

# Research Software Sustainability

**LaTech CSC 532: Advanced Topics: Software Engineering**  
**29 September 2020**

Daniel S. Katz

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

Assistant Director for Scientific Software & Applications

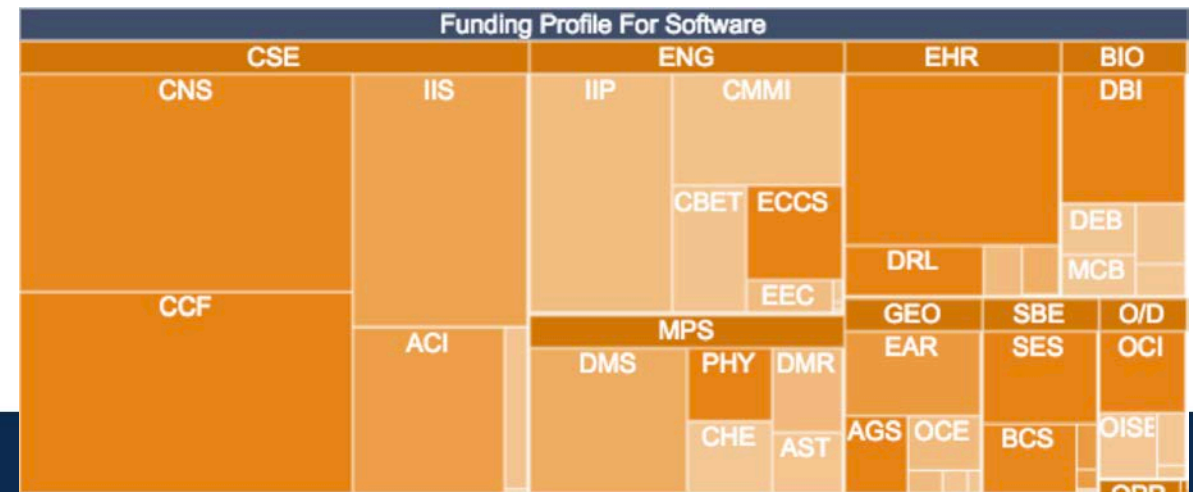
Research Associate Professor, CS, ECE, iSchool



NCSA | National Center for  
Supercomputing Applications

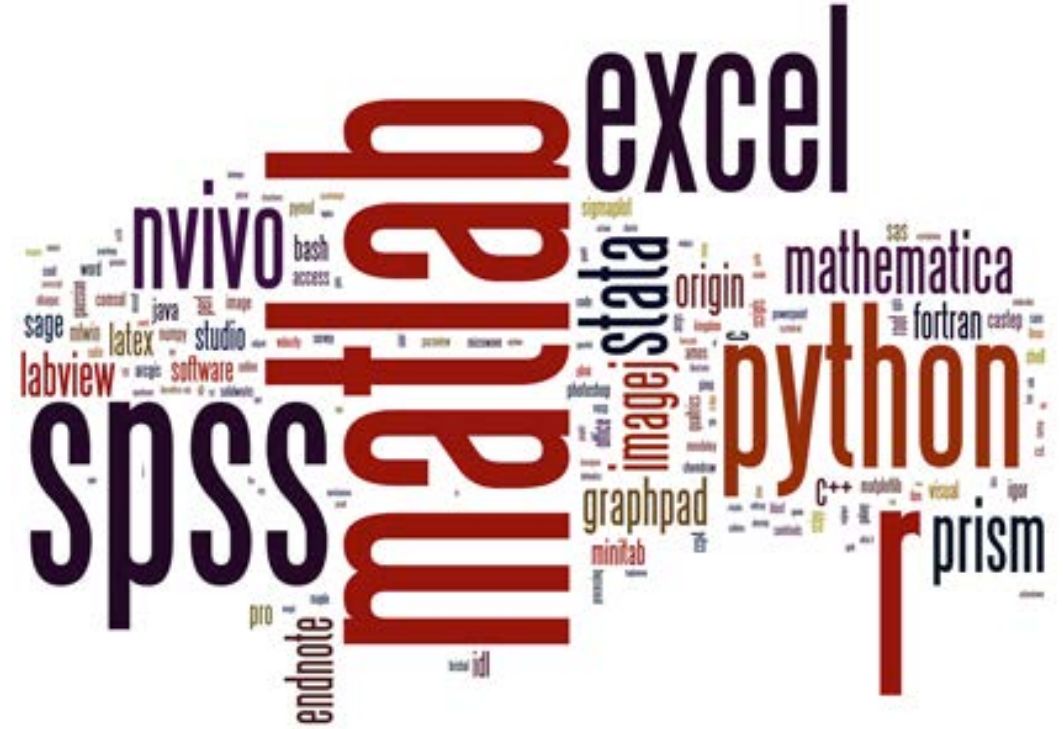
# Why do we care about research software?

