Elements of Reproducibility

Handing off your project and hoping it still works

Andy Boughton

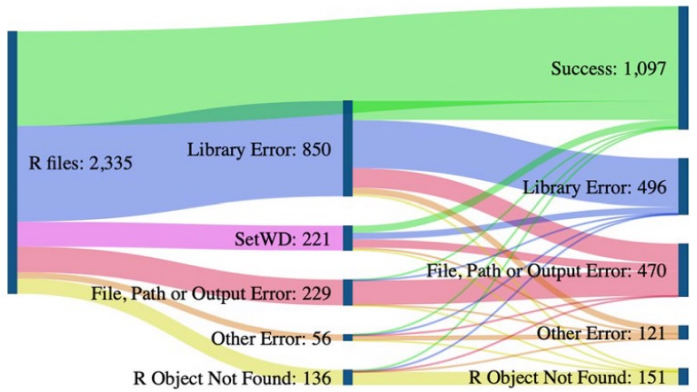
Center for Statistical Genetics

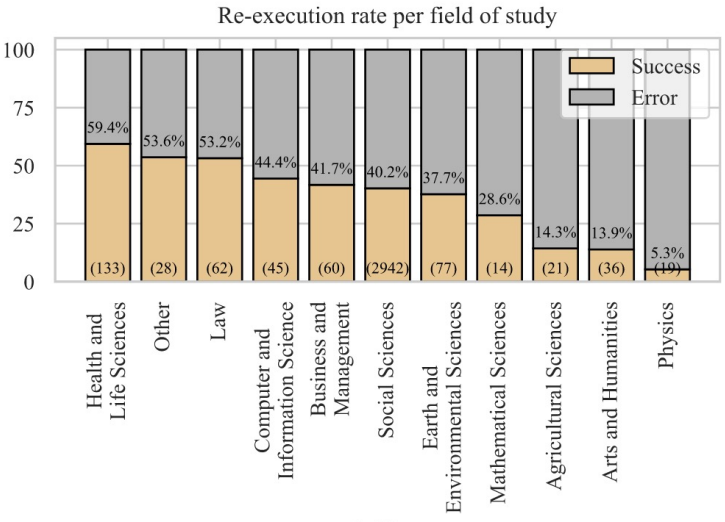
University Research: An overview

1. Students build software
2. They write a paper that makes a bunch of promises
3. Students graduate
4. People use software and have questions
5. Flail and run around

Key personnel (students) depart at a predictable rate, and take knowledge with them. What does this mean for community resources?

"Data Sharing and management snafu in 3 short acts" - https://www.youtube.com/watch?v=66oNv_DJuPc





"...more than 2000 replication datasets... execute[d] ... in a clean runtime environment... 74% of R files failed to complete without error in the initial execution, while 56% failed when code cleaning was applied...

...our automated study has a comparable success rate to the reported manual reproducibility...."

"...mandated data archiving policies that require the inclusion of a data availability statement in the manuscript improve the odds of finding the data online almost 1000-fold compared to having no policy... rates at journals with less stringent policies were only very slightly higher than those with no policy at all."

Trisovic, A. *et al*. A large-scale study on research code quality and execution. *Sci Data* **9**, 60 (2022). DOI: 10.1038/s41597-022-01143-6

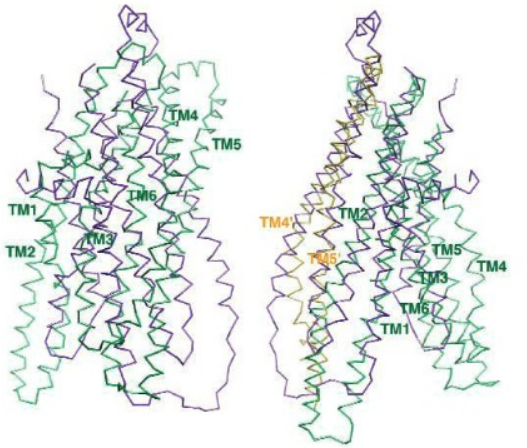
Vines, T. H. *et al*. Mandated data archiving greatly improves access to research data. *FASEB j*. **27**, 1304–1308 (2013). DOI: 10.1096/fj.12-218164

"...Over 95% of sites were running in the first 2 years, but this rate dropped to 84% in the third year and gradually sank afterwards ($P < 1e\text{-}16$). The mean half-life of Web services is 10.39 years. Working Web services were published in journals with higher impact factors..."

"Krystalli runs workshops called ReproHacks for researchers to submit their own published papers, code and data, and challenge participants to reproduce it. Often, she says, they cannot..."

Ősz, Á., Pongor, L. S., Szirmai, D. & Győrffy, B. A snapshot of 3649 Web-based services published between 1994 and 2017 shows a decrease in availability after 2 years. *Briefings in Bioinformatics* **20**, 1004–1010 (2019). DOI: 10.1093/bib/bbx159

Perkel, J. M. Challenge to scientists: does your ten-year-old code still run? *Nature* **584**, 656–658 (2020). DOI: 10.1038/d41586-020-02462-7



Flipping fiasco. The structures of MsbA (purple) and Sav1866 (green) overlap little (*left*) until MsbA is inverted (*right*).

