

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

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

6
7 *corresponding author(s)

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



9
10 Keywords: reliability, reproducibility, software development, coding errors, research process,
11 open science, transparency.

12
13 ***Abstract***

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

25 *Introduction*

26 Across scientific disciplines, researchers increasingly rely on code written in open-source
27 software, such as R and Python, to clean, manipulate, visualise, and analyse data (Lai *et al.*,
28 2019; Peikert & Brandmaier, 2021; Peikert *et al.*, 2021). Such software allows for increased
29 transparency and reproducibility compared to software that operates through point-and-click
30 interfaces (“User Interface” or “UI-based”), such as Minitab and SPSS (Obels *et al.*, 2020). One
31 of the key benefits of this code-based software is flexibility, because researchers can tailor
32 analyses to their specific research needs which would otherwise be unavailable. However, the
33 flexibility of code comes at a cost, as it means that it can be more error-prone (Budd *et al.*,
34 1998). These errors may be conceptual (e.g., implementing the wrong function for a given task),
35 programmatic (e.g., indexing the wrong column of a data frame), or syntactic (e.g., the incorrect
36 spelling of a statement or function). Although UI-based software is also prone to conceptual
37 errors, programmatic and syntactic errors are more common in code-based software. These
38 errors can contribute to a lack of reproducibility or to the propagation of incorrect results (see
39 Obels *et al.*, 2020 for a review of code and data in psychology). Indeed, several high-profile
40 retractions have centred on these types of mistakes (Miller, 2006; Ma & Chang, 2007; Bolnick &
41 Paull, 2009; Huijgen *et al.*, 2012; Williams & Bürkner, 2020). One way to minimise potential
42 errors, besides carefully annotating code and following best coding practices, is to undergo a
43 process of code review. However, unlike in some disciplines (such as in computer science and
44 software development) where code review is routinely implemented (Nelson & Schumann,
45 2004; Badampudi *et al.*, 2019), it is noticeably absent from the research and publication
46 processes in other academic disciplines that rely on code to make inferences and predictions
47 (Indriasari *et al.*, 2020), including ecology and evolutionary biology.

48

49 To address this, we advocate for a fundamental shift in research culture that brings code review
50 into all stages of the research process, as reviewing of code is necessary to facilitate error
51 correction and to confirm the reproducibility and reliability of reported results. This is particularly
52 important as analyses are becoming ever more complicated, especially in the fields of ecology
53 and evolutionary biology (Touchon & McCoy, 2016). But how can we implement code review?
54 By whom, when, and how can it take place? In this paper, we provide some suggestions about
55 how to conduct a code review and how to produce code that facilitates this form of review.
56 Finally, we discuss the application of code review throughout the entire process of publication,
57 from the early stages of pre-publishing right through to after work is published. Although we
58 focus mainly on issues and techniques related to the R and Python coding languages due to
59 their popularity in the fields of ecology and evolutionary biology (Mislán *et al.*, 2016; Lai *et al.*,
60 2019), the concepts and principles we discuss are widely applicable.

61

62 *What should code review evaluate?*

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

70

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

72

73 *Is the code as Reported?*

74 Code is a key research output and a critical component of scientific methodology. As such, open
75 code accompanying written methods sections is becoming more common, following similar
76 pushes for Open and FAIR data (Lamprecht *et al.*, 2020). Therefore, it is imperative that code is
77 checked for consistency when presented with the corresponding manuscript. These questions
78 help us avoid conceptual errors in code. Does the code match the description of what is
79 “Reported” within the methods section (Fig. 1, SM Box 1)? Ensuring code matches the methods
80 reported is imperative to evaluate whether the code is doing what is stated in the manuscript
81 and what it is intended to do by the user. For instance, methods may state that an analysis uses
82 a generalised linear model with Poisson error, but the code instead fits a Gaussian error
83 structure. Reviewing for this mismatch must be part of code review. In addition, and equally
84 important for reproducibility is whether the relevant packages (with appropriate version
85 numbers) are stated somewhere in the manuscript. In general, it is good practice to, at the very
86 least, list the packages (with version numbers) that are integral to the analysis or to visualisation
87 in the manuscript. These can be obtained by using the “citation()” function in R or using the
88 “setuptools” package in Python. A full list of all packages used (and versions), for instance those
89 involved with cleaning and tidying of data, could be given elsewhere such as in an associated .R
90 or .py file. Packages such as {renv} (Ushey, 2023; which replaces {packrat}, Ushey *et al.*, 2022),
91 {groundhog} (Simonsohn & Gruson, 2023), or {poetry} (Eustace, 2023) and {pipenv} (Pipenv
92 Maintainer Team, 2023) in Python can help with ensuring a reproducible environment and allow
93 for specific loading of desired package versions. Another option is containerisation through the
94 use of Docker (Boettiger, 2015). Detailed tutorials already exist which highlight the use of this
95 reproducible method in far more detail than we will discuss here.

96

97 *Does the code Run?*

98 Even if code matches the methodology reported in a paper, this does not mean the code is
99 executable (i.e., can “Run”). Programmatic and syntactic errors can make code fail to rerun. For
100 example, code will not be able to be run if it includes calls to libraries (or modules) that are
101 not installed in the current computing environment or if there are spelling mistakes (Fig. 1,
102 SM Box 1). Data sharing, where possible, should accompany code sharing, so that code can be
103 fully rerun with the original data. If data sharing is not possible, simulated data or a data snippet
104 should be provided so that the code can be rerun. In cases where it would take a long period of
105 time to rerun code (for instance with some forms of Bayesian modelling), the code should be
106 accompanied with appropriate model outputs (readily provided by the author, see below “Output
107 reproducibility”).