- NSF
  - 1995-2016: 18,592 awards totalling \$9.6 billion with project abstracts that topically include “software”
  - ~20% of the overall NSF research budget
- DOE
  - Of 3 ECP areas, most of 2 (application development & software) technology are research software
  - According to Paul Messina in 2017, “ECP is a 7-year project with a cost range of \$3.5B–\$5.7B”



# Why do we care about research software?

- 40 papers in Nature (Jan-Mar 2016)
  - 32 explicitly mentioned software
  - Average of 6.5 software tools/paper
  - Most of which were research software
- Top 100-cited papers:
  - 6 of top 13 are software papers
  - “... the vast majority describe experimental methods or software that have become essential in their fields.”



# Why do we care about research software?

- Surveys of UK academics at Russell Group Universities (2014) and members of (US) National Postdoctoral Research Association (2017):
  - I use research software: 92% / 95% (UK/US)
  - My research would not be possible without software: 67% / 63%
  - My research would be possible but harder: 21% / 31%
  - I develop my own software: 56% / 28%

# Science of research software organizations

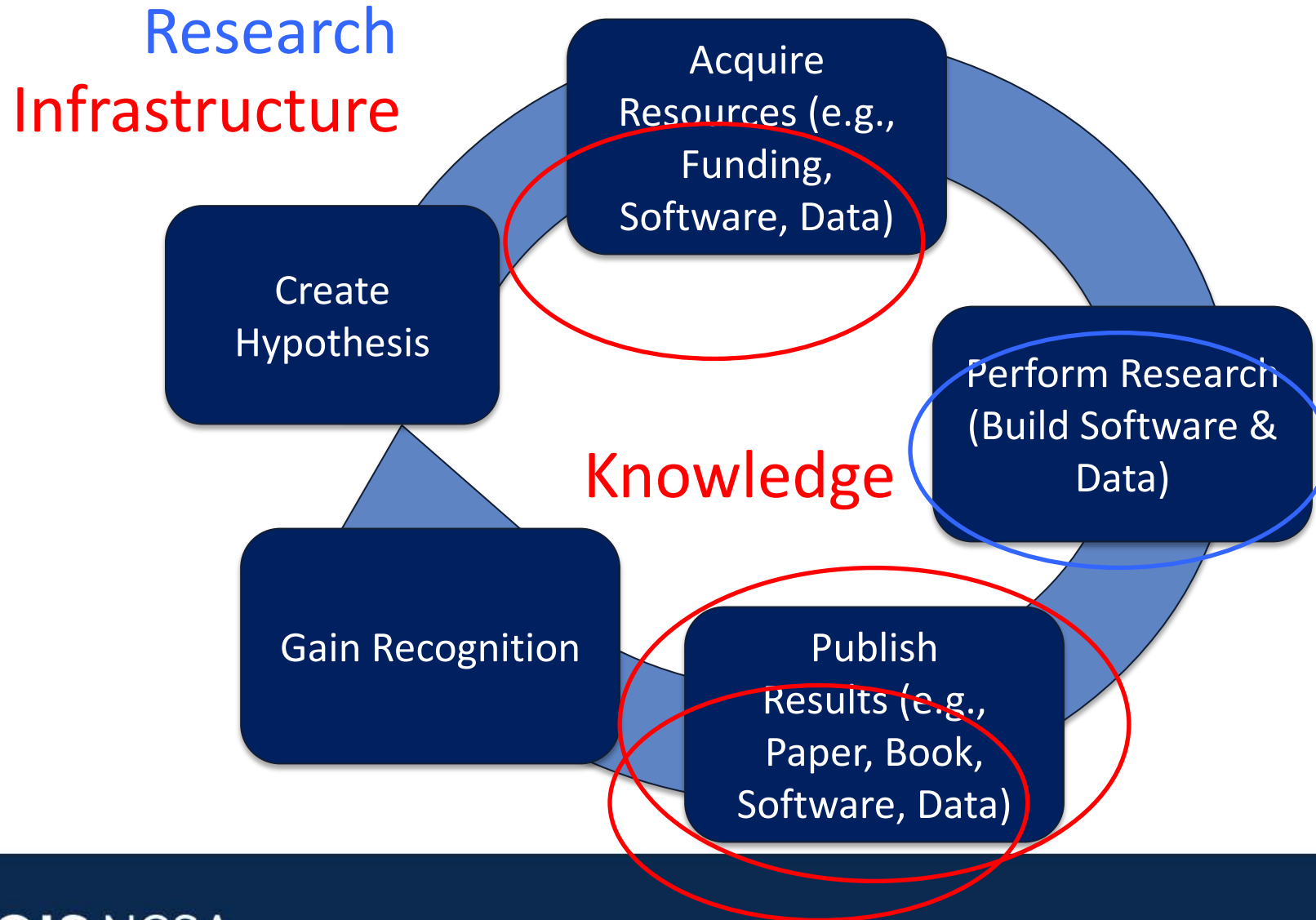
- Software Sustainability Institute (SSI)
  - In third period of funding, 10+ years
  - Now funded by all UK research councils
- Better Scientific Software (BSSw)
  - Clearinghouse to gather, discuss, and disseminate experiences, techniques, tools, and other resources to improve developer productivity and software sustainability
  - DOE funded
- United States Research Software Sustainability Institute (URSSI)
  - Conceptualization project under NSF funding
  - Institute proposal planned in 2020
  - Interest from private foundations
- Research Software Alliance (ReSA)
  - Intended to coordinate the above & others internationally

