Implementing Code Review in the Scientific Workflow: Insights from Ecology and Evolutionary Biology

Edward R. Ivimey-Cook*, Joel Pick, Kevin Bairos-Novak, Antica Culina, Elliot Gould, Matt Grainger, Benjamin M. Marshall, David Moreau, Matthieu Paquet, Raphaël Royauté, Alfredo Sánchez-Tójar, Inês Silva, and Saras M. Windecker*

*corresponding author(s)

e.ivimeycook@gmail.com & saras.windecker@gmail.com

Keywords: reliability, reproducibility, software development, coding errors, research process, open science, transparency.

Abstract

Code review increases reliability and improves reproducibility of research. As such, code review is an inevitable step in software development and is common in subjects such as computer science. However, despite its importance, code review is noticeably lacking in ecology and evolutionary biology. This is problematic as it facilitates the propagation of coding errors and a reduction in reproducibility of published results. To address this, we provide a detailed commentary on how to effectively review code, how to set up your project to enable this form of review and detail its possible implementation at several stages throughout the research process. This guide serves as a primer for code review, and adoption of the principles and advice here will go a long way in promoting more open, reliable, and transparent ecology and evolutionary biology.

Introduction

Across scientific disciplines, researchers increasingly rely on code written in open-source software, such as R and Python, to clean, manipulate, visualise, and analyse data (Lai et al. 2019; Peikert and Brandmaier 2021; Peikert et al. 2021). Such software allows for increased transparency and reproducibility compared to software that operates through point-and-click interfaces ("User Interface" or "UI-based"), such as Minitab and SPSS (Obels et al. 2020). One of the key benefits of this code-based software is flexibility, because researchers can tailor analyses to their specific research needs which would otherwise be unavailable. However, the flexibility of code comes at a cost, as it means that it can be more error-prone (Budd et al. 1998). These errors may be conceptual (e.g., implementing the wrong function for a given task), programmatic (e.g., indexing the wrong column of a data frame), or syntactic (e.g., the incorrect spelling of a statement or function). Although UI-based software is also prone to conceptual errors, programmatic and syntactic errors are more common in code-based software. These errors can contribute to a lack of reproducibility or to the propagation of incorrect results (see Obels et al. 2020 for a review of code and data in psychology). Indeed, several high profile retractions have centred on these types of mistakes (Miller 2006; Ma and Chang 2007; Bolnick and Paull 2009; Huijgen et al. 2012; Williams and Bürkner 2020). One way to minimise potential errors, besides carefully annotating code and following best coding practices, is to undergo a process of code review. However, unlike in some disciplines (such as in computer science and software development) where code review is routinely implemented (Nelson and Schumann 2004; Badampudi et al. 2019), it is noticeably absent from the research and publication processes in other academic disciplines that rely on code to make inferences and predictions (Indriasari et al., 2020), including ecology and evolutionary biology.

To address this, we advocate for a fundamental shift in research culture that brings code review into all stages of the research process, as reviewing of code is necessary to facilitate error correction and to confirm the reproducibility of reported results. This is particularly important as analyses are becoming ever more complicated, especially in the fields of ecology and evolutionary biology (Touchon and McCoy 2016). But how can we implement code review? By whom, when, and how can it take place? In this paper, we provide some suggestions about how to conduct a code review and how to produce code that facilitates this form of review. Finally, we discuss the application of code review throughout the entire process of publication, from the early stages of pre-publishing right through to after work is published. Although we focus mainly on issues and techniques related to the R and Python coding languages due to their popularity in the fields of ecology and evolutionary biology (Mislan et al. 2016; Lai et al. 2019), the concepts and principles we discuss are widely applicable.

What should code review evaluate?

Code review is the process of either formally (as part of the peer review process) or informally (as coauthors or colleagues) checking and evaluating each other's code. It is critical to help avoid conceptual, programmatic, and syntactic errors in code and can take place at any stage of the research cycle; pre-submission, during formal peer review, or post-publication. Although the manner and scope in which code review occurs may vary depending on the position in the research cycle, the core priorities remain the same: to ensure code is as reported in methods, is able to successfully run, is reliable, and is able to reproduce stated results

Below we describe these key priorities as the four R's of code review (Figs. 1 and 2):

Is the code as Reported?

Code is a key research output and a critical component of scientific methodology. As such, open code accompanying written methods sections is becoming more common, following similar pushes for Open and FAIR data (Lamprecht et al. 2020). Therefore, it is imperative that code is checked for consistency when presented with the corresponding manuscript. These questions help us avoid conceptual errors in code. Does the code match the description of what is "Reported" within the methods section (Fig. 1)? Ensuring code matches the methods reported is imperative to evaluate whether the code is doing what is stated in the manuscript and what it is intended to do by the user. For instance, methods may state that an analysis uses a generalised linear model with Poisson error, but the code instead fits a Gaussian error structure. Reviewing for this mismatch must be part of code review. In addition, and equally important for reproducibility is whether the relevant packages (with appropriate version numbers) are stated somewhere in the manuscript? In general, it is good practice to, at the very least, list the packages (with version numbers) that are integral to the analysis or to visualisation in the manuscript. These can be obtained by using the "citation()" function in R or using the "setuptools" package in Python. A full list of all packages used (and versions), for instance those involved with cleaning and tidying of data, could be given elsewhere such as in an associated .R or .py file.

Does the code Run?

