Research Software Sharing, Publication, & Distribution Checklists

Best practices for sharing your code from basic analysis scripts to bioinformatic pipelines

Richard J. Acton

2026-05-20

Outline

- Role of software in research & Defining Terms
- Case studies of errors
- Current state of reproducibility
- What can we do about it?
  - The Checklists
- Example checklist items

> ⚠ **Warning**
>
> This projecct is in Alpha.
>
> Some checklists are incomplete & I am looking for collaborators to refine them.

Slides

rsspdc.org

https://rsspdc.gitlab.io/slides/hpc-best-practices_2026-05-20.html

archived repository

Software in research: Ubiquitous & Integral

- **Essentially all modern research makes use of software in some way**
  - Embedded in Instruments
  - Statistical Analysis & Vizualisation
  - Modelling & Simulation
  - etc.
- **Important decisions, in the clinic, in public policy, and in future investment of time and resources are made based on the outputs of research software**

Many research projects include some software as a research output:

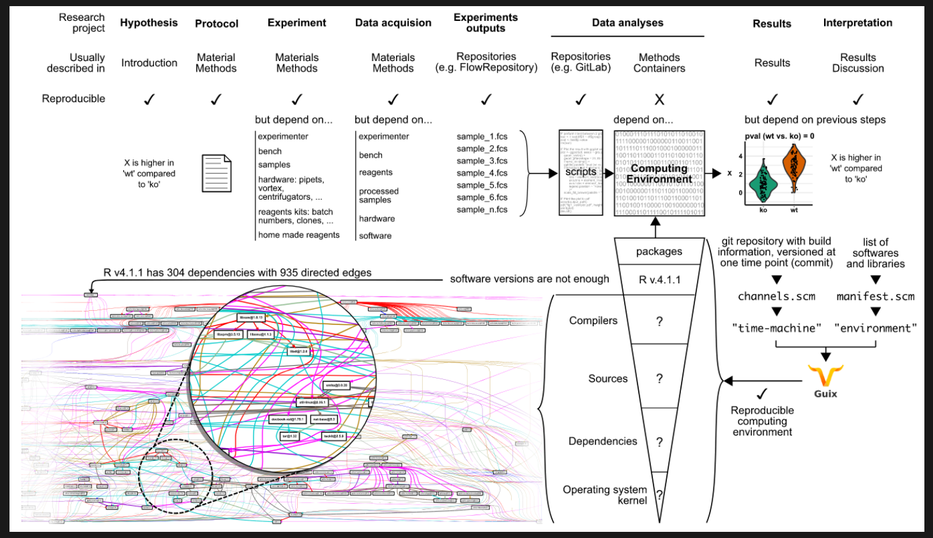
- Not limited to software tools for use by others
- May only be ‘one-off’ scripts specific to your analysis
  - **These are a part of the Methods the work**

Code as Scholarship

> An article about computational science in a scientific publication is **not** the scholarship itself, it is merely **advertising** of the scholarship. The actual scholarship is the complete software development environment and the complete set of instructions which generated the figures.
>
> “Claerbout’s Doctrine” - Codified by Buckheit & Donoho **1995** [1] CiTO:provides quotation for

Computational Verifiability

> [V]erifiability should be treated as an issue of human–computer interaction. In fact, verifiability requires scientists to have a clear notion of the scientific models and methods applied by a piece of software, irrespectively of whether they are performing or reviewing research.
>
> …
>
> An essential part of verification is … to check if the computation correctly implements the informal description given in the article, or inversely if the journal article correctly describes what the software does.
>
> – *"Verifiability in computer-aided research: the role of digital scientific notations at the human-computer interface"* [2] CiTO:credits



Reproduced from: 'Toward practical transparent verifiable and long-term reproducible research using Guix' [10] Figure 1

Beyond the ability to re-run the code and get that same result

Source code describing both the analysis and the environment in which it is run pre-requisite. Ideally a Full Source Bootstrap of the environment should be possible [3,4]. Container or virtual machine images which cannot themselves be reproducibly rebuilt don't satisfy verifiability [5]. Verifiability falls off over time if source code and build instructions are not archived [6–9].

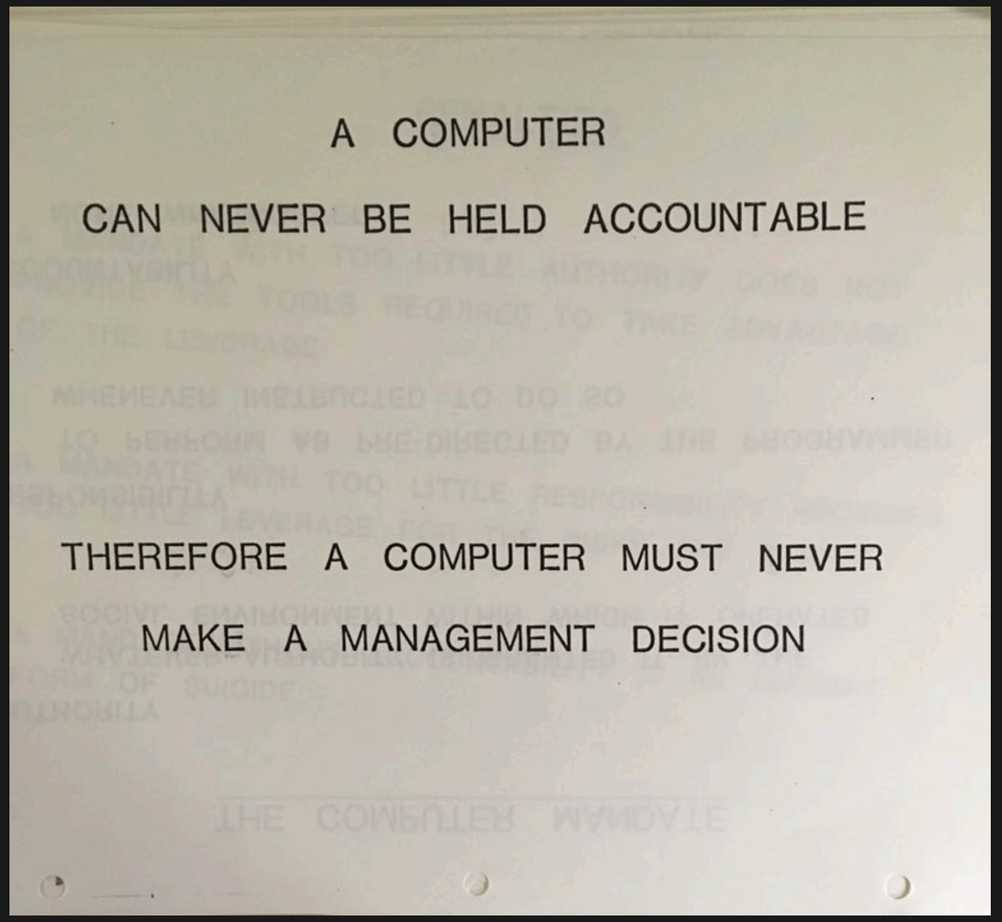
- Complex problem in modern software stacks
  - The R 4.1.1 interpreter requires 304 binaries from a dependency graph with 935 edges [10] CiTO:cites as data source
  - That's before we get to any R packages
- Not well addressed by most currently popular tools for managing software environments

What about AI?