# Open source & software growth

- 2001: 208K SourceForge users
- 2017: 20M GitHub users
- 2019: 37M GitHub users
- 1998: 180K downloads of Netscape (app) in 2 weeks
- 2017: 21M downloads of lodash (javascript library) in 2 weeks
- 2018 survey of scientist-developers found that 82% of respondents felt that they were spending spending “more time” or “much more time” developing software than they did 10 years ago



# Software in research cycle



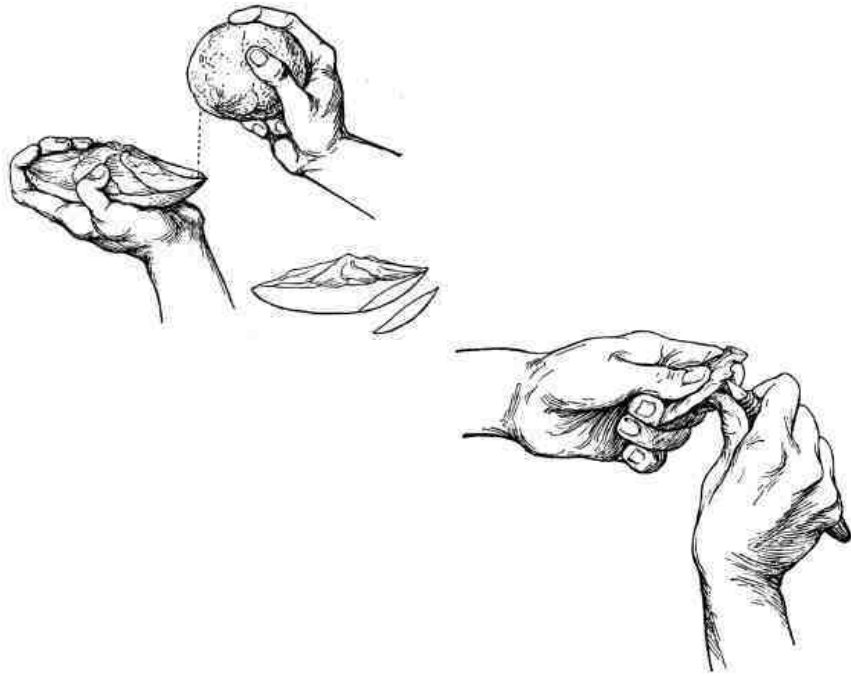
# Research software vs. infrastructure software

- Some research software is intended just for research
  - Funded by many agencies, sometimes explicitly, often implicitly
  - Intended for immediate use by developer
  - Maybe archived for future use and reproducibility
  - And probably dependent on infrastructure software
- Other research software is intended as infrastructure
  - To be shared
  - Funded by some agencies, almost always explicitly
  - Intended for use by community
  - Appreciation and reward easier because of sharing
- Software intended for research can be turned into infrastructure software
  - Requires making a conscious choice
  - Has consequences, both positive and negative



# 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

# Parsl: Interactive parallel programming in Python

*Apps* define opportunities for parallelism

- Python apps call Python functions
- Bash apps call external applications

Apps return “futures”: a proxy for a result that might not yet be available

Apps run concurrently respecting data dependencies. Natural parallel programming!

Parsl scripts are independent of where they run. Write once run anywhere!

```
pip install parsl
```

```
@python_app
def hello():
    return 'Hello World!'

print(hello().result())
```

Hello World!



```
@bash_app
def echo_hello(stdout='echo-hello.stdout'):
    return 'echo "Hello World!"'

echo_hello().result()

with open('echo-hello.stdout', 'r') as f:
    print(f.read())
```

Hello World!



# Parsl project summary

- Based on improving ideas in Swift workflow system/language & 10+ years of CS/infrastructure research
- Initially funded by NSF, \$3m over 3 years (stretched to 4)
- 2.5 core developer FTEs, PI, co-PIs, chemistry & education application developers, undergraduate & graduate students
- Open source, intended as open community, including library of reusable workflows
- Interesting milestones
  - First outside user
  - First outside user who didn't contact us
  - First outside contributor
  - First outside contributor who didn't contact us
- Some success with purely external contributions to code, more success with collaborating projects (e.g., LSST-DESC, Thain group @ ND), some providing funding
- New follow-on NSF project (funcX) will use (and support) Parsl as dependency

