Project Hand-offs: A Guide

Andy Boughton

Center for Statistical Genetics

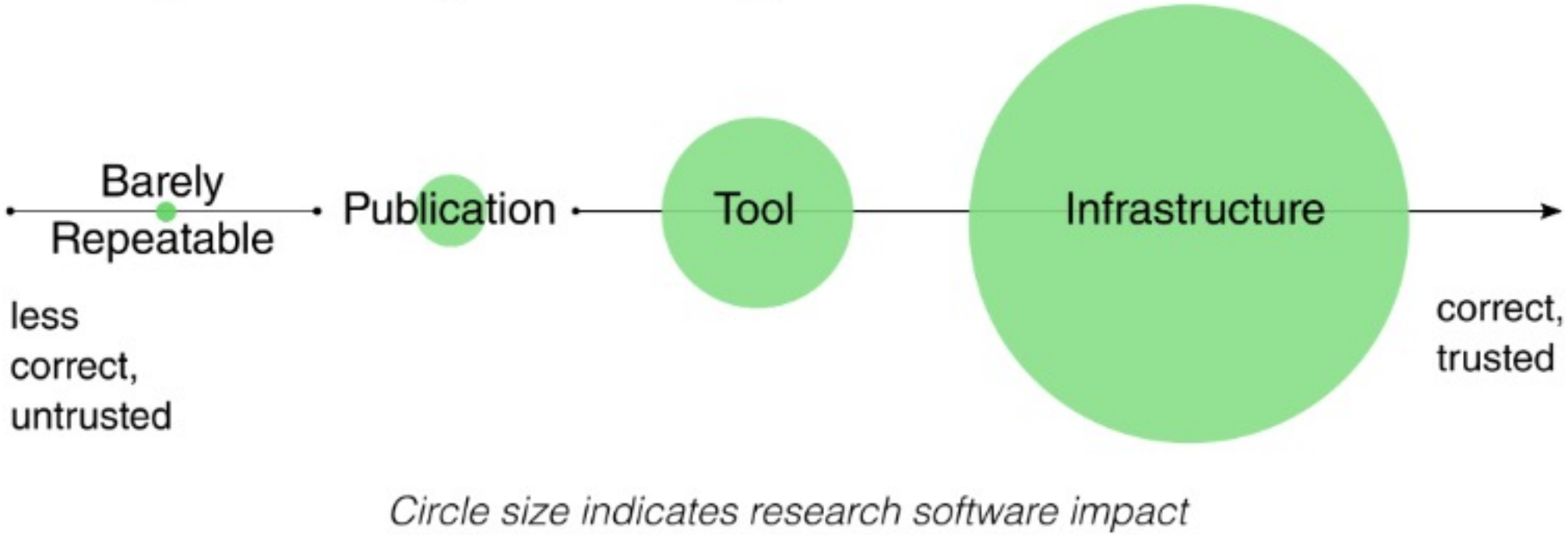
April 22, 2022

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?