> A computer can never Be held accountable
> Therefore a computer must never author a scientific paper

- As the author of a scientific work you are accountable for its contents
- If you use LLMs[1] to generate code it is your responsibility to understand that code well enough to check its correctness

Specifics of best practices for responsible LLM use are not otherwise addressed here - yet



Internal IBM training document, 1979

Four possibilities of reproducible computation

- **Inspectability**
  - the possibility to inspect all input data and source code
- **Executability**
  - the possibility to run the code on a suitable computer to verify the results
- **Explorability**
  - the possibility to explore the behavior of the code by inspecting intermediate results, making small modifications, or using code analysis tools
- **Confirmability**
  - the possibility to confirm that the published executable versions correspond to the available source code

My modification to Pol Dellaiera's restatement [11] CiTO:credits of Konrad Hinsen's four possibilities of reproducible scientific computations [12] CiTO:credits. Inspired by the free software foundations four freedoms.

What Is Reproducibility?

		Data	
		Same	Different
Analysis	Same	**Reproducible** A result is reproducible when the same analysis steps performed on the same dataset consistently produces the same answer.	**Replicable** A result is replicable when the same analysis performed on different datasets produces qualitatively similar answers.
	Different	**Robust** A result is robust when the same dataset is subjected to different analysis workflows to answer the same research question (for example one pipeline written in R and another written in Python) and a qualitatively similar or identical answer is produced. Robust results show that the work is not dependent on the specificities of the programming language chosen to perform the analysis.	**Generalisable** Combining replicable and robust findings allow us to form generalisable results. Note that running an analysis on a different software implementation and with a different dataset does not provide generalised results. There will be many more steps to know how well the work applies to all the different aspects of the research question. Generalisation is an important step towards understanding that the result is not dependent on a particular dataset nor a particular version of the analysis pipeline.

[13] CiTO:provides quotation for

Errors in Research Code: Examples & Case Studies

A Glitch with the “Willoughby-Hoye” Scripts for Calculating NMR Chemical Shifts [14] CiTO:gives background to

- Characterising chemicals produced by cyanobacteria using nuclear magnetic resonance
- Authors happened to discover a platform dependent bug in scripts widely used with this analytical method
- “Willoughby-Hoye” scripts published in **2014** nature protocols paper **cited over 130 times** when issue was published in **2019**

The Bug:

- A python script read two sets of files and assumed that they would be sorted in the same order in order the pair them up and use information in the paired file to perform a calculation
- On windows this assumption held and both sets of files were listed in the same order, on Linux it did not files and were paired incorrectly
- Particularly pernicious failure mode as both outputs appear superficially valid and no errors are thrown

Simple bugs, data mislabelling & duplication [15] CiTO:gives background to

An exercise in “forensic bioinformatics” was undertaken by Keith Baggerly and Kevin Coombes after they noticed what they thought was a swapped group label in some published work they were trying to reproduce . This effort uncovered a number of simple yet potentially consequential errors.

These included but were not limited to:

- An off-by one error leading to a list of gene names corresponding to the wrong data.
- A label switching error leading to sensitive cell lines being labelled as resistant, and resistant lines as sensitive.
- A number of instances where supposedly distinct samples had identical data, including identical samples assigned to different groups.

They were looking at a series of studies aimed at predicting patient response to different drugs using gene expression data from cell lines treated with those drugs. **These predictions where used to allocate patients to different treatment arms in clinical trials.**

Quick-fire round

- 5 papers retracted after a bug in a script for processing protein structure data flipped two columns [16] CiTO:gives background to
- Paper retracted after treatment groups for a COPD [1] intervention were swapped, finding the intervention to result in fewer hospitalisations than existing treatment when it would actually result in more [17] CiTO:gives background to
  - *fortunately?* another error was found in imputation of missing values which meant the result was not significant
- Highly politically influential paper used to justify austerity spending policies in major economies contained an excel spreadsheet error which unintentionally excluded five countries entirely (Australia, Austria, Belgium, Canada and Denmark) from the analysis. This and other errors meant that: “Over 1946-2009, countries with public debt/GDP ratios above 90% averaged 2.2% real annual GDP growth, not -0.1% as published.” [18] CiTO:gives background to
- …

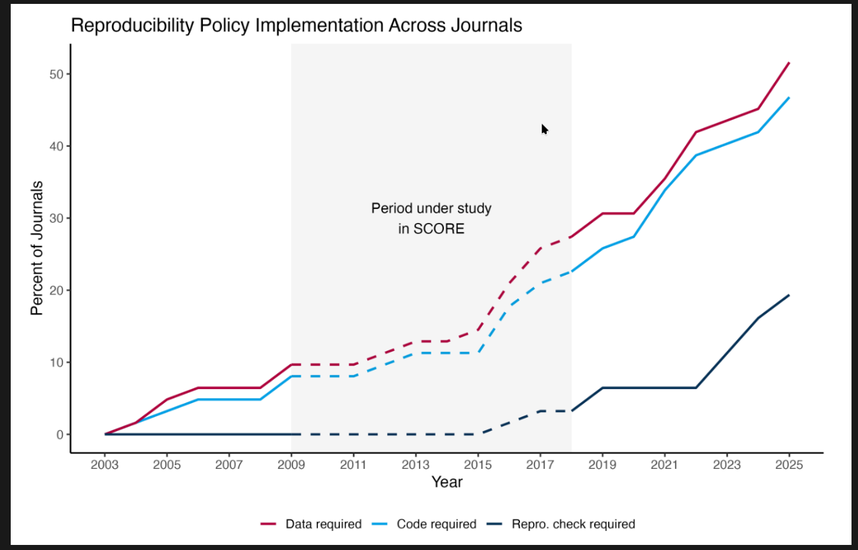
Pattern: When a single error is discovered an closer scrutiny applied more are often found

Current State of Computational Reproducibility

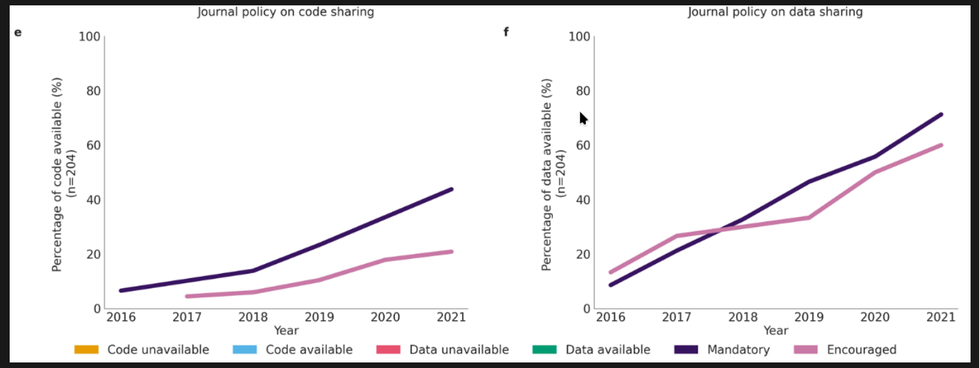
Code & Data Sharing, & policies requiring sharing are increasing

Good News!