"a homemade data-analysis program had flipped two columns of data... Chang and his colleagues retract[ed] three Science papers and report that two papers in other journals also contain erroneous structures.

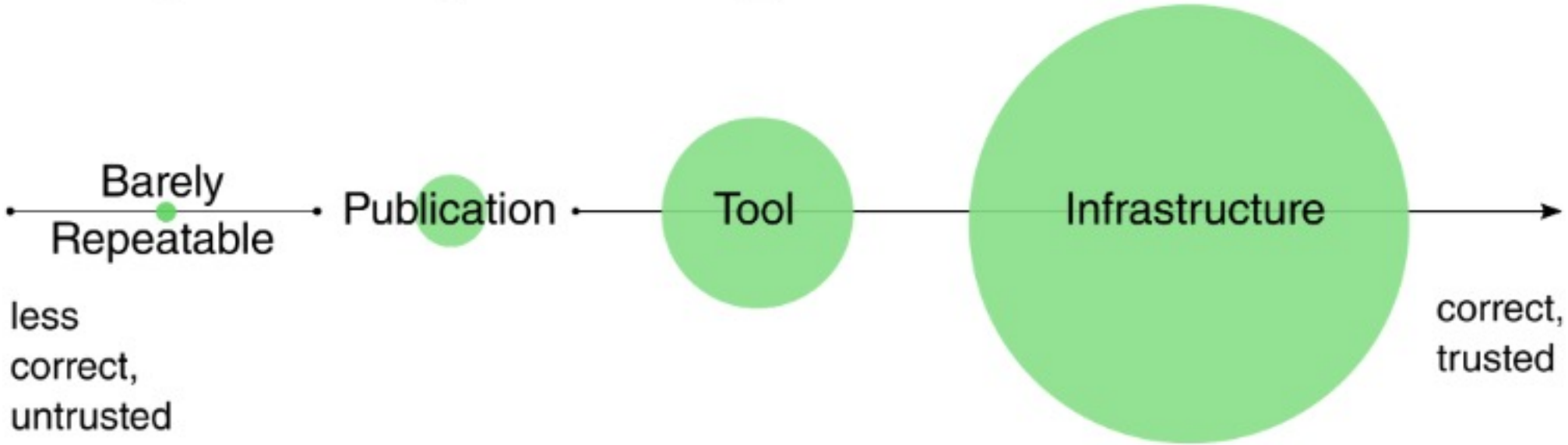
The ramifications of the software snafu extend beyond Chang's lab... [others] had a hard time persuading journals to accept their bio-chemical studies that contradicted Chang's MsbA structure. Those applications providing preliminary results that were not in agreement with the retracted papers were given a rough time..."

Miller, G. A Scientist's Nightmare: Software Problem Leads to Five Retractions. *Science* **314**, 1856–1857 (2006). DOI: 10.1126/science.314.5807.1856

Perkel, J. M. How to fix your scientific coding errors. *Nature* **602**, 172–173 (2022). DOI: 10.1038/d41586-022-00217-0

Is this going to take a lot of work?

