

# Principles of Research Software Sustainability

Daniel S. Katz

([d.katz@ieee.org](mailto:d.katz@ieee.org), <http://danielskatz.org>,  
[@danielskatz](https://twitter.com/danielskatz))

Chief Scientist, NCSA

Research Associate Professor, CS, ECE, iSchool

Rajiv Ramnath

([ramnath.6@osu.edu](mailto:ramnath.6@osu.edu))

Professor of Practice, Computer Science and  
Engineering



NCSA | National Center for  
Supercomputing Applications



**THE OHIO STATE UNIVERSITY**

[10.6084/m9.figshare.14138036](https://doi.org/10.6084/m9.figshare.14138036)

# Why do we care about research software?

- Funding

- ~20% of NSF projects over 11 years topically discuss software in their abstracts (\$10b)

Collected from <http://www.dia2.org> in 2017

- 2 of 3 main DOE ECP areas are research software (~\$4b)

- Publications

- Software intensive projects are a majority of current publications
- Most-cited papers are methods and software

Nangia and Katz; [10.1109/eScience.2017.78](https://doi.org/10.1109/eScience.2017.78)  
“Top 100-cited papers of all time,” Nature, 2014  
[10.1038/514550a](https://doi.org/10.1038/514550a)

- Researchers

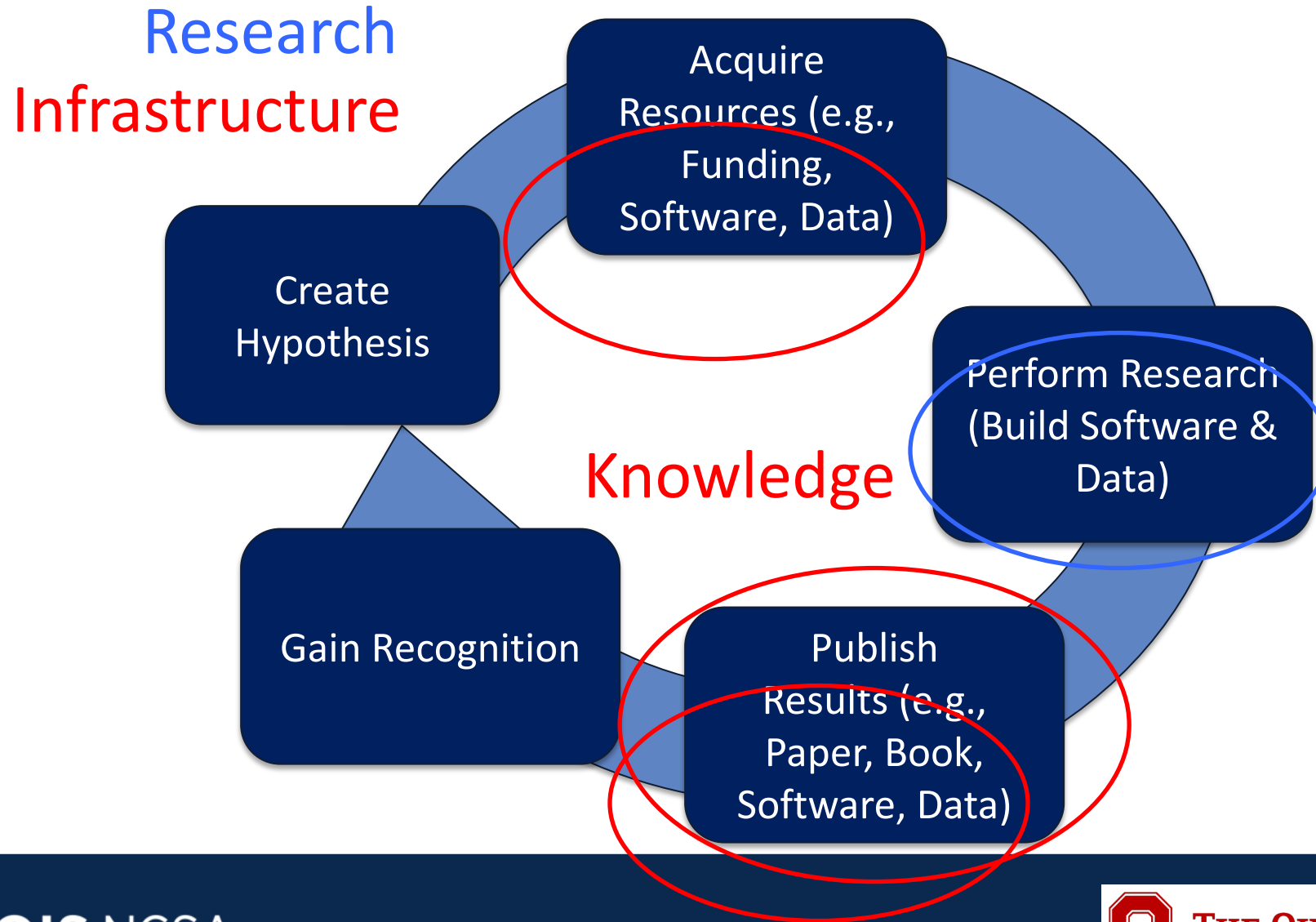
- >90% of US/UK researchers use research software
- ~65% would not be able to do their research without it
- ~50% develop software as part of their research