- Code & Data availability is increasing over time
- Code & Data Availability policies are becoming more prevelant over time
- Code is more likely to available from journals with “mandatory” code deposition
  - Enforcement of mandatory code deposition policies appears lacking



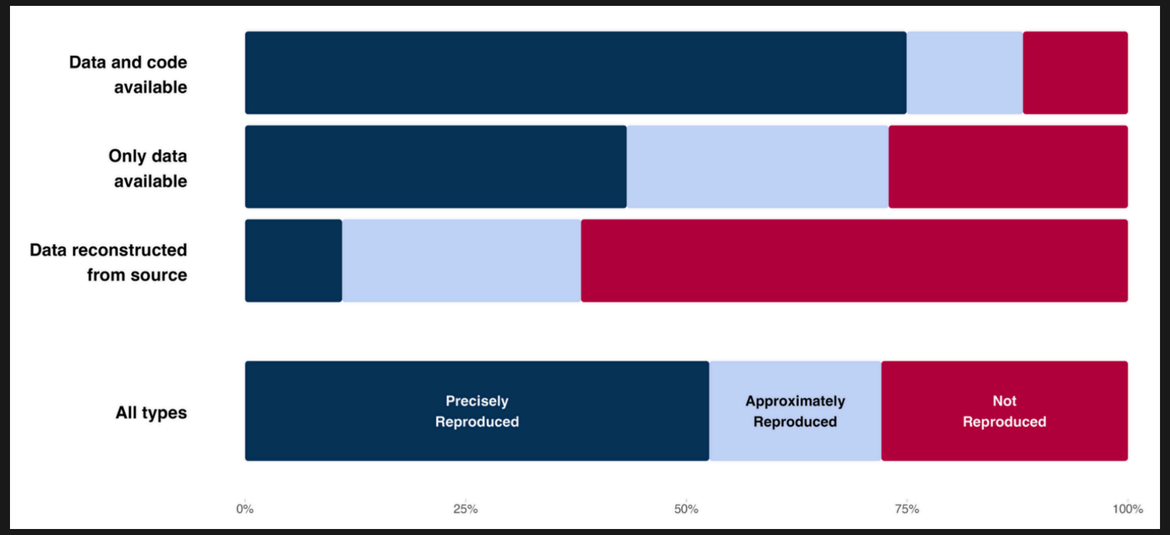
% of the 62 social sciences journals studied [19] CiTO:cites as data source



Papers with code/data available in 8 Biomedical Journals [20] CiTO:cites as data source

Code Sharing Increases Reproducibility

In both number of results and the precision with which they can be reproduced



[19] CiTO:cites as data source

- **Code and data** were available approximate or precise reproducibility for 73.3 of the 83.2 papers (88.1%) and precise reproducibility for 62.4 of the 83.2 papers (75.0%).
- **Only data** were available, attempt to reproduce the findings with new code following the analyses described in the paper. Of these, we observed approximate or precise reproducibility for 16.1 of the 22.1 papers (72.9%) and precise reproducibility for 9.5 of the 22.1 papers (43.2%).
- **Author-prepared data were unavailable, but source data were available**, attempted to reproduce the findings by preparing the data and generating new code. Of these, we observed approximate or precise reproducibility for 15.1 of the 39.7 papers (38.1%) and precise reproducibility for 4.4 of the 39.7 papers (11.0%)

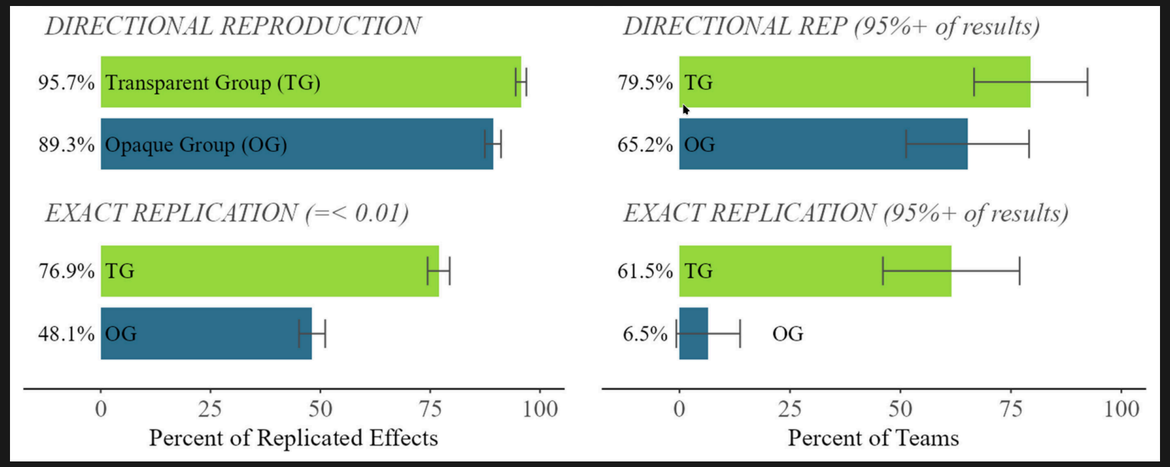
Take-aways:

- Data & Code is much better than data alone
- Code increases fraction of results precisly reproduced
- Data cleaning and preparation code makes a big difference

Code Sharing Increases Reproducibility

In both number of results and the precision with which they can be reproduced

Single study which reported multiple results which were replicated by multiple teams.



[21] CiTO:cites as data source

Teams were split into two groups:

- Transparent group with full access to data code, 39 teams, total of 1,872 effect replication attempts.
- Obfuscated group with no access to code or numerical results, 46 teams, total of 1,874 effect replication attemts.
- Directional reproduction: Results point in the same direction, exact numbers need not match
- Exact replication: Results are withing 1% of the original values

Take-aways:

- If enough people try someone can probably reproduce your result, even with opaque methods
  - Meaning a lot of those trying will fail or get incorrect results
- Transparency makes reproduciblility more accessible to a variety of analysts / more robust to between analyst variation

Code & Data Sharing Do Not Guarantee Reproducibility

- 2019 reproducibility study in hydrology & water resources found a **1.1% rate of 'full reproducibility'** sample design intended to increase likelihood of including reproducible papers [22] CiTO:cites as data source.
  - **5.6% made data, code / models, & Documentation available, only ~1/5 of those "doing it right" were 'fully' reproducible**
- Similar 2018 analysis on papers in Science had analogous rates of 'full' reproducibility **2.0%** (4/204) [23] CiTO:cites as data source
  - **Substantially higher rates after requesting code & data from authors**
    - 32% (65/204) 'potentially reproducible', extrapolating from a sub-set **26% reproducible**.
- **Introducing a policy improved availability** [23] CiTO:cites as data source
  - Uplift in 'potential reproducibility' upon request post-policy introduction indicates a 'deposition gap'
  - More **specific voluntary journal policies appeared more effective than vague mandatory ones** [22] CiTO:cites as data source

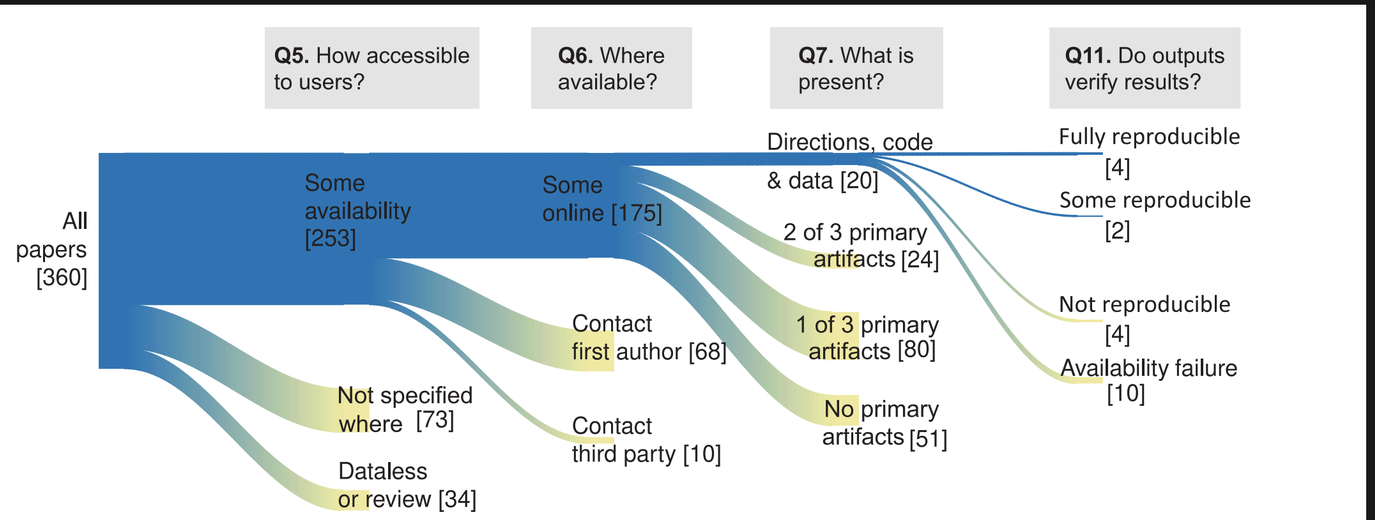


Figure 2. Number of papers progressing through the survey questions to determine availability and reproducibility requirements.