Answer: It depends

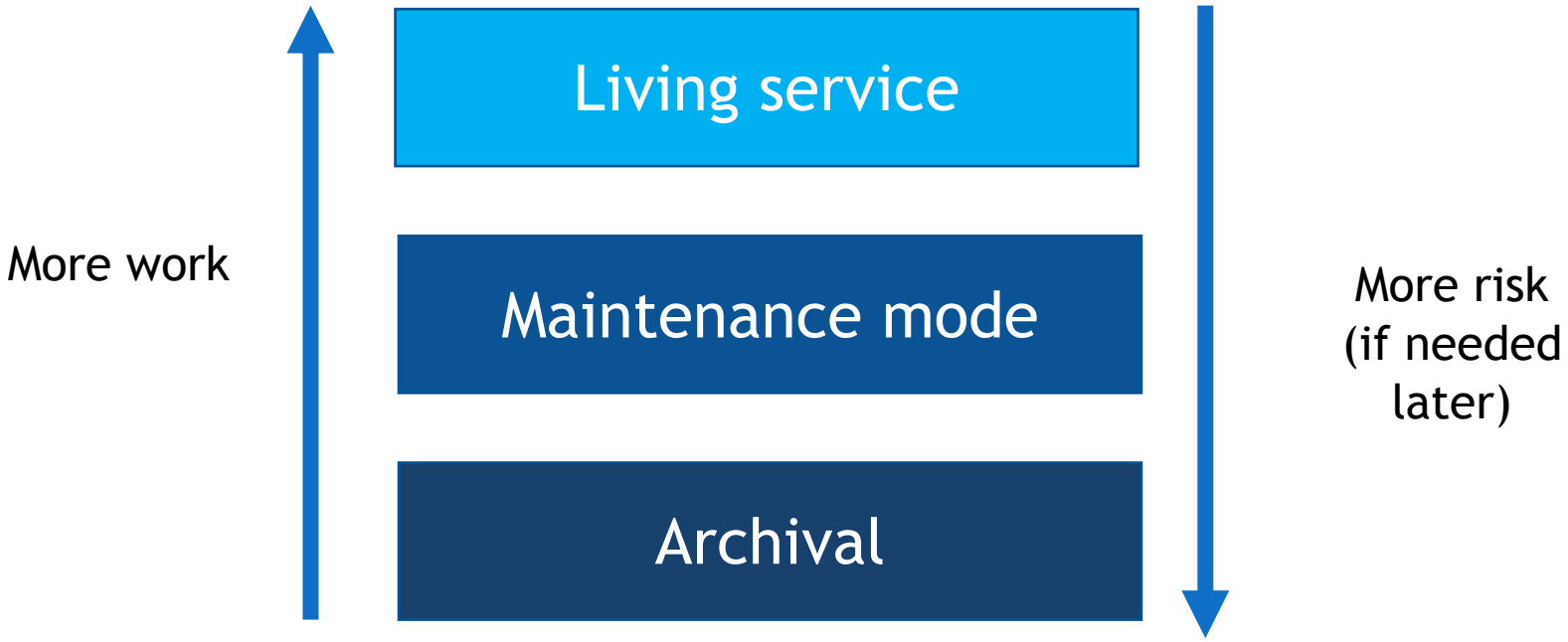


Circle size indicates research software impact

Harrison, S., Dasgupta, A., Waldman, S., Henderson, A. & Lovell, C. *How reproducible should research software be?* https://zenodo.org/record/4761866 (2021)

NASA Technology Readiness Levels: https://en.wikipedia.org/wiki/Technology_readiness_level

Levels of Preservation: What are your goals?



https://www.software.ac.uk/resources/approaches-software-sustainability

Side note: Data and code have different needs

- Software is more fragile to compile, but often smaller and easier to transfer than raw data
- Some funding agencies require researchers to file a plan for how they will archive project data. Each field has unique rules
  - Tools exist to help create plans

https://dmptool.org/
https://www.go-fair.org/fair-principles/

Wilkinson, M. D. *et al*. The FAIR Guiding Principles for scientific data management and stewardship. *Sci Data* **3**, 160018 (2016). 10.1038/sdata.2016.18

Archiving software

Baseline preservation for any project

A focus on what and how

- An archival project is not expected to receive new features or support; "at your own risk"
- Interested parties may have to rewrite the software to carry it forward
- Focus on scientific reproducibility: if an error is found, allow future readers to know what happened and why

- BUT: If you promise that your tool can be used by others, then there are additional requirements...

Table 2. Issues we encountered during code execution. Numbers in brackets show how often and in how many papers they occurred (overall occurrence/number of papers). In total, we encountered technical issues in 39 papers.

Minor (53, 25)	Substantial (73, 25)	Severe (41, 22)	Sys.-dependent (6, 5)
Library not found but available in repository (49, 24)	Wrong directory (34, 13)	Flawed functionality (13, 9)	Insufficient RAM (2, 2)
Faulty variable call (4, 3)	Deprecated function (10, 4)	Missing data or code (11, 9)	Function behaves differently across OSes (3, 3)
	Output not storable in local folder (10, 2)	Flawed data integration (11, 8)	Installing libraries on different OSes (1, 1)
	Function not found or missing library (9, 4)	Code in PDF (6, 6)	
	Library not found and not in repository (8, 7)		
	Broken link (2, 2)		