108

109 *Is the code **Reliable**?*

110 Errors can still propagate through code that runs and produces an output, because code can
111 produce incorrect results in a reproducible manner (i.e., every time the code is run). For
112 example, if code selects or modifies the wrong column in a dataset, the code will still run, but
113 produce a reproducible yet inaccurate result (i.e., the code is not “Reliable”; Fig. 1, SM Box 1).
114 This type of error could easily be conceptual, arising from a misunderstanding of the dataset, or
115 programmatic, such as from indexing by number and producing a mistaken column order or
116 from user-defined functions. Although some coding techniques, such as explicitly indexing by
117 column name or by performing unit testing of any user-defined function (see Cooper, 2017;
118 relevant packages include {testthat}, Wickham, (2011) in R or {pytest} in Python, Okken, 2022),
119 can help avoid many of these mistakes, this type of error is common and also extremely difficult
120 to pick up by anyone without deep familiarity with the dataset and code. In particular, these
121 errors are thought to scale with the number of lines and complexity of code (Lipow, 1982).
122 Although intrinsically linked to evaluating whether code can be run (the second “R”), evaluating

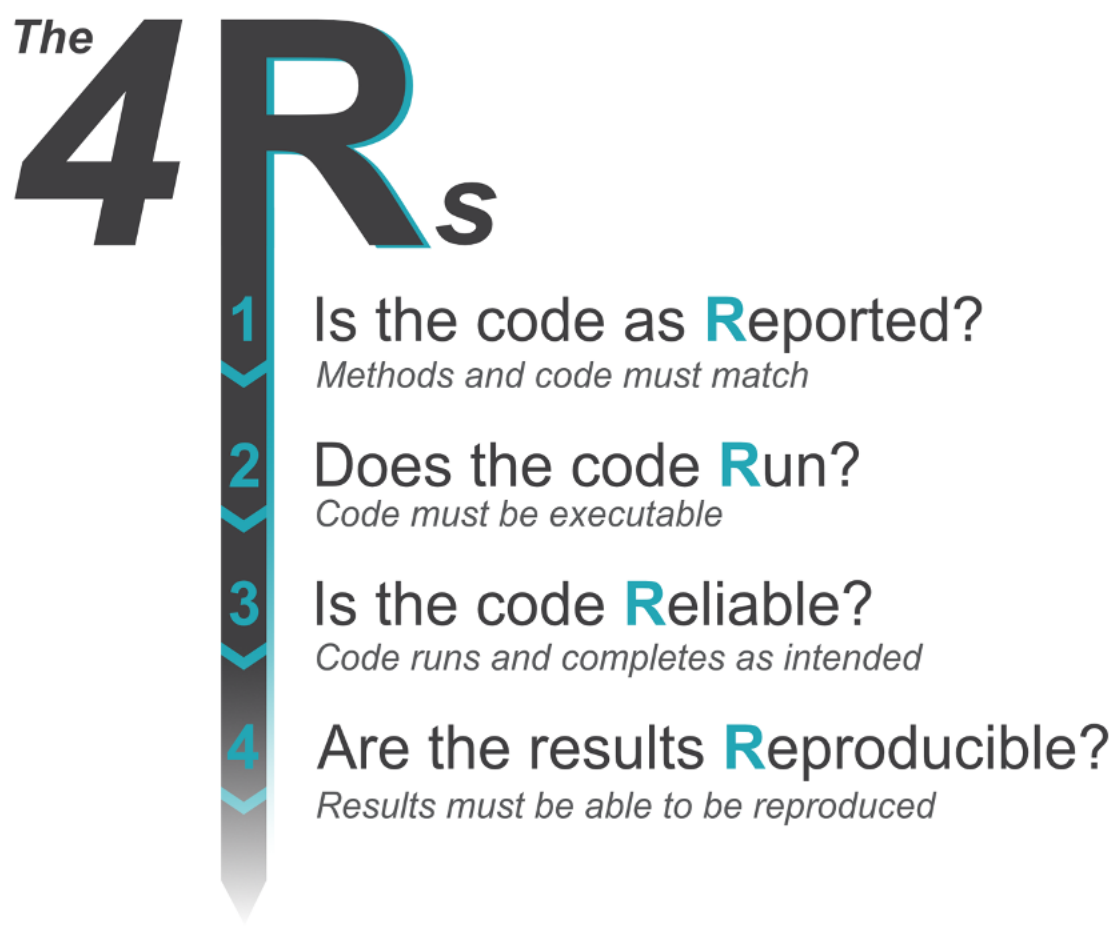
123 code reliability means not only ensuring that the code runs to completion without error, but
124 examining intermediate outputs of the code to ensure there are no mistakes. The functions
125 “`identical()`” in R and “`numpy.array_equal()`” in Python can be useful at this stage of code review
126 to compare object similarity between newly-generated and previously-saved intermediate
127 outputs.

128

129 *Are the results Reproducible?*

130 The last “R” of code review builds on the previous code review stages, and is perhaps the most
131 fundamental: can the code produce the output, and thus support the conclusions, given in the
132 paper (Goodman *et al.*, 2016; Fig. 1, SM Box 1)? As several recent papers have highlighted
133 (Archmiller *et al.*, 2020; Obels *et al.*, 2020; Errington *et al.*, 2021; Minocher *et al.*, 2021; Tiwari *et*
134 *al.*, 2021), reproducibility in research results is often very low. Therefore, the final step of code
135 review is ensuring that final outputs when code is rerun match those reported in the analysis
136 and results sections (including any relevant figures and narrative text contained within these
137 sections). With that said, at times obtaining the *exact* same result is not possible. Some level of
138 tolerance must therefore be applied especially when dealing with stochastic methods in which
139 parameter estimates will change between subsequent runs or with techniques that are
140 computationally demanding and slow. This can occur for example if the “`set.seed()`” function in
141 R or “`random.seed`” function in Python has not been used prior to stochastic sampling. Providing
142 model outputs can go some way in helping with this (see above), however it does not allow for
143 the code to be explicitly run to see if you can obtain similar results as stated in the paper
144 (regardless of potential time taken). In this case, newly generated results should be assessed to
145 see if they matched (and how closely) to the conclusion (the direction and significance level)
146 and the numbers (intervals matching within one significant figure) of the stated results
147 (Archmiller *et al.*, 2020). A useful example of this is also given in the supplementary material of