[22]

Table 5. Changes in disclosure practices

Disclosure practice	2009–2010, %	2011–2012, %
Citations to data and/or code in references	25	29
Data location given in acknowledgements	29	48
Code location given in acknowledgements	4	5

Table 6. Materials availability via inspection

Materials availability	2009–2010, %	2011–2012, %
Most or all relevant data locations given	52	75
Most or all relevant software locations given	43	54
Some data and software locations given	25	45
All major software and data locations given	15	25
Code, scripts, parameters, documentation	10	12
No supporting materials available	4	1

[23]

Why does computational reproducibility matter?

- Focus on the underlying or analytical sources of variation not technical ones
- Transparency of process, understanding how we came to our conclusions
- Finding & Correcting Errors with implications for consequential decisions
- Not losing time to ‘dependency hell’ & ‘it worked on my machine’

What Can We Do About This?

- **What are the best practices for treating software as a research output?**
- **How can we get them adopted?**
- Inspired by Community-developed checklists for publishing images and image analyses [24] CiTO:likes.

> For scientists wishing to publish obtained images and image-analysis results,
> there are currently no unified guidelines for best practices

The same applies to research software outputs

- The QUAREP-LiMi [25],[26] CiTO:cites as related community has been very successful at organising, & getting buy-in and adoption from various parties in the imaging space.
- Nature Communications, Nature Cell Biology, Nature Methods & Nature Structural & Molecular Biology are trialling adding light microscopy information to their reporting tables and cite QUAREP-LiMi [27],[28],[29] CiTO:cites for information.

I would like to try and emulate them for research software.

Existing resources

Checklist / Decision Tree

- Software Management Plan Decision Tree based on the Practical guide to Software Management Plans [30] CiTO:recommended reading
- Self-assessment for FAIR research software based on the FAIR Principles for Research Software [31,32] CiTO:recommended reading

Software Management Planning resources from ELIXIR, SSI, DRA Canada, & others [33–35] CiTO:recommended reading

All excellent resources

However

- Too developer focused
  - Advice primarily for software packages
- Insufficiently aspirational
- Long boring PDFs that aren't very fun 🥱
- Lack of integration into existing workflows

Checklists - Quick Overview

> Checklists for anyone publishing a research paper that has any analysis code associated with it, is developing a software tool which will be used by researchers, or is deploying a web service which will be used by researchers.

11 part **4** tiered checklists for **4** different types of software output

---

Software Output Types:

- Generic Tools
  - Unitary tool / software package
  - Multi-part workflows / Pipelines
- Web-based service
- Record of a specific analysis / Research Compendium
- (Embedded - help wanted)

Format:

Simple markdown text files with expandable details sections

NEW Issue templates for GitHub/GitLab issues

NEW Lite variants without the tips embedded

Parts: Generic Areas & Motivating Questions

📒 **Source control**	*How can you keep track of the history of your project and collaborate on it?*
© **Licencing**	*On what terms can others use your code, and how can you communicate this?*
📖 **Documentation**	*How do people know what your project is, how to use it and how to contribute?*
🔗 **Making Citable**	*How should people make reference to your project and credit your work?*
✅ **Testing**	*How can you test your project so you can be confident it does what you think it does?*
🤖 **Automation**	*What tasks can you automate to increase consistency and reduce manual work?*
👥 **Peer review / Code Review**	*How can you get third party endorsement of and expert feedback on your project?*
📦 **Distribution**	*How can people install or access the software emerging from your project?*
💿 **Environment Management / Portability**	*How can people get specific versions of your software running on their systems?*
🌱 **Energy Efficiency**	*How can you and your users minimise wasted energy?*
⚖ **Governance, Conduct, & Continuity**	*How can you be excellent to each other, make good decisions well, and continue to do so?*

Tiers:

🥉 Bronze (easy), 🥈 Silver, 🥇 Gold, 🏆 Platinum (very hard)

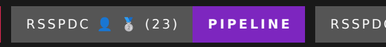
Gamified with badges & a points system 🎮

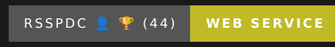
RSSPDC 👤 🥇 (34) **SOFTWARE PACKAGE**

Points & Medals System

- You must get at least bronze in all areas to get a bronze medal overall
  - Bronze across the board - Highly attainable
- You get 1 point for a bronze and 4 for a platinum
- After bronze your overall medal tier is the mean, rounded down to the nearest integer of your other scores
  - Platinum across across the board - practically impossible

Repo Badges





3rd party attestation

📒 **Source control**	1	1	4	1	2	3	3	4
© **Licencing**	1	1	1	1	2	4	3	4
📖 **Documentation**	1	1	1	1	2	3	3	4
🔗 **Making Citable**	1	1	4	1	2	4	3	4
✅ **Testing**	1	2	1	1	2	2	3	4
🤖 **Automation**	1	1	1	1	2	1	3	4
👥 **Peer review / Code Review**	1	1	4	1	2	3	3	4
📦 **Distribution**	1	1	1	1	2	3	3	4
💽 **Environment Management / Portability**	1	1	1	1	2	2	3	4
🌱 **Energy Efficiency**	1	1	4	1	2	3	3	4
⚖ **Governance, Conduct, & Continuity**	**0**	**0**	**0**	1	2	4	3	4
Total Score	10	11	22	11	22	32	33	44
Scaled Score (`floor(total score / 11)`)	0	1	2	1	2	2	3	4
Overall Medal	NA	NA	NA	🥉	🥈	🥈	🥇	🏆

Workflows

File in repo

- Add the appropriate checklist file to your repo when you begin your project
- Check boxes off as you meet them
  - Commit checks with their changes

```
---a/rsspdc-checklist.md
+++b/rsspdc-checklist.md
@@ -1,1 +1,1 @@
+ [x] README
- [ ] README
```

```
---a/README.md
+++b/README.md
@@ -0,0 +1,1 @@
+ This project is about...
```

Issue Templates

- Issue templates are available for GitHub/GitLab
  - Single ‘tracking’ issue
  - Seperate issues per theme

Website

rsspdc.org

Examples today from:

Record of a specific analysis Or Research Compendium Considerations for publishing code which runs a specific analysis that underpins some result to be published in the academic literature.

The emphasis here is on making the work narrowly reproducible i.e. the analysis of the same data can produce the same result when it is re-run. This is a starting point for making results robust (different analysis, same data) and replicable (same analysis, different data), and ultimately generalisable (different analysis, different data).

The other emphasis is on making the work ‘verifiable’, exposing the complete step-wise detail of the reasoning underpinning the analysis so that it can be scrutinised and understood.

📒 Source control

How can you keep track of the history of your project and collaborate on it?