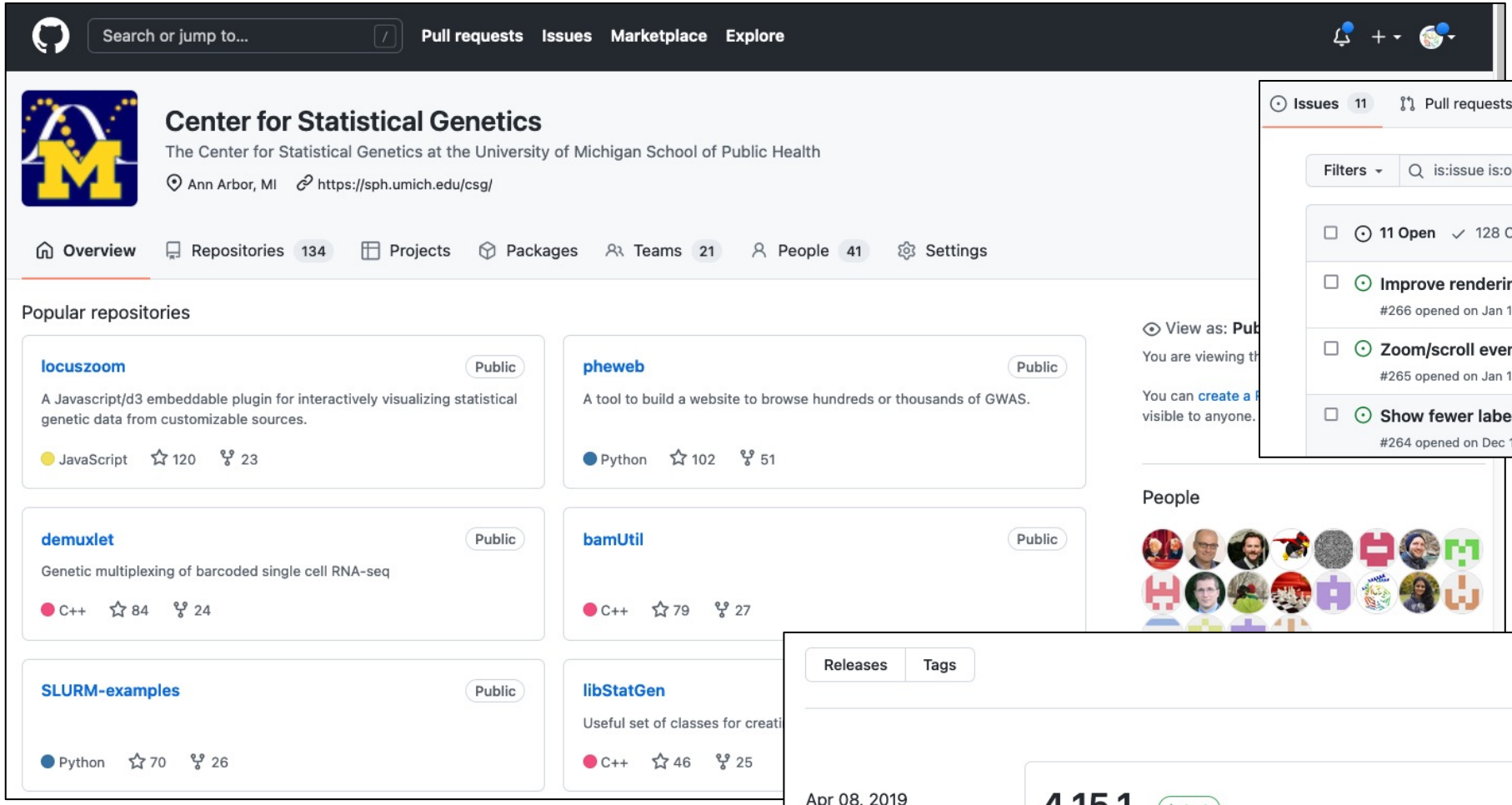
Konkol, M., Kray, C. & Pfeiffer, M. Computational reproducibility in geoscientific papers: Insights from a series of studies with geoscientists and a reproduction study. *International Journal of Geographical Information Science* **33**, 408–429 (2019). 10.1080/13658816.2018.1508687

