How to Read a Research Compendium

Daniel Nüst[1]

[1]**Institute for Geoinformatics, University of Münster, Münster, Germany**

Corresponding author:
Daniel Nüst[1]

Email address: daniel.nuest@uni-muenster.de

ABSTRACT

Researchers spend a great deal of time reading research papers. Keshav (2012) provides a three-pass method to researchers to improve their reading skills. This article extends Keshav's method to work for research compendia (RC). RC are an increasingly used form of publication packaging not only the research paper text and figures, but also all data and software used to conduct the computational workflow and create all outputs. We list the existing conventions for RC and suggest how to utilise their shared properties in a structured reading process. Unlike the original, this article is not build upon a long history but intends to provide guidance at the outset of an emerging practice.

1. INTRODUCTION

1.1 Motivation

Research compendia (RC) are an increasingly used form of publications. They comprise not only the research paper text and figures, but also all data and software used to conduct the computational workflow and create all outputs. They provide a lot of added value but potentially also increase complexity for understanding, if not done well. Therefore this work extends Keshav's three-pass method targeted at improving researcher's research paper reading skills (Keshav 2007) with additional steps to capture an RC's content.

Unlike the first version of the original (Keshav 2007), we cannot draw from a long history of experience. But we intend to provide guidance at the outset of an emerging practice to both authors and readers of research compendia to understand each others perspectives and needs. Authors may use this guide to improve their RC's structure and content, as they can better prepare for how readers may approach their work. Readers may avoid the trap of falling too deep into technology challenges by an iterative reading approach. This work may also improve the interaction between authors and mitigate unwarranted concerns, like giving support (Barnes 2010). Ultimately RCs can enhance and deepen the reading experience, if done right. Keshav's following introduction applies directly to RCs:

> *Researchers must read papers for several reasons: to review them for a conference or a class, to keep current in their field, or for a literature survey of a new field. A typical researcher will likely spend hundreds of hours every year reading papers.*
>
> *Learning to efficiently read a paper is a critical but rarely taught skill. Beginning graduate students, therefore, must learn on their own using trial and error. Students waste much effort in the process and are frequently driven to frustration.*
>
> *For many years I have used a simple 'three-pass' approach to prevent me from drowning in the details of a paper before getting a bird's-eye-view. It allows me to estimate the amount of time required to review a set of papers. Moreover, I can adjust the depth of paper evaluation depending on my needs and how much time I have. This paper describes the approach and its use in doing a literature survey.* (Keshav 2016)

The additions made in this work to accomodate for the content in a research compendium are quite extensive. At first glance this is naturally due to the complexity that an interactive compendium has compared to static classic "paper". A research compendium goes well beyond the "mere advertising of the

scholarship" (Claerbout 1994). Therefore we see the breadth of additions as a sign of potential, namely for unprecedented openness and collaboration.

1.2 Structure

In the remainder of this paper, the excellent original work that is taken over verbatim is in *italic font* based on the most recent online version: Keshav (2016). The term "paper" was not replaced with "research compendium" if no other changes were made in a paragraph.

First we briefly introduce research compendia and existing conventions, and also point out relevant publications as well as examples for authors of RCs. Then, matching the original paper's section numbering, Sections 2 extends the "Three-pass Approach" to include RC features in the reading process. Section 3 extends "Doing a Literature Survey" with aspects relevant reviewing many RCs.

1.3 Research compendia

The term *research compendium* was coined by Gentleman and Lang (2007) who "introduce[d] the concept of a compendium as both a container for the different elements that make up the document and its computations (i.e. text, code, data,...), and as a means for distributing, managing and updating the collection." According to (Marwick, Boettiger, and Mullen 2018) it provides "a standard and easily recognisable way for organising the digital materials of a research project to enable other researchers to inspect, reproduce, and extend the research". This standard may differ between scientific domains, yet the intentions and benefits are the same. RC improve transparency and quality of research [REF], enable enhanced review and publication workflows (Nüst et al. 2017), and answer readers' needs to understand complex analyses (Konkol and Kray 2018). RC improve citations since code and data are openly available (Vandevalle 2012). Ultimately, their goal is to improve reproducibility (see Barba (2018) for definitions of terms) in the light of a "reproducibility crisis".

