Reproducibility and Reuse
of your scientific code

The role of (basic!) software engineering in computational research

Dr Maeve Murphy Quinlan
Research Software Engineer | Research Computing Team | Research IT

M.MurphyQuinlan@leeds.ac.uk | earmmu@leeds.ac.uk

Research Computing Team and Service

- Here to support research(ers)
  - Provide training
  - Support users of Grid and Cloud Computing platforms
  - Provide consultancy
    - To develop project proposals
    - To help recruit people with specialist skills
    - Working directly on research projects
- For more information, please see our Website
- Contact us via the IT Service Desk

My background as an RSE

- A researcher in planetary science
  - Interdisciplinary research using astrophysical and geophysical models alongside laboratory analysis of billion-year-old meteorite samples to answer fundamental questions about the formation of the Solar System
  - PhD from the Institute of Geophysics and Tectonics, 2023
    - Development of novel numerical models to estimate the conductive cooling of planetesimals in the early Solar System
    - Pre-processing and analysis of large multi-dimensional microscopy datasets
    - Crystallographic analysis
  - Code: a fundamental tool in my research, on par with my microscopy lab access – but without the same robust peer review, benchmarking, validation in my field
- A frustrated code user → a novice software engineer

My role as an RSE

- Troubleshoot code for use on HPC systems such as ARC3/ARC4
- Teach introductory Python, R, version control (git), software development, data visualisation
- Migrate legacy codebases (spanning ~15 years) from SVN to git, maintaining attributions and developing automated testing and workflows
- Develop software packages with researchers for analysis on secure TREs
- Design, develop and maintain data visualisations and webapps with researchers
- Develop good workflows using package management and containerisation with researchers to aid their workflow

Before we begin

- A lot of information in this talk – the aim isn't for you to absorb it all, but to give you some jumping-off points for further reading
- By simply listening to this you are already doing more than is the norm across all of research computing
- These slides are available to download here so you have access to all the materials and links: murphyqm.github.io/slides/
- Some of this material might be very basic and obvious to you; some of it might be complex and overwhelming: researchers have very different backgrounds/competencies/comfort levels with research computing

Software sustainability

- The capacity of the software to *endure*
- *"Sustainability means that the software will continue to be available in the future, on new platforms, meeting new needs"* (DS Katz, 2016)
- *"Sustainable software is software which is:* (P Lago, 2016)
  - *Easy to evolve and maintain*
  - *Fulfils its intent over time*
  - *Survives uncertainty*
  - *Supports relevant concerns (Political, Economic, Social, Technical, Legal, Environmental)"*
- Perhaps a little far reaching for the snippet of code you wrote for scientific analysis…
- How do we make this more relevant to research code?

FAIR Principles

- The FAIR Guiding Principles for scientific data management and stewardship (Wilkinson et al., 2016)
  - Promoting good data stewardship practises
  - But can also be applied to scientific software
- **F**indable
- **A**ccessible
- **I**nteroperable
- **R**eusable

FAIR Principles

- The FAIR Guiding Principles for scientific data management and stewardship (Wilkinson et al., 2016)
  - Promoting good data stewardship practises
  - But can also be applied to scientific software
- **F**indable – use of a DOI, rich human *and* machine readable metadata, indexing in a searchable registry
- **A**ccessible – retrievable via their identifier (DOI) using a standardised protocol
- **I**nteroperable – use of a formal, accessible, widely used format
- **R**eusable – metadata provides sufficient detail to allow reuse, adhering to domain-relevant standards, detailed provenance

For sensitive, confidential, secure data, analysed in a trusted research environment or other secure server where data **cannot** be open, it is even **more important** to ensure *software* is as open, reproducible as possible

FAIR Principles for software

How do we apply these to software?

- **F**indable
- **A**ccessible
- **I**nteroperable
- **R**eusable

→ The 5 Guidelines for FAIR software: fair-software.nl

(See endorsing organisations here)

Guidelines for FAIR software

- Repository
- License
- Registry
- Citation
- Checklist

Guidelines for FAIR software

- Repository
- License
- Registry – register your code in a [searchable community](#) registry or index
- Citation
- Checklist – use a [software quality checklist](#) to validate your code

For another day!

Guidelines for FAIR software

- **Repository**
- **License**
- ~~Registry~~
- **Citation**
- ~~Checklist~~

Use a public **repository** with version control

- What is a repository with version control?
  - A folder with your scientific code
    - Everything used to analyse your data – Python scripts, R scripts, versions of libraries and languages used
  - With an automated track-changes function
    - A more robust version of file naming systems like… `draft_v01_edits_v2_2024-06-20_final_final2_FINALFINAL.py`
  - Shared, so that users/reviewers/researchers can find the exact version of the code you used to analyse data from start to end

Use a public **repository** with version control

- Why public?
  - Collaboration, reusability of code
  - Reproducibility of your results
  - Scrutiny of your code – peer review process, transparency
  - Good code practise – no fragments of identifying data, public shaming
- Why version control?
  - Track changes!
  - Back-ups of every version of the software, as it evolves
  - Saves on headaches with increasingly complex filenames
  - Track author contributions
  - Recommendation: git version control, with GitHub – but lots of options

- Read more here

Add a **license** to the repository

- Creative work including software is automatically protected by copyright, meaning the code you published in your repository cannot be legally used by anyone unless you grant explicit permission
- A license allows people to use your code, subject to certain requirements (**such as citation/attribution**) and limits your liability if something goes wrong
- It is very easy to add an OpenSource license to your GitHub repository, however you should research in detail whether your funding body/institute has specific requirements with regards licensing
- Read more here