Even if code matches the methodology reported in a paper, this does not mean the code is executable (i.e. can "Run"). Programmatic and syntactic errors can make code fail to rerun. For example, code will not be able to be run (Fig. 1) if the appropriate packages are not listed (and thus not installed) or if there are spelling mistakes. Data sharing, where possible, should accompany code sharing, so that code can be fully rerun with the original data. If data sharing is not possible, some simulated data or a data snippet should be provided so that the code can be rerun. In cases where it would take a long period of time to rerun code (for instance with some forms of Bayesian modelling), the code should be accompanied with appropriate model outputs (readily provided by the author, see below “Output reproducibility”).

Is the code Reliable?

Errors can still propagate through code that runs and produces an output, because code can produce incorrect results in a reproducible manner (i.e. every time the code is run). For example, if code selects or modifies the wrong column in a dataset, the code will still run, but produce a reproducible yet inaccurate result (i.e. the code is not “Reliable”). This type of error could easily be conceptual, arising from a misunderstanding of the dataset, or programmatic, such as from indexing by number and mistaking the column order. Although some coding techniques, such as explicitly indexing by column name, can help avoid many of these mistakes, this type of error is common and also extremely difficult to pick up by anyone without deep familiarity with the dataset and code. These errors are thought to scale with the number and complexity of code (Lipow 1982). Although intrinsically linked to evaluating whether code can be run (the second “R”), evaluating code reliability means not only ensuring that the code runs to completion without error, but examining intermediate outputs of the code to ensure there are no mistakes. The functions “identical()” in R and “numpy.array_equal()” in Python can be useful at this stage of code review to compare object similarity between newly-generated and previously-saved intermediate outputs.

Are results Reproducible?

The last “R” of code review builds on the previous code review stages, and is perhaps the most fundamental: can the code produce the output, and thus support the conclusions, given in the paper (Goodman et al. 2016)? As several recent papers have highlighted (Archmiller et al.

2020; Obels et al. 2020; Errington et al. 2021; Minocher et al. 2021; Tiwari et al. 2021), reproducibility in research results is often very low. Therefore, the final step of code review is ensuring that final outputs when code is rerun match those reported in analysis and results sections. With that said, at times obtaining the *exact* same result is not possible. Some level of tolerance must therefore be applied especially when dealing with stochastic methods in which parameter estimates will change between subsequent runs. This can occur for example if the "set.seed()" function in R or "random.seed" function in Python has not been used prior to stochastic sampling. Providing model outputs can go some way in helping with this (see above), however it does not allow for the code to be explicitly run to see if you can obtain similar results as stated in the paper (regardless of potential time taken). In this case, newly generated results should be assessed to see if they matched (and how closely) to the conclusion (the direction and significance level) and the numbers (intervals matching within one significant figure) of the stated results (Archmiller et al. 2020). A useful example of this is also given in the supplementary material of Archmiller et al. (2020), in which a mean of 4.12 and interval of 3.45 to 4.91 reproduces the conclusion and numbers of a study with a mean of 4.00 and interval of 3.3 to 5.0. Similar conclusions would be drawn if these means (and CIs) were higher (e.g., 6.5, 6.0 to 7.0), but the numbers would not be considered quantitatively reproduced. On the other hand, the conclusions and numbers would not be reproduced if the model instead produced a mean of 4.1 with an interval of -1 to 8.4 (as the confidence interval here overlaps with 0). It is worth noting and mentioning in your review how closely the numbers and conclusion matched with the reported results.

