Assuring positive value for open-source software

Author:
Thomas J. Loredo loredo@astro.cornell.edu
Senior Research Associate

620 Space Sciences Building
Cornell Center for Astrophysics and Planetary Science
Cornell University
Ithaca, NY 14853-6801

Short summary

Shared software can have **negative value** for the community of potential users. Producing software with positive shared value requires significant expertise and effort. New policies advocating or implementing code sharing requirements must consider the costs associated with making shared code genuinely useful.

The main point

Merely sharing a piece of research software does not make that software useful. Shared software can have **negative value** for the community of potential users. Software that is inadequately documented, poorly annotated (by in-source docstrings and comments), badly organized, poorly written, or inadequately validated can be unusable or have little use beyond trivial validation of published figures. Such software can waste time and other resources of potential users and reviewers. Any OSS policy NASA puts in place should **ensure that shared code provides significant positive scientific value** to the scientific community and/or to NASA itself. By "significant" I mean as measured, not only by its potential scientific value, but also considering the costs and burdens associated with producing and maintaining useful shared code. The policy must recognize that **producing useful open-source code is a burden above and beyond what has typically been required of NASA-funded researchers**. Also, it requires skills that are often not provided in the training of astronomers. Grant programs requiring code sharing will need to adjust their funding and publication expectations and allocations based on realistic assessments of the cost and value of shared software, across a range of project types and scales. Some nontrivial objective research on software practice and utility in astronomy is likely necessary to guide sound open-source software policy.

Justification

My vantage point

I began my undergrad career as an EE/CS major, and took a year of programming courses (LISP, PL/1, Algol; procedural, functional, and OOP paradigms) before switching to physics for my undergrad degree.

This by no means qualifies me as a software engineer, but I believe this background, and continued interest in programming, provides me a bit more of an engineer's perspective to scientific software development than many of my colleagues. My PhD is in Astronomy & Astrophysics; my research focuses on astrostatistics. I have a research appointment in astronomy and a teaching/advising appointment in statistics, where my teaching has included lab-based instruction in computational statistics.

I write from this background, and from the perspective of:

- A user of open-source software;
- A provider of open-source software;
- An associate editor of a high-impact applied statistics journal (*Annals of Applied Statistics*, AOAS) that has recently begun requiring software with submitted publications, in the name of scientific reproducibility.

Anecdotes & experience with shared software

I offer three anecdotes (among others) informing the viewpoint presented here, each providing a different lesson.

Many codes, none useful: A few years ago, amidst work on an image processing project, I needed a reasonably fast and flexible implementation of a standard image interpolation algorithm. I estimated it would take me half a day to a day to implement it as a C extension for Python. But such interpolation is frequently needed, and I figured I could save that time by using an open-source implementation. A Google search identified a handful of candidate implementations. But after most of a day of frustrating experimention with them, I ended up writing my own implementation. None of the shared implementations was readily useable, due either to insufficient or inconsistent documentation, peculiar and ambiguous conventions, or other shortcomings. None of the implementations had test cases (e.g., unit tests) or simple demo code.

Sharing code not specifically written (or revised) for use by third parties causes researchers to waste time attempting to decipher it, or using it in a manner eventually found to be incorrect.

Bad one-off code by a good programmer: I recently downloaded code linked from a highly cited paper on exoplanet demographics, hoping to explore some modifications of the model and algorithm described in the paper. The published code comprises one of the worst examples of shared software I've encountered. It is essentially undocumented. There are very few comments in the code; of the few comments, some are wryly humorous remarks (in one case about how obfuscated the code is). There is no evidence in the shared code base of any significant testing. An inefficient algorithm was implemented, taking an hour or more to compute one of the published figures; an obvious and simpler alternative algorithm is at least an order of magnitude faster.

The irony of this example is that the developer (an astronomer) is widely regarded as an unusually talented programmer. This reputation is in fact justified by *other* projects by this developer—those of broader use than for a single paper, written specifically for community use, and with a plan for ongoing maintenance. A point of this anecdote is that even an astronomer with significant engineering talent can produce negative-shared-value software, if the software is not specifically written or revised for reuse by others. My own one-

off codes for various projects would fall in this negative-shared-value category were I to share it without significant investment of effort to make it useful to others.