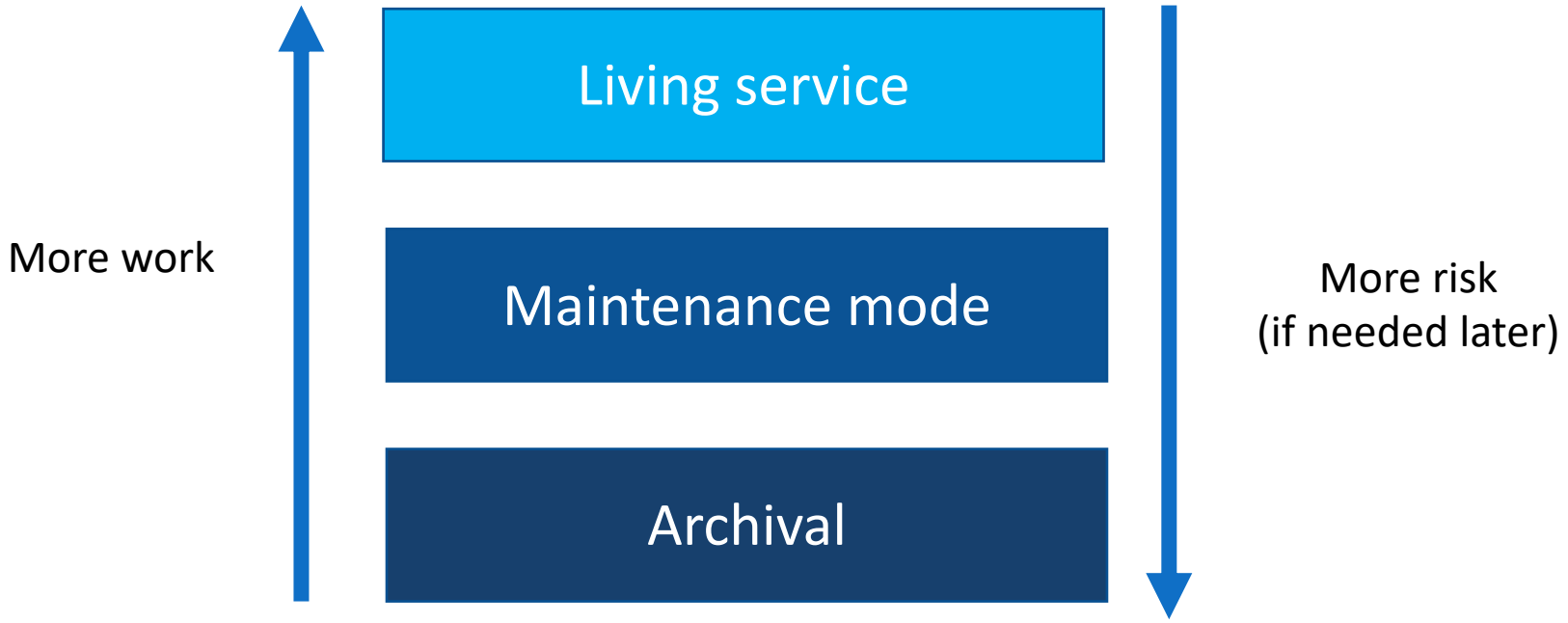
Our theme song: https://www.youtube.com/watch?v=EVZh4WcdC3s



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

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...

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).
https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1005412

"...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."

https://www.nature.com/articles/s41597-022-01143-6
https://faseb.onlinelibrary.wiley.com/doi/full/10.1096/fj.12-218164

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



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

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

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

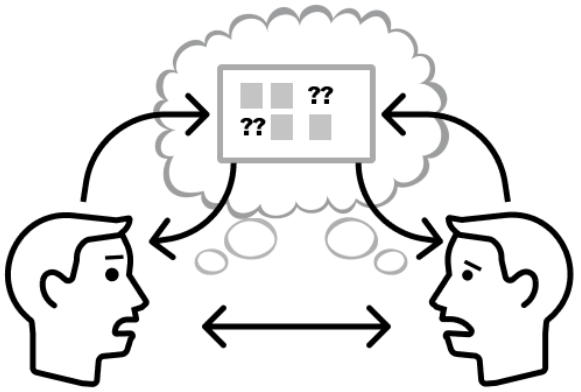
```
##INFO=<ID=ID,Number=number,Type=type,Description="description",Source="source",Version="version">
```

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

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

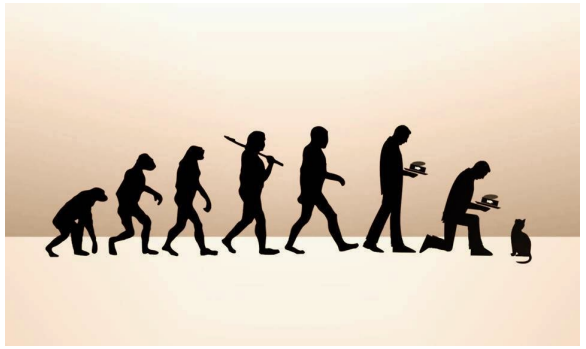
https://www.wired.com/story/parler-hack-data-public-posts-images-video/
https://www.csoonline.com/article/3444488/equifax-data-breach-faq-what-happened-who-was-affected-what-was-the-impact.html

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

GitHub organization: https://github.com/statgen/

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
- **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
- **Runbooks and Playbooks:** https://www.pagerduty.com/resources/learn/what-is-a-runbook/

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)
- KeePassXC encrypted password manager: https://keepassxc.org/
- "Requester pays" option for big datasets (cloud storage)
- Group wiki
  - https://genome.sph.umich.edu/wiki/Main_Page
  - 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