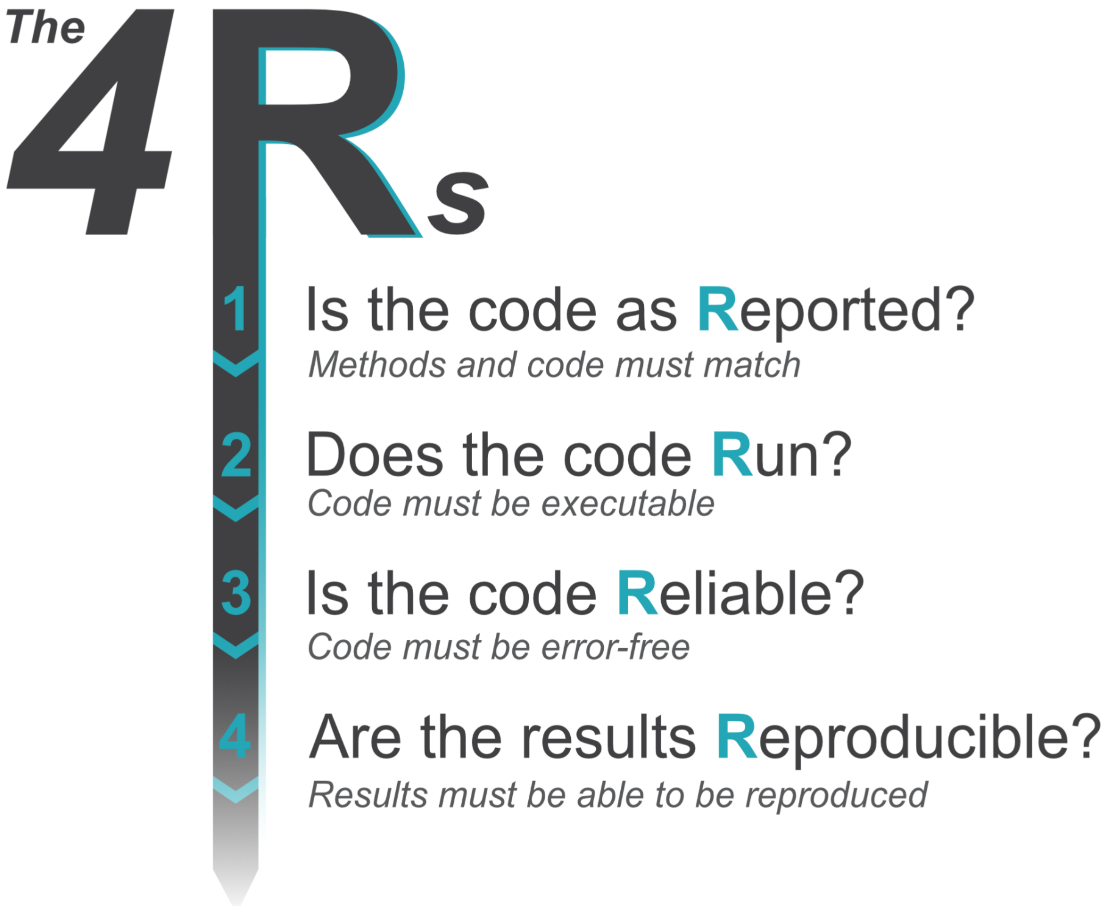

Figure 1. The four “R’s of code review.

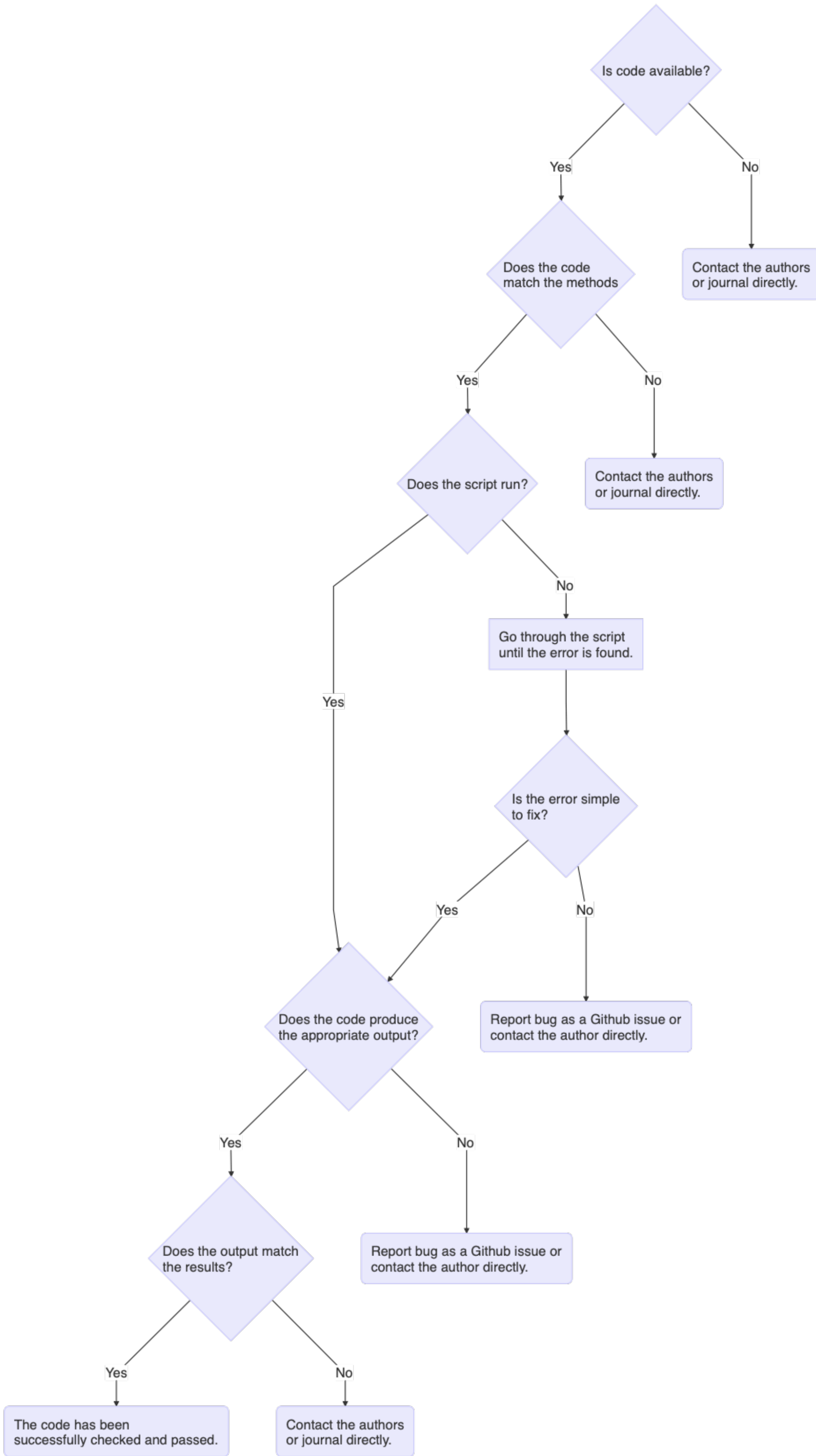

Figure 2. An example peer code review flowchart.

Setting up your code for effective code review

Code review should evaluate if code matches reported methods, whether code runs and is reliable, and lastly, if results can be reproduced. But in order for these questions to be addressed, code must be written and shared in a way that it is possible for someone else to rerun an analysis; both to allow for code to be reviewed and to be reused in the future when properly maintained and contained (see Boettiger 2015). For this to happen, all necessary scripts must be shared along with appropriate metadata indicating how the scripts interact with one another, along with describing all other necessary software and appropriate versions. Often, researchers lack formal training in coding, and learn to code in an *ad-hoc* fashion that excludes training on general styling, appropriate use of workflows, and project organisation. As a result, researchers may often not be aware of the steps necessary to set up code for a project in a manner that reflects best coding practices. Therefore, below we list key principles (Fig. 3) that will help make code reviewable at any stage of the research cycle.

