When is done *done*? Towards the fundamentals of automated testing for collaborative data analysis development

Charles T. Gray, Daniel Fryer, Elise Gould, and Aaron todo

Fundamentals

In Chuang C. Chang's *Fundamentals of Piano Practice* the author sets out to address a gap in piano pedagogy (Chang 2009). A similar gap exists in the implementation of *good enough* practices (Wilson et al. 2017), which is to say, what we might reasonably expect of a analyst, in reproducible computing for research data analyses. The objective is different from advanced pianism, however, in this manuscript we shall chracterise this gap by considering automated testing workflow in research data analysis through piano pedgogy. Through attempting to identify the fundamentals of automated testing for collaborative data analysis development, this manuscript aims to articulate the gap in understanding automated testing implementation for analysts.

As Chang notes, whilst there a rich history of technical pedagogy, there is a dearth of guidance for pianists on *learning* the technique (Chang 2009). There are many canonical texts of pianistic technique pedagogy. Bach provides a pathway from small canons [todo: cite], to two-part inventions[todo: cite], three-part sinfonia, and the challenge of *The Well-Tempered Clavier* and *The Art of Fugue*. Bartok provides the *Mikrokosmos* [todo:], and Czerny's *School of Velocity* [to do]. In each case technical exercises of increasing difficulty are provided. In piano pedagogy, a *technical* exercise isolates a particular aspect of pianistic technique [todo examples]. For example, [todo Czerny staccato]. Or, [todo voicing technique].

The dearth that Chang attempts to address is in pianistic *practice* habits that will lead to succesful adoption of these techniques. In science, we might call this *work flow* [todo cite].

> .. practically every piano learning method consists of showing students what to practice, and what kinds of techniques (runs, arpeggios, legato, staccato, trills, etc.) are needed. There are few instructions on how to practice order to be able to play them, which is mostly left to the student and endless repetitions (Chang 2009).

Wilson et al. followed their work on *best practices* in scientific computing (Wilson et al. 2014), with a reflection on *good enough* practices (Wilson et al. 2017), in recognition that we must account for what we might reasonably request of practitioners of data analysis. In this manuscript, we consider one component of *good enough* practice in data analysis: *automated testing*.

Automated tests are a formalised way of implementing checks that inputs and outputs of algorithsms are as expected (Wickham 2015). Informative messages are output that assist the developer in identifying where code is not performing as expected.

Collaboration via automated testing

At heart, automated tests are collaborative. This may be with others, but applies at least as much to a analyst approaching their own past work with which they have become unfamiliar. Automated tests provide an efficient way of returning to and extending an analysis.

Reproducible research compendia provide a means by which analysts can share their work so that it may be extended by others, and automated tests provide `code::proof` (Gray 2019), a measure of confidence in the

algorithm for others. Hayes expresses concern about using algorithms published without automated tests (Hayes 2019). However, only one quarter of the largest repository of R packages, The Comprehensive R Archive Network, have any automated tests (Hester 2016), highlighting that, despite testing identified as a 'vital' part of data analysis (Wickham 2015), automated testing is yet to be widely adopted.

Indeed, as noted in Wilson's testing primers (in development) for RStudio ("RStudio Cloud," n.d.),

> Almost all (92%) of the catastrophic system failures are the result of incorrect handling of non-fatal errors explicitly signaled in software. In 58% of the catastrophic failures, the underlying faults could easily have been detected through simple testing of error handling code (Yuan et al. 2014).

Scientific errors that are inadvertently introduced by faulty workflow are characterised in metascience as *questionable research practices*.

Questionable research practices in scientific computing

One way to characterise automated tests is to consider them part of a suite of methodologies that are being developed to mitigate *questionable research practices* (Fraser et al. 2018). For social scientists and ecologists, a literature is emerging on how our methodologies inadvertently lead to errors in scientific reasoning, p-hacking, altering a statistical model to achieve a desired outcome. But there are others, too, *HARKing*, hypothesising after the fact, where the question is changed to achieve a desired outcome from the model, and *cherry picking*, leaving out troublesome aspects of the analysis that don't fit a desired narrative.

Gould is extending these ideas to an examination of the workflow of ecological models, adapting Gelman and [todo]

[todo: gould extension bridge to computing]

As researcher software engineers, it behoves us to consider what are questionable research practices in software produced for data analyses. Version control and open code sharing via a platform such as GitHub, is one way to mitigate questionable research practices in scientific computing (Bryan 2017). There is also a growing literature on reproducible research compendia via packaged analyses (Marwick, Boettiger, and Mullen 2018; Wilson et al. 2017).

