Commentary: 'Responsible Research Assessment II: A specific proposal for hiring and promotion in psychology'

Andreas M. Brandmaier[1,2,3], Maximilian Ernst[2,4], & Aaron Peikert[2,3,4]

[1] MSB Medical School Berlin

[2] Max Planck Institute for Human Development

[3] Max Planck UCL Centre for Computational Psychiatry and Ageing Research

[4] Humboldt-Universität zu Berlin

A commentary on: Gärtner, A., Leising, D., & Schönbrodt, F. D. (2022, November 25). Responsible Research Assessment II: A specific proposal for hiring and promotion in psychology. https://doi.org/10.31234/osf.io/5yexm

To be submitted to: Meta-Psychology

Author Note

The authors made the following contributions. Andreas M. Brandmaier: Conceptualization, Writing - Original Draft Preparation, Writing - Review & Editing, Supervision; Maximilian Ernst: Conceptualization, Writing - Original Draft Preparation, Writing - Review & Editing; Aaron Peikert: Conceptualization, Writing - Original Draft Preparation, Writing - Review & Editing.

Correspondence concerning this article should be addressed to Andreas M. Brandmaier, Rüdesheimer Str. 50, 14197 Berlin. E-mail: andreas.brandmaier@medicalschool-berlin.de

Commentary: 'Responsible Research Assessment II: A specific proposal for hiring and promotion in psychology'

Introduction

Based on four principles of a more responsible research assessment in academic hiring and promotion processes (Schönbrodt et al., 2022), Gärtner, Leising, and Schönbrodt (2022) suggested a concrete evaluation scheme for published manuscripts, reusable datasets, and research software. We strongly support the increased emphasis on research software as a creditable and commendable scientific contribution. Why are research software contributions important scientific contributions? We would like to respond with a quote from the Science Code Manifesto (Climate Code Foundation, 2011): "Software is a cornerstone of science. Without software, twenty-first century science would be impossible." and "Software is an essential research product, and the effort to produce, maintain, adapt, and curate code must be recognized." However, despite the heavy reliance on computational infrastructure, the current academic infrastructure does not adequately incentivise software development and, specifically, good software engineering practice (Baxter, Hong, Gorissen, Hetherington, & Todorov, 2012). In line with principle 3 of Schönbrodt et al. (2022), we suggest that criteria for research software contributions must capture two major dimensions: rigour and impact. Impact measures whether the scholarly effort, implementation, and dissemination actually had a visible effect on the field. rigour means implementing high standards and best practices for ensuring transparency, correctness, and reusability of a piece of software. By setting a high bar of rigour in research software assessment in academic hiring and promotion, we hope to foster the creation of better software and, thus, better science. From this perspective, we comment on some indicators of the proposed evaluation scheme for research software contributions (Table 3 of Gärtner et al. (2022)).

Proposed Criteria

ID 5: Date of first fully functional public version/ID 9: Citations

Computing the citations in relation to age of software seems to be inconsistent with the proposal of Schönbrodt et al. (2022), who reminded us that a core principle of the implementation of DORA is to "abandon the use of invalid quantitative metrics of research quality and productivity in hiring and promotion" (p.2). It is unclear why a citation-based metric for software would be more valid than the equivalent for articles. In fact, there are several additional shortcomings in relation to software. We discuss shortcomings of the numerator (number of citations) here and the issues of the denominator (software age) in the next section.

Citations suffer from serious shortcomings when used to evaluate the impact of research software, particularly when used comparatively in the context of hiring and promotion. For example, citing data analysis packages is much more commonly accepted than citing supporting packages (such as papaja, Aust & Barth, 2022, used to render this article).

Further, functionality of successful, modular scientific software is ideally reused in other software packages to avoid code duplication and enable faster development of new software. While we highly encourage reuse from both a software engineering perspective and for scientific progress, it challenges the validity of citation-based metrics for impact. For example, consider the `NLopt` optimization suite, which counts 1,711 citations on Google scholar at the time of writing. `NLopt` is a backbone for many scientific packages both because it implements various optimization algorithms but also because it is open-source and can inspire re-implementations of these algorithms. One such example is the famous `lme4` package (Bates, Mächler, Bolker, & Walker, 2014) for generalized linear mixed-effects models, that is partly based on `NLopt`. `lme4` has more than 58,000 citations on Google scholar, yet it is unlikely that researchers will cite the underlying optimization algorithm. For another example, the `pdc` package (Brandmaier, 2015) offers functions to cluster time series based on

one specific algorithm. The `TSclust` package (Montero & Vilar, 2014) is a wrapper package, which imports and makes accessible functionality from the `pdc` package as well as various other clustering approaches, which is very useful from a users' perspective; however, we noticed that researchers now cite `TSclust` instead of `pdc`, which challenges the citation-based assessment of impact.

ID 6: Date of most recent substantive update

Both ID 5 and 6 are difficult to ascertain because it is not always clear when updates are considered 'substantive' or software 'fully functional'. We appreciate the importance of assessing maintenance as part of rigour. To assess this, we propose to provide a simple checkbox, in which the author indicates whether a scientific software package is actively maintained (e.g., the software has a regular release cycle or an update within the last six months). In addition, authors have a chance to explain why their active maintenance may be different from these guidelines.

ID 14: Lines of Code

We discourage the lines of code (LOC) metric to measure effort. LOC highly depends on programming language, mastery, and personal programming style. In particular, many LOC may simply mean that a researcher writes inefficient and repetitive code, one of the great sins of programming. On the contrary, a feature of good software is modularity because it enables reusing functions both inside and outside the project, resulting in fewer LOC.

