IJDC | *General Article*

Software Must be Recognised as an Important Output of Scholarly Research

Caroline Jay
University of Manchester

Robert Haines
University of Manchester

Daniel S. Katz
University of Illinois at Urbana-Champaign

Abstract

Software now lies at the heart of scholarly research. Here we argue that as well as being important from a methodological perspective, software should, in many instances, be recognised as an output of research, equivalent to an academic paper. The article discusses the different roles that software may play in research and highlights the relationship between software and research sustainability and reproducibility. It describes the challenges associated with the processes of citing and reviewing software, which differ from those used for papers. We conclude that whilst software outputs do not necessarily fit comfortably within the current publication model, there is a great deal of positive work underway that is likely to make an impact in addressing this.

Submitted 3 November 2020 ~ *Revision received* 6 December 2021 ~ *Accepted* date 20 December 2021

Correspondence should be addressed to Caroline Jay, Kilburn Building, University of Manchester, Oxford Road, Manchester M13 9PL. Email: Caroline.Jay@manchester.ac.uk

The *International Journal of Digital Curation* is an international journal committed to scholarly excellence and dedicated to the advancement of digital curation across a wide range of sectors. The IJDC is published by the University of Edinburgh on behalf of the Digital Curation Centre. ISSN: 1746-8256. URL: http://www.ijdc.net/

Introduction

Software is transforming scholarly research[1] practice, increasing the scale of knowledge production (Hettrick, 2018), and—through the automation of analysis pipelines—putting genuine reproducibility of experiments within reach. Where once studies were conducted *in vivo*, or *in vitro*, they are increasingly being conducted *in silico*. Software has also led to the creation of new forms of analysis and representation, enabling research or thinking that was not previously possible: computational models now form the backbone of many research domains, shifting the way in which we represent and understand the world.

Alongside the opportunities offered by computation, there is a conundrum for the research community: whilst software is now central to the production of research, it is difficult---arguably impossible---to represent it adequately in standard scholarly publications. Documents, in particular peer-reviewed papers, are currently the primary currency of scholarly research. Articles, alongside lab notes, books and reports, combine mathematical or logical formalisms with a descriptive narrative, allowing others to understand what has been discovered, and the context in which this has been achieved.

Software exists to perform processes and calculations that would otherwise be impossible in practical terms. Whilst we can endeavour to express an algorithm in pseudocode (a process fraught with problems, as the proliferation of inaccurate versions of Porter's stemming algorithm demonstrates (Thimbleby, 2003)), many computational analyses simply cannot be translated into words or equations (Jay et al., 2020). Explaining what a piece of software does will remain an essential part of reporting research, but providing access to the code itself is vital to ensuring the integrity, transparency and reproducibility of the research. This is part of the process of making the software FAIR, increasingly recognized as a key element in enabling better and more productive scholarship (Lamprecht et al., 2020; Directorate-General for Research and Innovation (European Commission), 2018; FAIR for Research Software (FAIR4RS) Working Group [2]).

If a computational model or analysis has complexity that cannot be adequately expressed in the form of a traditional, text-format publication, then it follows that the software should be treated as a research output in its own right, and its creators should be credited with making a contribution to scholarship. Whilst this may be acceptable in theory, the paper still rules as the primary measure of academic achievement in practice, so a rethink of how we understand and value scholarly endeavour is required. Here, we examine the reasons that software should be considered as a first-class citizen of scholarly research, and outline the challenges that we must overcome to achieve this.

The role of software in scholarly research

Software is changing the way we conduct scholarly research, in terms of the sophistication of the analyses we perform and the volume of data we can process. It supports real documentation of the research process (known as provenance), and makes it possible to verify results, by improving the reproducibility of the analysis pipeline. Executable notebooks, such as Jupyter, or R markdown, are a good example of this: by interleaving explanations with software, they make it straightforward to understand and rerun the way an author has processed data. Making software methods, models and analyses open to others can greatly accelerate the rate at which we gather knowledge and make discoveries. In spite of its value, however, a great deal of research software remains unpublished and unavailable (Peng, 2011). This is potentially a huge

[1] We use "scholarly research" as a general term for research in science, engineering, humanities, etc.
[2] https://www.rd-alliance.org/groups/fair-4-research-software-fair4rs-wg

loss to scholarly research: whilst very few recent papers would exist without the aid of software, software stands on its own, and may have uses that extend far beyond a single publication (de Souza, Haines & Jay, 2014; de Souza, Haines, Vigo & Jay, 2019).