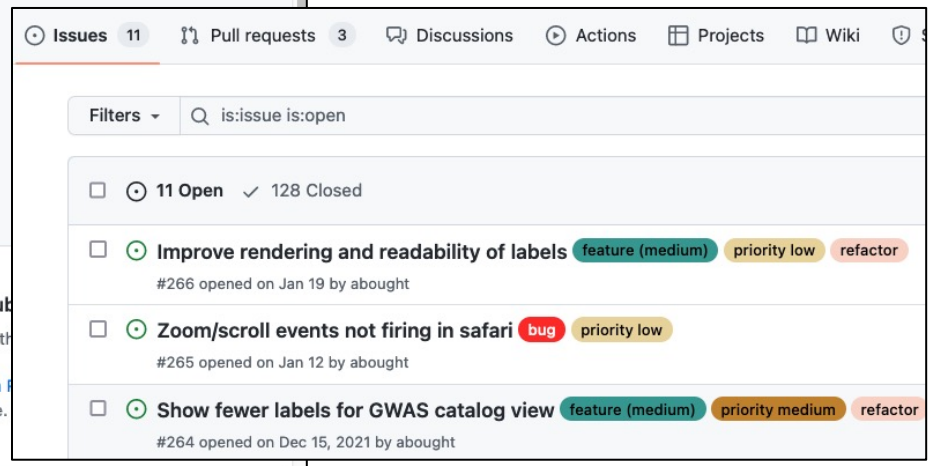
Make it findable

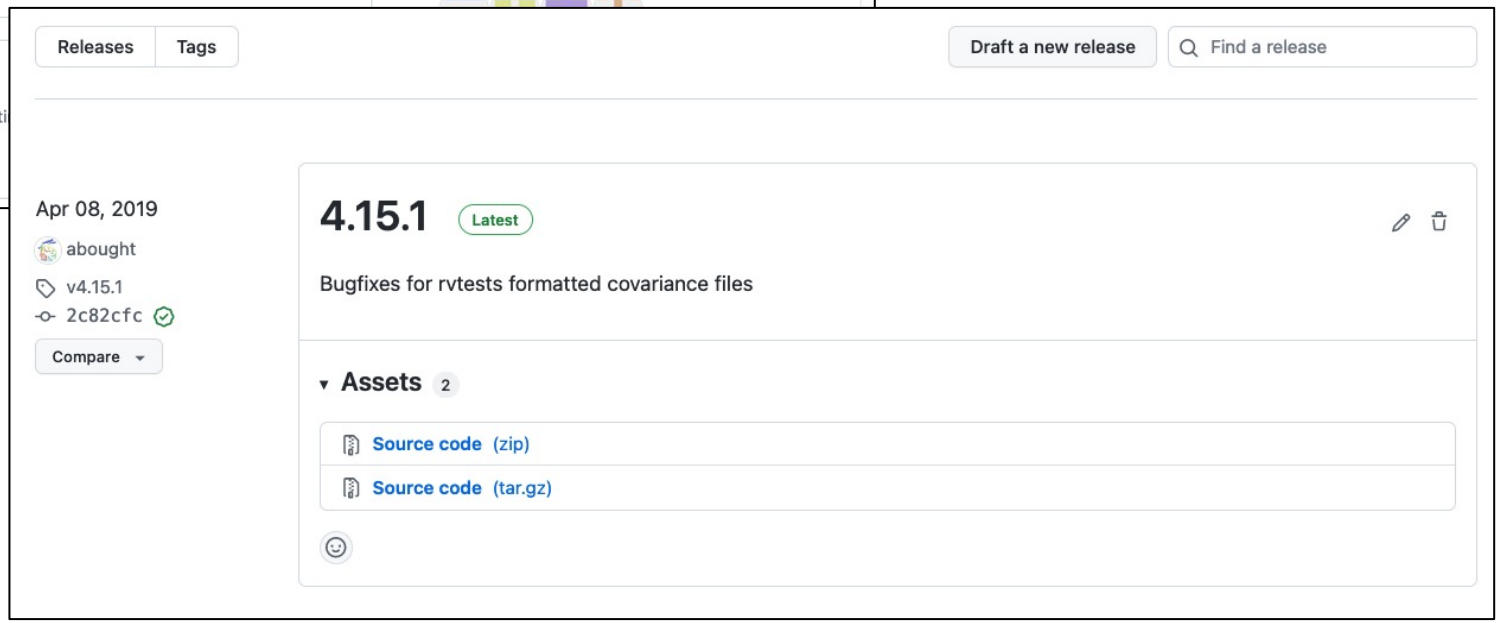
- GitHub is a popular place to store and share “living” code
- Private repos exist- you can start using version control before you are ready to publish
- Useful tips:
  - Before you publish, transfer to our organization so that people can release bugfixes to the “official” URL in the paper [1]
  - Tag your releases so that people can see exact version used for a paper [2]
- If you are publishing code or data as a separate item, Zenodo handles archiving and gives a citable DOI. Can create from GitHub repo!

[1] https://github.com/statgen
[2] git tag v1.0 -a && git push --tags

https://docs.github.com/en/repositories/archiving-a-github-repository/referencing-and-citing-content







https://github.com/statgen/

Help people to use your archive

- Contains exact code, not just a prose description of methods
- Clean up dead code, broken methods, or "test" files
- Tag releases if your program changes over time

Good archives start with good development habits. Don't wait until the end!

Taschuk, M. & Wilson, G. Ten simple rules for making research software more robust. *PLoS Comput Biol* **13**, e1005412 (2017). DOI: 10.1371/journal.pcbi.1005412

Capture the environment

- Much of the behavior of code is driven by external factors
  - Avoid hard-coded paths like "/User/broken_for_you/Documents"
  - Specify exact required dependencies using a dependency manager
- Data matters!
  - Include a small test dataset to run the program
  - Provide example configuration or parameter files with critical options
- Consider containers (Docker, Singularity, etc) to distribute dependencies along with the program

Be able to follow your own setup instructions on a new computer



Package managers: What to look for

```
[tool.poetry]
name = "impunity"
version = "0.1.0"
description = ""
authors = ["Andy Boughton <abought@umich.edu>"]

[tool.poetry.dependencies]
python = "^3.9"
pysam = "^0.19.0"

[tool.poetry.dev-dependencies]
pytest = "^5.2"
```

Compare to old-style requirements.txt:

numpy ?

flask ?

matplotlib ?

- Metadata: who and what
- Dependencies: What is required to run?
- Dev dependencies: extra stuff if you're modifying the code
- Track versions used- packages change over time
  - Semver: allow your package to use slightly newer bugfix versions if they promise to be compatible (^, ~, *)
  - Lockfiles: track exactly what got installed, regardless of future updates

https://python-poetry.org/docs/dependency-specification/

Automation tells you what is true*

- As you develop, automate your routine validation steps via test frameworks + continuous integration (eg pytest + GitHub actions)
- Instead of a markdown file specifying commands that are out of date, use bash scripts or workflow tools (makefile / snakemake)
- For REST APIs, use an API framework that auto-generates documentation (eg OpenAPI format). (for CLIs, use *argparse*)
  - Instructions maintained by hand quickly become lies

GoogleTest


GitHub Actions
R Markdown
from RStudio

Example GitHub actions file for Python tests: https://github.com/abought/zorp/blob/develop/.github/workflows/pythonpackage.yml
Standard CLI tooling that auto-generates documentation: https://docs.python.org/3/library/argparse.html

Workflow automation tools

nextflow

Imperative

- An exact list of commands to be run in order on a specific input
- Markdown / text file
  - Hard to keep up to date; examples quickly stop working
  - In a process with many steps, you'll probably forget one
- Bash script
  - Runs the commands exactly as written, so it stays more up to date
  - ... but unintuitive language rules can lead to *bad things*

Declarative

- Tell it the result you want ("out.txt"), and it decides what needs to be done
- Efficient: Only re-run parts of the process where something has changed
- Good when the amount of work to do is variable ("all files in folder")
- Examples: Make, snakemake, nextflow, workflowr
- It takes time to learn a new language- start small and start early.
  - Back-automating a project at the end is much harder.