148 Archmiller et al. (2020), in which a mean of 4.12 and interval of 3.45 to 4.91 reproduces the
149 conclusion and numbers of a study with a mean of 4.00 and interval of 3.3 to 5.0. Similar
150 conclusions would be drawn if these means (and CIs) were higher (e.g., 6.5, 6.0 to 7.0), but the
151 numbers would not be considered quantitatively reproduced. On the other hand, the conclusions
152 and numbers would not be reproduced if the model instead produced a mean of 4.1 with an
153 interval of -1 to 8.4 (as the confidence interval here overlaps with 0). It is worth noting and
154 mentioning in your review how closely the numbers and conclusion matched with the reported
155 results.
156



157

158 Figure 1. The four “Rs” of code review. Figure design by B.M.M.

159

160 *Setting up your code for effective code review*

161 Code review should evaluate if code matches reported methods, whether code runs and is
162 reliable, and lastly, if results can be reproduced. But in order for these questions to be
163 addressed, code must be written and shared in a way that it is possible for someone else to
164 rerun an analysis; both to allow for code to be reviewed and to be reused in the future when
165 properly maintained and contained (see Boettiger, 2015). For this to happen, all necessary
166 scripts must be shared along with appropriate metadata indicating how the scripts interact with
167 one another, along with describing all other necessary software and appropriate versions. Often,
168 researchers lack formal training in coding, and learn to code in an *ad-hoc* fashion that excludes
169 training on general styling, appropriate use of workflows, and project organisation. As a result,
170 researchers may often not be aware of the steps necessary to set up code for a project in a
171 manner that reflects best coding practices. Therefore, below we list key principles (Fig. 2) that
172 will help make code reviewable at any stage of the research cycle.

173

174 *Project organisation*

175 Every project needs some form of directory organisation and folder structure. This is likely to be
176 largely driven by the function and form that your research takes, but an efficient and transparent
177 folder structure that keeps raw data separate from code and intermediate outputs should be
178 created. This helps to ensure that raw data is not accidentally modified or overwritten if any data
179 cleaning or wrangling techniques are applied. A simple folder and file structure such as this will
180 go a long way to help researchers from all coding skill levels understand the order and flow of
181 the data analysis, particularly when the user creates sequentially labelled subfolders and scripts
182 where someone following the code knows which order things must be run (e.g., files beginning

183 with “01...” in addition to dividing and naming folders to fit their purpose (e.g., data, scripts,
184 function). Several incredibly useful examples already exist (Cooper, 2017; Alston & Rick, 2021;
185 Chure, 2023; see also <https://coderefinery.github.io/reproducible-research/> and
186 https://lakens.github.io/statistical_inferences/14-computationalreproducibility.html). Project code
187 should be stored and available on any data or code repository. Another option for organising a
188 project is to use pipeline or workflow tools (for instance see
189 <https://github.com/pditommaso/awesome-pipeline>), such as the {targets} (Landau, 2021) and
190 {workflowR} R packages (Blischak *et al.*, 2019) or the {luigi} package (The Luigi Authors, 2023)
191 in Python (see
192 <https://www.martinalarcon.org/2018-12-31-a-reproducible-science/>). These tools allow users to
193 automate the process of data analysis, taking a raw dataset through the steps necessary to
194 produce data analysis and visualisation. The advantage to the user is that the code is
195 compartmentalised into logical steps (e.g., import raw data, data cleaning, data wrangling, data
196 analysis, data visualisation) and any changes to the code only affects the downstream steps.
197 For example, if we change the type of analysis we do, we do not need to re-import the data or
198 clean it again. This saves time in computation (especially important for complex, long-running
199 pipelines) but is also advantageous for reproducibility, and sharing and reuse of code.
200 Reviewers can effectively rerun the steps needed to produce a data analysis or figure without
201 having to rerun time consuming pre-processing steps.

202

203 *Project and input metadata*

204 Projects will instantly have better organisation and increased reproducibility when users know
205 *how* they should work through the various folders and subfolders. A README text file and
206 additional metadata gives users the signposts required to facilitate rerunning of code. This can
207 contain information on the packages used (e.g., the package name and version number), along
208 with a detailed description of the various data files, project aim, contact information of the

209 authors, and any relevant licences in place for code or for data (see
210 <https://choosealicense.com/licenses/> for more information). Furthermore, key information about
211 source data is critical for reproducing analysis code. If sharing data is inappropriate to your
212 study (for example when dealing with sensitive confidential data) or if data is so large it cannot
213 be easily shared, then a user can provide a sample of simulated data or a primer so that the
214 code can be checked and read (Quintana, 2020; Hennessy *et al.*, 2022). However, if data is
215 readily available, then providing detailed information about what the data is (preferably in an
216 associated README) and where the data is (e.g., stored on a free data repository such as The
217 Open Science Framework (OSF), Zenodo, or for ecology data, the Knowledge Network for
218 Biocomplexity) should be provided. Metadata should include information such as where the data
219 comes from, who the owners are, as well as what each column header entails, and any relevant
220 acronyms or shorthand notation (ideally following FAIR principles, so data is Findable,
221 Accessible, Interoperable, and Reusable; see Lamprecht *et al.*, 2020). This is particularly useful
222 when controlled vocabulary is used throughout, and R packages such as {codemeta} (Boettiger,
223 2017) and {dataReporter} (Petersen & Ekstrøm, 2019) or Python packages such as
224 {CodeMetaPy} (Gompel, 2023) and {cookiecutter} (Feldroy, 2022) can help with this. Lastly, it is
225 also crucial to explain what data cleaning or curation occurred before the analysis code. For
226 instance, outlining what previous data manipulation or pre-processing steps have been taken to
227 obtain the data in its current state or when an intermediate data file was used.