# Who starts new research software projects?

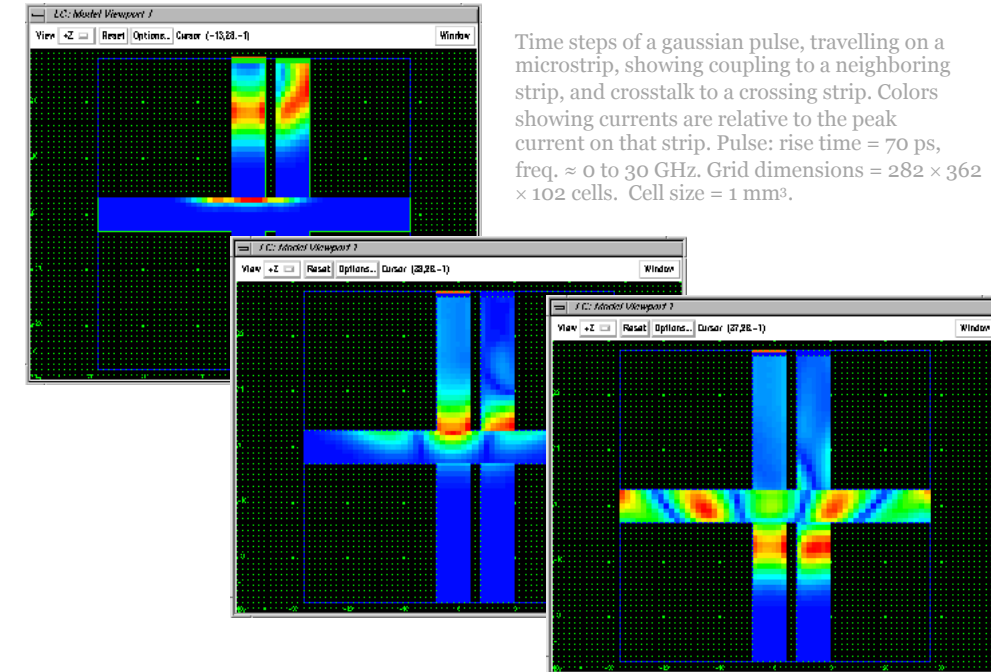
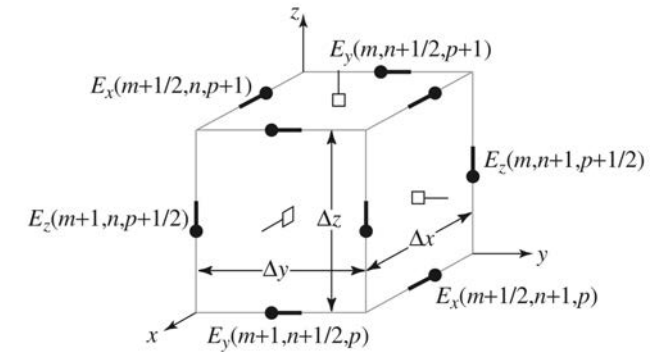
- User/Developer
  - To scratch their own itch



- 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

# FDTD electromagnetics

- My PhD dissertation was “Boundary treatments applied to the FD-TD method for solving problems of electromagnetic wave propagation” (1994)
- Adding methods to treat curved materials and infinite space to my advisor/lab’s time-marching code that solved EM problems on a finite cartesian grid
- Code originally written in late 1970s, FORTRAN77, no subroutines
- While I was working on it: heavily modified, better engineered, ported to various languages, vector computing, parallel computing
- But never shared – was viewed as part of the lab’s IP
- Algorithm was shared, leading others to redevelop code, now many versions including open source and commercial



# Project stages

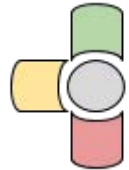
## Schematic stages of open community for research software





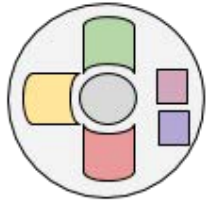
**Stage 0.** Some code and a user of it. No sustained team.



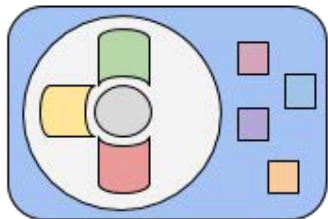
**Stage 1.** Software development team, internal use.  



**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.).

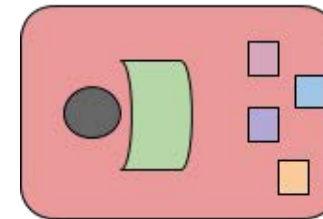
## Proprietary commercialization path...



Stage 0. Some code.



Stage 1. Software development team, internal use.  