Pitfalls of bad automation

> According to a bug report filed on GitHub, moving Steam's per-user folder to another location in the filesystem, and then attempting to launch the client may perform the following heart-stopping command:
>
> ```
> rm -rf /*
> ```
>
> "

The code in question is this in steam.sh:

```
# figure out the absolute path to the script being run a bit
# non-obvious, the ${0%/*} pulls the path out of $0, cd's into the
# specified directory, then uses $PWD to figure out where that
# directory lives - and all this in a subshell, so we don't affect
# $PWD
STEAMROOT="$(cd "${0%/*}" && echo $PWD)"

# Scary!
rm -rf "$STEAMROOT/"*
```

Yes, $STEAMROOT can end up being empty, but no check is made for that. Notice the `# Scary!` line, an indication the programmer knew there was the potential for catastrophe.

If your project has already committed to using Bash scripts...

- Put this at the start of every .sh file: https://explainshell.com/explain?cmd=set+-eux
- Sanity check scripts for common mistakes: https://github.com/koalaman/shellcheck

https://www.theregister.com/2015/01/17/scary_code_of_the_week_steam_cleans_linux_pcs/

Your archive should tell a story

- Use example commands to provide an obvious starting point
- Add value and remove distractions
  - Clean up dead code or unused files
  - Add example data (and file format docs)
- Use source code comments to explain tricky bits
  - Add citations for unusual methods, or to explain how parameters were chosen

https://stackoverflow.blog/2021/12/23/best-practices-for-writing-code-comments/
https://selftaughtcoders.com/how-to-quickly-and-effectively-read-other-peoples-code/

https://jupyter.org
https://rmarkdown.rstudio.com
https://en.wikipedia.org/wiki/Literate_programming

```
float Q_rsqrt( float number )
{
    long i;
    float x2, y;
    const float threehalfs = 1.5F;

    x2 = number * 0.5F;
    y  = number;
    i  = * ( long * ) &y;                       // evil floating point bit level hacking
    i  = 0x5f3759df - ( i >> 1 );               // what the !@#$ ?
    y  = * ( float * ) &i;
    y  = y * ( threehalfs - ( x2 * y * y ) );   // 1st iteration
//  y  = y * ( threehalfs - ( x2 * y * y ) );   // 2nd iteration, this can be removed

    return y;
}
```

https://en.wikipedia.org/wiki/Fast_inverse_square_root#Overview_of_the_code

File formats: Open to interpretation

1.1 An example

```
##fileformat=VCFv4.3
##fileDate=20090805
##source=myImputationProgramV3.1
##reference=file:///seq/references/1000GenomesPilot-NCBI36.fasta
##contig=<ID=20,length=62435964,assembly=B36,md5=f126cdf8a6e0c7f379d618ff66beb2da,
##phasing=partial
```

```
<contents of basic2.ped>
1   1   0  0  1   1       x    3 3    x x
1   2   0  0  2   1       x    4 4    x x
1   3   0  0  1   1       x    1 2    x x
1   4   1  2  2   1       x    4 3    x x
1   5   3  4  2   2   1.234    1 3    2 2
1   6   3  4  1   2   4.321    2 4    2 2
<end of basic2.ped>
```

1.4.2 Information field format

INFO fields are described as follows (first four keys are requ

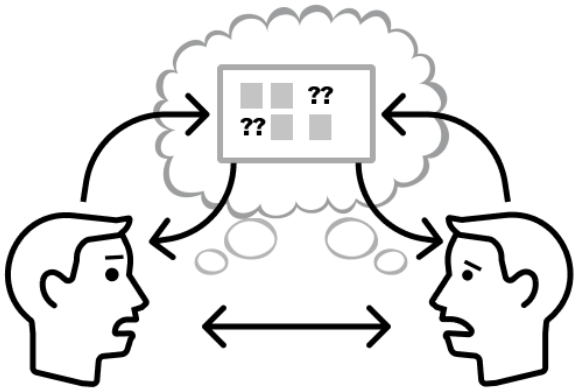
```
##INFO=<ID=ID,Number=number,Type=type,Description="description",Source="source",Version="version">
```

Possible Types for INFO fields are: Integer, Float, Flag, Character, and String. The Number entry is an Integer that describes the number of values that can be included with the INFO field. For example, if the INFO field contains a single number, then this value must be 1; if the INFO field describes a pair of numbers, then this value must be 2 and so on. There are also certain special characters used to define special cases:

https://samtools.github.io/hts-specs/VCFv4.3.pdf
http://csg.sph.umich.edu/abecasis/merlin/tour/input_files.html

Externalizing ownership

- All resources should be owned by the project, not one person
  - Mailing lists, Google Drive shared folders, domain names, etc
  - Key notes (including handoff documents) **should not** live in email, and test datasets should not live only in your user directory on a private project node
  - Inventory the services you depend on, and give someone access
- Handoff documents are not written for the author. Have a friend try to use the instructions without your help*



