Ten Simple Rules for Writing a Comparative Software Review

Authors:
- Amy Beeston, Department of Computer Science, University of Sheffield orcid.org/0000-0003-2796-1947
- Larisa Blazic, School of Media, Arts and Design, University of Westminster
- Neil Chue Hong, Software Sustainability Institute, University of Edinburgh orcid.org/0000-0002-8876-7606
- Richard Domander, Comparative Biomedical Sciences, Royal Veterinary College
- Ross Mounce, Department of Plant Sciences, University of Cambridge orcid.org/0000-0002-3520-2046
- Robin Wilson, Geography & Environment, University of Southampton orcid.org/0000-0001-7352-8912

Place, *Sustainable Software Institute Collaborations Workshop 2016, Edinburgh, UK*
Date: *23/03/2016*

Contributing to this draft:
The authors encourage the community to contribute to the ongoing development of this paper. People can suggest addition and changes, or provide comments via the Google Doc at http://bit.ly/compsoftrev - please identify your contributions if you want them to be acknowledged.

Abstract:

We provide the following set of rules as a framework for researchers in any domain to undertake a comparative software review and determine the best software for their task.

Rule 1: State your credentials and motivation
Rule 2: Define and justify your scope
Rule 3: Perform a comprehensive search
Rule 4: Make your longlist data available to others
Rule 5: Summarise your software shortlist
Rule 6: Define the software quality criteria
Rule 7: Define the task suitability criteria
Rule 8: Mind the gaps
Rule 9: Summarise the findings as clearly as possible
Rule 10: Involve your community

Introduction

Comparative reviews of scientific software are not a new thing, as shown for instance by the work of Schroeder (1969) and Frances (1979). However, in more recent decades software and tools have proliferated hugely, which can make it difficult for researchers to understand which software is most appropriate for a particular task. While literature review papers are common in most discipline areas, enabling researchers to understand which are the most important papers for them to consider, there are fewer software review papers which compare the different tools available for a given task.

Writing a good comparative software review can be difficult, since to perform an objective review with clear scope and well-defined comparison criteria, reviewers should ideally have extensive experience in using each of the possible software packages for the particular task in question. Despite this difficulty, it is important for individual researchers to share lessons learned from their practical experiences using software with the wider academic community, because the majority of researchers rely on software to do their work (Hettrick, 2014). Encouragingly, this has prompted an increase in comparative software review papers in certain disciplines.

For example, in the field of audio engineering, two very recent papers compare tools used for room acoustic characterisation (Álvarez-Morales et al, 2016; Cabrera, Xun and Guski, 2016). A further two contemporaneous audio software reviews make plain the differing approaches that authors may be required to take. At one extreme, a narrow focus may be required for technically detailed or closely-scoped tasks, for example, in evaluating background noise levels in a space (Cuff, 2016). At the other extreme, a broad overview of related software packages may be more relevant when the tasks themselves are more likely to vary from project to project, e.g. in the varied types of annotation that audio signals may require (Finlayson & Erjavec, 2016).

To date, the academic community has not yet described in detail the steps required to approach and carry out a comparative software review. In the spirit of the existing papers mentioned above, we therefore offer Ten Simple Rules for writing a comparative software review which we hope will assist others in producing such documents. The resulting reviews can then be used by other researchers to make sense of the software in the area, to identify software they may not otherwise have been aware of, to help determine which criteria are important to them, and ultimately to decide which software to use. Additionally, such reviews can also be used by the author to identify important gaps in provision, for instance a lack of open-source alternatives or a lack of software available for common platforms.

Rule 1: State your credentials and motivation

We recommend placing a credentials and motivation section at the start of your paper. It is important for readers to understand why they should trust the information provided in your comparative software review paper. You should honestly state your:

1. Background and credentials: Why are you qualified to write this paper? What is your experience? Conversely you should make it clear if this is not an expert review.
2. Viewpoints: What is your bias or special interest? For example, do you work exclusively with open source software?
3. Motivation: Why are you writing the review? Are you trying to justify your choice of tools, or to explain why you should write a new piece of software?

Rule 2: Define and justify your scope

It is important to clearly define the scope of the review. In particular, exclusion criteria will help to focus your paper, and make it easier to narrow down the potential field of candidates for comparison.

1. State the scope of your review clearly, listing your inclusion and exclusion criteria and providing justification for your scope.
2. Explain what you have chosen to include in your review, and why.
3. Are exclusion criteria based on fitness for use (e.g. functionality, minimum performance, documentation, license), domain knowledge (e.g. commonly used platforms), community health (e.g. currently maintained, regular commits) or personal preference (e.g. openly licensed)?
4. Sometimes it might be appropriate to apply an arbitrary cut-off to make the paper easier to write and more useful to readers, e.g. “new software released in the past three years”.

Rule 3: Perform a comprehensive search

It can be difficult to perform an exhaustive search of software, since the law of diminishing returns suggests that you may expend significant effort to identify only a few additional pieces of software. Compared to literature reviews, the tools (such as reference managers) and norms (such as how items are cited) relating to software are not as well-developed. Nevertheless, there are approaches you can take to ensure you have given appropriate attention to the search for software to include in your review.

