Why we should review research software

Konrad Hinsen

Published April 11, 2025

Citation

Hinsen, K. (2025, April 11). Why we should review research software. *Konrad Hinsen's Blog*. https://doi.org/10.59350/6pbdw-v8d04

At the recent SciCodes Symposium, I brought up the question of reviewing research software during the panel discussion. One panelist then raised the question of *why* we should review research software. I found this question surprising at first, but I do agree that it deserves an answer. Here is mine.

My goal with bringing up the question was to learn out the state of the art: which institutions, publishers but possibly also others, encourage, enforce, or conduct scientific or technical reviews of research software? The answer turned out to be "very few". There are evaluations of research software, but they are mostly restricted to checking if it is published, if it can be compiled, if it reproduces published results, or if it follows Open Source best practices. The question of whether it does what it claims to do, and whether what it does is scientifically relevant, is rarely asked to reviewers.

Why is this state of the art deplorable? I see two reasons for reviewing research software:

1. For the same reason that we review papers: to detect mistakes, biases, and tacit assumptions that should be made explicit.

2. In order to incentivize software authors to write their software in such a way that is is understandable by an outsider to the development process.

It's the first reason that I had considered obvious, but apparently it isn't. An important aspect of the scientific method is peer criticism, i.e. the critical inspection of everyone's work by their competent peers. This doesn't strictly require a formal reviewing process. The only strict requirement is to make all material available for inspection. However, formal reviewing processes have proven valuable in many fields of science and engineering. The label "peer reviewed" has justly been considered a reason to trust some publications more than others. Peer review has been criticized a lot recently, and I agree that the *specific* processes that we have used since the 1950s to review publications is no longer adequate, but that means that we should update those processes, not abolish them.

In today's state of affairs, if I put an obviously unreasonable assumption into a paper, there's a good chance that it will not pass the peer review process of the journal that I submit it to. But if I make that same unreasonable assumption in software source code, it is highly probable that nobody will ever notice it. The discovery of such a case in 1997 was actually one of the major events that got me doubting about the level of rigor in computational science. I was re-implementing from scratch a popular model for protein energetics, the Amber force field. In the course of exploring how this model was actually defined in detail, I discovered that the value it assigned to certain atomic interaction energies depended on the order in which the atoms appeared in an input file. That's physically completely unreasonable, and no reviewer would have let it pass in a paper. But expressed in Fortran and well hidden in unpublished source code, its users remained blissfully ignorant of this anti-feature. Yes, in Open Source software there would have been a slightly higher chance of early discovery. But unless an institution explicitly asks some experts to proof-read the source code, it can easily take decades before

somebody directs a critical look at the right place. And unless the result of such an expert review is made easily findable, it won't have an impact on how software actually gets used.

The second reason is more subtle but perhaps even more important in the long run. If research software has a good chance of being reviewed, research software developers have to write their code and documentation in such a way that it is actually reviewable, i.e. understandable by someone else. And that would also make it more understandable to its users. Today's state of the art in software understandability in the small is much better than it was when I started in computational science, 30 years ago. At that time, most code was written by a single person with no training in software engineering. It was still considered good practice to use single-letter variable names in order to save compile time. Today, most software is developed by teams, and that requires a higher level of cross-individual comprehensibility. But software complexity has also increased in those 30 years. My personal feeling is that the net effect is software being *less* understandable today than it was 30 years ago, but I can't back that up with any hard evidence. In any case, reviewing should lead to a clear improvement.

Are there any reasons *not* to review research software? I can think of only one: it takes time and effort. A lot of time and effort initially, to come up with suitable reviewing processes and appropriate tooling support. Less but lasting effort thereafter, to apply these processes in practice. But then, doing research with unreliable software tools also consumes time and effort, if quality control is done at a later stage and research projects have to be repeated after fixing software bugs. And since some mistakes will never be identified, research should overall be less reliable. You could expect something like a replicability crisis. Is that what we want to happen?