# Reviewer Response for: Software engineering principles to improve quality and performance of R software

Seth Russell1, Tellen D. Bennett1,2, Debashis Ghosh1,3

1 University of Colorado Data Science to Patient Value (D2V), Anschutz Medical Campus, Aurora, CO, United States of America

2 Pediatric Critical Care, University of Colorado School of Medicine, Aurora, CO, United States of America

3 Biostatistics and Informatics, Colorado School of Public Health, Aurora, CO, United States of America

Corresponding Author:

Seth Russell

CU Data Science to Patient Value

13199 E. Montview Blvd, Suite 210-15

Aurora, CO 80045 USA

# Email address: [seth.russell@ucdenver.edu](mailto:seth.russell@ucdenver.edu)

# General Comments

We thank the reviewers and editor for their time and effort in reviewing this work. This resubmission is a much better work due to the feedback we have received.

As we have had the opportunity to participate in this review and resubmit cycle, we have re-run the analysis to pull current packages, dependencies, and source code from CRAN. All tables, figures, and in text numbers have been updated.

In updating data, we noticed that Supplemental Table S2 “Pkgs w/Dep” row had been inadvertently left out, causing subsequent rows to be off by one from the row heading and the last row to be a duplicate of the first. We have added the missing row.

# Reviewer Response: Rory Nolan

**Response to “Basic Reporting”**

*I think for the sake of saving people the need for a dictionary, "autodictacts" should be replaced by "self-taught individuals" and "codified through" should be replaced by "constructed using" or something like that.*

We appreciate your suggestion for more clear and plain language. Corrected.

*Line 69 is missing a full stop.*

*Line 129 is missing its last word (years)*

Corrected.

*Line 140 has a misformatted citation (jpresse)*

*Line 483 has a misformed citation*

Corrected. You will also find we resolved some inconsistencies with several of the other citations. The PeerJ Zotero style had unexpectedly left out some important URLs and dates.

*Line 166 has "overtime" where it should say "over time"*

*Line 167 should say "percentage" and not "percent"*

Corrected.

**Response to “Comments for the author”**

*I find the examples of unit testing using the pccc package a little difficult to understand. Sure, they can be understood with some effort, but I think there must be an easier example out there which would permit a more concise code listing…*

Thank you for the advice and guidance. Unit testing and optimization principles are most easily demonstrated as part of the development process. While the glue package has simple and clear functionality and has extensive unit tests, getting examples from the glue development process that addresses tests and optimization is nontrivial. Based on your recommendations and the recommendations of the other reviewer, we have adjusted the provided code listings to make the examples clearer.

Please note that based on the other reviewer’s feedback, the example code is now farther back in the paper, but still organized by examples of testing and performance.

*The study of packages employing optimization is very interesting. However, with testing I think it's easy to say if a package isn't tested, it should be, but a package without obvious optimization attempts could just be very well written (using only vectorized code from other packages) and hence not need more explicit optimizations of its own. I think this should be stated: a package without obvious optimization isn't necessarily in need of optimization. Having said this, figure 3 is interesting, giving the change in explicit optimization efforts over the years.*

This has been addressed in the section “Key recommendations for profiling and benchmarking.” Some of the relevant text:

“The first step to software optimization is to understand the functional and non-functional requirements of the software being built. Based on expected input, output, and platform the software will be run on, one can make a decision as to what is good enough for the software being developed. A pragmatic approach is best – do not spend time optimizing if it does not add value. Once the functional requirements have been correctly implemented and validated, a decision point is reached: decide if the software is slow and in need of evaluation and optimization. While this may seem a trivial and unnecessary step, it should not be overlooked; a careful evaluation of costs versus benefit from an optimization effort should be evaluated before moving forward.”

An at the end of the same section:

“’Beware the dangers of premature optimization of your code. Your first duty is to create clear, correct code.’ (Knuth, 1974; Burns, 2012) Never optimize before you actually know what is taking all the time/memory/space with your software. Different compilers and core language updates often will change or reverse what experience has previously indicated as sources of slowness. Always benchmark and profile before making a change.”

*In figure 1, I think the "No" colour should be on the bottom and in particular in fig 4, "None" should be on the bottom, not in the middle. "None" should maybe also be given a striking colour to show that it's different to the other results.*

A consistent color selected for None/No. None/No has been moved to the bottom of the stacked bar charts.

*It should be made clear in the caption of fig 4 what the n>14 is about.*

We have updated the caption of figure 4 as well as added additional details in the captions of other figures.

*Consider using bench::mark instead of microbenchmark…*

Thank you for the recommendation; bench does provide additional information on memory allocation and garbage collection not available from microbenchmark. bench and benchr have been added to the list of benchmarking software and included in the “Optimization of R Packages” section. Based on monthly direct download statistics from RDocumentation, microbenchmark is the most popular at 204 (<https://www.rdocumentation.org/packages/microbenchmark/versions/1.4-6>), benchr has 46 (<https://www.rdocumentation.org/packages/benchr/versions/0.2.2>), bench has 36 (<https://www.rdocumentation.org/packages/bench/versions/1.0.1>). As microbenchmark is more commonly used, we’ve provided one example using microbenchmark and one example using bench.