As this article should provide hands-on guidance on using, and to some extend also creating, research compendia, we refer the reader to the references for details. All of them provide more information than the specific aspect they are mentioned for in this article. Instead, for the remainder of this work, we assume a minimal view on a research compendium suitable for *readers* (for a more detailed discussion see Marwick, Boettiger, and Mullen (2018), Gentleman and Lang (2007)) who examine a research compendium manually. An RC has three integral parts: text, code, and data. Text can be instructions, software documentation, or a full manuscript. Code can be scripts, software packages, specifications of dependencies and computational environments, or even virtual machines. Data can be just about anything, but probably comprises inputs and outputs.

For *authors*, there is a wealth of generic recommendations guiding researchers in creating open research software (Sandve et al. 2013,Taschuk and Wilson (2017),Prlić and Procter (2012),Stodden and Miguez (2014),Wilson et al. (2017)). When a research compendium is published, we assume the author has the intention to help the reader understanding the work and accepts there are "no excuses" to not publishing your code (Barnes 2010). An author may attempt to reach the ideal of having one "main" file that can be executed with "one-click" (Pebesma 2013), enabling re-use with proper licensing (Stodden 2009), and interweaving code and text following the literate programming paradigm (Knuth 1984). The following conventions are specifically for research compendia:

- Marwick, Boettiger, and Mullen (2018) discuss the structure and tooling of the R programming language and software engineering tools for a variety of disciplines with real-world examples, including several templates for R
- https://github.com/ropensci/rrrpkg
- popper: http://falsifiable.us/ and Jimenez et al. (2017)
- R and Python packages (Gentleman and Lang 2007), also npm packages?
- Researchcompendia.org (Stodden, Miguez, and Seiler 2015)

2. THE THREE-PASS APPROACH

> *The key idea is that you should read the paper in up to three passes, instead of starting at the beginning and plowing your way to the end. Each pass accomplishes specific goals and builds upon the previous pass: The first pass gives you a general idea about the paper. The second pass lets you grasp the paper's content, but not its details. The third pass helps you understand the paper in depth.* (Keshav 2016)

The fourth pass gets you started on building upon and extending the paper.

2.1 The first pass

> *The first pass is a quick scan to get a bird's-eye view of the paper. You can also decide whether you need to do any more passes. This pass should take about five to ten minutes and consists of the following steps:* (Keshav 2016)

1. *Carefully read the title, abstract, and introduction*
2. *Read the section and sub-section headings, but ignore everything else*
3. *Glance at the mathematical content (if any) to determine the underlying theoretical foundations*
4. *Read the conclusions*
5. *Glance over the references, mentally ticking off the ones you've already read*
6. Glance over the text looking for (a) URLs and `formatted` **names** referencing software and data (repositories) not yet mentioned in the sections read so far, mentally ticking off the ones you've heard about or used, and (b) tables or figures describing computational environments, deployments, or execution statistics

At the end of the first pass, you should be able to answer the seven *Cs:*

1. *Category: What type of paper is this? A measurement paper? An analysis of an existing system? A description of a research prototype?*
2. *Context: Which other papers is it related to? Which theoretical bases were used to analyze the problem?*
3. *Correctness: Do the assumptions appear to be valid?*
4. *Contributions: What are the paper's main contributions?*
5. *Clarity: Is the paper well written?*
6. Construction: What are the building blocks of the analysis workflow and how accessible are they (data set(s), programming language(s), tools, algorithms, scripts)? Under what licenses is the code published?
7. Complexity: What is the scale of the analysis (e.g. HPC, required OS/cores/memory, typical execution time, data size) and the software (number of dependencies and is installation possible with dependency management tools)?