Project organisation

Every project needs some form of directory organisation and folder structure. This is likely to be largely driven by the function and form that your research takes, but an efficient and transparent folder structure that keeps raw data separate from code and intermediate outputs should be created. This helps to ensure that raw data is not accidentally modified or overwritten if and when any data cleaning or wrangling techniques are applied. A simple folder and file structure such as this will go a long way to help researchers from all coding skill levels understand the order and flow of the data analysis. Particularly when the user creates sequentially labelled subfolders and scripts where someone following the code knows which order things must be run (e.g., files beginning with “01…”) in addition to dividing and naming folders to fit their purpose (e.g., data, scripts, function). Several incredibly useful examples already exist (Cooper 2017; Alston and Rick 2021; Chure 2023; see also https://coderefinery.github.io/reproducible-research/ and https://lakens.github.io/statistical_inferences/14-computationalreproducibility.html). Project code should be stored and available on any data or code repository. Another option for organising a project is to use pipeline tools (for instance see https://github.com/pditommaso/awesome-pipeline), such as the targets R package (Landau 2021) or the Luigi package in Python (see https://www.martinalarcon.org/2018-12-31-a-reproducible-science/). These tools allow users to automate the process of data analysis, taking a raw dataset through the steps necessary to produce data analysis and visualisation. The advantage to the user is that the code is compartmentalised into logical steps (e.g. import raw data, data cleaning, data wrangling, data analysis, data visualisation) and any changes to the code only affects the downstream steps. For example, if we change the type of analysis we do, we do not need to re-import the data or clean it again. This saves time in computation (especially important for complex long running pipelines) but is also advantageous for reproducibility, and sharing and reuse of code. Reviewers can effectively re-run the steps needed to produce a data analysis or figure without having to re-run time consuming pre-processing steps.

Project and input metadata

Projects will instantly have better organisation and increased reproducibility when users know *how* they should work through the various folders and subfolders. A README text file and additional metadata gives users the signposts required to facilitate rerunning of code. This can contain information on the packages used (e.g. the package name and version number), along with a detailed description of the various data files, project aim, contact information of the authors, and any relevant licences in place for code or for data (see https://choosealicense.com/licenses/ for more information). Furthermore, key information about source data is critical for reproducing analysis code. If sharing data is inappropriate to your study (for example when dealing with sensitive confidential data) then a user can provide a sample of simulated data or a primer so that the code can be checked and read (Quintana 2020; Hennessy et al. 2022). However, if data is readily available, then providing detailed information about what the data is (preferably in an associated README) and where the data is (e.g., stored on a free data repository such as The Open Science Framework or OSF, Zenodo, or Dryad) should be provided. Metadata should include information such as where the data comes from, who the owners are, as well as what each column header entails, and any relevant acronyms or shorthand notation (ideally following FAIR principles, so data is Findable, Accessible, Interoperable, and Reusable; see Lamprecht et al. (2020). This is particularly useful when controlled vocabulary is used throughout, and R packages such as codemeta (Boettiger 2017) and dataReporter (Petersen and Ekstrøm 2019) or Python packages such as CodeMetaPy (Gompel 2023) and cookiecutter (Feldroy 2022) can help with this. Lastly, it is also crucial to explain what possible data cleaning or curation occurred before the analysis code. For instance, outlining what previous data manipulation or pre-processing steps have been taken to obtain the data in its current state or when an intermediate data file was used.

Code Readability

Good readability of code is extremely important in enabling effective code review. Several quick solutions exist to provide increased clarity: explicitly calling packages (via a package's namespace, e.g. package::function() in R or package.module.function in Python), using relative file paths (preferably with an associated R project file, if using R with RStudio or in a virtual environment if using Python), removing redundant packages, and writing analysis code with clear subheadings and easy-to-understand object names. Best practice coding tips, aided by R packages such as styler (Müller and Walthert 2020) or pycodestyle in Python (Rocholl 2022) can format code in a number of standardised styles (e.g., Google, tidyverse in R, or PeP8 in Python) with a single line of code or a click of a button. Fortunately, several recent guides and primers have been written that focus on increasing coding cleanliness (Sweigart 2020; Filazzola and Lortie 2022; see Table 1 from Hunter-Zinck et al. 2021), so we urge the reader to consult these guidelines for tips and advice on improving code readability.

Output reproducibility

One of the key principles and requirements of code is the ability to correctly reproduce published graphs, statistics, and results. In order to do so, a user's code needs to provide a clear link between each section of the code and the various reported graphs and outputs to enable comparison of code to paper and to results. This should then facilitate checking that the results produced by the code match the stated results in the publication. In some cases, reproducing analysis from models can take considerable time to complete, for instance when re-running complicated Bayesian models. In this case the "exact" reproducibility of results is not always possible if code must simulate a stochastic process (e.g., Monte Carlo sampling methods). In this case using set seed (see above) or saving simulation outputs still allows for reproducible results (e.g., with the "saveRDS" function in R or the "pickle.dump" function in Python) and can enable code reviewers to check the reproducibility of the reported results.