- ‘git history’ as part of the ‘lab notebook’ for the comuptational project
  - Temporal sequence
  - Curated but sometimes complex and non-linear
- Ecosystem of tooling & services around git and git hosting that enable other checklist items
  - Distribution, automations for testing, making citable & documentation, platforms for collaboration including review
- 🏆 Advanced uses
  - Commits can be cryptographically signed and timestamped
    - Allows for strong guarantees of provenance for code and data (via git lfs / annex)

📒 Source control - Checkbox items

- [ ] Uses git (or other source control tool)
  - [ ] 🥉Bronze *(Easy)*: Using version control but has a shallow project history, just placed in git for distribution
  - [ ] 🥈Silver *(Intermediate)*: Longer project history, commit messages of mixed quality, some large messy changes
  - [ ] 🥇Gold *(Hard)*: Silver plus - Well written commit messages, nice granular commits making discrete self-contained changes. Tags, releases, or branches at major project milestones, maybe some contributions from other users
  - [ ] 🏆Platinum *(MAXIMUM OVERKILL)*: Gold plus - Some from: conventional commits; Clean history with a consistent rebasing/merging strategy; Signed commits from all contributors; Contributions go through a consistent workflow like, issues, then a pull request from a branch.

© Licencing

On what terms can others use your code, and how can you communicate this?

- Must explicitly allow others to use your code if you don't want them to potentially be liable for copyright infringement
  - Similar to open access for a manuscript
  - without a license code posted publicly is a type of 'source available' code, but not 'open source'
- Types of open license:
  - Permissive - *mostly* do whatever you want with it
  - Copy left - heritably open, derived works must also be open
- May need to license your figures and prose with a creative commons license

Proprietary [1] == Alchemy

- Don’t expect people to believe your results if you won’t show them your code

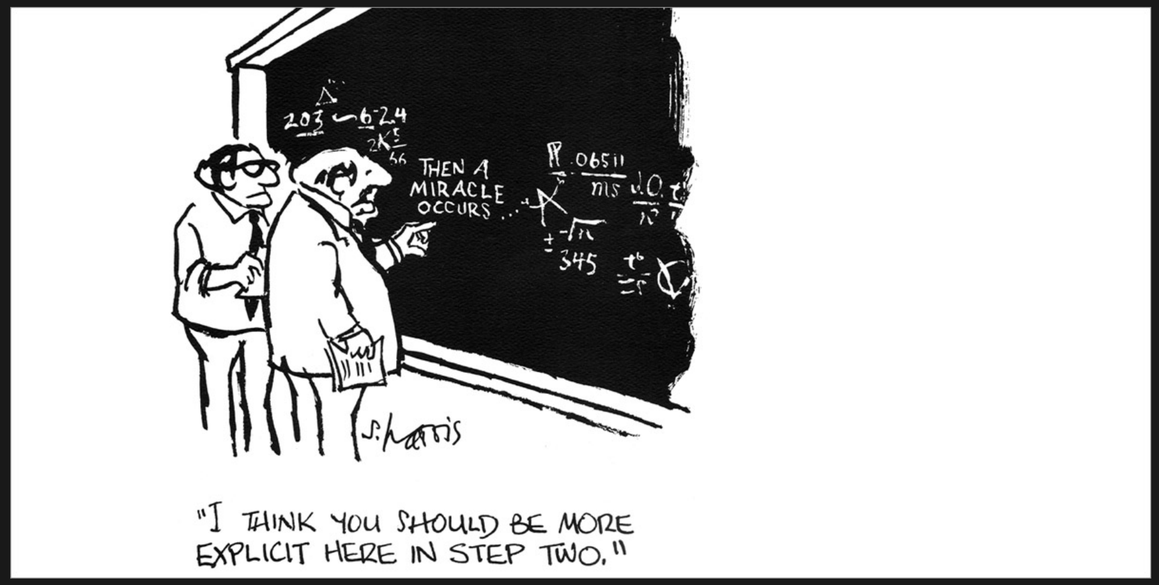


Image Credit: Sidney Harris

© Licencing - checkbox items

- [ ] Project is suitably licensed
  - [ ] 🥉Bronze *(easy)*: There is a LICENSE file in the repository for a license which meets one of the OSI, Debian, or FSF/GNU definitions of free/libre or open source software. Or for any contents that are not software a Creative Commons license.
  - [ ] 🥈Silver *(easy)*: If any prose/documentation or images are licenced differently from the code in the project this is indicated and those licences provided. If licences have an attribution requirement there is easy to copy text/links for appropriate attribution.
  - [ ] 🥇Gold *(intermediate)*: Uses Software Package Data Exchange (SPDX) license identifiers for every file/suitable unit of code. With a tool such as REUSE.software to automate and standardise the process.
  - [ ] 🏆Platinum *(intermediate)*: all previous tiers plus any images have licensing information embedded in their metadata.

📖 Documentation

How do people know what your project is, how to use it and how to contribute?

- Clear documentation is ***critical*** for reproducibility of a one off analysis
- Important to support peer / code review - clear enough for someone other than you to run your analysis
- Usage
  - Environment
    - Dependencies - libraries you've used in your code (versioned)
    - file layout - where your code expects everything to be
  - Installations
    - how to get your code and its dependencies ready to run
  - Run
    - What steps to execute in what order with what commands, flags, inputs and outputs
    - Format / Structure expectations for inputs / outputs
  - Modify
    - Details such as project structure, enough that someone else can understand it, pick it up and usefully alter the code
- Why & How?
  - Explanations of the problem the code tackles and details of how

📖 Documentation - checkbox items

- [ ] Project has suitable documentation
  - [ ] 🥉Bronze *(easy)*: Project has README file that either contains directly or links to resources which answer these questions:
    - [ ] Provides a description of the project structure so that the user knows which directories to find things in, possibly including a visual representation of the structure
    - [ ] Contains instructions with sufficient detail for someone else to re-run the analysis
    - [ ] What is the name or title of the project for which this code is record?
    - [ ] Why did you peform this analysis?
    - [ ] What problem does it address and how?
    - [ ] How do I set up an environment to re-run it? (with example)
    - [ ] What inputs do I need?
    - [ ] What outputs should I expect?
    - [ ] How do I cite the project?
    - [ ] Who contributed to the project and what did they contribute?
    - [ ] Who should I contact, about what, and how?
    - [ ] What should I expect / Not expect if I contact you?
    - [ ] How is the project Licensed?
    - [ ] Table of Contents (if long enough to benefit from one)
    - [ ] Bibliography (if reference is made to external resources)
  - [ ] 🥈Silver *(intermediate)*: Documentation / comments explain WHY things are done in the code
  - [ ] 🥇Gold *(intermediate)*: You have a simple worked example of your analysis methods with example data to illustrate its soundness in a simple case
  - [ ] 🏆Platinum *(MAXIMUM OVERKILL)*: The authors manuscript of you paper is a literate programming artefact with numbers, tables and figures programaticaly generated and built in a reproducible computational environment. Additional documentation of details not suitable for the main manuscript are in the supplementary material also as literate programming artefacts, more lengthy and complex elements of the analysis might be run as a pipeline or scripts then read from for inclusion in the manuscript. Building the manuscript (without caching) reruns the complete analysis and produces the same output (you may need to set random seeds for any analysis that makes use of pseudorandom number generation).

🔗 Making Citable

How should people make reference to your project and credit your work?