228

229 Code Readability

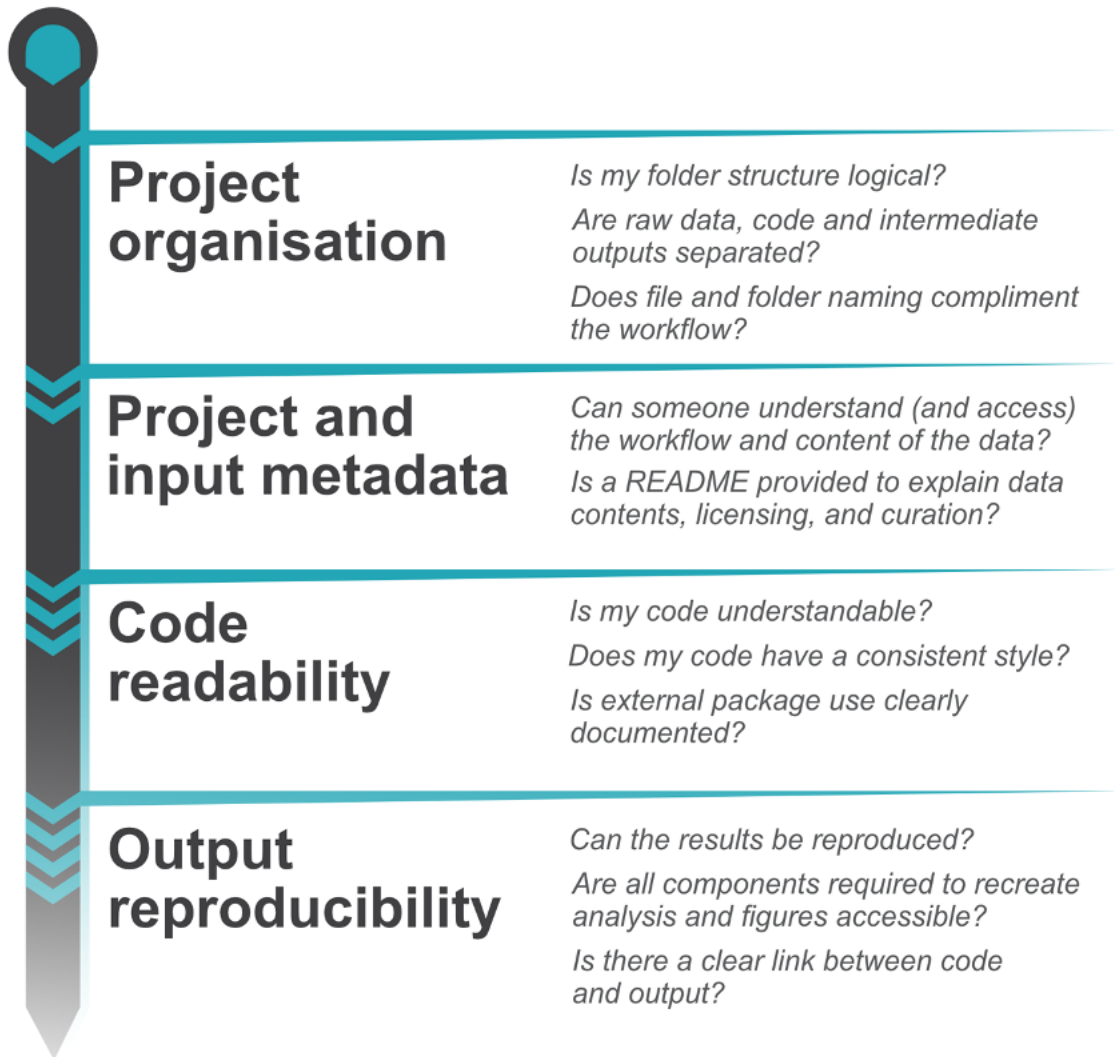
230 Good readability of code is extremely important in enabling effective code review. Several quick
231 solutions exist to provide increased clarity: explicitly calling packages (via a package's
232 namespace, e.g., `package::function()` in R or `package.module.function` in Python), using relative
233 file paths (for instance using the {here} package (Müller, 2020) and preferably with an
234 associated R project file, if using R with RStudio or in a virtual environment if using Python),

235 removing redundant packages, and writing analysis code with clear subheadings and easy-to-
236 understand object names. Best practice coding tips, aided by R packages such as `{styler}`
237 (`Müller & Walthert, 2020`) or `{pycodestyle}` in Python (`Rocholl, 2022`) can format code in a number
238 of standardised styles (e.g., Google, tidyverse in R, or PeP8 in Python) with a single line of code
239 or a click of a button. Fortunately, several recent guides and primers have been written that
240 focus on increasing coding cleanliness (`Sweigart, 2020`; `Hunter-Zinck et al., 2021`; `Filazzola &`
241 `Lortie, 2022`), so we urge the reader to consult these guidelines for tips and advice on improving
242 code readability.

243

244 Output reproducibility

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



258

259 Figure 2. A basic workflow for reviewable code that can be adopted from the onset of a project.

260 See Supplementary Material for a printable checklist of the points listed here. Figure and

261 checklist design by B.M.M.