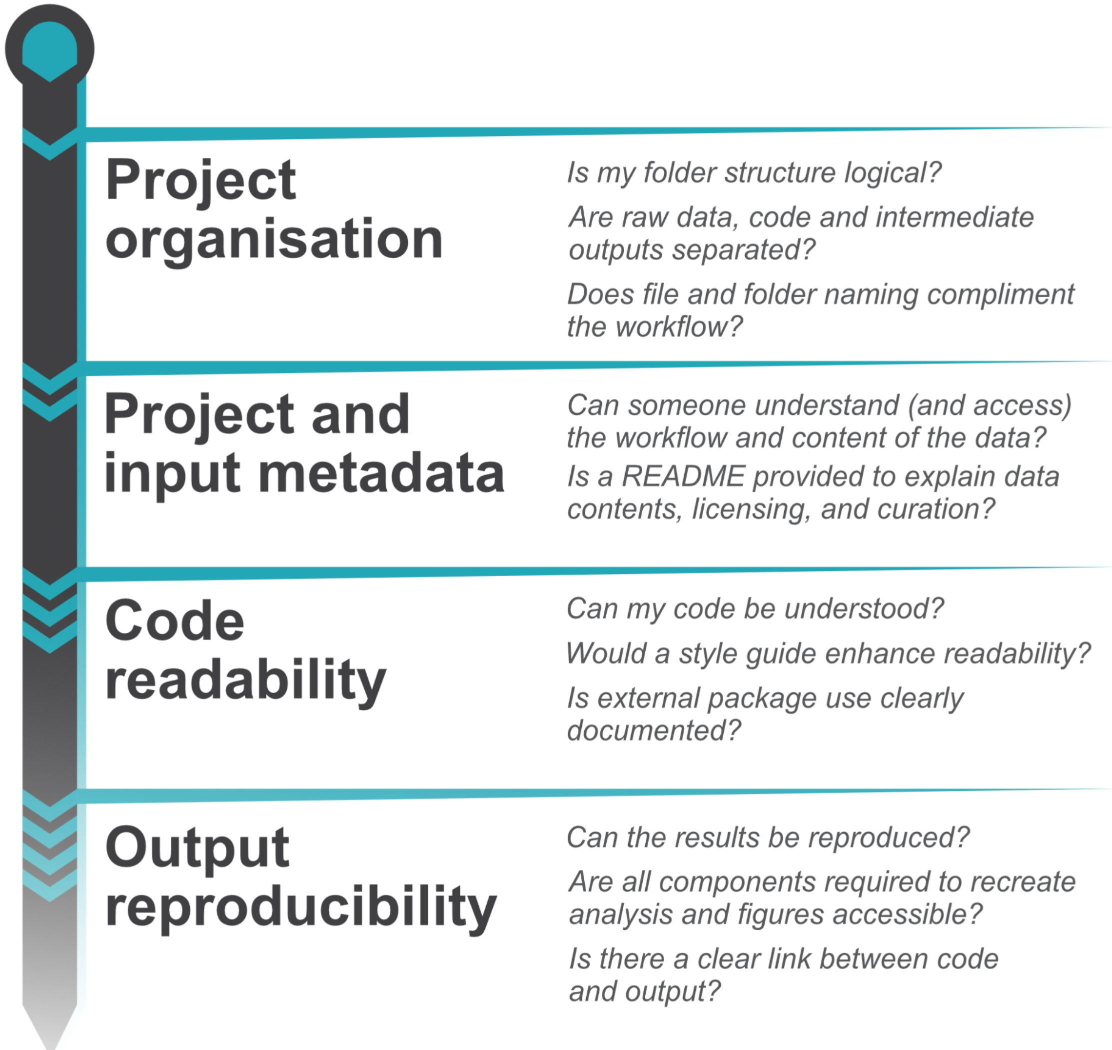

Figure 3. A basic workflow for reviewable code that can be adopted from the onset of a project.

Pre-Publication: Setting up a code review group

Informal training coupled with insufficient time and incentives (Touchon and McCoy 2016), means that coding and subsequent analysis are often the responsibility of a single member of a team throughout a project's entire lifetime. This is in stark contrast to the research-team wide collaboration typical when developing methodology and experimental design. The often individual nature of writing research code is part of what makes pre-publication code review so unlikely, but all the more critical. Although code review has a place in the formal peer review process and post-publication, one of the most important places for code review to take place is before publication.

To achieve this, there must be a culture of peer code review among research teams. One of the most effective methods by which researchers can establish a culture of peer code review in research labs or among colleagues is by setting up a code review group. Here we draw on our experience building a code review club (which we set up in collaboration with the Society for Open, Reliable, and Transparent Ecology and Evolution, SORTEE) to present tips for establishing this type of community. In particular, we focus on advice for removing the barriers people have towards sharing their code and receiving feedback; be these due to a lack of time or incentive, a lack of technical knowledge and unclear workflows, or due to social pressures and the fear of being judged by peers (Gomes et al. 2022).

Encourage collaboration from the start of a project

Code review can begin as early as the first initiation of a project and play a role beyond publication; it is useful to keep continuous code collaboration at all stages of a manuscript. Collaboration can be facilitated through various code-sharing platforms such as GitHub where users can submit and comment on pull requests (see Braga et al. 2023). At SORTEE we established a peer review group and used GitHub issues to summarise discussion of an individual’s code during an interactive zoom session (see https://github.com/SORTEE/peer-code-review/issues/8 for an example including a summary). However, it is important to find a method of facilitating code review that works for your group.

Set clear goals for the review

Setting out what you want to achieve with each code review session is particularly important when it comes to organising peer review meetings. Is the focus on general learning and improving readability or is it to error-check and scrutinise the reproducibility of your code? Having a clear structure and goal for each peer review session is important in order to focus comments and advice to address the precise reason for review. Similarly, unless the aim of a code review is to evaluate different analytical options, it would be better to leave methodological questions aside to ensure code review is streamlined.