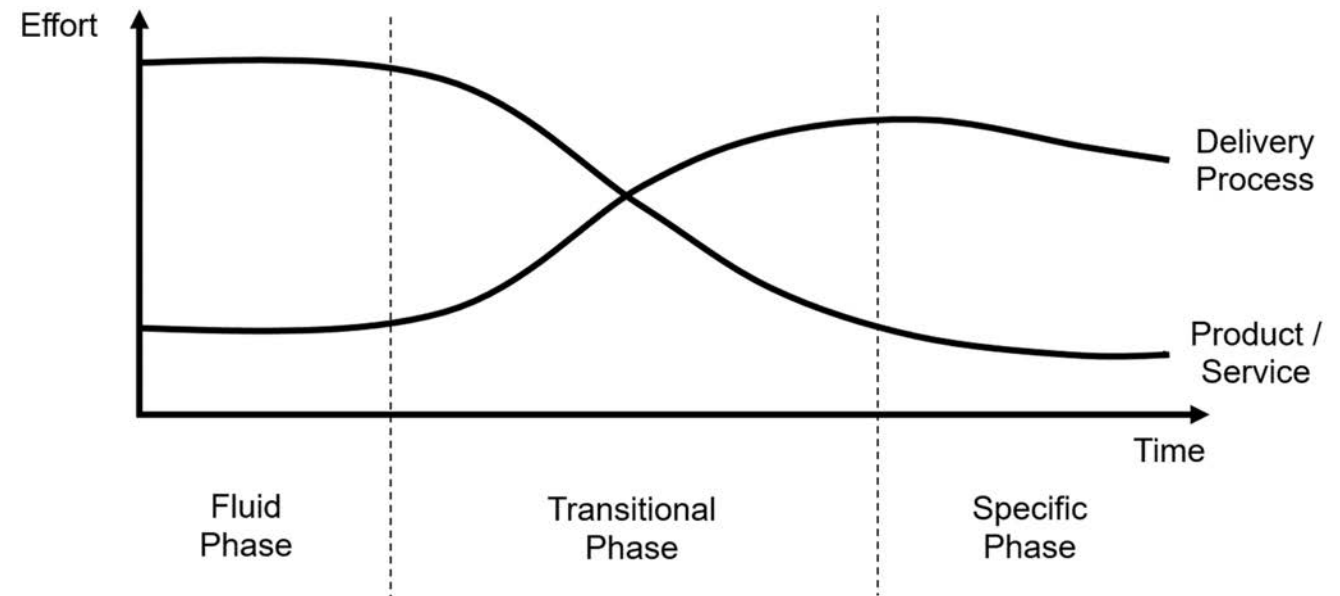


Stage X. Project goes commercial, without developing an open community.



# 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?





# 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

# Software sustainability

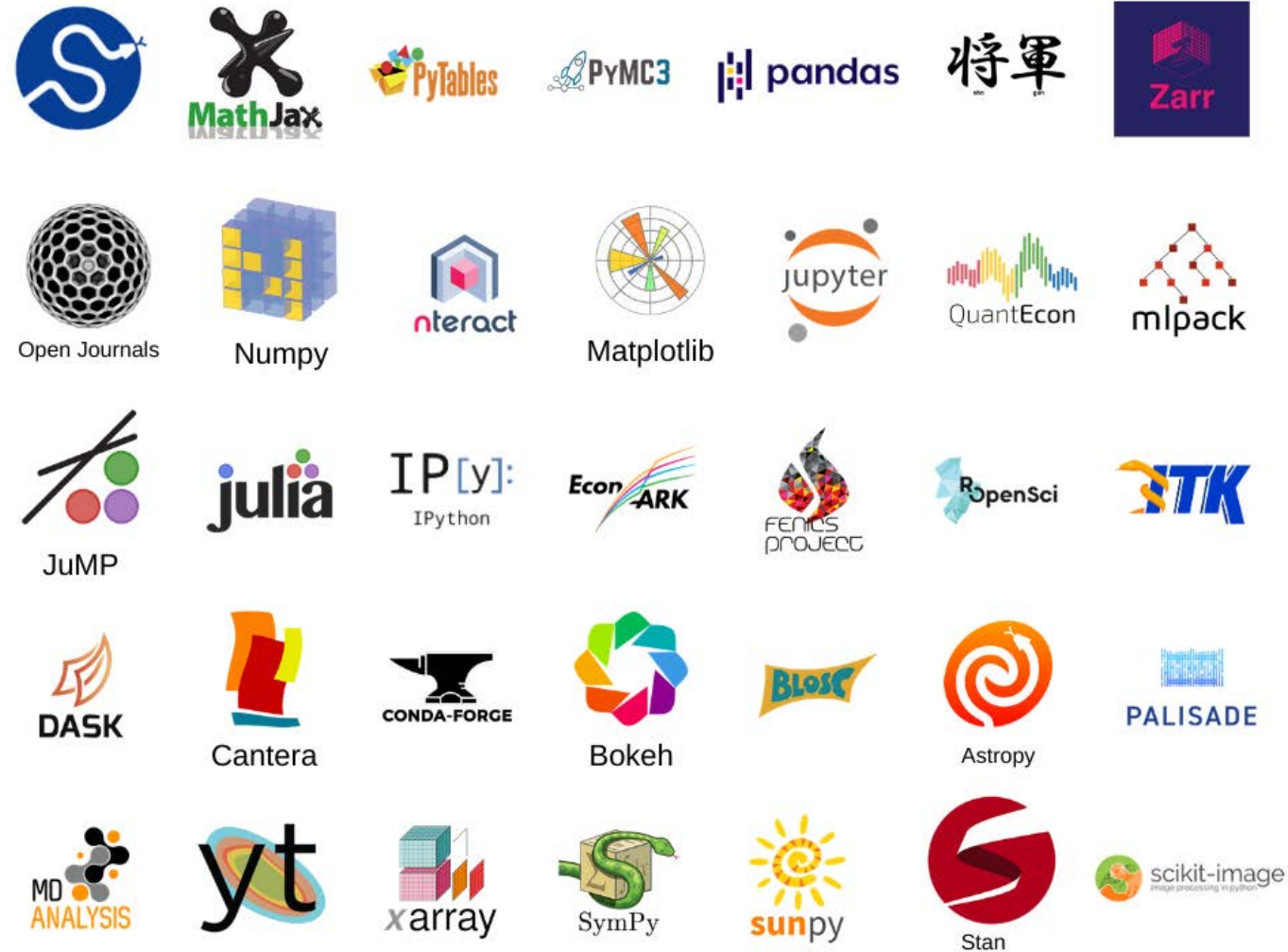
- Software sustainability  $\equiv$  the capacity of the software to endure
  - Will the software will continue to be available in the future, on new platforms, meeting new needs?
- Software development and maintenance requires human effort
- Human effort  $\Leftrightarrow$  \$
  - All human effort works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

