

## Opinion

# Barely sufficient practices in scientific computing

Graham Lee,<sup>1,\*</sup> Sebastian Bacon,<sup>2</sup> Ian Bush,<sup>3</sup> Laura Fortunato,<sup>4,5</sup> David Gavaghan,<sup>1</sup> Thibault Lestang,<sup>1</sup> Caroline Morton,<sup>2</sup> Martin Robinson,<sup>1</sup> Philippe Rocca-Serra,<sup>3</sup> Susanna-Assunta Sansone,<sup>3</sup> and Helena Webb<sup>1</sup>

<sup>1</sup>Department of Computer Science, University of Oxford, Oxford, UK

<sup>2</sup>Nuffield Department of Primary Care Health Sciences, University of Oxford, Oxford, UK

<sup>3</sup>Oxford e-Research Centre, Department of Engineering Science, University of Oxford, Oxford, UK

<sup>4</sup>Institute of Cognitive and Evolutionary Anthropology, University of Oxford, Oxford, UK

<sup>5</sup>Santa Fe Institute, Santa Fe, NM, USA

\*Correspondence: [grahamlee@acm.org](mailto:grahamlee@acm.org)

<https://doi.org/10.1016/j.patter.2021.100206>

The importance of software to modern research is well understood, as is the way in which software developed for research can support or undermine important research principles of findability, accessibility, interoperability, and reusability (FAIR). We propose a minimal subset of common software engineering principles that enable FAIRness of computational research and can be used as a baseline for software engineering in any research discipline.

The importance to research of data and how it is collected, processed, and analyzed is accepted by the community. The principles of findability, accessibility, interoperability, and reusability (collectively, the FAIR principles) were first published in 2016<sup>1</sup> and have become widely adopted norms in many disciplines, with best practices recommended or mandated to support FAIRness of data. Some in the community are considering how the idea of FAIRness applies to research software.<sup>2</sup>

Many results in the scientific literature now depend on software. Hypotheses are constructed based on the results of computational models. Data are gathered and analyzed using commercial packages, open source modules, and custom scripts. Software is critical to the accessibility and reusability of scientific data, and the verification of scientific methods. Of scientists surveyed by Pinto et al. in 2018,<sup>3</sup> 86% report that developing scientific software is important to their own research, and 63% that it is important to the research of others. The average respondent reported spending 30% of their working time on writing scientific software.

Research software design and construction is still largely a cottage industry, with 99% of respondents in the Pinto survey reporting that self-study was important to their software development skills acquisition. Far from agreeing on best practice across all computational

research, disciplines have divergent software practices leading to different levels of reliability, replicability, and reusability of research software. In the next section, we give recent examples of problems in the production or impact of research due to the way in which related software was constructed. We then propose ADVerTS (availability of software, documenting software, version control, testing, and support), a minimal selection of “barely sufficient” software engineering practices chosen to address these problems and suggested as a baseline for research software development.

### The problem

Software written without due attention to relevant engineering practices can suffer from many issues. Two cases presented here from the literature on coronavirus disease 2019 (COVID-19) demonstrate unsupportable conclusions that must be retracted, and distrust of (valid) results.

*The Lancet* published, then retracted, analysis of the effects of hydroxychloroquine or chloroquine treatment for COVID-19.<sup>4</sup> In the retraction, Prof. Mehra indicates that the data and software on which the analysis was based were not made available for independent peer review or replication.

A model of pandemic spread produced at Imperial College, London was used to justify government policies related to lockdown in March 2020. Initially unavailable for external scrutiny and replication,

the software and the decisions based upon it received intense criticism. It was not until June 2020, well into lockdown, that the model was conclusively validated and its results replicated, after significant input from academic and commercial software communities.<sup>5</sup> By this time, public trust in the model had already eroded, a situation that could have been avoided by documenting and testing the software when it was first used.

### ADVerTS: pragmatic practices in research software

The availability, portability, and reusability of scientific research software has important effects on the reliability and reproducibility of the scientific record, and the trustworthiness of results gained using that software. It would be fruitless to demand that all researchers skill up as fully trained software engineers, a distraction to many whose interests lie in exploring scientific questions, not in producing commercial-quality code.

We propose the ADVerTS practices: availability of software, documenting software, version control, testing, and support. This minimal set of five practices should be considered a baseline in data science or any research discipline. The name ADVerTS is chosen because adherence to the practices “advertises” the fact that authors have taken an end-to-end approach to FAIRness of their research that includes software artifacts.