This manuscript contributes to this literature by focussing on workflows for automated tests that assist the developer communicate what they have coded to others and their future self, as a means of mitigating questionable research practices in scientific computing.

A test-driven toolchain walkthrough for an in-development analysis

In this section we now consider the practicality of implementing tests through a *toolchain walkthrough*, an opinionated documetation of a scientific workflow , towards a measure of `code::proof`, confidence in the algorithm implemented (Gray 2019). In particular, a toolchain walkthrough is a reflection of *one* workflow, whilst others, indeed, better, workflows might exist. This is in contrast to a comprehensive review of tools. Instead, a toolchain walkthrough ruminates on considerations *after* a set of tools have been chosen.

Toolchain walkthroughs aim to identify obstacles and advantages in implementation of scientific workflows. By necessity, a toolchain walkthrough is language specific, but, ideally, observations emerge that are useful for any appropriate language employed for data analysis, such as Python. This toolchain walkthrough examines the *process*, analogous to piano *practice*, of implementing tests, as opposed to defining comprehensively the nature of *good enough* tests, analogous to guidance on pianistic technique.

Objective of this toolchain walkthrough

This toolchain walkthrough aims to provide guidance on implementing a test-driven workflow for an in-development analysis. Many analyses begin as scripts that develop [todo find blog post on this]. The central question of this manuscript is what constitutes a minimal level of testing for in-development analysis, where code is still being written and features implemented.

This is a first effort in identifying the fundamentals of automated testing for the collaborative process of developing an analysis in R. Analgous to Riederer's *RMarkdown-driven development* (Riederer 2019), which deconstructs the workflow of developing an analysis from .Rmd notebook-style reports to packaged analyses, we consider a set a computational tools that form a workflow to assist in the coherent development of automated tests for data analysis. This is an extension of the workflow suggestions provided in *R Packages*, with a specific focus on collaborative workflows in research.

Devops and assumed expertise

This toolchain walkthrough assumes a knowledge of R advanced enough to be using the language to be answering scientific research claims.

References

Bryan, Jennifer. 2017. "Excuse Me, Do You Have a Moment to Talk About Version Control?" *PeerJ PrePrints* 5: e3159.

Chang, Chuan C. 2009. *Fundamentals of Piano Practice*.

Fraser, Hannah, Tim Parker, Shinichi Nakagawa, Ashley Barnett, and Fiona Fidler. 2018. "Questionable Research Practices in Ecology and Evolution." *PLOS ONE* 13 (7): e0200303. https://doi.org/10.1371/journal.pone.0200303.

Gray, Charles T. 2019. "Code::Proof: Prepare for Most Weather Conditions." In *Statistics and Data Science*, edited by Hien Nguyen, 22–41. Communications in Computer and Information Science. Singapore: Springer. https://doi.org/10.1007/978-981-15-1960-4_2.

Hayes, Alex. 2019. "Testing Statistical Software - Aleatoric." Blog.

Hester, Jim. 2016. "Covr: Bringing Test Coverage to R."

Marwick, Ben, Carl Boettiger, and Lincoln Mullen. 2018. "Packaging Data Analytical Work Reproducibly Using R (and Friends)." e3192v2. PeerJ Inc. https://doi.org/10.7287/peerj.preprints.3192v2.

Riederer, Emily. 2019. "RMarkdown Driven Development (RmdDD)." *Emily Riederer*.

"RStudio Cloud." n.d. https://rstudio.cloud/learn/primers.

Wickham, H. 2015. *R Packages: Organize, Test, Document, and Share Your Code*. O'Reilly Media.

Wilson, Greg, D. A. Aruliah, C. Titus Brown, Neil P. Chue Hong, Matt Davis, Richard T. Guy, Steven H. D. Haddock, et al. 2014. "Best Practices for Scientific Computing." Edited by Jonathan A. Eisen. *PLoS Biology* 12 (1): e1001745. https://doi.org/10.1371/journal.pbio.1001745.

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

Yuan, Ding, Yu Luo, Xin Zhuang, Guilherme Renna Rodrigues, Xu Zhao, Yongle Zhang, Pranay U. Jain, and Michael Stumm. 2014. "Simple Testing Can Prevent Most Critical Failures: An Analysis of Production Failures in Distributed Data-Intensive Systems." In *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation*, 249–65. OSDI'14. Broomfield, CO: USENIX Association.