# Reviewer Response: Greg Wilson

**Response to “Basic Reporting”**

*- Professional structure (but I have asked for some reorganization).*

*- References provided are good, but more are needed to substantiate specific claims (see general discussion below).*

See “Response to Specific” where we address reviewers specific concerns by line number.

**Response to “Validity of the findings”**

*- Empirical results on testing and performance optimization seem trustworthy.*

*- High-level discussion of rules to follow only partly backed up by empirical evidence (see general discussion).*

See “Response to Specific” where we address reviewers specific concerns by line number.

**Response to “General”**

*1. Put the empirical material on testing and optimization in one section near the front of the paper to show that there is significant room for improvement.*

Done

*2. Replace the paragraph-length high-level advice on how to test and tune with bulleted lists of rules, each having pointers to longer-form discussions. This will help experienced readers (who will nod at the lists of rules), while also helping newcomers operationalize those rules (which I think they would struggle to do based on the current brief explanations).*

We appreciate the reviewer’s suggestion and have attempted to make our recommendations sections more accessible to both expert and novice readers. To do so, we provided bulleted lists of key points and shortened the discussion of each recommendation. We agree with the reviewer that most of our key recommendations require further discussion and have provided additional references for readers who want to learn more.

*3. Expand the examples to show specific applications of the general rules. For example, I would like to see the performance results for the PCCC code on line 413 and following, and then see what changes the authors made to the code to speed it up, and a second set of performance figures. Similarly, in the testing example starting on line 301, I do not know what the bug was that the test found, or how the bug was detected before the test was written and its detection then translated into a test.*

We have expanded example code and provided output of benchmarking tests.

The example from line 413 is a performance validation test and is intended to catch performance regression as development progresses. We have clarified the purpose of this test and provided benchmarking output. A more focused example of optimizations can be found in the “Key recommendations for profiling and benchmarking” section.

The example from line 533 is intended to demonstrate how code run time can be improved. We have provided additional details and link to sourcecode to help user who want to read further.

**Response to “Specific”**

*28: "we show that reproducible and replicable software tests are not available" -> as written, this is a very strong claim.*

We have amended this wording to reflect more accurately the results as shown in the paper; i.e. about 70% of R packages on CRAN do not have tests available with the package.

39: Introduction of software engineering seems disconnected with preceding material.

Addressed this in the Introduction.

*69: missing "." between "maintenance" and "Software maintenance"*

Resolved.

*71: "chief factor" -> isn't people's time the chief factor in \_all\_ maintenance, not just that for statistical software?*

You are correct, removed limitation of statement to research/statistical software.

*76: "As research scientists tend to think..." -> Are they wrong? I.e., if I assert that the majority of software written by researchers exists to solve one-off problems, rather than to be used repeatedly, is there data to show that I'm wrong?*

Added some references to address this:

“As research scientists tend to think of their software products as unique tools that will not be used regularly or for a long period, they often do not consider long term maintenance issues during the development phase (Sandve et al., 2013; Prins et al., 2015).”

*81: Here and elsewhere, I worry that "best practices" is not validated. There is, for example, no published research showing that the use of version control makes people more productive. (Believe me, I've looked.) I think the authors need to present evidence that various practices actually improve productivity and/or reliability before calling them "best".*

Introduction has been updated to clarify what is meant by ‘best’:

“The term ‘best’ as referenced in the previous cited work and others cited later refers to expert consensus based on knowledge and observational reporting of results from application of the practices.”

In other locations we use the term ‘best’ as used in original source of cited papers.

*102: Do the authors have data showing that commercial software and/or open source software are tested any more frequently? (I've seen a \_lot\_ of projects on GitHub that don't have any tests...)*

We make no claims to testing coverage of commercial nor open source software; our intent was to state that testing is always recommended. Updated wording to clarify:

“Software testing is a well-established component of any software development lifecycle (Atchison et al., 1968) and should also be a key component of research software.”

*140: What is "jpreese"?*

This is a malformed citation which has been corrected. You will also find we resolved some inconsistencies with several of the other citations. The PeerJ Zotero style had unexpectedly left out some important URLs and dates.

*149: I'm unclear what is meant by "Grep for...directories" - are the authors working from textual manifests of projects, or are they using "grep" as a synonym for "search for"?*

Clarified wording for the Testing section as well as Optimization section.

*160: should "updated" be "most recently updated"?*

Clarified wording

*166: "over time" rather than "overtime"*

Corrected

*172: a table might be a better way to display this summary of "have X but not Y".*

See supplemental tables.

*184: There is also growing evidence that TDD doesn't actually make developers more productive (see e.g. http://people.brunel.ac.uk/~csstmms/FucciEtAl\_ESEM2016.pdf).*

Thank you for the excellent reference; we’ve updated text and reference list to include it.