S. Hettrick; <https://www.software.ac.uk/blog/2016-09-12-its-impossible-conduct-research-without-software-say-7-out-10-uk-researchers>

S.J. Hettrick, et al.; [10.5281/zenodo.14809](https://doi.org/10.5281/zenodo.14809)

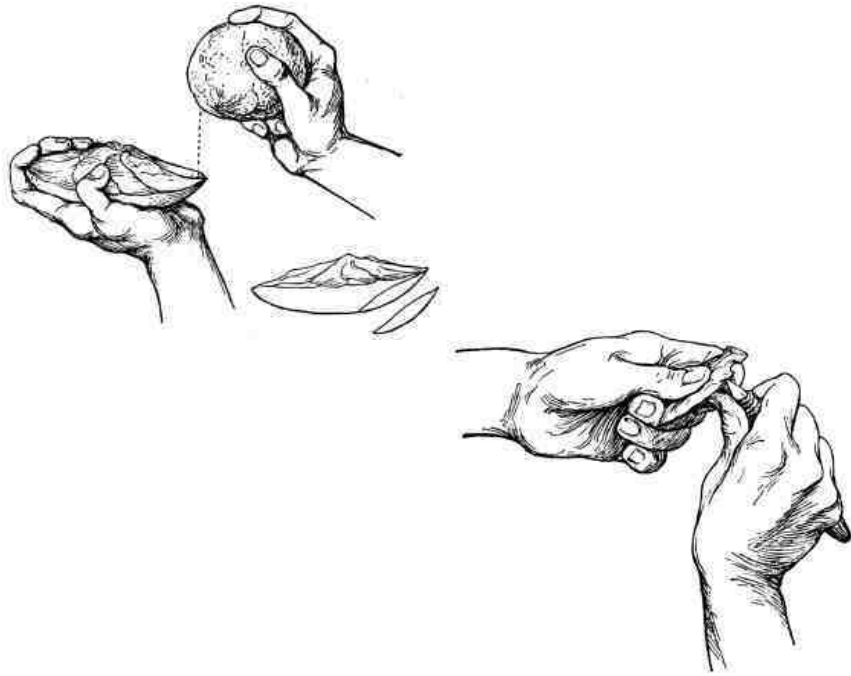
U. Nangia and D. S. Katz; [10.6084/m9.figshare.5328442.v1](https://doi.org/10.6084/m9.figshare.5328442.v1)

# Software in research cycle



# Who starts new infrastructure software projects?

- Tool makers
  - To make something useful to others



- Then options:
- Accept contributions? and if so:
  - a. Broaden focus?
    - Bring together other (related) packages
  - b. Broaden governance?
    - Collaborate with other developers

# Who starts new research software projects?

- User/Developer
  - To scratch their own itch



- Then options:
  1. Keep it private
  2. Share it
  3. Accept contributions?  
and if so:
    - a. Broaden focus?
      - Bring together other (related) packages
    - b. Broaden governance?
      - Collaborate with other developers

# Project stages

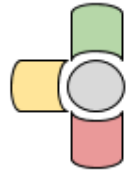
## Schematic stages of open community for research software





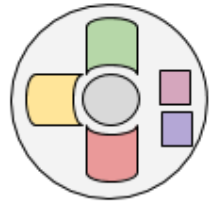
**Stage 0.** Some code and a user of it. No sustained team.



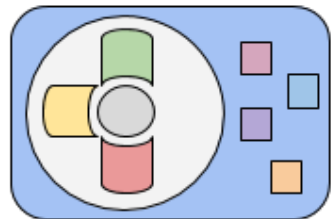
**Stage 1.** Software development team, internal use.  