- Providing metadata readable by reference managers
  - Include a CITATION.cff in your repository

```yaml
authors:
  - family-names: Druskat
    given-names: Stephan
    orcid: https://orcid.org/1234-5678-9101-1121
title: "My Research Software"
version: 2.0.4
```

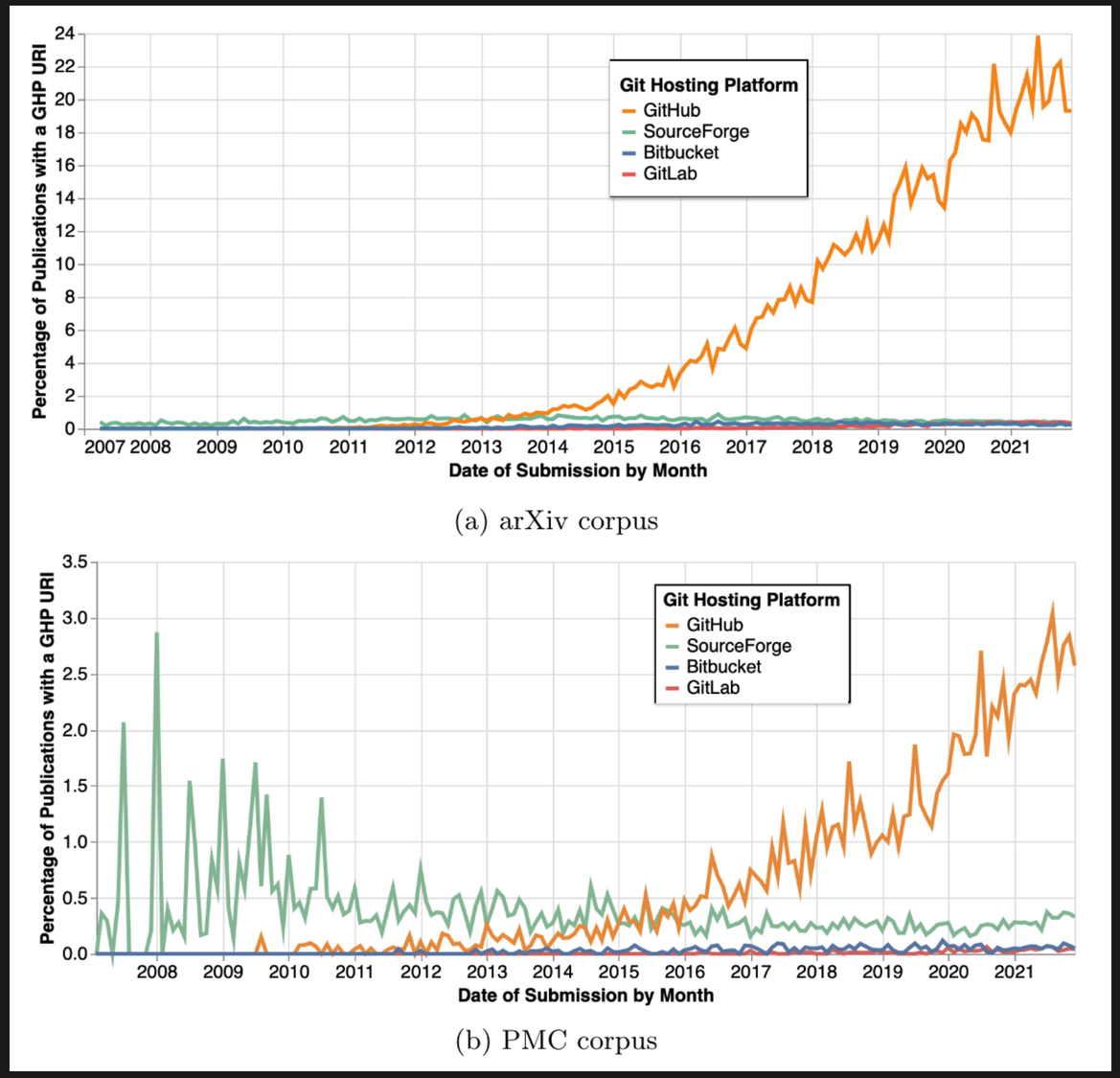
- Minting a **unique**, **persistent**, **resolvable** identifier
  - Zenodo for DOIs (suitable for one off analyses)
  - software heritage for SWHIDs (arguably better for packages - commit level granularity)
  - Use in the manuscript, bench protocol, and dataset metadata
- Opportunity to emphasise credit for this part of the work that is due to any technical specialists that worked on it

The Rise of GitHub

- Links to GitHub in scholarly papers have risen precipitously
- A GitHub URL is not a persistent identifier
  - github repos can be deleted and/or changed
- GitHub makes no commitment to long term archival storage

Suitable persistent identifiers:

- Software Heritage SoftWare Hash IDentifier (SWHID)
  - Captures entire git history
- Zenodo DOI
  - Only captures snapshots



[36] CiTO:cites as data source

🔗 Making Citable - checkbox items

- [ ] Record is Citable
  - [ ] 🥉Bronze *(easy)*: A CITATION.cff file exists in the code repository to provide citational metadata about your project
  - [ ] 🥈Silver *(easy)*: bronze plus the project has persistent resolvable identifier such as a DOI or SWHID, with which it can be referenced, which has been minted for the project using a tool like zenodo or Software Heritage's Archive to store an archival copy of the project.
  - [ ] 🥇Gold *(intermediate)*: silver plus:
    - [ ] Contributions are credited using a suitable contributor roles ontology or taxonomy (CROT) such as CrediT, ScoRo, CRO, or TaDiRAH.
    - [ ] All contributors are identified by their ORCID or other suitable persistent identifier
  - [ ] 🏆Platinum *(mixed)*: gold plus any two or more from:
    - [ ] (easy) All research institutions are identified by their ROR ID
    - [ ] (intermediate) Versioned persistent identifier with automation to update snapshots on zenodo or similar tool when a new version is created.
    - [ ] (intermediate) Annotating work cited in this work with the Citation Typing Ontology (CiTO)
    - [ ] (hard) Your environment is defined with Nix or Guix - this might not seem like it contributes to making software more citable see details below for why this is the case.

👥 Peer review / Code Review

How can you get third party endorsement of and expert feedback on your project?

- Simple check - can someone else you know install and run your code from your repo using only the included instructions?
- Slightly more formal external validation someone else can run your code
  - CODECHECK
  - ReproHack
- More complete / academic code peer review e.g. from rOpenSci/pyOpenSci, JOSS is typically limited to software packages, though should in theory also form part of the regular peer review process

👥 Peer review / Code Review - checkbox items

- [ ] Code has been subject to a review indicating that someone else could re-run the analysis
  - [ ] 🥉Bronze *(easy)*: Someone other than you has checked over your project, given you feedback and told you they are reasonably confident they could re-run your analysis without your help.
  - [ ] 🥈Silver *(easy)*: Someone other than you has scessfully re-run your analysis using only your documentation, (preferably in a different compute environment, such as a different computer/compute cluster)
  - [ ] 🥇Gold *(intermediate)*: You have a review from CODECHECK, ReproHack or equivalent and have incorporated suggestions for improving reproducibility from these reviews.
  - [ ] 🏆Platinum *(intermediate)*: You have reviews which go beyond checking the ability to re-run your code but which also review it's technical correctness

💿 Environment Management / Portability

How can people get specific versions of your software running on their systems?

- List of software dependencies and their versions / dependencies
  - Ideally in a format which allows their automated install
  - Wherever possible use cross-platform tools for this (Containers, Nix)
- No hard coded absolute paths, paramaterise Input / Output paths and document defaults
  - ❌ `/home/richardjacton/project/data/sample01.csv`
  - ✅ `./data/sample01.csv` relative to project root
- Is it possible to package all or part of your project to make it easier to install?

