Journal of
open research software

Blanton, B and Lenhardt, C 2014 A Scientist's Perspective on Sustainable Scientific Software. *Journal of Open Research Software,* 2(1): e17, pp. 1-4, DOI: http://dx.doi.org/10.5334/jors.ba

ISSUES IN RESEARCH SOFTWARE

A Scientist's Perspective on Sustainable Scientific Software

Brian Blanton* and Chris Lenhardt*

Software underpins most of our daily activities, from banking and finance to interactions with the internet, to weather forecasts and reports. Software also impacts individuals, groups, and societies through policy implementation, since information for decision and policy making is frequently derived from software ranging from climate and weather models to financial forecasting systems. One way to gauge the extent to which specific software needs to be sustainable, accessible, and transparent essentially hinges on the degree to which scientific analysis software, models, and model output are used to help inform and guide policy. Climate models and related *scientific* results are perhaps the most obvious example of the need for sustainable and transparent software, due in part to the public forum in which the results are scrutinized and the implications on environmental management policy. Without almost ubiquitous adoption of best practices for scientific software development, maintenance, and use, the credibility of scientific results and of ourselves as scientists is substantially at risk; sustainable and transparent research processes are thus at the heart of maintaining and increasing our collective reputations. [The authors want to make clear that, by using climate models as an example of software with policy impacts, we are *not* claiming that these models, are written with little "best practices" in mind, nor that they are inherently unsustainable as software.]

Keywords: scientific software; credibility; credibility risk

Software is at the core of most modern scientific activities. As societal awareness of, and impacts from, extreme weather, disasters, and climate and global change continue to increase, scientific software is put more in the spotlight because it is often used to understand, analyze, and predict these types of phenomena. Some of this scientist-written research software has indirect impacts on decision- and policy-making, and so reproducibility of research results becomes an essential component to establishing and maintaining *credibility* of both scientists and scientific results. This is referred to as a *crisis of credibility* by Donoho [1], Stodden [2], and others. For results to be independently reproducible, the software (and data) should meet certain levels of best practices that help ensure sustainable and open access to the software, data, and results.

One aspect of the credibility crisis has been highlighted in a recent article [3] that describes reasons for particular software being chosen by scientists. These reasons include that the "developer is well-respected" and on "recommendation from a close colleague". This taking of software for granted, assuming that it performs as advertised and that the software itself has been validated and results verified, is one of several big *hazards* facing the entire scientific community (others include data, methods, and results openness and transparency, and competition of scarce resources). Failure to adequately understand and address this hazard and its potential consequences puts our collective scientific credibility at substantial risk.

It is inevitable that scientific software will frequently be taken for granted, requiring that some level of credibility risk being taken on. Not all scientists have the same level of expertise in software development, computational sciences, and other related fields. This is hardly a disparagement. As pointed out by Hanney et al [4], a fundamental difference between science software development and other software development enterprises is that developers of science software generally need some level of knowledge of the science domain. Degrees are granted in the fields of computational sciences and software engineering, and it may be unreasonable for software engineers to become proficient in multiple scientific domains throughout a career in software development. In general, it is easier for scientists to acquire the basics of writing functioning software (whether or not they follow best practices) than for software engineers to become adept in one or more scientific domain. This is partly why so many scientists write their own computer

* Renaissance Computing Institute, The University of North Carolina at Chapel Hill, Chapel Hill, NC, USA
brian_blanton@renci.org

Corresponding author: Brian Blanton

programs. This is not a direct risk avoidance measure, but rather a reaction to necessity. It likely results in a *perceived* reduction in risk.

The issue of taking scientific software for granted essentially becomes, "How should this risk be managed and mitigated?" Hedging against this risk has several aspects, most related to software development itself. In some rough order of increasing complexity, these range from adoption and adherence of software development best practices, to peer-review of software, to formal training in software engineering and related fields for scientists. Ultimately, some balanced approach is needed that incorporates parts of the entire range of approaches, and the goal should be to arm scientists with reasonable best-practices and provide opportunities for collaborations with professional software developers.

One software development path is as follows. Scientific software might begin as a component of a research idea funded and developed in response to an RFP or similar request. The software is a by-product of the research efforts, and since the research end-goal is delivery of "science", the software may be written in a *get it done* sense. Some software emerges from this cookery as having substantive value for other applications and research; continued development of the software then typically occurs in sporadic fits and with multiple developers (including students).