# Example/Problem 1: OpenSSL

- 1998: UK group built internet encryption tools: OpenSSL
- 2011: Heartbleed bug introduced
- 2014: 2/3 of web sites rely on OpenSSL
- One full-time developer: Steven Hensen, barely supported by OpenSSL Software Foundation (OSF)
  - Private, for-profit company
- 2014: Heartbleed bug discovered
- OpenSSL bug fixed
- OSF requests donations
  - \$9,000 given initially
- Further campaign led to support for 4 developers for 3 years
- Today: active project with 18 committers and financial & in-kind (including staff time) support from companies

# Example/Problem 2: Bus factor

- NumFOCUS: Umbrella non-profit to support scientific software
- NumFOCUS sustainability summit annually since 2017
  - 2017 bus factor survey
  - Bimodal, ~half 1-2, ~half 4-6
  - One project's story: developer support, backlog, students
- Wider open source community
  - Two-thirds of popular projects: bus factor of 1 or 2



# Example/Problem 3: Faculty recognition

- Young faculty member, very involved in open source, open science, reproducibility
- In recent pre-promotion case meeting with chair, told:
  - Committee won't count software efforts or papers
    - They're not "traditional" journal papers
  - Visibility and recognition for open science & reproducibility are ok ...
    - but official h-index is more important and not high enough

# Example/Problem 4: Expectations



**Eric O LEBIGOT (EOL)**

@lebigot

Follow



Pandas is a key data library. But its

documenta

does DataF

default?). c

4:14 AM - 18 Sep 201



**Wes McKinney** ✓

@wesmckinn

Follow



If you aren't happy with the documentation,

please submit

getting paid to



**Wade Johnston**

@wj\_chicago

Follow

Replying to @lebigot @wesmckinn

As you're a

allocate some

contribute to



**Terry Koch**

@tkoch\_a

Follow



Replying to @wj\_chicago @lebigot @wesmckinn

Probably can & probably won't. Too many look at OSS as "You give me sw for free, and in return I'll feel free to use & complain about it."

# 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



# Max Planck

- Eine neue wissenschaftliche Wahrheit pflegt sich nicht in der Weise durchzusetzen, daß ihre Gegner überzeugt werden und sich als belehrt erklären, sondern vielmehr dadurch, daß ihre Gegner allmählich aussterben und daß die heranwachsende Generation von vornherein mit der Wahrheit vertraut gemacht ist
- A new scientific truth does not triumph by convincing its opponents and making them see the light, but rather because its opponents eventually die, and a new generation grows up that is familiar with it
- Or: science advances one funeral at a time
- My version: culture of science advances one funeral at a time

- Enough bad news
- What can we do?
- Wait
  - cf Planck
- Or act

# 12 scientific software challenges

- Incentives, citation/credit models, and metrics
- Career paths
- Training and education
- Software engineering
- Portability
- Intellectual property
- Publication and peer review
- Software communities and sociology
- Sustainability and funding models
- Software dissemination, catalogs, search, and review
- Multi-disciplinary science
- Reproducibility

All are tied together

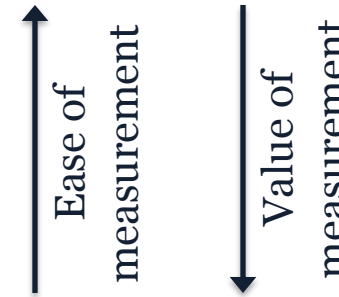
# Credit for software: What to measure?

## 1. Developer of open source physics simulation

### – Possible metrics

- How many downloads?
- How many contributors?
- How many uses?
- How many papers cite it?
- How many papers that cite it are cited?

impact



## 2. Developer of open source math library

- Possible metrics are similar, but citations are less likely
- May not be able to measure downloads
  - It's part of a distribution
  - It's pre-installed (and optimized) on an HPC system
  - It's part of a cloud image
  - It's a service

# Directly measuring software

- Downloads
  - From web/repository logs/stats
- Installations/Builds
  - Add “phone home” mechanism at build level
    - As done by Homebrew package manager for OS X
  - e.g., add ‘curl URL’ in makefile
- Uses/Runs
  - Track centrally; add “phone home” mechanism at run level
    - Per app, e.g. add ‘curl URL’ in code, send local log to server
    - For a set of apps, e.g., sempervirens project, Debian popularity contest
  - Track locally; ask user to report
    - e.g., duecredit project, or via frameworks like Galaxy
- Overall, some working tools, some proofs-of-concept, all with privacy concerns