262

263 *Pre-Publication: Setting up a code review group*

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

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

281

282 *Encourage collaboration from the start of a project*

283 Code review can begin as early as the first initiation of a project and play a role beyond
284 publication; it is useful to keep continuous code collaboration at all stages of a manuscript.
285 Collaboration can be facilitated through various code-sharing platforms such as GitHub where
286 users can submit and comment on pull requests (see Braga *et al.*, 2023). At SORTEE we
287 established a peer review group and used GitHub issues to summarise discussion of an
288 individual's code during an interactive zoom session (see

289 <https://github.com/SORTEEE/peer-code-review/issues/8> for an example including a summary).

290 However, it is important to find a method of facilitating code review that works for your group.

291

292 *Set clear goals for the review*

293 Setting out what you want to achieve with each code review session is particularly important

294 when it comes to organising peer review meetings. Is the focus on general learning and

295 improving readability or is it to error-check and scrutinise the reproducibility of your code?

296 Having a clear structure and goal for each peer review session is important in order to focus

297 comments and advice to address the precise reason for review. Similarly, unless the aim of a

298 code review is to evaluate different analytical options, it would be better to leave methodological

299 questions aside to ensure code review is streamlined.

300

301 *Normalize coding errors and establish a judgement-free environment*

302 Code review volunteers often feel very anxious about showing code that may have errors. It is

303 therefore vital to normalise the existence of errors and highlight that perfection is never possible.

304 It is also useful to stress that there is no such thing as bad code (Barnes, 2010) and there are

305 usually multiple ways to approach the same problem (Silberzahn *et al.*, 2018; Botvinik-Nezer *et*

306 *al.*, 2020). One of the most important statements for peer code review is that there is no single

307 way to code. It is important for code review not to get bogged down by modifying or

308 homogenising style; as long as code is readable, then coding diversity should be encouraged. It

309 is important to create a relaxed environment where people can learn and correct mistakes

310 without judgement or fear of failure and everyone in the peer review group should have a

311 chance to contribute and speak.

312

313 *Carefully consider group size*

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

323

324 Consider the incentives

325 Code review, outside of paper submission and the formal peer review process, can have a large
326 impact on an individual's project, from error-checking, to validation of appropriate statistical
327 analyses. This then poses the question: what incentives should reviewers of code get? If
328 deemed appropriate, the reviewer could be acknowledged using the MeRIT (Method Reporting
329 with Initials for Transparency) system (Nakagawa *et al.*, 2023), "e.g., J.L.P. ran a linear mixed
330 model with a Gaussian error distribution. Code was checked by E.I.C.". In some circumstances,
331 it may even be appropriate for the reviewer to obtain co-authorship of the paper, if the review
332 fundamentally altered the project and subsequent paper. For instance, a situation may arise
333 where a code reviewer(s) finds a major coding error which, when fixed after highlighting and
334 reproducing the issue to the author(s), alters the subsequent results and conclusions of the
335 manuscript. Ultimately, incentives should be relative to the impact of the reviewer on the
336 project.

337

338 *During Publication: Formal code review*

339 One of the most crucial aspects of code review can take place during the formal peer review
340 process. This is where reviewers are able to carefully follow and understand the logic of
341 analyses, much like the flow of writing from the introduction to the discussion of a paper
342 (Powers & Hampton, 2019). In some journals, such as The Royal Society (Data sharing and
343 mining | Royal Society, 2023), Behavioural Ecology and Sociobiology (Bakker & Traniello,
344 2020), and The American Naturalist (Bolnick, 2022) *both* code and data are available for
345 reviewers to assess right from the submission stage. In some cases, such as in Journal of Open
346 Source Software, the entire process of formal peer review, including that of code and
347 manuscript is hosted on GitHub and implemented via GitHub issues (see
348 <https://github.com/openjournals/joss-reviews/issues> for several useful examples). This, as
349 Fernández-Juricic (2021) points out, has several benefits. For authors, providing code during
350 peer review could lead to an increase in the quality of the manuscript, and for reviewers,
351 available code allows for a far deeper insight into the manuscript as there is a clearer link
352 between experimental methodology and statistical analysis (the First R; code as “Reported”).
353 These benefits are substantial and could ultimately contribute to the adoption of code review
354 during the publication process by journals.

355 However, beyond the availability of code during submission, there are numerous other hurdles
356 before effective and in-depth code review can be reasonably formalised as part of the peer
357 review process. One of the most pressing issues is finding suitable individuals to review code
358 given there is already a lack of willing reviewers in the current system. It is reasonable to expect
359 reviewers to check that code is as reported, but anything more in-depth could take up the time
360 of already overworked academics, who may not necessarily have the exact expertise needed to
361 check other people’s code. A potential first step is for journals to appoint official journal code
362 reviewers/editors. Although similar to data editors (see below), this role’s sole responsibility
363 would be to check that code adheres to the four R’s and would be considered a separate (but

364 parallel) process from the responsibilities of “typical” reviewers. However, all of these concerns
365 need to be fully considered and sufficiently addressed before code review becomes a standard
366 part of the peer review process.