Occasionally, software developed largely in the scientific and engineering research realm gains acceptance and applicability in different sectors like applied research, operations, or commercial and industry use. This happens neither quickly nor by coincidence. Years of model/software application establish a track record of success, and (presumably) studies conducted with the model demonstrate good predictive skill and consequently good representations of the underlying physical laws and science (such as the Navier Stokes or shallow-water equations for hydrodynamic models). While not directly peer-reviewed as a software construct, the results have been reasonably well vetted in the community through the publication and peer-review process.

Consider the following scenario: Flood insurance is required for federally backed mortgage loans, with rates essentially set according to maps that delineate areas expected to flood with an annual chance of occurrence of 1%. Determining the spatial locations of these zones requires knowledge of the flood hazard in an area, detailed topographic elevations, and records of observed events for developing the statistical models that represent likely storm occurrences in a region. Most of the statistical and physics-based models used have their origins in research, but the process of conducting the storm surge and statistical study is largely ad hoc, even though there are requirements for data management and archiving. There is no requirement on the software design, maintenance, and evolution, even though the ultimate results of a flood insurance study are regulatory.

In a *coastal* flood insurance study, several numerical models are used that simulate storm surge and wind-wave responses to tropical cyclones and extra-tropical storms. Because these computational and statistical studies of the coastal environment need to resolve small scale features that impose hydraulic constraints on the physical system, we necessarily use models that are themselves research tools into the physics and the numerical/computational methods used to solve the problems on high-performance computers. It is easy to see that this situation (using sophisticated research software for results that become regulatory) contains many of the issues associated with sustainable software development and best-practices adoption and adherence.

So, given the above use case, the need for sustainable scientific software and development practices extends well beyond just the credibility argument. Since issues like insufficient documentation, limited test cases, and software unavailability (e.g., in the case of proprietary software or unwillingness to share) are significant barriers to informed and intelligent science software usage, the consensus is that adoption of, and adherence to, best practices in scientific software development will substantially increase intelligent software usage, thus promoting a sustainable evolution of both the scientific software and the science as encoded in the software. Best practices, for example as described by G. Wilson [5], include designing for people and not computers, defensive programming, optimize only after the software has been validated, use bugs as new test cases, and the use of software versioning systems (e.g., Subversion, GitHub).

Perhaps the biggest problem inhibiting development of sustainable science software is the tension and time scale differences between *get it done* and *get it done right*. This problem is directly related to how scientific software development is funded, since this directly impacts the extent to which best practices are adopted, implemented, and maintained. Much scientific software has been developed in an ad hoc manner, with an inconsistent funding stream, and with variable adherence to and application of core software engineering best practices. This situation is exacerbated when the scientist is also the software developer. This could be out of necessity due to resource constraints, or because a scientist likes writing software.

There is naturally a spectrum of scientists' participation in software development activities. Toward one end, scientists remain somewhat distanced from software development activities, but rather cede software responsibility to software experts. This requires substantial planning of research activities within which software best practices and engineering are given equal weight to scientific development. It is possible that this approach may only be practicable when the science and software engineering are *co-funded*, wherein science and software are funded within project grants at relatively equal levels of effort. Co-funding, however, implies a co-dependence between the groups, which ultimately depends on sustained funding for the co-development of scientific software. *Sustainability of the software then implies and requires sustained funding*. Additionally, professional software development is generally expensive and time consuming.

Research project budgets generally cannot withstand this level of cost, *unless* they are co-funded. In the academic, scientific area, co-funded development is rare and will likely remain so without direct and explicit acknowledgement of the problem at the highest levels.

The other extreme is for scientists to become fully engaged in the software development process from inception and design through an agile (where software is iteratively and incrementally written and tested frequently) development and delivery. Scientists become relatively expert in software engineering best practices, an approach advocated by some and instanced in several science curricula in the US and Europe.

Both of these extremes are relevant and important, primarily because some scientists like writing software and would prefer to be deeply involved in its development, and some scientists don't like writing software. In terms of sustainable and reusable scientific software, it ultimately doesn't matter which path is taken. What matters is that some approach is adopted by individual projects, and the end product is the result of best practices, regardless of who carried out the development. Some level of expertise is essential for scientists to work within the computational and software development communities. Many scientists write good software by following best practices, but errors are inevitable. Following best practices will make errors in the software easier to find and correct, and generally these will be found much earlier in the software lifetime.

