkeley
Berkeley, CA
Karthik.ram@berkeley.edu

Noam Ross
EcoHealth Alliance
New York, NY
noamross@gmail.com

Scott Chamberlain
The rOpenSci project
Berkeley, CA
scott@ropensci.org

***Abstract*— Code review, in which peers manually inspect the source code of software written by others, is widely recognized as one of the best tools for finding bugs in software. Code review is relatively uncommon in scientific software development, though. Scientists, despite being familiar with the process of peer review, often have little exposure to code review due to lack of training and historically little incentive to share the source code from their research. So scientific code, from one-off scripts to reusable R packages, is rarely subject to review.**

Most R packages are subject only to the automated checks required by CRAN, which primarily ensure that packages can be installed on multiple systems. As such, the burden is on software users to discern well-written and efficient packages from poorly written ones.

rOpenSci is a community of developer-scientists, creating R packages for other scientists, and our package contributors have a mix of backgrounds. We aim to serve our users with high-quality software, and also promote best practices among our author base and in the scientific community in general. So for the past year, rOpenSci has been piloting system of peer code review for submissions to our suite of R packages (https://ropensci.org/packages/). In this paper, we outline how our system works, and what we've learned from our authors and reviewers.

Our System

rOpenSci's package review process owes much to the experiments of others (such as Marian Petre (http://mcs.open.ac.uk/mp8/) and the Mozilla Science Lab (https://mozillascience.org/code-review-for-science-what-we-learned)), as well as the active feedback (https://discuss.ropensci.org/t/code-review-onboarding-milestones/180) from our community (https://discuss.ropensci.org/t/how-could-the-onboarding-package-review-process-be-even-better/302).

Here's how it works: When an author submits a package, our editors evaluate it for fit according to our criteria (https://github.com/ropensci/policies#package-fit), then assign reviewers who evaluate the package for usability, quality, and style based on our guidelines (https://github.com/ropensci/packaging_guide#ropensci-packaging-guide).

After the reviewers evaluate and the author makes recommended changes, the package gets the rOpenSci stamp in its README and is added to our collection.

We work entirely through the GitHub issue system. To submit authors open an issue (https://github.com/ropensci/onboarding/issues/new). Reviewers post reviews as comments on that issue. This means the entire process is open and public from the start. Reviewers and authors are known to each other and free to communicate directly in the issue thread. GitHub-based reviews have some other nice features: reviewers can publicly consult others by tagging if outside expertise is wanted. Reviewers can also contribute to the package directly via a pull request when this is more efficient than describing the changes they suggest.

This system deliberately combines elements of traditional academic peer review (external peers), with practices from open-source software review. One design goal was to keep reviews **non-adversarial** - to focus on improving software quality rather than judging the package or authors. We think the openness of the process has something to do with this, as reviews are public and this incentivizes reviewers to do good work and abide by our code of conduct (https://github.com/ropensci/policies#code-of-conduct). We also do not explicitly reject packages, except for turning some away prior review when they are out-of-scope. We do this because submitted packages are already public and open-source, so "time to publication" has not been a concern. Packages that require significant revisions can just remain on

hold until authors incorporate such changes and update the discussion thread.

.

I. Some lessons learned

So far, we've received 16 packages. Of these, only 1 was rejected due to lack of fit. 11 were reviewed, 6 of which were accepted, and 5 are awaiting changes requested by reviewers. 4 are still awaiting at least one review.

We also recently surveyed (https://docs.google.com/spreadsheets/d/1zaE5MvqXyD0I7LWONh1HlQu98wTIZ6Uls4QVmKs2u-w/edit?usp=sharing) our reviewers and reviewees, asking them how long it took to review, their positive and negative experiences with the system, and what they learned from the process.

Reviewers and reviewees like it!

Pretty much everyone who responded to the survey, which was most of our reviewers, found value in the system. While we didn't ask anyone to rate the system or quantify their satisfaction, the length of answers to "What was the best thing about the software review process?" and the number of superlatives and exclamation marks indicates a fair bit of enthusiasm. Here are a couple of choice quotes:

"I don't really see myself writing another serious package without having it go through code review."

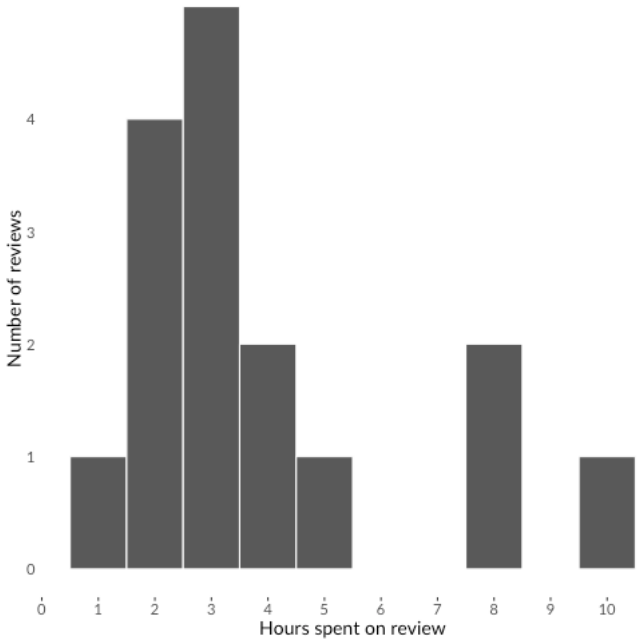
"I learnt that code review is the best thing that can ever happen to your package!"