Currently many researchers are not working as openly as they could. The main reasons researchers give for this are embarrassment due to perceived poor quality code, a lack of confidence the software is robust for other users and usages, and the time required to prepare it for release, including the provision of appropriate licensing and documentation (Jay, Sanyour & Haines, 2016). The second point is particularly troubling: if a researcher is not confident in their own software, how can they be confident in the results it produces? Improving the visibility, and therefore scrutiny, of research software would mitigate these problems, increasing both the openness of a project, and the confidence in its conclusions. It is important to note here that valuing the software in its own right is an important catalyst to good development. Where the software is simply regarded as a means to an end, rather than an integral part of the research, the temptation to minimize the time and resources that go into its creation is high.

Increased openness may be viewed by some as a burden, but it ultimately has the potential to benefit researchers and the culture they work in. A report from the UK Parliamentary Office of Science and Technology, "Integrity in Research" (Auckland & Bunn, 2017), puts an emphasis on enforcing the integrity of research outcomes, potentially via regulation, but does not address *how* researchers' everyday practices should evolve to ensure this outcome is achieved. Telling researchers that they are not working with integrity—in effect that they are not doing research *well*—is applying pressure in the wrong place; while mistakes happen, the vast majority of researchers are working honestly. Instead, a focus on promoting openness is likely to have a much larger impact while fixing the same problem, as it will naturally increase the chances of mistakes being caught. Valuing software in its own right, and giving credit to those who produce it, is an excellent way of motivating this shift in practice.

When is software an output?

Software plays different roles in the research process. It can 1) be a tool for supporting the work -- software as infrastructure, 2) embody the research itself, for example, in a scientific simulation, or 3) be an object of study, such as in software engineering research. The role of the same piece of software can vary according to the context. To a computer scientist in the field of workflow management, the workflow software would be considered a direct output, as it is the manifestation of the research. To a biologist, this same software would be considered a tool: useful for analysing results, but not itself an output of the research. For a bioinformatician, both using and developing the tool, the answer is somewhere in the middle: whilst the core research may be in the life science domain, the modifications made to the tool as a result of this work could also be considered an output, advancing workflow management (Jay & Haines, 2016).

Drawing a hard line between these categories is difficult. Another way of considering software within the research process is from the angle of reproducibility and reusability. If any bespoke software is developed as part of the research, even if it is just an analysis script, then making it available is an important part of the reproducibility pipeline. This is only part of the challenge however; to maintain the integrity of the software as a part of the research process, it is important not just to be able to access it, but also to be able to refer to it accurately.

Citing software

Many venues now mandate that data, and increasingly analysis software, be archived and made available alongside a paper (Katz, 2021), in a welcome step towards improving the reproducibility of research. This works well when the software is a straightforward analysis script, but the process of archiving quickly becomes complex with anything beyond this. A

preserved `snapshot' of the environment in which a discovery was made is crucial to fully understanding the provenance and reliability of the data, and the potential permanence of software promises to greatly increase the rigour of the scientific process. Most publication venues lack guidelines that encourage citing software directly, however, and doing so is not general practice. While work is ongoing in improving how repositories work with software (Task Force on Best Practices for Software Registries, 2020), a common workaround is to cite a related paper instead. This might be a paper describing a larger study, where the software was integral to that research and is described in the methods section, or it might be a "software paper": a paper that exists solely to describe the software, in a venue such as SoftwareX, the Journal of Open Research Software (JORS), the Journal of Open Source Software (JOSS) or F1000 Research, which require the authors to deposit the software in an archive after peer-review, for example, archiving a tagged release into Zenodo or Figshare directly from GitHub (Referencing and citing content[3]). In either case the software referred to in those papers will be out of date very quickly. Software does not stay still—bugs are fixed, new functionality is added and optimizations are made—and development is rarely paused for lengthy journal submission processes to complete. The specific release of software must be preserved (archived) and then cited directly, in each publication in which it is used, to be sure that the correct version is referenced each time, and can be used for reproducibility. Providing information that will help people find the latest version of the software in a repository is also helpful, as this may be the one most useful to someone who wishes to use or develop the software further (Software and repositories in the context of FAIR[4]). Recently, GitHub has added a feature that allows software authors to include a CITATION.cff file in their repository, which is then surfaced in a widget which provides quick access to citation text in an APA-like format, and BibTeX format (About citation files[5], Druskat 2021).