Normalize coding errors and establish a judgement-free environment

Code review volunteers often feel very anxious about showing code that may have errors. It is therefore vital to normalise the existence of errors and highlight that perfection is never possible. It is also useful to stress that there is no such thing as bad code (Barnes 2010) and there are usually multiple ways to approach the same problem (Silberzahn et al. 2018; Botvinik-Nezer et al. 2020). One of the most important statements for peer code review is that there is no single way to code. It is important for code review not to get bogged down modifying or homogenising style; as long as code is readable, then coding diversity should be encouraged. It is important to create a relaxed environment where people can learn and correct mistakes without judgement or fear of failure and everyone in the peer review group should have a chance to contribute and speak.

Carefully consider group size

Usually a smaller group is a friendly starting point for peer code review because it allows people to feel more comfortable speaking up and participating. Small peer review groups (potentially even one-to-one) can better facilitate peer-to-peer learning and more focused review of code. However, there are also times when larger groups are more effective, such as having wider discussions on general themes and tips. It is worth considering the aims in establishing the group to help guide the ideal size. For instance, if your goal is to facilitate more general discussions, then a big group size is more likely to enable this. However, if your goal is to enable more focused review of code, then perhaps it is better to reduce the size of the peer review group for this purpose.

Consider the incentives

Code review, outside of paper submission and the formal peer review process, can have a large impact on an individual's project, from error-checking, to validation of appropriate statistical analyses. This then poses the question: what incentives should reviewers of code get? If deemed appropriate, the reviewer could be acknowledged using the MeRIT (Method Reporting with Initials for Transparency) system (Nakagawa et al. 2023), "e.g., J.L.P. ran a linear mixed model with a Gaussian error distribution. Code was checked by E.I.C.". In some circumstances, it may even be appropriate for the reviewer to obtain co-authorship of the paper, if the review fundamentally altered the project and subsequent paper. Incentives should be relative to the impact of the reviewer on the project.

During Publication: Formal code review

One of the most crucial aspects of code review can take place during the formal peer review process. This is where reviewers are able to carefully follow and understand the logic of analyses, much like the flow of writing from the introduction to the discussion of a paper (Powers and Hampton 2019). In some journals, such as The Royal Society (see Data sharing and mining | Royal Society 2023), Behavioural Ecology and Sociobiology (Bakker and Traniello 2020), and The American Naturalist (see Bolnick 2022) *both* code and data are available for reviewers to assess right from the submission stage. In some cases, such as in Journal of Open Source Software, the entire process of formal peer review, including that of code and manuscript is hosted on GitHub and implemented via GitHub issues (see https://github.com/openjournals/joss-reviews/issues for several useful examples). This, as Fernández-Juricic (2021) points out, has a number of benefits. For authors, providing code during peer review could lead to an increase in the quality of the manuscript, and for reviewers, available code allows for a far deeper insight into the manuscript as there is a clearer link between experimental methodology and statistical analysis (the First R; code as "Reported"). These benefits are substantial and could ultimately contribute to the adoption of code review during the publication process.

However, beyond the availability of code during submission, there are numerous other hurdles before effective and in-depth code review can be reasonably formalised as part of the peer review process. One of the most pressing issues is finding suitable individuals to review code given there is already a lack of willing reviewers in the current system. It is reasonable to expect reviewers to check that code is as reported, but anything more in-depth could take up the time of already overworked academics, who may not necessarily have the exact expertises needed to check other people's code. This needs to be fully considered and sufficiently addressed before code review becomes a standard part of the peer review process.

Post-Publication: Reviewing code after publication

Reviewing code post-publication is another facet of code review but one that has been much less discussed. Although it does not prevent *publication* of incorrect results, it does enable checking if code is indeed adhering to the R's listed above. However, the initial question should be, has all code used to produce the results been made available? This can either be a yes (stored and available on any data or code repository) or a no. Fortunately, an increasing number of journals are now *requesting* code be shared alongside scientific articles (Culina et al. 2020), such as in supplemental materials or by linking to an online repository. This then allows for any open and shared code to be checked and verified alongside methods section statements (Stodden, 2011; Light *et al.*, 2014). However, unlike data, code is a lot less likely to be made available regardless of these mandatory journal policies. As Figure 2 from Culina et al. (2020) shows, although the number of journals that possesses a mandatory code rule is increasing (from 15% to 79%; from 2015 to 2020) the number of articles that actually provide open code is still around 27% (although this number varies considerably among journals). This suggests that not many authors are adhering to this policy, which is an impediment to computational reproducibility (Culina et al. 2020). However, there is hope to be found here. As Culina *et al.* have shown, journals requiring code to be shared are increasing in number yearly and, as a field, we already have improved substantially (Mislan et al. 2016; Culina et al. 2020; Jenkins et al. 2023). In some cases, journals have implemented far stricter (and rightly so) data and code requirements along with assigning corresponding data editors (see Bolnick 2022). However, the first necessary step is for all journals to make it a requirement for *both* code and data to be present from the very start of the submission stage (Powers and Hampton 2019; Fernández-Juricic 2021). But what happens if the code is not available? In this case, the main option is to reach out to the corresponding author (or perhaps the journal itself) and ask if the code could be made available; similar to data being made available “upon reasonable request”.

The next part is relevant to the previous section above (“What should code review evaluate?). If you find that the code associated with a manuscript does not adhere to any of the “R”s listed above, then the first step is to contact the corresponding author (or if the paper uses the MeRIT system, the person who actually conducted the analysis; Nakagawa et al. 2023). This could be in the form of a GitHub issue if there is a repository for the code or an email (see Fig. 2). If there is indeed an error in code, and it is not due to differences in software version (e.g., differences in R and package versions) or due to inherent stochasticity (e.g., simulations or MCMC sampling), then the authors should be given a chance to contact the journal themselves to highlight and correct their mistakes. For instance, as per American Naturalist's stance (see Bolnick, 2022), authors who contact the journal to correct code or data errors will not be penalised and corrections are encouraged (when warranted). However, in cases where updated results would alter the narrative of a published paper, corrections may be more difficult to address without newer methods of documenting changes. Publication versioning or "living" documents may present a solid first step in such a scenario (Kane and Amin 2023). By encouraging post-publication code review, we can both decrease the proliferation of coding errors and also increase the reliability of published science.