367

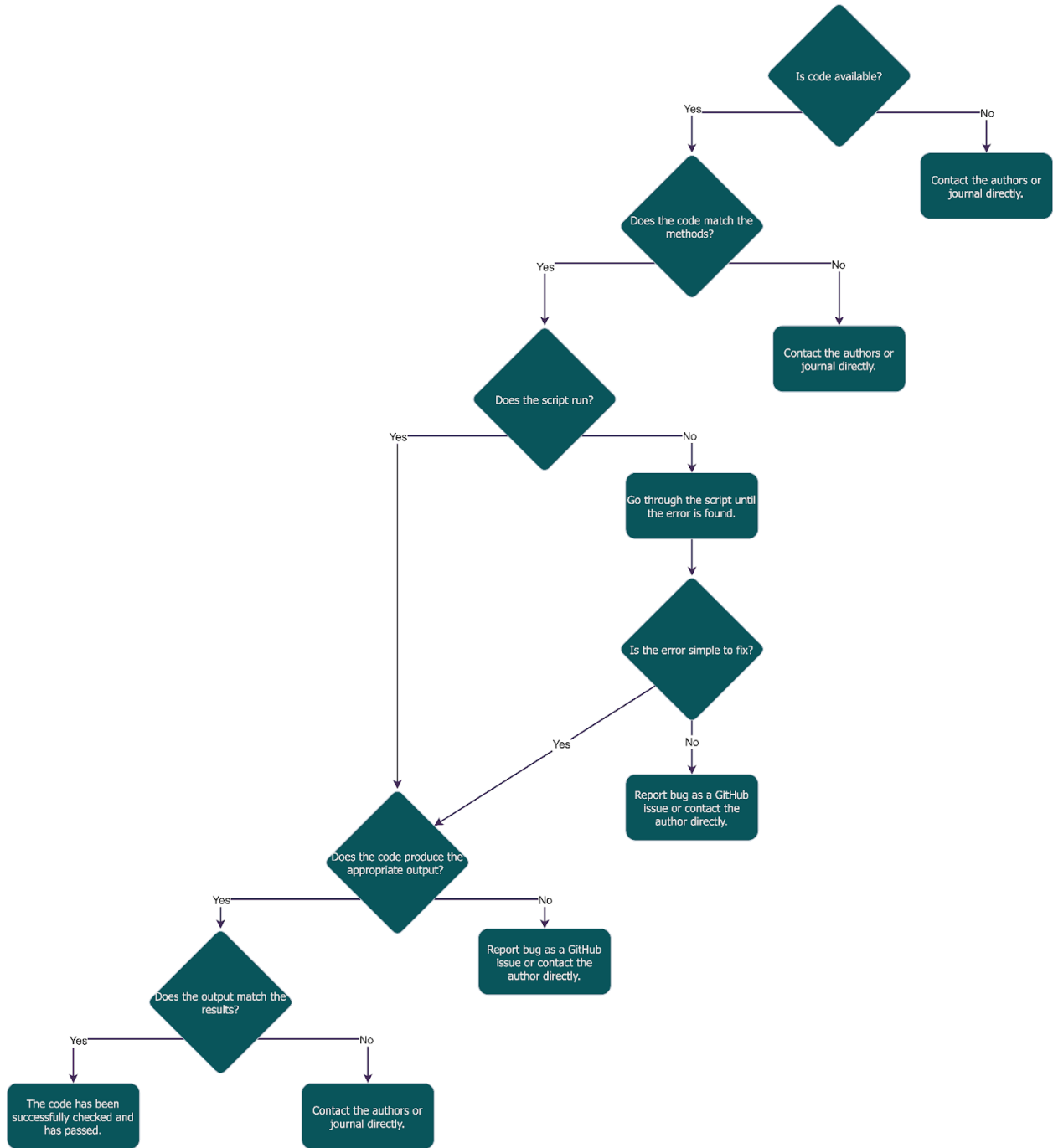
368 *Post-Publication: Reviewing code after publication*

369 Reviewing code post-publication is another facet of code review but one that has been much
370 less discussed. Although it does not prevent *publication* of incorrect results, it does enable
371 checking if code is indeed adhering to the R’s listed above (Fig. 1). However, the initial question
372 should be, has all code used to produce the results been made available? This can either be a
373 yes (stored and available on any data or code repository) or a no. Fortunately, an increasing
374 number of journals are now *requesting* code be shared alongside scientific articles (Culina *et al.*,
375 2020), such as in supplemental materials or by linking to an online repository. This then allows
376 for any open and shared code to be checked and verified alongside methods section statements
377 (Stodden, 2011; Light *et al.*, 2014). However, unlike data, code is a lot less likely to be made
378 available regardless of these mandatory journal policies. As Figure 2 from Culina *et al.* (2020)
379 shows, although the number of journals that possesses a mandatory code rule is increasing
380 (from 15% in 2015 to 79% in 2020) the number of articles that actually provide open code is still
381 around 27% (although this number varies considerably among journals). This suggests that not
382 many authors are adhering to this policy, which is an impediment to computational
383 reproducibility (Culina *et al.* 2020). However, there is hope to be found here. As Culina *et al.*
384 have shown, journals requiring code to be shared are increasing in number yearly and, as a
385 field, we already have improved substantially (Mislán *et al.*, 2016; Culina *et al.*, 2020; Jenkins *et*
386 *al.*, 2023). In some cases, journals have implemented far stricter (and rightly so) data and code
387 requirements along with assigning corresponding data editors (Bolnick, 2022). However, the first

388 necessary step is for all journals to make it a requirement for *both* code and data to be present
389 from the very start of the submission stage (Powers & Hampton, 2019; Fernández-Juricic,
390 2021). But what happens if the code is not available? In this case, the main option is to reach
391 out to the corresponding author (or perhaps the journal itself) and ask if the code could be made
392 available; similar to data being made available “upon reasonable request”.

393 The next part is relevant to the previous section above (“What should code review evaluate?). If
394 you find that the code associated with a manuscript does not adhere to any of the “R”s listed
395 above, then the first step is to contact the corresponding author (or if the paper uses the MeRIT
396 system (Nakagawa *et al.*, 2023), the person who actually conducted the analysis). This could be
397 in the form of a GitHub issue if there is a repository for the code or an email (see Fig. 3). If there
398 is indeed an error in code, and it is not due to differences in software version (e.g., differences
399 in R and package versions) or due to inherent stochasticity (e.g., simulations or MCMC
400 sampling), then the authors should be given a chance to contact the journal themselves to
401 highlight and correct their mistakes. For instance, as per American Naturalist’s stance (Bolnick,
402 2022) authors who contact the journal to correct code or data errors will not be penalised and
403 corrections are encouraged (when warranted). However, in cases where updated results would
404 alter the narrative of a published paper, corrections may be more difficult to address without newer
405 methods of documenting changes. Publication versioning or “living” documents may present a solid
406 first step in such a scenario (Kane & Amin, 2023). By encouraging post-publication code review,
407 we can both decrease the proliferation of coding errors and also increase the reliability of
408 published science.