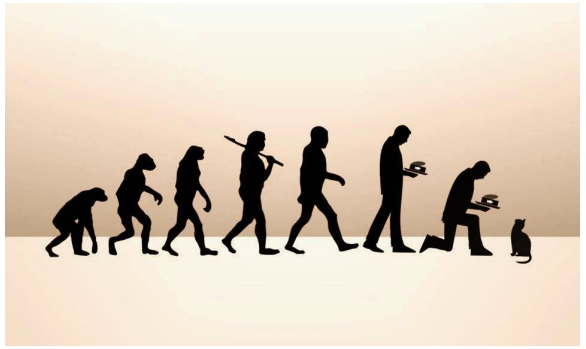
* Then take your friend to lunch and apologize

Living projects

Sustaining momentum after you're gone

Gardner, P. P. *et al. Sustained software development, not number of citations or journal choice, is indicative of accurate bioinformatic software.* http://biorxiv.org/lookup/doi/10.1101/092205 (2016) doi:10.1101/092205.

Recognize that needs may change



- In a living project, no decision is final. Both maintenance and domain requirements are equally important
  - Do everything from the previous lists
  - Disaster recovery plans are no longer optional
- Plan to revisit methods and implementation based on scale change
  - Without good requirements capture and process automation, the team will be crushed by maintenance of old stuff
- Make sure that people with different expertise can work together

Cechova, M. Ten simple rules for biologists initiating a collaboration with computer scientists. *PLoS Comput Biol* **16**, e1008281 (2020). https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1008281

Anna "Legacy Archaeologist" Filina @afilina

Replying to @afilina

Today I did a particularly in-depth archaeology exercise, trying to reconstruct a business rule from multiple contradicting sources, with small pieces scattered all over the place.

Barriers to collaboration

- Brittle systems
  - Stuff that never actually worked ("did I create this bug?")
  - Inaccurate documentation ("none of the sample commands work")
- No shared picture of reality
  - Lack of version control
  - Servers that are routinely edited by hand (test environments != production)
- Secret or overly specialized knowledge
  - Bug reports and renewal notices go to former employee
  - Configuration-driven workflow systems whose behavior is only defined at deployment time, and cannot be inferred from code
  - No standard languages, tools, or habits

https://medium.com/capital-one-tech/lessons-from-the-leadership-trenches-how-do-i-choose-the-right-programming-language-for-my-team-36222aa78b38

Example: Languages I've worked with at CSG

Lack of a common team baseline is a hidden tax on productivity

Choose technologies that can be supported by your team after you leave

Learning is great, but no one wants to maintain your custom malbolge interpreter

- Python (2 and 3)
  - Flask, Django/ DRF, FastAPI, pandas, jupyter, numpy, matplotlib
- PHP (v5 and v7) (Frameworkless and partial adoption of Laravel)
- JavaScript (ES5, ES2015+, node.js)
  - D3 (v3 and v5), JQuery, Vue.js, Backbone.js
- Presentation layer: HTML, SVG, bootstrap 3-4, material UI, tailwind CSS
- R
- Go
- Java
- C++
- Build/ dependency /automation/ deployment systems
  - Make, snakemake, cmake, bash scripts, Docker, some ansible
  - Webpack, babel, gulp, parcel, rollup, browserify, vue cli
  - pip, pipenv, poetry, conda, yarn, npm, go, cargo, cget, composer
- Infrastructure environments
  - Google Cloud, AWS, on prem
- Databases / Queryable storage
  - MySQL, PostGres, Sqlite, Mongo, Tabix, Parquet

Sample archival checklist: CLI Tools

- ❑ Code is available in a stable "source of record" (eg public version control)
  - ❑ Ensure that at least one other person can release "official" bugfixes
  - ❑ Tidy up: ADD comments / citations, and REMOVE unused code and files
- ❑ Instructions can be followed without additional help
  - ❑ Code compiles and runs on a "clean" environment (eg docker or CI server)
- ❑ Identify how to validate the software
  - ❑ Automate with small example test set if possible
  - ❑ Given same code and input files, can someone reproduce the figures in your paper?
  - ❑ Tag the version of code used to generate a paper
- ❑ Externalize ownership
  - ❑ Have someone read and try the instructions
  - ❑ Provide an inventory of key resources and make sure someone else has access
  - ❑ Transfer ownership of mailing lists, shared documents, monitoring services, etc

Web Apps: sample inventory of key resources

- Where to download key resource files
  - If access is required, identify point of contact (people are resources too!)
  - Provide a citation in case the FTP site is reorganized
- Things not stored in version control
  - Configuration files, application secrets
  - External dependencies (Box/ Google Drive folders, Facebook/ Google Login, etc)
- How to find AND obtain access to key services- give PI access as fallback
  - List of servers and databases + access credentials
  - Third party services (analytics, mailing lists, issue tracker, domain registrations)
  - Monitoring systems (Nagios, Sentry) and billing reminders
  - Grant PERMISSIONS, not just ACCESS. A locked console is not useful.
- Process
  - Data dictionary for config/resource files: "the entire script depends on column 11, what's that?"
  - Secret knowledge: "Things I check manually but never found time to automate"
  - Tips to the next person: "Stuff I think is worth doing in the future"
  - Critical files that you must be able to find fast (eg IRB documents)
- Operations and disaster recovery
  - Standard operating procedures / automated scripts / service-specific repositories (Terraform, ansible, etc)
  - Call out key items that are time sensitive and can't be ignored (billing alerts, IRB paperwork, domain renewals, etc)
  - Backups