Concluding remarks

In this brief overview, we have provided a basic set of guidelines for peer code review, recommendations for producing reviewable code, and considerations for how it should be adopted at every level of research throughout the publication process. The principles and advice listed here should form a baseline for code review that should be improved upon. We hope that this encourages coders at all levels to try and promote more reproducible, transparent, and open coding practices. In addition, we hope that this provides a primer to start a code reviewing club of your own!

Acknowledgements

This work began during workshops at the 2021 and 2022 annual conferences of the Society for Open, Reliable, and Transparent Ecology and Evolutionary biology run by E.I-C. and S.M.W. This work was partially funded by the Center of Advanced Systems Understanding (CASUS), which is financed by Germany's Federal Ministry of Education and Research (BMBF) and by the

Saxon Ministry for Science, Culture and Tourism (SMWK) with tax funds on the basis of the budget approved by the Saxon State Parliament. C.H.F. and J.M.C. were supported by NSF IIBR 1915347.

References

Alston, J. M., and J. A. Rick. 2021. A Beginner's Guide to Conducting Reproducible Research. Bulletin of the Ecological Society of America 102:1–14.

Archmiller, A. A., A. D. Johnson, J. Nolan, M. Edwards, L. H. Elliott, J. M. Ferguson, F. Iannarilli, et al. 2020. Computational Reproducibility in The Wildlife Society's Flagship Journals. The Journal of Wildlife Management 84:1012–1017.

Badampudi, D., R. Britto, and M. Unterkalmsteiner. 2019. Modern code reviews - Preliminary results of a systematic mapping study. Pages 340–345 *in*Proceedings of the Evaluation and Assessment on Software Engineering, EASE '19. Association for Computing Machinery, New York, NY, USA.

Bakker, T. C. M., and J. F. A. Traniello. 2020. Ensuring data access, transparency, and preservation: mandatory data deposition for Behavioral Ecology and Sociobiology. Behavioral Ecology and Sociobiology 74:132.

Barnes, N. 2010. Publish your computer code: it is good enough. Nature 467:753.

Boettiger, C. 2015. An introduction to Docker for reproducible research. ACM SIGOPS Operating Systems Review 49:71–79.

———. 2017. Generating CodeMeta Metadata for R Packages. The Journal of Open Source Software 2:454.

Bolnick, D. n.d. EIC Update: American Naturalist policy on data and code archiving.

Bolnick, D. I., and J. S. Paull. 2009. Morphological and dietary differences between individuals are weakly but positively correlated within a population of threespine stickleback. Evolutionary Ecology Research 11:1217–1233.

Botvinik-Nezer, R., F. Holzmeister, C. F. Camerer, A. Dreber, J. Huber, M. Johannesson, M. Kirchler, et al. 2020. Variability in the analysis of a single neuroimaging dataset by many teams. Nature 582:84–88.

Braga, P. H. P., K. Hébert, E. J. Hudgins, E. R. Scott, B. P. M. Edwards, L. L. Sánchez Reyes, M. J. Grainger, et al. 2023. Not just for programmers: How GitHub can accelerate collaborative and reproducible research in ecology and evolution. Methods in Ecology and Evolution 1–17.

Budd, J. M., M. Sievert, and T. R. Schultz. 1998. Phenomena of retraction: reasons for retraction and citations to the publications. JAMA 280:296–297.

Chure, G. 2023. Git + GitHub As A Platform For Reproducible Research. Python.

Cooper, N. 2017. A Guide to Reproducible Code in Ecology and Evolution.

Culina, A., I. van den Berg, S. Evans, and A. Sánchez-Tójar. 2020. Low availability of code in ecology: A call for urgent action. PLOS Biology 18:e3000763.

Data sharing and mining | Royal Society. n.d.

Errington, T. M., A. Denis, N. Perfito, E. Iorns, and B. A. Nosek. 2021. Challenges for assessing replicability in preclinical cancer biology. (P. Rodgers & E. Franco, eds.)eLife 10:e67995.

Feldroy, A. 2022. cookiecutter: A command-line utility that creates projects from project templates, e.g. creating a Python package project from a Python package project template. Python.

Fernández-Juricic, E. 2021. Why sharing data and code during peer review can enhance behavioral ecology research. Behavioral Ecology and Sociobiology 75:103.

Filazzola, A., and C. Lortie. 2022. A call for clean code to effectively communicate science. Methods in Ecology and Evolution.

Gomes, D. G. E., P. Pottier, R. Crystal-Ornelas, E. J. Hudgins, V. Foroughirad, L. L. Sánchez-Reyes, R. Turba, et al. 2022. Why don’t we share data and code? Perceived barriers and benefits to public archiving practices. Proceedings of the Royal Society B: Biological Sciences 289:20221113.

Gompel, M. van. 2023. CodeMetaPy: Generate and manage CodeMeta software metadata. Python.

Goodman, S. N., D. Fanelli, and J. P. A. Ioannidis. 2016. What does research reproducibility mean? Science Translational Medicine 8.

Hennessy, E. A., R. L. Acabchuk, P. A. Arnold, A. G. Dunn, Y. Z. Foo, B. T. Johnson, S. R. Geange, et al. 2022. Ensuring Prevention Science Research is Synthesis-Ready for Immediate and Lasting Scientific Impact. Prevention Science 23:809–820.