ID 7: Contributor Roles and Involvement

We support the standardised assessment of project contributor roles, similar to the Contributor Roles Taxonomy (CRediT; https://credit.niso.org). However, we like to point out that the current evaluation schemes yields lower scores for the same effort of the individual researcher if they are part of a larger software project.

ID 8: License

At present, whether a piece of software is open source is not evaluated in the prescreening phase. However, an open license is central for assessing both rigour and (potential) impact, and should be part of the phase I assessment. Above, we already discussed the different ways research software can have an impact — not only by direct usage, but also by reusing software in other packages. Open-source software makes broader impact more likely and is a prerequisite for full transparency and reproducibility (Peikert, Van Lissa, & Brandmaier, 2021). In addition, many aspects of rigour are impossible to evaluate for closed-source software — for example, whether it is well-tested or bugs have been fixed. Therefore, we propose to penalize software if it does not adhere to an open-source license (those approved in a review process by the Open Source Initiative (https://opensource.org/licenses) by allowing them only half of the total achievable points.

ID 17: Reusability Indicator

This is one of only two criteria used in the prescreening phase of the proposal and therefore is of central importance. It is also important because it assesses aspects of rigour in software development: documentation, active maintenance, and testing. However, by incorporating the size of the user base, it confounds *usablity* (as an aspect of rigour) with *usage* (as an aspect of impact). In addition, the criteria for the different proposed categories are not clearly defined. For example, the difference between "fairly extensive" and "extensive" documentation is unclear. As a result, this indicator is more of a "gut-feeling" indicator, roughly assessing the "size" of the software project. Instead, in the following we propose to assess rigour and impact independently as primary aspects of a software contribution.

rigour for software implementations

We propose to use the following aspects as equally weighted indicators of rigour instead of the proposed broader reusability indicator (item 17 in Table 3 of the proposal).

Tests

Tests are essential to discover incorrect functionality, investigate code scalability and reveal poor design choices. There are a variety of useful tests, such as unit tests of subcomponents or tests of software functionality at a larger scale (e.g., see the testthat package in R, Wickham, 2011). It is possible to quantify aspects of software testing, for example, by assessing code coverage, defined as the percentage of code lines executed during testing. However, we believe that we should give points for software that promises that major functionality is covered by tests. Those tests should be automated or at least open-source and reproducible.

Documentation

Just like for tests, there are different types of documentation. For example, tutorials showcase software usage with examples, and there is application programming interface (API) documentation for individual functions and classes to enable reusing functions in other software packages. We propose to identify relevant categories of documentation for research software, e.g., installation instructions, tutorials, API, and community guidelines , and score the presence of each of them separately.

Maintenance

Maintaining a software package is often more work than writing it. This should be reflected in the assessment procedure. We propose to score two aspects of maintenance separately, maintaining the code base (such as active bug fixing and documenting changes in logs) and maintaining the community (such as providing the possibility to report bugs, feature requests, or support requests via tickets or mailing lists).

Measuring Impact

Total citation metrics, number of users, downloads per month, GitHub stars and similar may provide a coarse measure for the impact of a software package, even though it is important to note the shortcomings we described above.

We believe that the suggested merit statement is most useful to assess impact of research software and that this should be the primary statement for committee members to evaluate if they are less concerned with the technical aspects of research software development. In our view, researchers should be requested to indicate at least one (and up to three) research projects that directly benefited from their software contributions. We believe this to be a fair assessment of the actual impact the specific contribution of the individual researcher had.

Summary

In sum, we are thankful to Gärtner et al. (2022) for highlighting the importance of research software contributions as scientifically valuable products. We believe the current proposal should aim to better reflect the distinction between rigour and impact for software, similar to the guidelines proposed to evaluate journal articles. To this end, we suggest a more fine-grained assessment of rigour, and put emphasis on the merit statement, in which software authors should argue in how far their project impacted other scientific endeavours. Last, we hope to arrive at an evaluation scheme that incentivises the development of scientific open-source software.

References

Aust, F., & Barth, M. (2022). *papaja: Prepare reproducible APA journal articles with R Markdown*. Retrieved from https://github.com/crsh/papaja

Bates, D., Mächler, M., Bolker, B., & Walker, S. (2014). Fitting linear mixed-effects models using lme4. *arXiv Preprint arXiv:1406.5823*.

Baxter, R., Hong, N. C., Gorissen, D., Hetherington, J., & Todorov, I. (2012). The research software engineer. *Digital Research Conference, Oxford*, 1–3.

Brandmaier, A. M. (2015). pdc: An R package for complexity-based clustering of time series. *Journal of Statistical Software*, *67*(5), 1–23. https://doi.org/10.18637/jss.v067.i05

Climate Code Foundation. (2011). *Science code manifesto*. Retrieved from http://web.archive.org/web/20201112032938/http://sciencecodemanifesto.org/

Gärtner, A., Leising, D., & Schönbrodt, F. (2022). *Responsible research assessment II: A specific proposal for hiring and promotion in psychology*.

Montero, P., & Vilar, J. A. (2014). TSclust: An R package for time series clustering. *Journal of Statistical Software*, *62*(1), 1–43. Retrieved from http://www.jstatsoft.org/v62/i01/

Peikert, A., Van Lissa, C. J., & Brandmaier, A. M. (2021). Reproducible research in r: A tutorial on how to do the same thing more than once. *Psych*, *3*(4), 836–867.

Schönbrodt, F., Gärtner, A., Frank, M., Gollwitzer, M., Ihle, M., Mischkowski, D., et al.others. (2022). *Responsible research assessment I: Implementing DORA for hiring and promotion in psychology*.

Wickham, H. (2011). Testthat: Get started with testing. *The R Journal*, *3*, 5–10. Retrieved from https://journal.r-project.org/archive/2011-1/RJournal_2011-1_Wickham.pdf