# Measuring software impact

- Citations
  - Because citation system was created for papers/books
  - Need to jam software into current citation system
- Altmetrics
  - Not citations, but other structured measures of discussion (tweets, blogs, etc.)
- ImpactStory
  - Measures research impact: reads, citations, tweets, etc.
- Depsy (roughly ImpactStory specifically for software)
  - Measures software impact: downloads, software reuse (if one package is forked into another package), citations, tweets, etc.
- Libraries.io
  - Counts software dependencies

# Software citation today

- Software and other digital resources currently appear in publications in very inconsistent ways
- Howison & Bullard: random sample 90 articles in biology literature -> 7 different types of software mentions

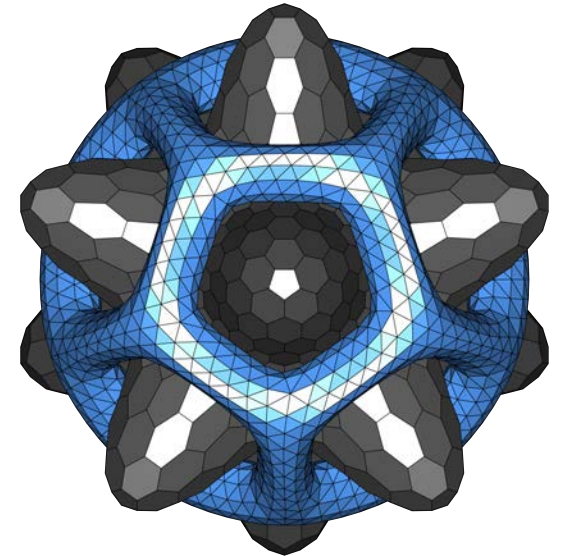
Mention Type	Count (n=286)	Percentage
Cite to publication	105	37%
Cite to users manual	6	2%
Cite to name or website	15	5%
Instrument-like	53	19%
URL in text	13	5%
In-text name only	90	31%
Not even name	4	1%

- Studies on data and facility citation -> similar results



# Journal of Open Source Software (JOSS)

- A developer friendly journal for research software packages
- “If you've already licensed your code and have good documentation then we expect that it should take **less than an hour** to prepare and submit your paper”
- Everything is open:
  - Submitted/published paper: <http://joss.theoj.org>
  - Code itself: where is up to the author(s)
  - Reviews & process: <https://github.com/openjournals/joss-reviews>
  - Code for the journal itself: <https://github.com/openjournals/joss>
- JOSS papers archived, have DOIs, increasing indexed
- First paper submitted 4 May 2016
  - 31 May 2017: 111 accepted papers, 56 under review and pre-review
  - 22 Sept 2020: 1023 accepted papers, 177 under review and pre-review
  - Pre-Covid-19 publication rate ~400 papers/year, now back to that rate
- Editors: 1 editor-in-chief and 11 editors at launch;  
1 EiC, 5 associate EiCs, 48 topic editors, 13 emeritus editors today



# Software citation principles

- FORCE11 Software Citation group started July 2015
- WSSSPE3 Credit & Citation working group joined September 2015
  - ~55 members (researchers, developers, publishers, repositories, librarians)
- Reviewed existing community practices & developed use cases
- Drafted & published software citation principles
  - Started with data citation principles, updated based on software use cases and related work, updated based working group discussions, community feedback, workshop
  - Smith AM, Katz DS, Niemeyer KE, FORCE11 Software Citation Working Group.(2016) Software Citation Principles. PeerJ Computer Science 2:e86. DOI: [10.7717/peerj-cs.86](https://doi.org/10.7717/peerj-cs.86)
  - Principles: **importance, credit and attribution, unique identification, persistence, accessibility, specificity**
- Software Citation Working Group ended April 2017

# Software citation implementation

- FORCE11 Software Citation Implementation Working Group in progress, started May 2017
  - Co-chairs: Neil Chue Hong, Martin Fenner, Daniel S. Katz
  - Goal is implementing software citation
  - Working with institutions, publishers, technology and service providers, funders, researchers, etc.
- Lots of good work being done, and good coordination of ongoing activities
- Metadata standards and translation (DataCite Schema 4.1, CodeMeta, citation.cff), being aligned with schema.org
- Open source archiving and identification (Software Heritage) developing
- Good work and initial acceptance in communities (astronomy, Earth science, math, ...)
- Published Software Citation Checklist for Authors v0.9 document
- Published Software Citation Checklist for Developers v0.9 document
- Software Citation Checklist for Reviewers document under review
- Repositories task force developing good/best practices for registries and repositories
- Journals task force started in Jan 2020, guidance document for journals (and conferences to use) in publication

# 12 scientific software challenges

- Incentives, citation/credit models, and metrics
- Career paths
- Training and education
- Software engineering
- Portability
- Intellectual property
- Publication and peer review
- Software communities and sociology
- Sustainability and funding models
- Software dissemination, catalogs, search, and review
- Multi-disciplinary science
- Reproducibility

All are tied together

# Challenge: better career paths?

- Career paths for software developers in universities unclear
- Should we give up in favor of national labs & government intramural researchers & industry?
  - More financial rewards, cohorts (others with similar problems and solutions) and promotion opportunities
- For researchers, published software or software papers make software a valued output similar to publications
- University centers, e.g. NCSA, SDSC, TACC, give programmers a home & critical mass for career paths
- Moore/Sloan Data Science program created new structure across universities
- Research Software Engineers (RSEs) ...