💿 Environment Management / Portability - Checklist Items

- [ ] Computational environment description provided
  - [ ] 🥉Bronze *(Easy)*: List of package versions, e.g. output of `sessionInfo()` in R, not in a machine readable format
  - [ ] 🥈Silver *(Intermediate)*: Structured language specific environment decription, language environment can be re-created e.g. `renv.lock` in a mostly automated fashion
  - [ ] 🥇Gold *(Hard)*: Structured full environment description, automated ability to recreate the complete environment including system dependencies
  - [ ] 🏆Platinum *(MAXIMUM OVERKILL)*: Your description allows the automated bootstrap of the entire* depencency tree of your environment from source with bitwise binary reproducibility (currently almost impossible to achieve, basically only approachable in Guix)

⚖ Governance, Conduct, & Continuity

How can you be excellent to each other, make good decisions well, and continue to do so?

- Continuity
  - What happens to your code when you have left the lab?
  - Who can pick up where you left off, made and needed updates?
- Governance
  - Who has the admin rights on the GitHub Org
- Conduct
  - Ground rules for using it
  - What should be put here, when, and by who?

⚖ Governance, Conduct, & Continuity - Checkbox Items

- [ ] The project has a suitable governance model
  - [ ] 🥉 Bronze *(easy)*: The governance model is clearly communicated
  - [ ] 🥈 Silver *(easy)*: Bronze Plus - Project has continuity planning in place (2 or more from)
    - [ ] Source archived and/or mirrored to other platforms
    - [ ] Public archives of key project governance documentation and plans for continuity of operations in the events such as the loss of key project infrastructure
    - [ ] Plans of action in the event project admin(s) are no longer available
  - [ ] 🥇 Gold *(intermediate)*: Project has a governance model appropriate to its scale and goals
    - [ ] Project has clear and transparent processes
  - [ ] 🏆Platinum *(Hard)*: Project has a track record of good governance and policy, any from:
    - [ ] Decisions have involved the appropriate person(s) and been well documented
    - [ ] Disputes are largely resolved in a respectful and amicable fashion
    - [ ] The project leadership has learned from any mistakes and implemented policy changes as a result

This Presentation

https://rsspdc.gitlab.io/slides/hpc-best-practices_2026-05-20.html

archived repository

Questions / Feedback ?

Richard.Acton@babraham.ac.uk

info@rsspdc.org

Acknowledgements

Peter Rugg-Gunn

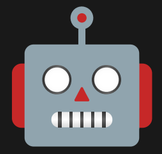
The Turing Way Community

Wellcome 215116/Z/18/Z

References

1. Buckheit JB, Donoho DL. WaveLab and Reproducible Research. Published online May 1995. https://purl.stanford.edu/gh803dm8254
2. Hinsen K. Verifiability in computer-aided research: The role of digital scientific notations at the human-computer interface. *PeerJ Computer Science*. 2018;4(7):e158. doi:10.7717/peerj-cs.158
3. Courtès L. Building a Secure Software Supply Chain with GNU Guix. *Programming*. 2022;7(1):1. doi:10.22152/programming-journal.org/2023/7/1
4. Nieuwenhuizen J, Courtès L. The Full-Source Bootstrap: Building from source all the way down. Published 2023--04-06. Accessed May 20, 2025. https://guix.gnu.org/en/blog/2023/the-full-source-bootstrap-building-from-source-all-the-way-down/
5. Nüst D, Sochat V, Marwick B, et al. Ten simple rules for writing Dockerfiles for reproducible data science. Markel S, ed. *PLoS Comput Biol*. 2020;16(11):e1008316. doi:10.1371/journal.pcbi.1008316
6. Malka J. Replication package for: Reproducibility of Build Environments through Space and Time. Published online January 16, 2024. doi:10.5281/ZENODO.10519820
7. Vallet N, Michonneau D, Tournier S. Toward practical transparent verifiable and long-term reproducible research using Guix. *Sci Data*. 2022;9(1):597. doi:10.1038/s41597-022-01720-9
8. Courtès L, Sample T, Zacchiroli S, Tournier S. Source Code Archiving to the Rescue of Reproducible Deployment. In: *Proceedings of the 2nd ACM Conference on Reproducibility and Replicability*. ACM; 2024:36-45. doi:10.1145/3641525.3663622
9. Perkel JM. Challenge to scientists: Does your ten-year-old code still run? *Nature*. 2020;584(7822):656-658. doi:10.1038/d41586-020-02462-7
10. Vallet N, Michonneau D, Tournier S. Toward practical transparent verifiable and long-term reproducible research using Guix. *Scientific Data*. Published online October 4, 2022. doi:10.1038/s41597-022-01720-9
11. Dellaiera P. Reproducibility in Software Engineering. Published online May 1, 2025. doi:10.5281/ZENODO.15315531
12. Hinsen K. The four possibilities of reproducible scientific computations. Published November 20, 2020. Accessed June 18, 2025. https://doi.org/10.59350/81ba8-dse91
13. Community TTW. The Turing Way: A handbook for reproducible, ethical and collaborative research. Published online April 14, 2025. doi:10.5281/ZENODO.15213042
14. Bhandari Neupane J, Neupane RP, Luo Y, Yoshida WY, Sun R, Williams PG. Characterization of Leptazolines A-D, Polar Oxazolines from the Cyanobacterium Leptolyngbya sp., Reveals a Glitch with the "Willoughby-Hoye" Scripts for Calculating NMR Chemical Shifts. *Org Lett*. 2019;21(20):8449-8453. doi:10.1021/acs.orglett.9b03216
15. Baggerly KA, Coombes KR. Deriving chemosensitivity from cell lines: Forensic bioinformatics and reproducible research in high-throughput biology. *Ann Appl Stat*. 2009;3(4). doi:10.1214/09-AOAS291
16. Miller G. A Scientist's Nightmare: Software Problem Leads to Five Retractions. *Science*. 2006;314(5807):1856-1857. doi:10.1126/science.314.5807.1856
17. Aboumatar H, Wise RA. Notice of Retraction. Aboumatar et al. Effect of a Program Combining Transitional Care and Long-term Self-management Support on Outcomes of Hospitalized Patients With Chronic Obstructive Pulmonary Disease: A Randomized Clinical Trial. *JAMA* . 2018;320(22):2335-2343. *JAMA*. 2019;322(14):1417. doi:10.1001/jama.2019.11954
18. Herndon T, Ash M, Pollin R. Does high public debt consistently stifle economic growth? A critique of Reinhart and Rogoff. *Cambridge Journal of Economics*. 2014;38(2):257-279. doi:10.1093/cje/bet075
19. Miske O, Abatayo AL, Daley M, et al. Investigating the reproducibility of the social and behavioural sciences. *Nature*. 2026;652(8108):126-134. doi:10.1038/s41586-026-10203-5
20. Sharma NK, Ayyala R, Deshpande D, et al. Analytical code sharing practices in biomedical research. *PeerJ Computer Science*. 2024;10:e2066. doi:10.7717/peerj-cs.2066
21. Breznau N, Rinke EM, Wuttke A, et al. The reliability of replications: A study in computational reproductions. *R Soc Open Sci*. 2025;12(3):241038. doi:10.1098/rsos.241038
22. Stagge JH, Rosenberg DE, Abdallah AM, Akbar H, Attallah NA, James R. Assessing data availability and research reproducibility in hydrology and water resources. *Sci Data*. 2019;6(1):190030. doi:10.1038/sdata.2019.30
23. Stodden V, Seiler J, Ma Z. An empirical analysis of journal policy effectiveness for computational reproducibility. *Proc Natl Acad Sci USA*. 2018;115(11):2584-2589. doi:10.1073/pnas.1708290115
24. Schmied C, Nelson MS, Avilov S, et al. Community-developed checklists for publishing images and image analyses. *Nature Methods*. Published online September 14, 2023. doi:10.1038/s41592-023-01987-9
25. Nelson G, Boehm U, Bagley S, et al. QUAREP-LiMi: A community-driven initiative to establish guidelines for quality assessment and reproducibility for instruments and images in light microscopy. *Journal of Microscopy*. 2021;284(1):56-73. doi:10.1111/jmi.13041
26. Boehm U, Nelson G, Brown CM, et al. QUAREP-LiMi: A community endeavor to advance quality assessment and reproducibility in light microscopy. *Nat Methods*. 2021;18(12):1423-1426. doi:10.1038/s41592-021-01162-y
27. Light microscopy reporting for reproducibility. *Nat Cell Biol*. 2025;27(6):877-877. doi:10.1038/s41556-025-01704-y
28. Reporting light microscopy data in our pages. *Nat Struct Mol Biol*. 2025;32(6):955-955. doi:10.1038/s41594-025-01605-6
29. Crediting early-career researchers in peer review. *Nat Methods*. 2025;22(6):1121-1122. doi:10.1038/s41592-025-02738-8
30. Martinez-Ortiz C, Martinez Lavanchy P, Sesink L, et al. *Practical Guide to Software Management Plans*. Zenodo; 2023. doi:10.5281/ZENODO.7038280
31. Wilkinson SR, Aloqalaa M, Belhajjame K, et al. Applying the FAIR Principles to computational workflows. *Sci Data*. 2025;12(1):328. doi:10.1038/s41597-025-04451-9
32. Barker M, Chue Hong NP, Katz DS, et al. Introducing the FAIR Principles for research software. *Sci Data*. 2022;9(1):622. doi:10.1038/s41597-022-01710-x
33. Alves R, Bampalikis D, Castro LJ, et al. ELIXIR Software Management Plan for Life Sciences. doi:10.37044/osf.io/k8znb
34. Institute TSS. Checklist for a Software Management Plan. Published online December 10, 2018. doi:10.5281/ZENODO.2159713
35. Zhang Q, Dhane F. Alliance Software Management Plan (SMP) Template. Published online August 6, 2024. doi:10.5281/ZENODO.13242503
36. Escamilla E, Klein M, Cooper T, Rampin V, Weigle MC, Nelson ML. The Rise of GitHub in Scholarly Publications. doi:10.48550/ARXIV.2208.04895