Huijgen, R., S. M. Boekholdt, B. J. Arsenault, W. Bao, J.-M. Davaine, F. Tabet, F. Petrides, et al. 2012. RETRACTED: Plasma PCSK9 Levels and Clinical Outcomes in the TNT (Treating to New Targets) Trial: A Nested Case-Control Study. Journal of the American College of Cardiology 59:1778–1784.

Hunter-Zinck, H., A. F. de Siqueira, V. N. Vásquez, R. Barnes, and C. C. Martinez. 2021. Ten simple rules on writing clean and reliable open-source scientific software. PLOS Computational Biology 17:e1009481.

Jenkins, G. B., A. P. Beckerman, C. Bellard, A. Benítez-López, A. M. Ellison, C. G. Foote, A. L. Hufton, et al. 2023. Reproducibility in ecology and evolution: Minimum standards for data and code. Ecology and Evolution 13:e9961.

Kane, A., and B. Amin. 2023. Amending the literature through version control. Biology Letters 19:20220463.

Lai, J., C. J. Lortie, R. A. Muenchen, J. Yang, and K. Ma. 2019. Evaluating the popularity of R in ecology. Ecosphere 10:e02567.

Lamprecht, A.-L., L. Garcia, M. Kuzak, C. Martinez, R. Arcila, E. Martin Del Pico, V. Dominguez Del Angel, et al. 2020. Towards FAIR principles for research software. (P. Groth, P. Groth, & M. Dumontier, eds.)Data Science 3:37–59.

Landau, W. M. 2021. The targets R package: a dynamic Make-like function-oriented pipeline toolkit for reproducibility and high-performance computing. Journal of Open Source Software 6:2959.

Lipow, M. 1982. Number of Faults per Line of Code. IEEE Transactions on Software Engineering SE-8:437–439. Presented at the IEEE Transactions on Software Engineering.

Ma, C., and G. Chang. 2007. Retraction for Ma and Chang, Structure of the multidrug resistance efflux transporter EmrE from Escherichia coli. Proceedings of the National Academy of Sciences 104:3668–3668.

Miller, G. 2006. A Scientist's Nightmare: Software Problem Leads to Five Retractions. Science 314:1856–1857.

Minocher, R., S. Atmaca, C. Bavero, R. McElreath, and B. Beheim. 2021. Estimating the reproducibility of social learning research published between 1955 and 2018. Royal Society Open Science 8:210450.

Mislan, K. A. S., J. M. Heer, and E. P. White. 2016. Elevating The Status of Code in Ecology. Trends in Ecology & Evolution 31:4–7.

Müller, K., and L. Walthert. 2020. Styler: Non-invasive pretty printing of R code. R package version 1.3. 2.

Nakagawa, S., E. R. Ivimey-Cook, M. J. Grainger, R. E. O'Dea, S. Burke, S. M. Drobniak, E. Gould, et al. 2023. Method Reporting with Initials for Transparency (MeRIT) promotes more granularity and accountability for author contributions. Nature Communications 14:1788.

Nelson, S., and J. Schumann. 2004. What makes a code review trustworthy? Page 10 pp.- *in*37th Annual Hawaii International Conference on System Sciences, 2004. Proceedings of the. Presented at the 37th Annual Hawaii International Conference on System Sciences, 2004. Proceedings of the.

Obels, P., D. Lakens, N. A. Coles, J. Gottfried, and S. A. Green. 2020. Analysis of Open Data and Computational Reproducibility in Registered Reports in Psychology. Advances in Methods and Practices in Psychological Science 3:229–237.

Peikert, A., and A. M. Brandmaier. 2021. A Reproducible Data Analysis Workflow With R Markdown, Git, Make, and Docker. Quantitative and Computational Methods in Behavioral Sciences 1–27.

Peikert, A., C. J. van Lissa, and A. M. Brandmaier. 2021. Reproducible Research in R: A Tutorial on How to Do the Same Thing More Than Once. Psych 3:836–867.

Petersen, A. H., and C. T. Ekstrøm. 2019. **dataMaid** : Your Assistant for Documenting Supervised Data Quality Screening in *R*. Journal of Statistical Software 90.

Powers, S. M., and S. E. Hampton. 2019. Open science, reproducibility, and transparency in ecology. Ecological Applications 29:e01822.

Quintana, D. S. 2020. A synthetic dataset primer for the biobehavioural sciences to promote reproducibility and hypothesis generation. eLife 9:e53275.

Rocholl, J. C. 2022. pycodestyle: Python style guide checker. Python.

Silberzahn, R., E. L. Uhlmann, D. P. Martin, P. Anselmi, F. Aust, E. Awtrey, Š. Bahník, et al. 2018. Many analysts, one data set: Making transparent how variations in analytic choices affect results. Advances in Methods and Practices in Psychological Science 1:337–356.

Sweigart, A. 2020. Beyond the Basic Stuff with Python: Best Practices for Writing Clean Code. No Starch Press.

Tiwari, K., S. Kananathan, M. G. Roberts, J. P. Meyer, M. U. Sharif Shohan, A. Xavier, M. Maire, et al. 2021. Reproducibility in systems biology modelling. Molecular Systems Biology 17:e9982.

Touchon, J. C., and M. W. McCoy. 2016. The mismatch between current statistical practice and doctoral training in ecology. Ecosphere 7:e01394.

Williams, D., and P.-C. Bürkner. 2020. Coding Errors Lead to Unsupported Conclusions: A critique of Hofmann et al. (2015). Meta-Psychology 4.