**Stage 2.** Multiple software teams (different institutions) on same code (team is *community*), for internal use.  



**Stage 3.** Self-governing developer community deliberately supporting broad user community.





**Stage 4.** Self-sustaining organization dedicated to supporting user and dev community (e.g. through commercial support, events, software foundation, etc.).

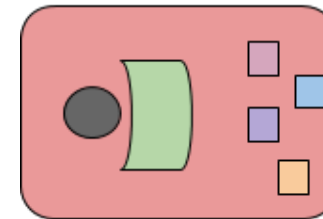
## Proprietary commercialization path...



Stage 0. Some code.



Stage 1. Software development team, internal use.  



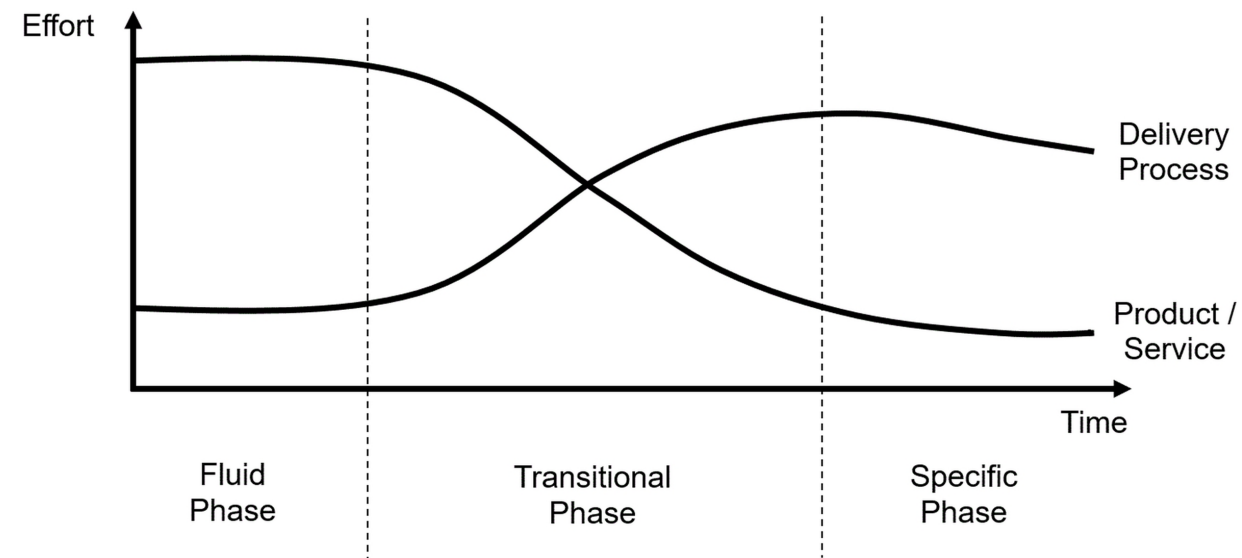
Stage X. Project goes commercial, without developing an open community.

S. P. Benthall, Software Incubator Workshop: A Synthesis,  
<http://urssi.us/blog/2019/02/25/software-incubator-workshop-a-synthesis/>



# Changing stages

- At each point/stage, decide consciously to go forward
- Think about methods, goals, and consequences
- What resources are available to help?
- What (type) of work will be needed?
- Are the right skills available?
- What are the incentives?
- How will success be measured?
- How will the institution(s) support this?



J. Leng , M. Shoura, T. C. B. McLeish, A. N. Real, et al. "Securing the future of research computing in the biosciences," *PLoS Comput Biol* 15(5): e1006958, 2019. <https://doi.org/10.1371/journal.pcbi.1006958>

# Software collapse

- Software stops working eventually if is not actively maintained
- Structure of computational science software stacks:
  1. Project-specific software (developed by researchers): software to do a computation using building blocks from the lower levels: scripts, workflows, computational notebooks, small special-purpose libraries & utilities
  2. Discipline-specific software (developed by developers & researchers): tools & libraries that implement disciplinary models & methods
  3. Scientific infrastructure (developed by developers): libraries & utilities used for research in many disciplines
  4. Non-scientific infrastructure (developed by developers): operating systems, compilers, and support code for I/O, user interfaces, etc.
- Software builds & depends on software in all layers below it; any change below may cause collapse