Authors appreciated that their reviews were thorough, that they were able to converse with (nice) reviewers, and that they picked up best practices from other experienced authors. Reviewers also praised the ability to converse directly with author, expand their community of colleagues and learn about new and best practices from other authors.

Interestingly, no one mentioned the credential of an rOpenSci "badge" as a positive aspect of review. While the badge may be a motivating factor, it seems from the responses that authors primarily value the feedback itself. There has been some argument (http://simplystatistics.org/2013/09/26/how-could-code-review-discourage-code-disclosure-reviewers-with-motivation/) whether code review will encourage or discourage scientists to publish their code. While our package authors represent a specific subset of scientists - those knowledgeable and motivated enough to create and disseminate packages - we think our pilot shows that a well-designed review process can be encouraging.

II. Review takes a lot of time

We asked reviewers to estimate how much time each review took, and here's what they reported:



Answers varied from 1-10 hours with an average of 4. This is comparable to how long it takes researchers to review scholarly papers (http://publishingresearchconsortium.com/index.php/112-prc-projects/research-reports/peer-review-in-scholarly-journals-research-report/142-peer-review-in-scholarly-journals-perspective-of-the-scholarly-community-an-international-study), but it's still a lot of time, and does not include further time corresponding with the authors or re-reviewing an updated package.

Package writing and reviewing are generally volunteer activities, and as one respondent put it, the process "still feels more like community service than a professional obligation." 7 of 16 reviewers respondents mentioned the time to it took to review and respond as a negative of the process. For this process to be sustainable, we have to figure out how to limit the burden on our reviewers.

We can be clearer about the beginning and end of the process

"It wasn't immediately clear what to do"

A few respondents pointed out we could be better at explaining the review process, both in how to get started and how it is supposed to wrap up. For the former, we've recently updated our reviewer guide (https://github.com/ropensci/onboarding/wiki/For-Reviewers), including adding links to previous reviews. We hope as our reviewer pool gets more experienced, and as software reviews become more common, this gets easier. However, as our pool of editors and reviewers grows, we'll need to ensure that our communication is clear.

As for the **end** of the review, this can be an area of considerable ambiguity. There's a clear endpoint when a package is accepted, but with no "rejections" some reviewers weren't sure how to respond if authors didn't follow up on their comments. We realize it can be demotivating to reviewers if their suggestions aren't acted upon. (One reviewer pointed out that seeing her suggestions implemented as a positive motivator.) It may be worthwhile to enforce a deadline for package authors to respond.

We are helping drive best practices with our author base

"I had never heard of continuous integration, and it is fantastic!"

We asked both reviewers and reviewees to tell us what they learned. While there was a lot of variety in the responses, one common thread was learning and appreciating best practices: continuous integration, documentation, "the right way to do X", were the common responses.

Importantly, a number of reviewers and reviewees commented that they *learned the value of review* through this process.

III. Questions and Ideas for the Future.

Scaling and reviewer incentives: Like academic paper review or contributing to free open-source projects, our package review is a volunteer activity. How do we build an experienced reviewer base, maintain enthusiasm, and avoid overburdening our reviewers? We will need to expand our reviewer pool in order to spread the load. As such, we are moving to a system of multiple "handling editors" to assign and keep up with reviews. Hopefully we will be able to bring in more reviewers through their networks.

Author incentives: Our small pool of early adopters indicated that they valued the review process itself, but will this be enough incentive to draw more package authors to do the extra work it takes? An area to explore is finding ways to help package authors gain greater visibility and credit for their work after their packages pass review. This could take the form of "badges", such as those being developed by The Center for Open Science (https://osf.io/tvyxz/wiki/home/) and Mozilla Science Lab (https://www.mozillascience.org/projects/contributorship-badges), or providing an easier route to publishing software papers.

Automation: How can we automate more parts of the review process so as to get more value out of reviewer and reviewees time? One suggestion has been to submit packages as pull requests (https://discuss.ropensci.org/t/how-%20could-the-onboarding-package-review-process-be-even-better/302/3) to take more advantage of GitHub review features such as in-line commenting. This may allow us to move the burden of setting up continuous integration and testing away from the authors and onto our own pipeline, and allow us to add rOpenSci-specific tests. We've also started using automated reminders](https://github.com/ropenscilabs/heythere) to keep up with reviewers, which reduces the burden on our editors to keep up with everyone.

We have learned a ton from this experiment and look forward to making review better! Many, many thanks to the authors who have contributed to the rOpenSci package ecosystem and the reviewers who have lent their time to this project.

Acknowledgments

We have learned a ton from this experiment and look forward to making review better! Many, many thanks to the authors who have contributed to the rOpenSci package ecosystem and the reviewers who have lent their time to this project.