Further reading

- (*) **Ten simple rules for making research software more robust -** https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1005412
- (*) **Good enough practices in scientific computing** https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1005510
- (*) **Ten simple rules for biologists initiating a collaboration with computer scientists** https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1008281
- (*) **Jeffrey Perkel – articles on scientific tools (*Nature* columnist):** https://jeffreyperkel.com/publications/publications-nature/
- **JOSS review checklist:** https://joss.readthedocs.io/en/latest/review_checklist.html
- **Best practices for writing code comments:** https://stackoverflow.blog/2021/12/23/best-practices-for-writing-code-comments/
- **Data dictionary**: https://library.ucmerced.edu/data-dictionaries

Hands-on opportunities

- UMICH Software Carpentry - https://umcarpentries.org/
- MIDAS events calendar: https://midas.umich.edu/calendar/
- BIOSTAT R Workshops - https://sph.umich.edu/biostat/computing/r-programming-final.html
- ...many others!

Example “archival” projects

- RP:CB data, methods, and analysis together - https://osf.io/nbryi/
- METAL: Code, unit tests, tagged versions - https://github.com/statgen/METAL
- Schloss lab manuscripts: Rmarkdown and snakemake workflows to generate the entire manuscript start to finish
  - http://www.schlosslab.org/papers/
  - Example: https://github.com/SchlossLab/Sovacool_OptiFit_mSphere_2022
- JOSS archives: Peer-reviewed software repos - https://joss.theoj.org/

People are really doing this, and you can too

Places to archive things

- Version Control: https://github.com/statgen/
  - Also provides public issue tracker for reporting bugs!
- Citable DOIs for research articles: https://help.zenodo.org/features/
  - Or other domain specific hosting services (GWAS catalog, Figshare, Dataverse, OSF, etc)
- Group wiki
  - https://genome.sph.umich.edu/wiki/
  - https://statgen.sph.umich.edu/wiki/
- UMich services:
  - https://its.umich.edu/accounts-access/leaving
  - Shared accounts: https://documentation.its.umich.edu/node/501
  - MCommunity Groups: https://documentation.its.umich.edu/node/370

Summary

- Archives should be self-contained
  - Don’t wait until the end for cleanup! Use good practices throughout the development process
- Keep a record of key services and how to access them
- Shared standards help people with different skills work together
- Expect people to graduate, and plan ahead

Extra material

Some challenges in maintenance mode

Maintenance mode

Minor bugfixes and keeping the lights on

Know how to find things

- Check your service inventory
  - Know how to find critical code, data, and infrastructure
  - Be able to interpret intent ("what is column 11 in this file? It seems important")
- Reference key points of contact
  - If key resources are restricted, who can give you access?
  - Are there special requirements (like IRB)?
- Write stuff down while you still remember it!

Things will break when left alone

- Command line tools
  - Dependencies may change; pin versions you depend on
  - Data standards may change, or FTP sites may vanish- capture sample input
- Web servers
  - Security updates must be installed, which could change how things work
  - Third party services may change or be turned off
  - Have a clear plan to validate new releases/ security updates
- Automate as much as possible

"Data Sharing and Management Snafu in 3 short acts" - https://www.youtube.com/watch?v=66oNv_DJuPc

Know when to pay attention


- Set clear priorities based on usage ("paper actively under review" vs "people only visit for nostalgia")
- Automated alerts tell you when something needs to be done
  - Some CSG machines have Nagios email alerts (examples: memory, disk usage, expired SSL certs, network issues, whether key processes are running)
  - Do not rely on users to report downtime. They rarely report bugs.
- Mailing lists and public bug trackers let users reach "the project" regardless of who is currently responsible

Nagios® SENTRY

Worst case scenario / Disaster Recovery

- If the server breaks, is the project dead?
  - For a web site, make this decision early on, and communicate when it changes
- Can a new environment be created automatically?
- Can you find and generate critical data files from scratch?
- Backups should be validated, frozen, and stored off site before you go into maintenance mode

“Hibernation” vs “Planned end date”

- Not every project is a good candidate to be maintained indefinitely
- Sensitive data must be protected
  - Unpatched machines could be exploited, and you would never know
- Routine operations may have unforgiving deadlines
  - Missed domain renewal → anyone could impersonate a major research study
  - Missed IRB renewal → (...you don’t want to know)
- Do not leave robots unattended
  - “Bring your own data” cloud services are a blank check against your bank account
  - Expect people to abuse your service eventually

https://www.wired.com/story/parler-hack-data-public-posts-images-video/
https://www.csoonline.com/article/3444488/equifax-data-breach-faq-what-happened-who-was-affected-what-was-the-impact.html