Enable **citation** of the software

- Citation ensures you are recognised for your software development work
- Also important for scientific accountability and reproducibility – but can be more complex than citing a paper for users
- Make it easy for the users – create a citation file that shows exactly how to cite your work
  - Get a persistent identifier – a DOI – for a specific version of your code. GitHub bundles “releases” with Zenodo to allow you to generate a new version with an updated DOI
  - Create a file using the Citation File Format, which is both human and machine readable, allowing various referencing softwares to parse it and add it to their library
- Read more here

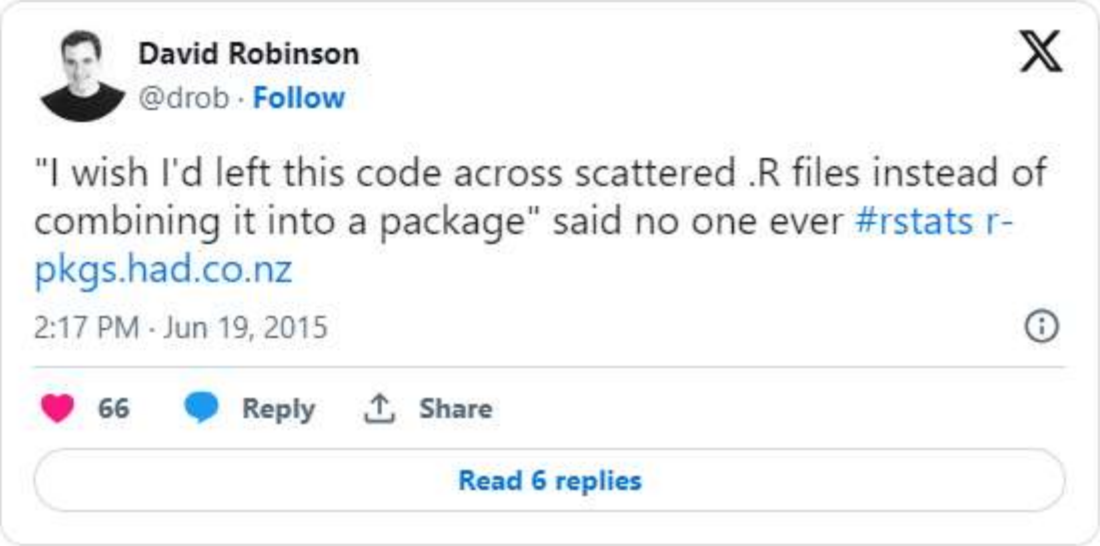
Anything worth doing,
is worth doing well

Anything worth doing,
is worth doing poorly at first

Worth implementing even for “bad” code

- Tracking your changes via a git repository is useful *even if* some of those changes are you hopping back and forth making silly mistakes – in fact, version control makes it a lot easier to undo mistakes
- Adding a license for reuse if useful *even if* you think your code is too niche and not re-useable – why add an additional barrier to entry. You never know when a desperate PhD student might need *that exact snippet of code* that you wrote
- Enabling citation of the software is useful *even if* it is a messy collection of scripts to analyse data – it allows you to correctly cite the exact version you did when you write up your results in a research article

Anything worth doing,
is worth doing well



David Robinson
@drob · **Follow**

"I wish I'd left this code across scattered .R files instead of combining it into a package" said no one ever #rstats r-pkgs.had.co.nz

2:17 PM · Jun 19, 2015

66 Reply Share

Read 6 replies

Maeve Murphy Quinlan | earmmu@leeds.ac.uk
murphyqm.github.io/slides | bit.ly/arc-help

You’ve poured so much time and effort in…

To write your code in the first place; shouldn’t you make it as reusable and robust as possible, so that you and other researchers in the future can save time by using little snippets of it?

Package your code for installation through a registry!

- Code organisation: puts in place a tried-and-tested organisation system so you can easily navigate your own code
- Consistent documentation: makes sure you properly note down what the code does, what data types it accepts, etc.
- Code distribution: makes it even easier for people to get up and running using your code

How to get started

- Using R:
  - “Making your first R package”: a tutorial/blog post to walk you through the basics
  - R Packages 2E: an open source (free!) textbook by Hadley Wickham and Jennifer Bryan
- Using Python:
  - Use cookiecutter – project templates to build your package
  - “Packaging Python Projects” – tutorial
  - Build Your Very First Python Package - tutorial

Other topics to keep in mind

Some links, jumping-off points, further reading:

- Test and benchmark your code – Medium article
  - Think of this in the same way you would lab analysis
- Pin library versions and record dependencies to ensure reproducibility
  - Managing dependencies for reproducible (scientific) software: blog post
  - Python
    - An unbiased evaluation of environment management and packaging tools: blog post
    - Python for Scientific Computing course notes: recording dependencies
  - R
    - Dependency Management in R: opensource textbook chapter
    - Managing R and Rstudio with conda: blog post
- If you're overwhelmed: reach out to an RSE! bit.ly/arc-help

Thank you for listening!

Anything worth doing, is worth doing poorly at first

Contact me if:

- Your code is broken, and you don't know why
- You want to submit work to ARC but don't know where to begin
- You want to meet to chat through a technology that might help your workflow, like
    - Containerisation
    - Parallel computing
    - HPC
    - Interactive data visualisation or webapp development
- You want to discuss costing an RSE in a grant application (I can pass you on to my manager to discuss specifics, or you can submit a request here: bit.ly/arc-help)
- You would like direction to useful documentation/resources/literature/courses to help develop your skills

Dr Maeve Murphy Quinlan | M.MurphyQuinlan@leeds.ac.uk | earmmu@leeds.ac.uk