1. Keep track of the search methods (and parameters) you use, and record instances of software in a sensible way e.g. tabular data (CSV, spreadsheet),
    a. Record the software name and the link to where you found it
    b. Start recording data relating to your comparison criteria
2. Use a variety of mechanisms to search for software

a. Review recent literature in the field: look for software which has been mentioned in key papers.
b. Look for existing comparative software papers: these may be in related areas.
c. Search language or domain-specific repositories, such as the Python Packaging Index (PyPI), the Comprehensive R Archive Network (CRAN) and so on.
d. Search GitHub and BitBucket: these are currently the largest repositories of scientific software.
e. Ask on academic mailing lists: this can be useful for identifying ‘tacit’ software which has not been officially released but may be in wide use.
f. Use generic internet search engines, keeping track of which search terms you have used.
g. Write up your search methodology in your paper, recognising this can never be truly objective. It can be useful to include instructions for readers or developers to alert you of any software you have missed so that it can be included in a future version of the review.

Rule 4: Make your longlist data available to others

Your comparative software review will not evaluate all the software you discovered in your search in detail, due to the exclusion criteria mentioned above. Nonetheless, providing the list of all the software you discovered, how they were found, why they were included in or excluded from the review, and any data you collected about the software during the search process can be useful to others who may wish to conduct their own review with a different scope. This data can be provided as an appendix or in a supplementary materials section.

Rule 5: Summarise the software shortlist

Provide the readers with a concise overview of the software that you are going to consider in greater detail. List the chosen software in a tabular format, listing its name, complete version number, date published, authors, and website URL, and full citation (Jackson, 2016).

Rule 6: Define the software quality criteria

Before you conduct the comparison itself, define the criteria, methods and benchmarks you will apply to the software shortlist in your comparative review, . Some of these measures can applied to any piece of software, giving an objective indicator of the “quality” of the software. Other measures will relate to the task you have in mind (see rule 7 below). Some key software quality criteria which might be relevant (with examples in brackets) include:

1. The *availability* of the software (as defined by its license, installation format) indicates how easy it is for a researcher to download and start using a piece of software.
2. The *robustness* of the software (as evidenced, for example, by its documentation, unit tests, positive user reviews).

3. The *size of the community* (as indicated by citations or *GitHub* stars) will give an idea of whether the software is currently being maintained, or is likely to become deprecated at short notice.
4. The *health of the community* (as indicated by communication, number of (happy) developers, frequency of releases, bug fixes, and commits) will also indicate whether there is a source of help beyond the documentation provided with the software itself.
5. The *technical requirements* (including supported hardware/software platforms and data formats, software dependencies, and programming languages) indicate there are likely issues when interfacing with other software.
6. The *performance* (as measured through benchmark tests or datasets) of the software will identify its support for missing - or massive - data, any known limitations, and likely compute power required. A variety of tests, parameters and/or input data are needed because some scientific programs are designed to optimise performance for specific use-cases and may perform relatively poorly in others - thus variety is needed to demonstrate true strengths and weaknesses

Rule 7: Define the task-suitability criteria

You should also consider a wider range of factors which determine the fitness for purpose of the shortlisted software candidates to the actual task at hand. That is, the criteria considered under rule 6 above should be weighted by a second set of criteria which establishes their relative importance in this *particular* use case.

Each comparative software review will involve a unique combination of personal preferences. Thus all stakeholders (researchers, developers, project team members, and possibly of target consumers of the project outcomes too) must be identified and consulted before any new software is adopted.

Factors to consider include:

1. Security and integrity concerns of the software, relative to the project aims and objectives.
2. Costs of purchasing or licensing the software, relative to project budgets more widely.
3. Computational functionality - the software should have necessary and sufficient capabilities, without being bloated by unused functionality.
4. Computational efficiency - ensure the software can run in a reasonable time on the machines available (for instance, whether it will require a high performance cluster or GPU setup).
5. Whether the prevalence of the software in the relevant community (Joppa, 2013), and the level of documentation provided is sufficient for the proposed userbase.
6. The fluency of any developers with the language in which the software is written, as this will affect the time taken to install and deploy the software in question.
7. The usability and acceptable steepness of the learning curve. If usability is the prime factor in your software comparison then it may be advisable to examine this point in

far greater detail (see e.g. Bax et al (2007) who gathered feedback from 30 independent researchers on the usability of various different programs).

It is inevitable that these factors will be influenced by psychological factors beyond the scope of this paper. Nonetheless, even a naive attempt will allow you to identify trade-offs between criteria under Rule 7. For example, a balance exists between the fluency of an individual developer with a given piece of software versus its relative *un*popularity in the community, since little support would be needed if the developer is already fluent with the selected software.