> *Using this information, you may choose not to read further (and not print it out, thus saving trees). This could be because the paper doesn't interest you, or you don't know enough about the area to understand the paper, or that the authors make invalid assumptions.* (Keshav 2016)

You may also choose not to pursue the parts of the RC further, i.e. not looking at data, code, or running the workflow, thus saving resources. Reasons related to code and data may be that you don't know have the expertise or resources to re-use data and code with acceptable efforts.

> *The first pass is adequate for papers that aren't in your research area, but may someday prove relevant.* (Keshav 2016)

It suits RC comprising potentially re-usable components, like workflows or algorithms using data sets or generic software transferable to your field of research. After the first pass, you should be able to judge if the software is useful if it works.

> *Incidentally, when you write a paper, you can expect most reviewers (and readers) to make only one pass over it. Take care to choose coherent section and sub-section titles and to write concise and comprehensive abstracts. If a reviewer cannot understand the gist after one pass, the paper will likely be rejected; if a reader cannot understand the highlights of the paper after five minutes, the paper will likely never be read.* For these reasons, a 'graphical abstract' that summarizes a paper with a single well-chosen figure is an excellent idea and can be increasingly found in scientific journals._ (Keshav 2016)

Take also care to add instructions to reproduce your work and provide all required parts, i.e. publish a research compendium. The instructions should start with a "blank" system to give readers an ideas what is needed to recreate your environment and execute the analysis, including expected or experienced execution times and resources. If your work requires specialised or bespoke hardware (HPC, specific GPUs), consider creating an exemplary (reduced) analysis that runs in regular environments.

Also ensure your code and data is properly deposited, citable and licensed, or these core parts of your work will likely never be evaluated or re-used. See Section "Research Compendia" for recommendations to make your reviewer's and reader's life easier. However, every little bit helps and you should "find what works for you" even in reproducible research (Whitaker 2017). It is better to come up with your own structure or solution, than providing no reproducible code or data at all.

2.2 The second pass

> *In the second pass, read the paper with greater care, but ignore details such as proofs. It helps to jot down the key points, or to make comments in the margins, as you read. Dominik Grusemann from Uni Augsburg suggests that you "note down terms you didn't understand, or questions you may want to ask the author." If you are acting as a paper referee, these comments will help you when you are writing your review, and to back up your review during the program committee meeting.* (Keshav 2016)

1. *Look carefully at the figures, diagrams and other illustrations in the paper. Pay special attention to graphs. Are the axes properly labeled? Are results shown with error bars, so that conclusions are statistically significant? Common mistakes like these will separate rushed, shoddy work from the truly excellent.* (Keshav 2016)
2. *Remember to mark relevant unread references for further reading (this is a good way to learn more about the background of the paper).* (Keshav 2016)
3. Skim over data and source code files. Are they reasonably named, or do they even follow a well-defined structure (e.g. a Python package or a research compendium convention)? Do they have comments? Is there a README file and/or structured documentation for functionalities?
4. Visit the source code repository, if available. Is it established and well maintained, or orphaned? Is there only one author or are there contributors? How responsive are they to issues? Does the repository have "stars" (i.e. public bookmarks) or forks? Are there regular releases, using semantic versioning?
5. Follow the instructions to execute the RC's workflow and compare the outputs with the ones reported in the paper. Note down errors or warnings during execution but do not try to fix any but trivial or known problems.
6. Check for differences in output figures. Do labels, legends etc. match?

The listing above gives a lot of examples how to estimate the quality of a software, but we recommend to be realistic as to what to expect and careful not to judge too fast. The software project you evaluate might be done by a researcher under a lot of pressure and have only a single user and use case. It is OK not to document beyond the information required by the author, and the documentation might be provided quickly once you as an external user show interest. Also, no recent changes or releases can also mean the software is stable and simply works!

> *The second pass should take up to an hour for an experienced reader.* (Keshav 2016)

This does not include the computation time of workflows in an RC. Use this time to complete first passes for one or several other papers. If the used software is familiar, you may attempt to reduce the computation time by subsetting data or simplifying the workflow. As an author, consider adding a reduced example to your RC for easier access by readers.