K. Hinsen, "Dealing With Software Collapse," 2019.  
<https://doi.org/10.1109/MCSE.2019.2900945>



# Research software summary

- Software developed and used for the purpose of research: to generate, process, analyze results within the scholarly process
- Increasingly essential in the research process
- But
  - Software will collapse if not maintained
  - Software bugs are found, new features are needed, new platforms arise
  - Software development and maintenance is human-intensive
  - Much software developed specifically for research, by researchers
  - Researchers know their disciplines, but often not software best practices
  - Researchers are not rewarded for software development and maintenance in academia
  - Developers don't match the diversity of overall society or of user communities

# Software sustainability

- Software sustainability  $\equiv$  the capacity of the software to endure
  - Will the software will continue to be available in the future, on new platforms, meeting new needs?
- Software development and maintenance requires human effort
- Human effort  $\Leftrightarrow$  \$
  - All human effort works (community open source)
  - All \$ (salary) works (commercial software, grant funded projects)
  - Combined is hard: effort  $\neq$  \$; humans are not purely rational

# Potential solutions

- Research software sustainability is the process of developing and maintaining software that continues to meet its purpose over time, which includes that the software adds new capabilities as needed by its users, responds to bugs and other problems that are discovered, and is ported to work with new versions of the underlying layers, including software as well as new hardware
- In order to sustain research software, we can
  - Do things that reduce the amount of work needed
  - Do things that increase the available resources
  - Do things that both reduce the amount of work needed and increase the available resources

# Methods to sustain research software (1)

- To reduce the amount of work needed
  - Train its developers, which involves finding or developing training material
    - Carpentries, discipline-specific materials, language-specific materials
  - Use best practices, which involves finding or developing best practices
    - Software Carpentry, Incubators (e.g. ESIP, Apache)

# Methods to sustain research software (2)

- To increase the available resources
  - Create incentives so that people want to work on the software
    - Citations that help in existing career paths
    - Adjusted existing career paths that they reward software work
    - New career paths
  - Increase available funding by first making the role of software in research clear to research funders, and then by clearly making the case for them to increase funding for new software, and to provide funding for software maintenance
  - Seek institutional resources if the software is considered sufficiently important to the institution, operationally or reputationally

# Methods to sustain research software (3)

- To both reduce work and bring in new resources, encourage collaboration
  - Using the work of others rather than reimplementing a function or package reduces what a software team (or its developers) needs to do themselves, even without assuming that the collaborators contribute to the software, which also may happen
  - Similarly, if others use a team's software and contribute to maintaining it, the team has less they need to do
  - To make this work, software has to be designed from the start to be modular and reusable, and it must also be clearly documented and explained to potential users, even those in fields other than the developer's
  - And the team has to put effort into engaging and working with the potential user and contributor community

# Volunteers & incentives (1)

- Why do volunteers or collaborators choose to put effort into our software project?
- How we can engage them?
- In the context of community activities and organizing, Porcelli defines

**Engagement = intrinsic motivation + extrinsic motivation + support – friction**

Joseph Porcelli, “How to grow users into active community members and get your community more engaged”, 2013 Open Source Software Summit

- Intrinsic motivation = self-fulfillment, altruism, satisfaction, accomplishment, pleasure of sharing, curiosity, real contribution to science
- Extrinsic motivation = job, rewards, recognition, influence, knowledge, relationships, community membership
- Support = ease, relevance, timeliness, value
- Friction = technology, time, access, knowledge