Rule 8: Mind the gaps

Comparing software in a way that highlights their strengths and weaknesses against your chosen criteria can be a good way of identifying situations for which no suitable software exists. When you are reporting the results of your comparison based on the quality and suitability criteria, don't forget to discuss any gaps and absences you discover. For instance, if there is no good open source software available for a given task, then this should be highlighted and discussed.

Rule 9: Summarise the findings as clearly as possible

Present your findings in such a way that the reader may quickly and intuitively understand which is the best tool for their job. There is no hard and fast rule about the best approach to take. Rather, the presentation should be sensitive to the domain and task in question, and to the number of possible options which remain at the end of your analysis.

If relatively few software options have survived the culling and weighting of the earlier rules, then it may be sufficient to tabulate results and provide these remaining candidates in a checklist for side-by-side comparison. Furthermore, if just a few key differences separate these options, it can be instructive to implement a flow-chart or walk-through approach, so that the single best choice can be selected in conversation with relevant stakeholders.

For cases where a larger number of possible software candidates still remain, it may be more instructive to provide a visualisation (Tufte, 2001) of the comparative software review results. For example, software candidates may be mapped onto particular locations on a three-dimensional terrain, where the three selected dimensions represent software-quality factors which received the highest suitability-weightings and which differ among the candidates. An alternative to this would be to follow an interactive “recipe” style, where the “ingredients” (personal priorities) you have can help narrow down the search among candidates.

Rule 10: Involve your community

While writing, be conscientious and objective at all times. The reviewed software might be the developer’s life work, and criticism may not be taken kindly. Ensure everything you write is as considerate as possible. Consider asking the developer for their opinion on anything about their software that you are unsure of.

To help ensure you haven’t overlooked anything in your review, ask your peers and colleagues to read and comment on your work. In addition, consider hosting an early version of your review on a pre-print server to seek feedback from a wider audience. Remember, it may also be necessary to alert the appropriate developer and researcher communities to the existence of your work by using appropriate mailing lists and forums.

Conclusions

Despite the proliferation of software in research, we lack tools for comparing different software options and judging which is most appropriate for a particular task. This paper discusses the issues involved in writing a comparative software review, and provides a set of ten simple rules which allow the reader to undertake such a review. If followed, the resulting review should have a clear scope, explicitly define the quality and suitability comparison criteria, and should present its findings concisely, and with clarity.

Acknowledgements

Many thanks to James Baker for his helpful discussion, and to the Software Sustainability Institute (www.software.ac.uk) for bringing the authors together in Edinburgh for the Collaborations Workshop 2016, where this paper was started.

References

Álvarez-Morales, L.; Galindo, M.; Girón, S.; Zamarreño, T.; Cibrián, R. M. 2016. Acoustic Characterisation by Using Different Room Acoustics Software Tools: A Comparative Study. *Acta Acustica united with Acustica* 102(3), 578-591(14). http://dx.doi.org/10.3813/AAA.918975

Bax, L., Yu, L.-M. M., Ikeda, N., and Moons, K. G. 2007. A systematic comparison of software dedicated to meta-analysis of causal studies. *BMC medical research methodology* 7:40. http://dx.doi.org/10.1186/1471-2288-7-40

Cabrera, D., Xun, J., and Guski, M. 2016. Calculating Reverberation Time from Impulse Responses: A Comparison of Software Implementations. *Acoustics Australia.* Technical Note. 1-10. http://dx.doi.org/10.1007/s40857-016-0055-6

Cuff, N. 2016. Predicting and controlling heating, ventilating, and air conditioning systems noise: A comparison of computer noise control prediction tools and their potential pitfalls. The Journal of the Acoustical Society of America, 139, 2059-2059. http://dx.doi.org/10.1121/1.4950101

Finlayson, M. A. and Erjavec, T. 2016. Overview of Annotation Creation: Processes & Tools. To appear in James Pustejovsky & Nancy Ide (2016) *Handbook of Linguistic Annotation.* New York: Springer https://arxiv.org/pdf/1602.05753.pdf

Francis, I. 1979. *A comparative review of statistical software.* International Association for Statistical Computing, Voorburg, Netherlands.

Hettrick, S. 2014. *It's impossible to conduct research without software, say 7 out of 10 UK researchers*. http://www.software.ac.uk/blog/2014-12-04-its-impossible-conduct-research-without-software-say-7-out-10-uk-researchers (accessed 27 June 2016).

Jackson, M. 2016. *How to cite and describe software*. http://www.software.ac.uk/how-cite-and-describe-software (accessed 27 June 2016).

Joppa, L. N. et al. 2013. Troubling trends in scientific software use. *Science*. 340 (6134): 814-815. http://dx.doi.org/10.1126/science.1231535

Schroeder, M. R. 1969. Computers in acoustics: Symbiosis of an old science and a new tool. *The Journal of the Acoustical Society of America*, 45, 1077-1088. http://dx.doi.org/10.1121/1.1911577

Tufte, E. R. 2001. *The visual display of quantitative information* (second edition). Graphics Press, Cheshire, Connecticut.