You may do quick or obvious fixes that prohibit a successful workflow execution, like fixing a path or install an undocumented dependency. However we do recommend not to dive too deep - it is the author's responsibility to guide you through her work. You may also face unsolvable problems, like access to specific infrastructure. But you should report issues, for example in the software's public code repository, if available.

> *After this pass, you should be able to grasp the content of the paper.* (Keshav 2016)

You should have re-executed the provided workflow or understand why you could not. At this point you should be able to judge whether you might want to re-use parts of the analysis, i.e. software or data, for your own work.

> *You should be able to summarize the main thrust of the paper, with supporting evidence, to someone else. This level of detail is appropriate for a paper in which you are interested, but does not lie in your research speciality. Sometimes you won't understand a paper even at the end of the second pass. This may be because the subject matter is new to you, with unfamiliar terminology and acronyms. Or the authors may use a proof or experimental technique that you don't understand, so that the bulk of the paper is incomprehensible. The paper may be poorly written with unsubstantiated assertions and numerous forward references.* (Keshav 2016)

The RC may have incomplete documentation, rely on unavailable (e.g. proprietary) software or (e.g. sensitive) data, or require infrastructure not available to you. It may use a programming language or programming paradigms unknown to you.

> *Or it could just be that it's late at night and you're tired. You can now choose to: (a) set the paper aside, hoping you don't need to understand the material to be successful in your career, (b) return to the paper later, perhaps after reading background material or (c) persevere and go on to the third pass.* (Keshav 2016)

2.3 The third pass

> *To fully understand a paper, particularly if you are a reviewer, requires a third pass. The key to the third pass is to attempt to virtually re-implement the paper: that is, making the same assumptions as the authors, re-create the work. By comparing this re-creation with the actual paper, you can easily identify not only a paper's innovations, but also its hidden failings and assumptions. This pass requires great attention to detail.* (Keshav 2016)

If a best practice for structuring data and code was followed, familiarise yourself with it now.

> *You should identify and challenge every assumption in every statement. Moreover, you should think about how you yourself would present a particular idea. This comparison of the actual with the virtual lends a sharp insight into the proof and presentation techniques in the paper and you can very likely add this to your repertoire of tools.* (Keshav 2016)

Take a close look at data, metadata, and code. This can be a time consuming very close study of the materials. If data is not publicly available, e.g. because it may contain identifying information about humans, decide if you have a reasonable request to contact the original authors and ask for data access. Work though the examples and analysis scripts included in the RC. Play close attention not only to code, but also to code comments as they should include helpful information. A good entry point for your code read may be a "main" script (if provided by the author) or literate programming document. If that is not given then start with the code creating the figures for the article (e.g. look for "`plot`" statements in the code) and trace your way back through the code until you reach a statement where the input data is read. Your impression on the code may support or refute your impression of the quality of the article's text.

If you did not succeed before but the work is relevant for you, spend more time on getting the analysis to run on your computer and do not hesitate to contact the author of the paper of authors of used software. For them it is a great experience to be contacted by an interested reader!

With respect to the analysis, you may re-implement core parts or the full workflow with a different software. For example using a tool one you know but which was not used in the RC. Does your code lead to the same results, or does it give different ones? Can the differences be explained or are they not significant? Note that such a replication is of very high value for scholarly publishing and you should share your results with the RC's authors and also with the scientific community. Depending on the efforts you put in, write a blog post or even publish a replication ~~paper~~ research compendium for one or more evaluated RCs.

If a full replication is not feasible, explore the assumptions you challenge with data and code. Play around with input parameters to get a feel for the changing results. Create explorative plots for the data as

if you would want to analyse it, without the knowledge of the existing workflow. With your understanding of the code now you can also extend the application to a new problem or different dataset. This deep evaluation of code and data increases your understanding of the author's reasoning and decisions, and may lead to new questions.

To make sure you can trace your own hands-on changes with the original code and configuration, we recommend initiating a local git repository. You can create branches for specific explorations and easily reset to the original functional state.