Fortunately, good software can and does emerge from the relatively ad hoc process of code development, either because at some point the software and supporting infrastructure is completely overhauled with best-practices and software engineering at the forefront, or because some level of good design was adopted early in the process. Many of the numerical models used for coastal ocean research (from process-based studies to forecasting of coastal ocean response to weather and climate) have been developed in this manner, with differing levels of best-practices strategies (e.g., HYCOM, ROMS, ADCIRC).

One specific idea to increase reproducibility of research results is to adopt a peer-review process for the software used in scientific research, analogous to that in the proposal review **and** publishing process. If peer-review of scientific results and the software itself becomes a requirement for publication and is fully implemented, then this would certainly promote early adoption of best practices. There are, however, several factors that complicate this. This implies that a research project and results could be rejected if best practices are not adhered to even if the results are sound. This also implies that best practices and standards be proposed, vetted, approved, adopted, *and* enforced. Who would constitute a set of peers for a review? The pool of potential reviewers that are expert enough in both software engineering and the science domain will be a very small group. How would a review process fit into an agile development cycle? Would each cycle be reviewed? How would this level of involvement be funded? The authors' worry is that only the co-funding model will work, which ultimately does not seem sustainable as a specific project ends but the software continues to evolve and grow.

NSF-funded projects can and should lead the way as to how software can be developed to simultaneously achieve research goals and produce sustainable, reusable software. The advent of funded software institutes such as the Water Science Software Institute [6] and the Institute for Sustainable Earth and Environmental Software, a part of the NSF's Cyberinfrastructure Framework for 21st Century Science and Engineering CIF21 [7], is clearly an effort to promote and instantiate software development and engineering best practices (among other cyberinfrastructure related concepts) in the academic research culture. An additional need is for all directorates, divisions, and offices to recognize and acknowledge the importance of the issue, subsequently require a software development and sustainability plan analogous to the data management plan, and (most importantly) enthusiastically fund software (and data management) activities explicitly. And, of course, to adopt standards for both data and software against which the software can be valued.

References

1. **Donoho, D, Maleki, A, Rahman, I U, Shahram, M** and **Stodden, V** 2009 Reproducible research in computational harmonic analysis. *Computing in Science and Engineering*, 11(1): 8–18. DOI: http://dx.doi.org/10.1109/MCSE.2009.15
2. **Stodden, V** 2010 The Scientific Method in Practice: Reproducibility in the Computational Sciences. *MIT Sloan Research Paper No. 4773–10.* DOI: http://dx.doi.org/10.2139/ssrn.1550193
3. **Joppa, L, McInerny, G, Harper, R, Salido, L, Takeda, K, O'Hara, K, Gavaghan, D** and **Emmott, S** 2013 Troubling trends in scientific software use. *Science*, 340(6134): 814–815. DOI: http://dx.doi.org/10.1126/science.1231535
4. **Hannay, J, MacLeod, C, Singer, J, Langtangen, H P, Pfahl, D** and **Wilson, G** 2009 How do scientists develop and use scientific software? In: Proceedings of the 2009 ICSE Workshop on Software Engineering for Computational Science and Engineering, SECSE '09, Washington, DC, USA, IEEE Computer Society, pp. 1–8.
5. **Wilson, G, Aruliah, D, Brown, C, Chue Hong, N, Davis, M, Guy, R, Haddock, S, Huff, K, Mitchell, I, Plumbley, M, Waugh, B, White, E** and **Paul, W** 2014 Best Practices for Scientific Computing. *PLoS Biology,*12(1): e1001745. DOI: http://dx.doi.org/10.1371/journal.pbio.1001745
6. **Ahalt, S, Band, L, Christopherson, L, Idaszak, R, Lenhardt, C, Minsker, B, Palmer, M, Shelley, M, Tiemann, M** and **Zimmerman, A** 2014 Water Science Software Institute: Agile and open source scientific software development. *IEEE Computing in Science and Engineering (CiSE)*, 6(3): 18–26.
7. See: http://www.nsf.gov/cise/aci/cif21/CIF21Vision-2012current.pdf

How to cite this article: Blanton, B and Lenhardt, C 2014 A Scientist's Perspective on Sustainable Scientific Software. *Journal of Open Research Software,* 2(1): e17, pp. 1-4, DOI: http://dx.doi.org/10.5334/jors.ba

Published: 9 July 2014

]u[*Journal of Open Research Software* is a peer-reviewed open access journal published by Ubiquity Press.

OPEN ACCESS