Precisely how to cite archived software remains an open question (Smith, Katz, Niemeyer & FORCE11 Software Citation Working Group, 2016), but an obvious mechanism for doing this is to use a Digital Object Identifier (DOI) for the particular version of the software, and include this in the reference list in the paper. As software and papers have a symbiotic relationship, it would be ideal to link back to the paper from the software. The publication workflow makes this difficult, however, as the paper will be published after the software, and at that point it is not possible to alter the software object and maintain the integrity of the DOI. Indeed, the nature of the DOI allocation process means that it is impossible for two objects to reference each other without careful planning and DOI reservation. This demonstrates the necessity for software to be considered an object in its own right, standing alone and independently of any paper. An alternative is to cite software via an automated archiving of it in code development platforms, e.g. the Software Heritage archive that archives GitHub (Cosmo, Gruenpeter & Zacchiroli, 2020).

Peer review of software

If software is to be considered an output of scholarly research it is important to ensure, as with text-based publications, it is valid and reliable. Peer review is currently the accepted method for determining the validity (and to some extent, value) of research outputs, and the format for the review of text publications is well-established. `Software-paper' venues (e.g., SoftwareX, JORS, JOSS) and initiatives such as ACM Artefact badging (ACM artefact review and badging[6]) have a review process for software, but the methodology currently followed often focuses primarily on checking that the software meets technical requirements (for example, that it is open source and

[3] https://docs.github.com/en/repositories/archiving-a-github-repository/referencing-and-citing-content
[4] https://danielskatzblog.wordpress.com/2020/10/20/fair-software-and-repositories/
[5] https://docs.github.com/en/repositories/managing-your-repositorys-settings-and-features/customizing-your-repository/about-citation-files
[6] https://www.acm.org/publications/policies/artifact-review-and-badging-current

has installation instructions), rather than fully evaluating its scientific contribution. Clear documentation, strategies for quality assurance, such as unit tests, and following relevant coding standards are indicators of rigour, but should be treated as proxies, rather than guarantees of this.

Code review—checking that the way in which software is written meets certain quality standards—is widely used in industry to check for defects and ensure that software is efficient and usable by others. This process, analogous to checking that a paper is free from language errors, and that the narrative is unambiguous, has an important role in assessing research software, where accuracy is of paramount importance. Code review is an extremely time-consuming process, however, particularly where the reviewer is unfamiliar with the software, and as such realising this will be a challenge, though work is ongoing in determining and promoting code review best practices, e.g., CODECHECK (Nüst & Eglen, 2021). Determining the scholarly 'contribution' of software as a research output (which remains a contentious issue for traditional publications) may be less important if we take the view that its value can be judged by the papers in which it is cited, or the number of people who go on to use or extend it.

Conclusion

Software is now integral to scholarly research, and it is thus essential that it is open, accessible, and valued by the research community. The present publication model falls short of guaranteeing any of these things, but a shift is gradually occurring. Peer review of software is likely to remain a challenge, and may require a different approach from that used for papers. Official recognition of software as a research output will ultimately be transformative, improving the quality, reproducibility and scalability of our knowledge production, as well as recognising the often hidden role of the increasing number of scholarly researchers who spend most of their time writing code.