> *During this pass, you should also jot down ideas for future work. This pass can take many hours for beginners and more than an hour or two even for an experienced reader. At the end of this pass, you should be able to reconstruct the entire structure of the paper from memory, as well as be able to identify its strong and weak points. In particular, you should be able to pinpoint implicit assumptions, missing citations to relevant work, and potential issues with experimental or analytical techniques.* (Keshav 2016)

You should be able to come up with useful extensions of the used software stack and be able to judge the transferability and reusability of the analysis' building blocks. You should most certainly have improved your programming skills by reading and evaluating other people's code or even trying to apply it to a new problem.

3. DOING A LITERATURE SURVEY

> *Paper reading skills are put to the test in doing a literature survey. This will require you to read tens of papers, perhaps in an unfamiliar field. What papers should you read? Here is how you can use the three-pass approach to help.* _First, use an academic search engine such as Google Scholar or CiteSeer and some well-chosen keywords to find three to five recent highly-cited papers in the area. (Keshav 2016)

No search capability comparable to scientific articles exists for RC, though you can of course use generic and academic search engines. More and more journals encourage reproducible research and software and data publication, so that extending your search regular search with keywords such as "reproduction", "reproducible", "open data/software/code" may give improve your results.

In addition, you can search online platforms where RC may be published and tagged as an RC (`research-compendium`):

- GitHub label: https://github.com/topics/research-compendium
- Zenodo community: https://zenodo.org/communities/research-compendium

There is no journal specifically for research compendia yet, but the following ones feature reproducibility, computational studies, or openness in some way and can be a starting point if they fit your topic:

- *ReScience*: https://rescience.github.io/
- *Information Systems* has a reproducibility editor and special track for invited reproducibility papers: https://www.journals.elsevier.com/information-systems/

A lateral approach takes advantage of the parts of a RC. If you work with a specific software (tool, extension package, library) or data, find out the recommended way to cite it (and follow it yourself). Most scientific software provides this information in their FAQ or might have a build in function to generate a citation, and scientific data is often accompanied by a "data paper" or published in repositories with citeable identifiers.

Then search for recent papers which cite the core software or data of interest.

> *Do one pass on each paper to get a sense of the work, then read their related work sections. You will find a thumbnail summary of the recent work, and perhaps, if you are lucky, a pointer to a recent survey paper. If you can find such a survey, you are done. Read the survey, congratulating yourself on your good luck. Otherwise, in the second step, find shared citations and repeated author names in the bibliography. These are the key papers and researchers in that area.*

You can also find shared software or data and use them as a seed for a next iteration.

> *Download the key papers and set them aside. Then go to the websites of the key researchers and see where they've published recently. That will help you identify the top conferences in that field because the best researchers usually publish in the top conferences.*

Also check where they publish their code and data. It will give you an idea where this community interacts online and even lead you to RC under development.

> *The third step is to go to the website for these top conferences and look through their recent proceedings. A quick scan will usually identify recent high-quality related work. These papers, along with the ones you set aside earlier, constitute the first version of your survey. Make two passes through these papers. If they all cite a key paper that you did not find earlier, obtain and read it, iterating as necessary.* (Keshav 2016)

If a majority cites or uses a key software, technology, or dataset, evaluate it.

4. RELATED WORK

> *If you are reading a paper to do a review, you should also read Timothy Roscoe's paper on "Writing reviews for systems conferences" (Roscoe 2007). If you're planning to write a technical paper, you should refer both to Henning Schulzrinne's comprehensive web site (Schulzrinne 2018) and George Whitesides's excellent overview of the process (Whitesides 2004). Finally, Simon Peyton Jones has a website that covers the entire spectrum of research skills (Peyton Jones 2018). Iain H. McLean of Psychology, Inc. has put together a downloadable 'review matrix' that simplifies paper reviewing using the three-pass approach for papers in experimental psychology (McLean 2012), which can probably be used, with minor modifications, for papers in other areas.* (Keshav 2016)

We have extended this matrix to provide space for notes about software, data, results of the reproduction, and application of the methods.

WIP: https://github.com/nuest/how-to-read-a-research-compendium/issues/2

Infrastructures supporting the creating and inspection of research compendia is an active field of research, but none of which have been widely deployed yet (Nüst et al. (2017),Brinckman et al. (2018),Stodden, Miguez, and Seiler (2015),Kluyver et al. (2016)).