*185: "A better approach..." Better by what criteria, and what data can the authors cite to support this contention?*

Clarified what was meant by “better” and provided some references for readers:

“An approach that more closely matches the theoretically based software development cycle and flexible nature of research software is to create tests after a requirement or feature has been implemented (Osborne et al., 2014; Kanewala & Bieman, 2014).”

*194: "In an ideal world...100% test coverage" Do the authors mean line coverage, statement coverage, branch coverage, combinatorial coverage, ...?*

Our new text reads: “In an ideal world, any software developed would be accompanied by 100% test coverage validating all lines of code, all aspects of functionality, all input, and all interaction with other software.”

*208: "Once answers to these questions are known..." I have found that giving high-level advice like this only frustrates most scientists, because they don't know how to operationalize it. Can the authors point at concrete examples of how to translate these general rules into specific decisions, priorities, and/or tests for specific software packages?*

This is now in addressed in the “SOFTWARE TESTING” and “SOFTWARE OPTIMIZATION” sections.

*226: "In addition to unit tests, users should perform..." I believe the authors mean "developers", not "users" (but could be wrong). I also think that this statement trips over an important distinction between testing what-if contingencies for software tools (which developers of packages should do), and testing specific users of those tools and their inputs for particular analyses (which analysts doing particular analyses should do). I think that repeatable sanity checks make sense for the latter, but that doesn't necessarily mean use of unit testing frameworks.*

By “users” we mean the person who will run the software for some purpose. We clarified language about potential division of labor in testing processes.

*233: A notebook demonstrating the use of the software is \_not\_ the same thing as an acceptance test, though the automatic validation of the expected end result may be.*

We did not intend to imply that an example of how to run the software is an acceptance test. We have clarified our language around how software demonstrations can be helpful in testing.

*238: Radcliffe et al have developed a nice framework for thinking about the ways in which data analysis can go wrong, which is summarized in the figure in http://stochasticsolutions.com/pdf/TDDA-One-Pager.pdf. It seems that most of the discussion in this paper is focusing on Step 2 of that model's 5-step process - if so, the authors may wish to cite that model and make this explicit, and if not, expand their recommendations for other phases.*

Our work is not based on the referenced framework by Radcliffe et al.

The Radcliffe framework does closely match the process computational researchers already follow: become knowledgeable in a specific domain, determine a scientific question of merit, determine methods for answering the chosen scientific question, perform experiments to provide answers to the chosen question, analyze results, reporting of results – and potentially re-evaluating the original question and repeating steps as necessary. This work focuses on the non-domain specific components of software developed to support research; our current scope of testing is focused primary in the area Radcliffe identified as “Errors of implementation”. To cover other areas of the framework would require domain specific guidance.

We do see a few problems in Radcliffe’s work. Radcliffe reports how likely errors are to arise in various phases of the framework with no evidence provided to support the provided error rates. Test Driven Development requires one to start any software implementation with first writing tests for unwritten software. The Test Driven Data Analysis framework does not recommend developing tests first; it proposes to address the 5 steps of the framework with 2 solutions. The first solution the authors recommend is to create an analysis processes that one can re-run (either the original developer or others others) and provide a mechanism to verify that the results produced are correct. The second solution focuses on data validation/verification. We have covered both reproducibility and data validation in this work; see “INTRODUCTION”, “SOFTWARE TESTING”, and “Key recommendations for identifying and validating performance targets” sections.

*301-302: some odd indentation.*

Indentation issues due to line wrapping; resolved.

*430: "run 10 timess and find the mean" No - this can easily give a misleadingly high performance result because of caching effects.*

You are correct that caching effects can affect how a repeated test performs. We have clarified the text related to interpretation of results of benchmarking tests.

Repeated iterations are the default for popular R benchmarking packages. See rbenchmark (defaults to running 100 repetitions; see https://cran.r-project.org/web/packages/rbenchmark/rbenchmark.pdf), microbenchmark (defaults to running a test 100 times; see <https://cran.r-project.org/web/packages/microbenchmark/microbenchmark.pdf>), and benchr (defaults to running a test 100 times; see <https://cran.r-project.org/web/packages/benchr/benchr.pdf>). The package bench uses adaptive timing that results in a variable number of runs (from 1 to 1000 by default, see <https://cran.r-project.org/web/packages/bench/bench.pdf>).

*437: "There are many optimizations that can be considered..." This is the kind of advice that my students used to find frustrating, because it tells readers that something exists without telling them what it is or where to find it.*

We have expanded this section and added some references.

*560: the summary of possible optimizations is very condensed - can the authors point at resources that have lengthier and more detailed coverage? (There are several guides to high performance R...)*

This is now included in:

“Key recommendations for identifying and validating performance targets:” “There are many resources beyond this work that can provide guidance on how to minimize RAM and disk resources (Kane, Emerson & Weston, 2013; Wickham, 2014b; Wickham et al., 2016; Klik, Collet & Facebook, 2018).”

And in: “Key recommendations for profiling and benchmarking:” “For some additional information on R optimization, see (Wickham, 2014b; Robinson, 2017).”