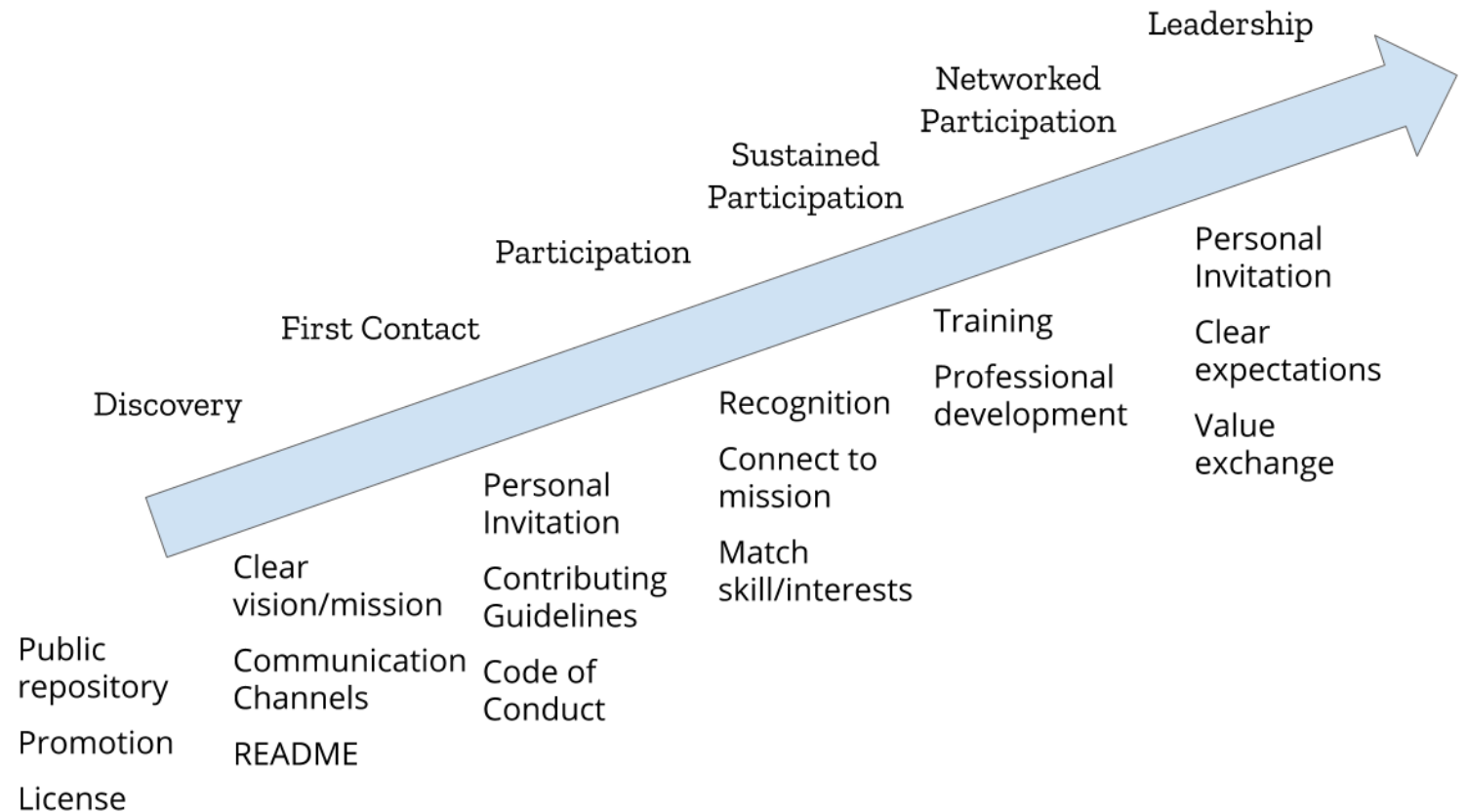


# Volunteers & incentives (2)

- Examples of things we can do:
  - Use GitHub for development
    - Reduce friction by using a known technology
  - Provide templates for issues and guidelines for good pull requests
    - Reduce friction by providing knowledge of how to work with our project
    - Increase support by easing the means of doing so
  - Provide a code of conduct and a welcoming and encouraging environment
    - Increase extrinsic motivation by helping develop relationships and sense of community
  - Add contributors to a list of authors who are cited when the software is used
    - Increase both intrinsic motivation and extrinsic motivation through recognizing accomplishments
  - Highlight examples of how the software is used
    - Increase intrinsic motivation by demonstrating the contribution to science

# Volunteers & incentives (3)

- Plan for a progression of types of engagements
- How a project can encourage the potential contributor to move to from level to another



Abigail Cabunoc Mayes, "Work Open, Lead Open," Chan-Zuckerberg Initiative (CZI) Essential Open Source Software (EOSS) Kickoff Meeting, 2020.

# Credits

- Thanks to Arfon Smith and Kyle Niemeyer for co-leadership in FORCE11 Software Citation WG
- And Neil Chue Hong & Martin Fenner for co-leadership in FORCE11 Software Citation Implementation WG
- And colleagues Gabrielle Allen, C. Titus Brown, Kyle Chard, Ian Foster, Melissa Haendel, Christie Koehler, Bill Miller
- And to the BSSw project (<http://bssw.io>) for a fellowship to pursue some parts of the citation work
- More of Dan's thinking
  - Blog: <http://danielskatzblog.wordpress.com>
  - Tweets: @danielskatz