Extra Slides



Automation

What tasks can you automate to increase consistency and reduce manual work?

- Computational Notebooks
- pipeline managers / Make-like systems
- automated environment management tool such as renv, poetry, conda, Nix, or GUIX
- git hosts also tend to have continuous integration and deployment (CI/CD) systems
  - Automatically do things when you ‘push’ changes to your code
  - For example build these slides and serve them on a website
  - Run tests, or if the analysis is small can run the whole thing
  - Great for checking environment and portability are properly described
- Linters and/or stylers to format code
  - git hooks

🤖 Automation - Checkbox Items

- [ ] Suitable automations are in place
  - [ ] 🥉Bronze *(easy)*: 1 from this list of processes are automated
    - use of an environment management tool
    - use of a literate programming / computational notebook
    - use of a pipeline manger or make-like tool
    - use of a linter / formatter
    - use of continuous integration / continuous deployment
    - use of git hooks
    - automated minting of new persistent identifiers on release tagging
    - ...
  - [ ] 🥈Silver *(easy)*: 2-3 from the above list of processes are automated
  - [ ] 🥇Gold *(intermediate)*: 4+ from the above list of processes are automated
  - [ ] 🏆Platinum *(hard)*: note that difficult is somewhat project dependent Your manuscript and its supplements are generated and served on a website after being built from your CI/CD pipeline. All statistics and data visualisations in your manuscript are generated programatically by your analysis pipeline from your raw data in CI/CD. Results are cached such that if you, for example, change the formatting of a graph only the plotting and rendering code needs to be re-run, but if you change the data the entire pipeline is re-rerun.

✅ Testing

How can you test your project so you can be confident it does what you think it does?

- For one off analyses automated unit testing frameworks for software packages can be:
  - a bit overkill, though helpful to know for quick tests of core logic
  - Not best suited to the sorts of ‘sanity’ tests it is good to do
- Dummy data
  - Simulate inputs which you expect to have certain outputs
- ‘fuzz’ testing
  - noisy random inputs, ‘graceful failure’ when inputs don’t make sense over potentially misleading outputs
- ‘unit’ testing
  - Check core logic is correct
- Particularly useful for pre-registered analyses where you plan and test the analysis prior to getting the primary dataset

✅ Testing - Checkbox Items

- [ ] Project has undergone suitable testing
  - [ ] 🥉Bronze *(easy)*: Includes a minimal test data set necessary to demonstrate the basic functionality of the analysis
  - [ ] 🥈Silver *(easy)*: Includes test datasets which cover a range of outcomes of the analysis
  - [ ] 🥇Gold *(intermediate)*: You are using unit tests and an automated testing framework to check the correctness of core steps of your analysis
  - [ ] 🏆Platinum *(hard)*: Have your analysis code written and tested on preliminary or simulated data in advance of recieving your principle dataset with a copy of your code from this time archived and referenced in a pre-registration or registered report.

🌱 Energy Efficiency

How can you and your users minimise wasted energy?

- less relevant for records of specific analyses than whole analyses which will be re-run many times
  - Consider profiling and refactoring any parts of your code which you will re-run many times
- Minimise unnecessary materials to archive
  - Favour Succinct efficient representations of environments and test data

🌱 Energy Efficiency - Checkbox Items

- [ ] Consideration has been given to the energy efficiency of the code
  - [ ] 🥉Bronze *(easy)*: minimise unnecessary output files
  - [ ] 🥈Silver *(easy)*: bronze plus: Profile your code and refactor inefficient parts
  - [ ] 🥇Gold *(intermediate)*: silver plus: Estimate and share the carbon footprint of your computations with a tools such as green algorithms calculator
  - [ ] 🏆Platinum *(intermediate)*: gold plus: Offload suitable computations to hardware accelerators where possible

Distribution

How can people install or access the software emerging from your project?

- Deposit the code publicly (a suitable license)
  - Use a suitable format for code distribution e.g. a git repo
    - No PDFs/Word Docs, plain text files are better than this but not as good as git
- Make it discoverable
  - Reference it in appropriate places like the metadata for any datasets you used, your protocols, your manuscript
  - keywords, author information, etc to make searchable
  - Archive in searchable repositories like Zenodo
- Use standard formats for any packaging of the software that you do
- Use an environment management tool to record and facilitate installing dependencies of the correct versions and distribut this alongside the code
- Share not just the code but the inputs, outputs, and environment in which it ran in a way others can easily copy modify and re-run

📦 Distribution - Checkbox Items

- [ ] Project is distributed in a suitable fashion
  - [ ] 🥉 Bronze *(Easy)*: Code and data (barring privacy related access restrictions) are in public repositories.
  - [ ] 🥈 Silver *(Intermediate)*: Detailed instructions on how to fetch, install and configure the tools and data needed re-run your analysis, and how to re-run the analysis in the described environment.
  - [ ] 🥇 Gold *(Intermediate)*: Project is in a reproducible interactive environment such as those offered by binder or renku.
  - [ ] 🏆Platinum *(Hard)*: Gold plus - Your project is built and served as a website using continuous integration and deployment tools such that your analysis is run on your data in a reproducible compute environment and computational results like graphs and statistics are programatically inserted into your output. (It is best to have some form of caching when doing this).