If you are reviewing a research compendium, a more detailed checklist is given in the "rOpenSci Analysis Best Pratice Guidelines" (rOpenSci 2017), which are partially even automated for R-based research compendia (DeCicco et al. 2018).

5. ACKNOWLEDGMENTS

> *The first version of this document was drafted by my students: Hossein Falaki, Earl Oliver, and Sumair Ur Rahman. My thanks to them. I also benefited from Christophe Diot's perceptive comments and Nicole Keshav's eagle-eyed copy-editing. I would like to make this a living document, updating it as I receive comments. Please take a moment to email me any comments or suggestions for improvement. Thanks to encouraging feedback from many correspondents over the years.* (Keshav 2016)

In the spirit of the original paper, we would like to make this a living document and invite readers to provide comments or suggestions for improvement via email, as part of this preprint, or on the GitHub repository: https://github.com/nuest/how-to-read-a-research-compendium. The repository also includes open questions and is where the paper's authors openly discuss.

REFERENCES

Barba, Lorena A. 2018. "Terminologies for Reproducible Research." *arXiv:1802.03311 [Cs]*, February. http://arxiv.org/abs/1802.03311.

Barnes, Nick. 2010. "Publish Your Computer Code: It Is Good Enough." *Nature News* 467 (7317): 753–53. doi:10.1038/467753a.

Brinckman, Adam, Kyle Chard, Niall Gaffney, Mihael Hategan, Matthew B. Jones, Kacper Kowalik, Sivakumar Kulasekaran, et al. 2018. "Computing Environments for Reproducibility: Capturing the 'Whole Tale'." *Future Generation Computer Systems*, February. doi:10.1016/j.future.2017.12.029.

Claerbout, Jon. 1994. "Seventeen Years of Super Computing." http://sepwww.stanford.edu/sep/jon/nrc.html.

DeCicco, Laura, Noam Ross, Alice Daish, Molly Lewis, Nistara Randhawa, Carl Boettiger, Nils Gehlenborg, Jennifer Thompson, and Nicholas Tierney. 2018. "Checkers: Automated Checking of Best Practices for Research Compendia." rOpenSci Labs. https://github.com/ropenscilabs/checkers.

Gentleman, Robert, and Duncan Temple Lang. 2007. "Statistical Analyses and Reproducible Research." *Journal of Computational and Graphical Statistics* 16 (1): 1–23. doi:10.1198/106186007X178663.

Jimenez, I., M. Sevilla, N. Watkins, C. Maltzahn, J. Lofstead, K. Mohror, A. Arpaci-Dusseau, and R. Arpaci-Dusseau. 2017. "The Popper Convention: Making Reproducible Systems Evaluation Practical." In *2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 1561–70. doi:10.1109/IPDPSW.2017.157.

Keshav, S. 2007. "How to Read a Paper." *SIGCOMM Comput. Commun. Rev.* 37 (3): 83–84. doi:10.1145/1273445.1273458.

———. 2016. "How to Read a Paper." Manuscript. Waterloo, ON, Canada. http://blizzard.cs.uwaterloo.ca/keshav/home/Papers/data/07/paper-reading.pdf.

Kluyver, Thomas, Benjamin Ragan-Kelley, Fernando Pérez, Brian Granger, Matthias Bussonier, Jonathan Frederic, Kyle Kelley, et al. 2016. "Jupyter Notebooks - a Publishing Format for Reproducible Computational Workflows." *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, 87–90. doi:10.3233/978-1-61499-649-1-87.

Knuth, Donald E. 1984. "Literate Programming." *Comput. J.* 27 (2): 97–111. doi:10.1093/comjnl/27.2.97.

Konkol, Markus, and Christian Kray. 2018. "In-Depth Examination of Spatio-Temporal Figures in Open Reproducible Research." *EarthArXiv*, April. doi:10.17605/OSF.IO/Q53M8.