Well-meaning policy gone awry: For a research proposal I recently prepared, I needed a modest-sized database of galaxy photo-z estimates. I had just learned that a number of major statistical journals had started requiring authors to provide code and data for publications with papers. I knew of a recent photo-z paper with plots showing just the kind of examples I needed, so I downloaded the code. There was no README file or other overall documentation in the main directory. There were dozens of code files across multiple folders and subfolders, with only one README file in one of those folders (merely showing docstrings of a few R functions). The code had hardly any comments. Most surprisingly, there were no data files. I presumed some of the code must have included queries obtaining data from public databases. An hour or so of exploration of the undocumented code base turned up nothing obvious. I contacted an author, who apologized for the oversight and promised that a coauthor (the lead developer) would fix the problem with the journal editor shortly. Four months later, the published code is unchanged. I eventually got the data I needed by doing my own queries of a public catalog database (thankfully, well documented!), but I'd lost hours of my time because of shared software that was not suitable for sharing.

This is a troubling example because the code was provided to satisfy a new journal policy, motivated by extended deliberations by the American Statistical Association (ASA) on how to improve reproducibility of statistical research. However well-motivated the policy, its implementation clearly is inadequate, at least for that journal (AOAS). What is particularly troubling, from my perspective, is that (1) I am an associate editor for this journal, and was never informed of the policy; and (2) as of this writing, the journal's instructions to reviewers say nothing about software requirements or testing. What was overlooked here is that merely requiring software to be provided, and publishing whatever is provided, may not add value to a publication, and in fact can very easily add *negative value* to the publication.

My overall experience with shared software may be summarized thusly:

- I have benefitted enormously from large, well-organized, ongoing open-source projects (e.g., Linux, Python, the PyData stack).
- I have benefitted significantly from modest-sized team software projects, e.g., on the scale of software being developed by LSST Science Collaborations.
- I have benefitted positively from small projects, where a single author, perhaps aided by a few contributors, has committed to maintaining a focused package for community use.
- I have yet to encounter publication-specific one-off software (other than for publications describing maintained packages) that has been worth the time I've spent trying to use or understand it.

Implications

A naive line of argument may contend that since NASA-funded projects produce code funded by the public, that the resulting code should be made publicly available as a matter of principle. But merely sharing code does not make it useful, and can do more harm than good.

Shared software provides significant positive scientific value to 3rd-party users only when significant effort is invested in making the code useful to scientists other than the developer(s). Policy deliberations that

consider requiring code sharing must consider the costs to the community (in dollars, time, or other resources) of sharing inadequate code, or of the work required to make code truly useful.

I am open-minded about there being net positive value to a wide-ranging code sharing requirement, provided that it recognizes the costs. These may include fewer publications per unit time or dollar, as effort gets reallocated to support sound code sharing. It may well be the case that, over the long run, this path produces better science (e.g., if the published work is of higher quality as a result of better programming practice, and if shared code is significantly reusable, accelerating future related research). But a concrete case needs to made for this, accompanied by explicit adjustments of expectations and funding allocations.

Alternatively, with current constraints, expectations, and practices, it may be best to have more modest goals, e.g., to require proposers to identify specific methods or algorithms from their research likely to be of broad interest, and to require that such methods be provided in implementations that meet some kind of standard. E.g., a revised publication review process, which includes code review and testing, could enable publication itself to provide such a standard. But this will require buy-in from the broader community; it could not be unilaterally imposed by a funding agency.

A broader underlying issue is whether most astronomers are capable of writing good shared code. Both my research and teaching experience lead me to doubt that this is true, but that's a topic deserving separate discussion.

Speaking more generally, open-source code needs to be recognized as important **science-enabling technology** (like instrumentation), with funding channels that recognize the technological aspects of this potential community resource. E.g., there should be funding channels enabling *maintenance and updating* of software that NASA has invested in creating. Such a program or program element would likely resemble APRA more than a science-focused program like ADAP or ATP or the other domain- or mission-specific ROSES research and analysis programs.

This may well be a topic better handled at the level of a decadal survey or a multi-agency study, than by a single funding agency.

Appendix: American Statistical Association policy

The resources below address code sharing for *statistical computation*, which may be just one component of an astronomical computation, perhaps more amenable to "reproducible research" requirements than other components. There is a broad literature on this; I point to these documents because they represent outcomes of ASA society-level deliberations. The first item is specifically addressed to funding agencies.

ASA Develops Reproducible Research Recommendations: "[T]he ASA has developed a guidance document that provides funding agencies with suggestions for how they can help support reproducible research."

Reproducible Research in JASA | Amstat News: *JASA* (Journal of the ASA) is a premier US applied statistics journal. This web site describes *JASA*'s current code publication policy. It established a new editorial role—associate editor for reproducibility (AER)—to manage code sharing for *JASA*. This highlights a nontrivial added burden for managing code sharing with publications.

Trust Your Science? Open Your Data and Code | AmStat News (AmStat = ASA): Makes the case for sharing statistical code with publications, citing the consensus document from a multidisciplinary roundtable including astronomer Alyssa Goodman (CfA). The document highlights a variety of costs/burdens associated with useful code sharing.