Each ADVerTS practice is described along with a justification of how adopting that practice supports the scientific method, improving the FAIRness of associated research. This choice of practices was designed to be small enough that every researcher can either acquire the relevant skills or find someone locally to provide training, yet impactful enough to bring about a meaningful improvement to the state of research software.

#### **Availability of software for others to use**

Replicating and building on another researcher's results requires access to the same tools used by the original researcher. Software is, in principle, the easiest tool to distribute, as doing so is free. Some academics choose not to share their codes because of embarrassment with their programming skills, or due to "Gollum syndrome," in which they view their software as their personal, precious artifact.

Software used in research, particularly custom tools or scripts made by and on behalf of researchers for particular projects, should be made available to the community in open repositories with metadata using the standard Citation File Format.<sup>6</sup> The software's repository entry should be associated with a digital object identifier (DOI). The DOI should be used in connection with any publication using the software.

This practice supports the findability, accessibility, and reusability of software artifacts by making the software readily available to other researchers. Publishing the software up front reduces the friction of trying to obtain software on request from the original authors and obviates reliance on those authors' computing equipment, backup regimens, and approaches to filing. It also enables the growing practice of digital archaeology—the need to recover and reuse software produced a long time ago—when the original author may have moved on to other things. The benefits (earlier and more detailed peer review, greater FAIRness, opportunities to reuse others' code and have your code used by others) of publishing research software early and often outweigh the embarrassment of sharing work-in-progress code.

Once available, the software also needs to be licensed for re-use by others; for

example, under an open-source software (OSS) license like the Massachusetts Institute of Technology (MIT) license or GNU General Public license (GPL).

#### **Document setup, use, and expectations of code**

With access to the software, the would-be replicator must be able to execute the software in the same way that the original author did. Respondents to the Pinto et al. survey<sup>3</sup> describe documentation as one of the biggest "pain points" in scientific software. This contradicts the statement in commercial software development that "we have come to value working software over comprehensive documentation."<sup>7</sup> The audience for commercial application source code is typically other professional developers who benefit from the training, and access to their colleagues, necessary to interpret undocumented source code.

Additionally, we ask not for *comprehensive* documentation of software, but for *barely sufficient* documentation. It should be possible for a motivated individual with no prior knowledge of the project to obtain a copy of the code, get it running, then make productive use of the software, following just the steps included in a README or "getting started" file.

We acknowledge that software portability is a complex problem that many researchers are neither equipped nor necessarily motivated to handle, but that non-portable software is a hindrance to re-use. We suggest that authors of research software document the setup requirements of the software, including specific version numbers of programming languages, packages, modules, or other programs used. If it is not possible to run the software in other environments, it should at least be possible to replicate the author's configuration. This will bring advantages to the original author, coming back to the software they have written sometime later and attempting to reuse it.

#### **Version control**

Scientific software typically goes through many revisions, whether trial-and-error improvements on a single author's workstation or multiple releases as part of an ongoing program of research. It is critical for the reproducibility and re-use of research that replicators get to use not merely the same script or application, but the exact same *version* of the software that was used in the original research.

A version control system works like an intentional change tracker, allowing authors to record significant changes to their code as "commits" in a permanent record. It is always possible to retrieve earlier versions of the code from the repository of commits, and to browse the log to discover when particular changes were made.

Version control makes modifying software a less stressful experience by reducing the cost of change, as any modifications that lead to a dead end can always be reverted. While we do not go as far as to propose that any particular version control system be adopted, we acknowledge that Git is already popular among both academic and commercial programmers and suggest it as a good example of a version control tool.

#### **Test, mostly at the unit level**

It can be difficult to know what the expected behavior of research software should be when run on real research problems, though some research software, for example, that tasked with curation or presentation of data, can be clearly specified. Any software can be tested for conformance with the expected algorithm even if the results expected in an experiment are not known. Testing reduces the risk that results gained in research are erroneous, by ruling out failures due to incorrect software behavior. Risks due to problems with the input data and the underlying algorithms can be secondarily uncovered, thanks to the test's status as a precise and critical specification of the program's expected behavior.