409



410

411 Figure 3. An example peer code review flowchart that can occur post-publication. Figure design
 412 by J.L.P and E.I.C.

413

414 *Concluding remarks*

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

422

423 *Acknowledgements*

424 This work began during workshops at the 2021 and 2022 annual conferences of the Society for
425 Open, Reliable, and Transparent Ecology and Evolutionary biology (SORTEE) run by S.M.W.
426 and E.I.C. (in 2022). We also would like to acknowledge Fonti Kar for their help with organising
427 and delivering the 2021 workshop. This work was partially funded by the Center of Advanced
428 Systems Understanding (CASUS), which is financed by Germany's Federal Ministry of
429 Education and Research (BMBF) and by the Saxon Ministry for Science, Culture and Tourism
430 (SMWK) with tax funds on the basis of the budget approved by the Saxon State Parliament. We
431 also thank Corina Logan, Serena Caplins, the PREreview group (Gracielle Higino, Varina
432 Crisfield, Mobina Gholamhosseini, Katherine Hébert, and Tanya Strydom), and one anonymous
433 reviewer for thoughtful comments on the manuscript. We thank Melina de Souza Leite for
434 allowing us to use their GitHub issue as an example of SORTEE peer code review.

435

436 *References*

437 Alston, J.M. & Rick, J.A. (2021) A Beginner's Guide to Conducting Reproducible Research.
438 *Bulletin of the Ecological Society of America*, 102, 1–14.

439 Archmiller, A.A., Johnson, A.D., Nolan, J., Edwards, M., Elliott, L.H., Ferguson, J.M., *et al.*
440 (2020) Computational Reproducibility in The Wildlife Society's Flagship Journals. *The Journal of*
441 *Wildlife Management*, 84, 1012–1017.

442 Badampudi, D., Britto, R. & Unterkalmsteiner, M. (2019) Modern code reviews - Preliminary
443 results of a systematic mapping study. In *Proceedings of the Evaluation and Assessment on*
444 *Software Engineering*, EASE '19. Association for Computing Machinery, New York, NY, USA,
445 pp. 340–345.

446 Bakker, T.C.M. & Traniello, J.F.A. (2020) Ensuring data access, transparency, and preservation:
447 mandatory data deposition for Behavioral Ecology and Sociobiology. *Behavioral Ecology and*
448 *Sociobiology*, 74, 132.

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

450 Blischak, J.D., Carbonetto, P. & Stephens, M. (2019) Creating and sharing reproducible
451 research code the workflowr way. *F1000Research*, 8, 1749.

452 Boettiger, C. (2015) An introduction to Docker for reproducible research. *ACM SIGOPS*
453 *Operating Systems Review*, 49, 71–79.

454 Boettiger, C. (2017) Generating CodeMeta Metadata for R Packages. *The Journal of Open*
455 *Source Software*, 2, 454.

456 Bolnick, D. (2022) EIC Update: American Naturalist policy on data and code archiving.

457 Bolnick, D.I. & Paull, J.S. (2009) Morphological and dietary differences between individuals are
458 weakly but positively correlated within a population of threespine stickleback. *Evolutionary*
459 *Ecology Research*.

460 Botvinik-Nezer, R., Holzmeister, F., Camerer, C.F., Dreber, A., Huber, J., Johannesson, M., *et*
461 *al.* (2020) Variability in the analysis of a single neuroimaging dataset by many teams. *Nature*,
462 582, 84–88.

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

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

468 Chure, G. (2023) Git + GitHub As A Platform For Reproducible Research.

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

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

472 Data sharing and mining | Royal Society [WWW Document]. (2023) . URL
473 <https://royalsociety.org/journals/ethics-policies/data-sharing-mining/> [accessed on 2023].

474 Errington, T.M., Denis, A., Perfito, N., Iorns, E. & Nosek, B.A. (2021) Challenges for assessing
475 replicability in preclinical cancer biology. *eLife*, 10, e67995.

476 Eustace, S. (2023) poetry: Python dependency management and packaging made easy.

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

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

481 Filazzola, A. & Lortie, C. (2022) A call for clean code to effectively communicate science.
482 *Methods in Ecology and Evolution*.

483 Gomes, D.G.E., Pottier, P., Crystal-Ornelas, R., Hudgins, E.J., Foroughirad, V., Sánchez-
484 Reyes, L.L., et al. (2022) Why don't we share data and code? Perceived barriers and benefits to
485 public archiving practices. *Proceedings of the Royal Society B: Biological Sciences*, 289,
486 20221113.

487 Gompel, M. van. (2023) CodeMetaPy: Generate and manage CodeMeta software metadata.

488 Goodman, S.N., Fanelli, D. & Ioannidis, J.P.A. (2016) What does research reproducibility
489 mean? *Science Translational Medicine*, 8.

490 Hennessy, E.A., Acabchuk, R.L., Arnold, P.A., Dunn, A.G., Foo, Y.Z., Johnson, B.T., *et al.*
491 (2022) Ensuring Prevention Science Research is Synthesis-Ready for Immediate and Lasting
492 Scientific Impact. *Prevention Science*, 23, 809–820.