Marwick, Ben, Carl Boettiger, and Lincoln Mullen. 2018. "Packaging Data Analytical Work Reproducibly Using R (and Friends)." *The American Statistician* 72 (1): 80–88. doi:10.1080/00031305.2017.1375986.

McLean, Iain H. 2012. *Literature Review Matrix*. http://archive.org/details/LiteratureReviewMatrix.

Nüst, Daniel, Markus Konkol, Edzer Pebesma, Christian Kray, Marc Schutzeichel, Holger Przibytzin, and Jörg Lorenz. 2017. "Opening the Publication Process with Executable Research Compendia." *D-Lib Magazine* 23 (1/2). doi:10.1045/january2017-nuest.

Pebesma, Edzer. 2013. "Earth and Planetary Innovation Challenge (EPIC) Submission 'One-Click-Reproduce'." http://pebesma.staff.ifgi.de/epic.pdf.

Peyton Jones, Simon. 2018. "Simon Peyton Jones at Microsoft Research." *Simon Peyton Jones at Microsoft Research*. Accessed May 25. https://www.microsoft.com/en-us/research/people/simonpj/.

Prlić, Andreas, and James B. Procter. 2012. "Ten Simple Rules for the Open Development of Scientific Software." *PLOS Comput Biol* 8 (12): e1002802. doi:10.1371/journal.pcbi.1002802.

rOpenSci. 2017. "rOpenSci Analysis Guide (Unconf 2017)." *Google Docs*. https://docs.google.com/document/d/1OYcWJUk-MiM2C1TIHB1Rn6rXoF5fHwRX-7_C12Blx8g/edit?usp=embed_facebook.

Roscoe, Timothy. 2007. "Writing Reviews for Systems Conferences," March, 6. https://people.inf.ethz.ch/troscoe/pubs/review-writing.pdf.

Sandve, Geir Kjetil, Anton Nekrutenko, James Taylor, and Eivind Hovig. 2013. "Ten Simple Rules for Reproducible Computational Research." *PLoS Comput Biol* 9 (10): e1003285.

doi:10.1371/journal.pcbi.1003285.

Schulzrinne, Henning. 2018. "Writing Systems and Networking Articles." Accessed May 25. https://www.cs.columbia.edu/~hgs/etc/writing-style.html.

Stodden, Victoria. 2009. "The Legal Framework for Reproducible Scientific Research: Licensing and Copyright." *Computing in Science & Engineering* 11 (1): 35–40. doi:10.1109/MCSE.2009.19.

Stodden, Victoria, and Sheila Miguez. 2014. "Best Practices for Computational Science: Software Infrastructure and Environments for Reproducible and Extensible Research." *Journal of Open Research Software* 2 (1). doi:10.5334/jors.ay.

Stodden, Victoria, Sheila Miguez, and Jennifer Seiler. 2015. "ResearchCompendia.org: Cyberinfrastructure for Reproducibility and Collaboration in Computational Science." *Computing in Science & Engineering* 17 (1): 12–19. doi:10.1109/MCSE.2015.18.

Taschuk, Morgan, and Greg Wilson. 2017. "Ten Simple Rules for Making Research Software More Robust." *PLOS Computational Biology* 13 (4): e1005412. doi:10.1371/journal.pcbi.1005412.

Vandevalle, Patrick. 2012. "Code Sharing Is Associated with Research Impact in Image Processing." *Computing in Science & Engineering* Reproducible Research for Scientific Computing (July): 42–47.

Whitaker, Kirstie. 2017. "Publishing a Reproducible Paper." doi:10.6084/m9.figshare.5440621.v2.

Whitesides, G. M. 2004. "Whitesides' Group: Writing a Paper." *Advanced Materials* 16 (15): 1375–7. doi:10.1002/adma.200400767.

Wilson, Greg, Jennifer Bryan, Karen Cranston, Justin Kitzes, Lex Nederbragt, and Tracy K. Teal. 2017. "Good Enough Practices in Scientific Computing." *PLOS Computational Biology* 13 (6): e1005510. doi:10.1371/journal.pcbi.1005510.