We suggest small tests of isolated components, called unit tests, which give good feedback on what has gone wrong when a test failure is reported, and motivate decomposition of codes into small modules and procedures that facilitate sharing and re-use. End-to-end tests are often implemented using "toy" problems that, while well understood, do not fully exercise edge cases or alternate code paths in the logic of the program under test. Nonetheless, end-to-end tests provide confidence that the whole system is correctly set up and integrated. We suggest that any tests are better than no tests and that effort should be spent mostly, though not exclusively, on unit tests.

Tests should be considered part of the software's documentation, as they succinctly encapsulate information about

the software's expected behavior. Tests should be part of the standard software distribution, along with documentation of how to run the tests.

### Support or maintenance of the software clarified

Re-use of research software may be expected or even sought out when the software is created; for example, when the author sets out to make a library of algorithms or a framework for modeling particular problems. On other occasions, re-use arises despite expectations that the software is "done." A PhD student finds a useful script on the shared drive, or you bundle up some files to send to a peer and they use one of your algorithms in their analysis. Either way, re-use is almost inevitable and should be planned for.

Making clear at time of publication the expectations other researchers can have regarding the sustainability of the software, both in terms of the expected workflow and the resources available, will simplify the authors' work and that of people who wish to extend that work. The authors can direct requests or contributions to a particular mailing list, or issue tracker on a website like Github, to reduce communication overhead and have everything in one place to aid prioritization and visibility; additionally, they can open up the work to their group or a volunteer community. Potential contributors and collaborators can be sure that their requests and contributions are going to the correct place.

Frequently, the situation will be that the authors have no time or resources to sup-

port the software they published, because there is no funding or they are working on another project. This is still important information to share so that the community knows what to expect and can make informed judgements about the sustainability of the software it depends on.

### Conclusion

Calls in the scientific literature for improvements in software engineering practice among computational researchers are not new.<sup>8</sup> Here, we propose a minimalist subset of practices deliberately chosen to support the robustness and veracity of the scientific record. Rather than demand that researchers also become professional software engineers, we merely suggest acquiring a small collection of skills that will quickly improve the FAIRness of their research. We also suggest that journal editors and funding bodies use evidence of these practices as signals of the reliability of software associated with research.

We welcome further discussion on defining or adapting these baseline practices for computational research. We also look forward to experience reports on teaching and adopting these practices across the community.

### REFERENCES

1. Wilkinson, M.D., Dumontier, M., Aalbersberg, I.J., Appleton, G., Axton, M., Baak, A., Blomberg, N., Boiten, J.W., da Silva Santos, L.B., Bourne, P.E., et al. (2016). The FAIR Guiding Principles for scientific data management and stewardship. *Sci. Data* 3, 160018.

2. Research Data Alliance (2019). fair for Research Software (fair4rs) wg. <https://www.rd-alliance.org/groups/fair-research-software-fair4rs-wg>.
3. Pinto, G., Wiese, I., and Felipe Dias, L. (2018). How do scientists develop scientific software? An external replication. *IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (saner) 2018*, 582–591.
4. Mehra, M.R., Desai, S.S., Ruschitzka, F., and Patel, A.N. (2020). Retracted: Hydroxychloroquine or chloroquine with or without a macrolide for treatment of COVID-19: a multinational registry analysis. *Lancet* 395, 10240.
5. Chawla, D.S. (2020). Influential Pandemic Simulation Verified by Code Checkers. <https://media.nature.com/original/magazine-assets/d41586-020-01685-y/d41586-020-01685-y.pdf>.
6. Druskat, S., Bast, R., Hong, N.C., Kononov, A., Rowley, A., and Silva, R. (2017). A standard format for CITATION files. <https://www.software.ac.uk/blog/2017-12-12-standard-format-citation-files>.
7. Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., et al. (2001). A Manifesto for Agile Software Development. <https://agilemanifesto.org/>.
8. Wilson, G., Bryan, J., Cranston, K., Kitzes, J., Nederbragt, L., and Teal, T.K. (2017). Good Enough Practices in Scientific Computing. *PLOS Computational Biology* 13, <https://doi.org/10.1371/journal.pcbi.1005510>.

### About the author

Graham Lee is a senior research software engineer in the Oxford RSE group at Oxford's computer science department, where he works with researchers of all disciplines to ensure that their research goals are well supported by the software their groups create and use. He is also a PhD student in the same department, researching the values and practices of research software engineering. Graham came to academia after 15 years as a professional software engineer in industry, with experience spanning startups, blue chip companies, and established Silicon Valley corporations.