493 Huijgen, R., Boekholdt, S.M., Arsenault, B.J., Bao, W., Davaine, J.-M., Tabet, F., *et al.* (2012)
494 RETRACTED: Plasma PCSK9 Levels and Clinical Outcomes in the TNT (Treating to New
495 Targets) Trial: A Nested Case-Control Study. *Journal of the American College of Cardiology*, 59,
496 1778–1784.

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

500 Indriasari, T.D., Luxton-Reilly, A. & Denny, P. (2020) A Review of Peer Code Review in Higher
501 Education. *ACM Transactions on Computing Education*, 20, 1–25.

502 Jenkins, G.B., Beckerman, A.P., Bellard, C., Benítez-López, A., Ellison, A.M., Foote, C.G., *et al.*
503 (2023) Reproducibility in ecology and evolution: Minimum standards for data and code. *Ecology*
504 *and Evolution*, 13, e9961.

505 Kane, A. & Amin, B. (2023) Amending the literature through version control. *Biology Letters*, 19,
506 20220463.

507 Lai, J., Lortie, C.J., Muenchen, R.A., Yang, J. & Ma, K. (2019) Evaluating the popularity of R in
508 ecology. *Ecosphere*, 10, e02567.

509 Lamprecht, A.-L., Garcia, L., Kuzak, M., Martinez, C., Arcila, R., Martin Del Pico, E., *et al.*
510 (2020) Towards FAIR principles for research software. *Data Science*, 3, 37–59.

511 Landau, W.M. (2021) The targets R package: a dynamic Make-like function-oriented pipeline
512 toolkit for reproducibility and high-performance computing. *Journal of Open Source Software*, 6,
513 2959.

514 Light, R.P., Polley, D.E. & Börner, K. (2014) Open data and open code for big science of
515 science studies. *Scientometrics*, 101, 1535–1551.

516 Lipow, M. (1982) Number of Faults per Line of Code. *IEEE Transactions on Software*
517 *Engineering*, SE-8, 437–439.

518 Ma, C. & Chang, G. (2007) Retraction for Ma and Chang, Structure of the multidrug resistance
519 efflux transporter EmrE from Escherichia coli. *Proceedings of the National Academy of*
520 *Sciences*, 104, 3668–3668.

521 Miller, G. (2006) A Scientist’s Nightmare: Software Problem Leads to Five Retractions. *Science*,
522 314, 1856–1857.

523 Minocher, R., Atmaca, S., Bavero, C., McElreath, R. & Beheim, B. (2021) Estimating the
524 reproducibility of social learning research published between 1955 and 2018. *Royal Society*
525 *Open Science*, 8, 210450.

526 Mislán, K.A.S., Heer, J.M. & White, E.P. (2016) Elevating The Status of Code in Ecology.
527 *Trends in Ecology & Evolution*, 31, 4–7.

528 Müller, K. (2020) *here: A Simpler Way to Find Your Files*.

529 Müller, K. & Walthert, L. (2020) Styler: Non-invasive pretty printing of R code. *R package*
530 *version 1.3. 2*.

531 Nakagawa, S., Ivimey-Cook, E.R., Grainger, M.J., O’Dea, R.E., Burke, S., Drobniak, S.M., *et al.*
532 (2023) Method Reporting with Initials for Transparency (MeRIT) promotes more granularity and
533 accountability for author contributions. *Nature Communications*, 14, 1788.

534 Nelson, S. & Schumann, J. (2004) What makes a code review trustworthy? In *37th Annual*
535 *Hawaii International Conference on System Sciences, 2004. Proceedings of the*. Presented at
536 the 37th Annual Hawaii International Conference on System Sciences, 2004. Proceedings of
537 the, p. 10 pp.-.

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

541 Okken, B. (2022) *Python Testing with pytest*. Pragmatic Bookshelf.

542 Peikert, A. & Brandmaier, A.M. (2021) A Reproducible Data Analysis Workflow With R
543 Markdown, Git, Make, and Docker. *Quantitative and Computational Methods in Behavioral
544 Sciences*, 1–27.

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

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

549 Pipenv maintainer team. (2023) pipenv: Python Development Workflow for Humans.

550 Powers, S.M. & Hampton, S.E. (2019) Open science, reproducibility, and transparency in
551 ecology. *Ecological Applications*, 29, e01822.

552 Quintana, D.S. (2020) A synthetic dataset primer for the biobehavioural sciences to promote
553 reproducibility and hypothesis generation. *eLife*, 9, e53275.

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

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

558 Simonsohn, U. & Gruson, H. (2023) groundhog: Version-Control for CRAN, GitHub, and GitLab
559 Packages.

560 Stodden, V. (2011) Trust Your Science? Open Your Data and Code, 2.

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

563 The Luigi Authors. (2023) luigi: Workflow mgmt + task scheduling + dependency resolution.

564 Tiwari, K., Kananathan, S., Roberts, M.G., Meyer, J.P., Sharif Shohan, M.U., Xavier, A., *et al.*
565 (2021) Reproducibility in systems biology modelling. *Molecular Systems Biology*, 17, e9982.
566 Touchon, J.C. & McCoy, M.W. (2016) The mismatch between current statistical practice and
567 doctoral training in ecology. *Ecosphere*, 7, e01394.
568 Ushey, K. (2023) renv: Project Environments.
569 Ushey, K., McPherson, J., Cheng, J., Atkins, A., Allaire, J.J. & Allen, T. (2022) packrat: A
570 Dependency Management System for Projects and their R Package Dependencies.
571 Wickham, H. (2011) testthat: Get Started with Testing. *The R Journal*, 3, 5–10.
572 Williams, D. & Bürkner, P.-C. (2020) Coding Errors Lead to Unsupported Conclusions: A critique
573 of Hofmann et al. (2015). *Meta-Psychology*, 4.