References

Auckland, C. & Bunn, S. (2017). Integrity in research. *POSTNote number 544*. Retrieved from http://researchbriefings.files.parliament.uk/documents/POST-PN-0544/POST-PN-0544.pdf

Cosmo, R. D., Gruenpeter, M. & Zacchiroli, S. (2020). Referencing source code artifacts: A separate concern in software citation. *Computing in Science & Engineering*, *22*(2), 33-43. doi:10.1109/MCSE.2019.2963148

de Souza, M. R., Haines, R. & Jay, C. (2014). Defining sustainability through developers' eyes: Recommendations from an interview study. In *2nd Workshop on Sustainable Software for Science: Practice and Experiences (WSSSPE2)*. doi:10.6084/m9.figshare.1111925.v1

de Souza, M. R., Haines, R., Vigo, M. & Jay, C. (2019). What makes research software sustainable? An interview study with research software engineers. In *2019 IEEE/ACM 12th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)* (pp. 135–138). Retrieved from https://arxiv.org/abs/1903.06039

Directorate-General for Research and Innovation (European Commission). (2018). *Turning FAIR into reality: Final Report and Action Plan from the European Commission Expert Group on FAIR Data*. Retrieved from https://ec.europa.eu/ info/publications/turning-fair-reality_en

Druskat, S., Spaaks, J. H., Chue Hong, N., Haines, R., Baker, J., Bliven, S., . . . Konovalov, A. (2021, 8). *Citation File Format.* doi:10.5281/zenodo.5171937

Hettrick, S. (2018, February). *2014 software in research survey.* doi:10.5281/zenodo.1183562

Jay, C. & Haines, R. (2016). Software as Academic Output. In C. Goble, J. Howison, C. Kirchner, O. Nierstrasz & J. J. Vinju (Eds.), *Engineering Academic Software.* Dagstuhl Publishing. doi:10.5362/DagRep.6.6.62

Jay, C., Sanyour, R. & Haines, R. (2016). "Not everyone can use Git": Research Software Engineers' recommendations for scientist-centred software support (and what researchers really think of them). Retrieved from https://figshare.manchester.ac.uk/account/articles/17313215. doi:10.48420/17313215

Jay, C., Haines, R., Katz, D. S., Carver, J. C., Gesing, S., Brandt, S. R., . . . Turk, M. J. (2020). The challenges of theory-software translation [version 1; peer review: 1 approved, 1 approved with reservations]. *F1000Research*, *9*(1192), 1192. doi:10.12688/f1000research.25561.1

Katz, D. S., Chue Hong, N. P., Clark, T., Muench, A., Stall, S., Bouquin, D., . . . Yeston, J. (2021). Recognizing the value of software: a software citation guide [version 2; peer review: 2 approved]. *F1000Research*, *9*(1257). doi:10.12688/f1000research.26932.2

Lamprecht, A.-L., Garcia, L., Kuzak, M., Martinez, C., Arcila, R., Pico, E. M. D., . . . Capella-Gutierrez, S. (2020, June). Towards FAIR principles for research software. *Data Science*, *3*(1), 37–59. doi:10.3233/ds-190026

Nüst, D. & Eglen, S. (2021). CODECHECK: an open science initiative for the independent execution of computations underlying research articles during peer review to improve reproducibility [version 2; peer review: 2 approved]. *F1000Research*, *10*(253). doi:10.12688/f1000research.51738.2

Peng, R. D. (2011). Reproducible research in computational science. *Science*, *334*(6060), 1226–1227. doi:10.1126/science.1213847

Smith, A. M., Katz, D. S., Niemeyer, K. E. & FORCE11 Software Citation Working Group. (2016, September). Software citation principles. *PeerJ Computer Science*, *2*, e86. doi:10.7717/peerj-cs.86

Task Force on Best Practices for Software Registries, Monteil, A., Gonzalez-Beltran, A., Ioannidis, A., Allen, A., Lee, A., . . . Morrell, T. (2020, December). Nine Best Practices for Research Software Registries and Repositories: A Concise Guide. *arXiv e-prints*, arXiv:2012.13117. Retrieved from https://arxiv.org/abs/2012.13117

Thimbleby, H. (2003). Explaining code for publication. *Software: Practice and Experience*, *33*(10), 975-1001. doi:10.1002/spe.537