# Who are Research Software Engineers?

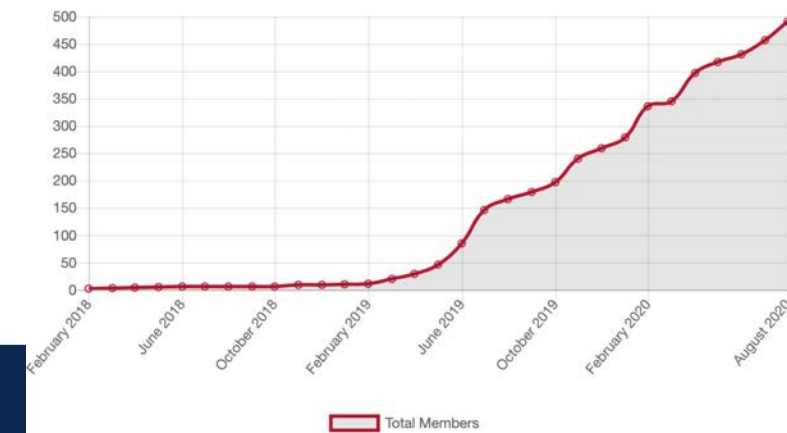
- Not independent researchers
  - No personal research agenda
- Facilitative, supportive, and collaborative
  - Part of the academic community
  - With professional IT skills
  - Deep engagement with research groups
  - Understand, study, and be part of group research activities
  - Can read and understand the papers
- Sustainable and long term
  - Institutional memory
  - Continuity, stability, maintenance
- But in most institutions
  - Without a formal home in academia, or a career path

# The Story of RSEs

- April 2012: Idea & name at SSI Collaborations Workshop
- September 2012: University College London group founded
- More groups: Manchester (2014); Sheffield, Southampton, and Cambridge (2015)
- January 2016: EPSRC awards first RSE Fellowships
- RSE Conferences in the UK
  - September 2016: First RSE conference, 202 attendees, 14 countries
  - September 2017: Second RSE conference, 224 attendees
  - September 2018: Third RSE conference, 340 attendees
  - September 2019: Fourth **UK** RSE conference, 360 attendees
- Society of Research Software Engineering – formed 2019
  - Independent organization for RSEs: international, membership fee, voting rights
- Other countries at various stages of development
  - Germany, Netherlands, Nordic, US, South Africa, Canada, Australia/NZ, Belgium, ...
  - First conferences in Germany and Netherlands occurred in 2019
  - 491 US RSE members (as of Sept 1 2020)



Membership in the US Research Software Engineer Association



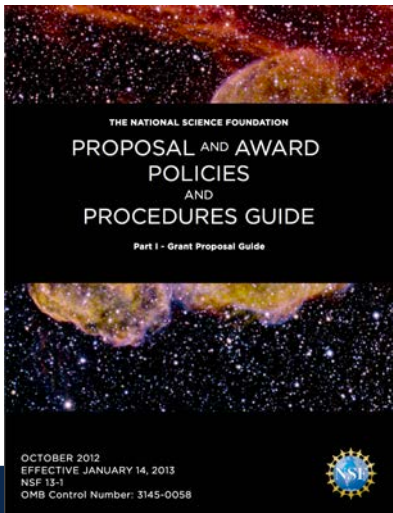
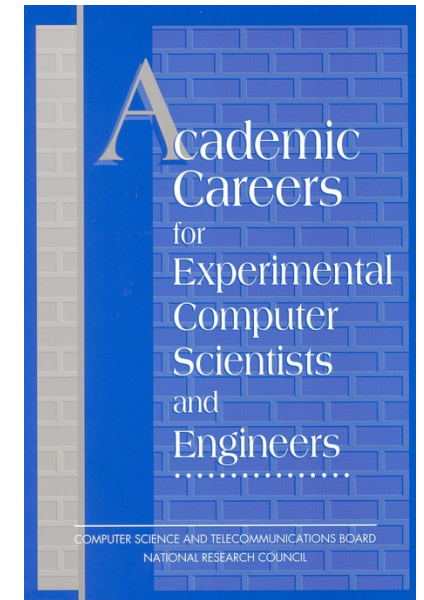


# Promotion and evaluation

- Guidelines for promotion and evaluation important
  - Say what's valued; shape activities people undertake
  - Promotion guidelines written by senior people, how can they be changed?
- We can influence these processes when we participate in these evaluations
- We can provide templates and guidelines for recognizing software contributions and encourage respected organizations to adopt them
  - Multiple groups working in this space

# Promotion and evaluation are not fixed

- National Academies (1994): “Academic Careers for Experimental Computer Scientists and Engineers”
  - Experimental artifacts are important in CS, should be part of evaluation
  - Intended to provide a reference point for change
  - Has been quoted in many tenure recommendation letters
- NSF 2013 biosketch change: products, not publications
  - Acknowledges software contributions as a primary research product
  - Intended to signal to universities that they should do the same



# Potential systematic solutions

- Convince governments and funders of importance of software (and sustained funding for some of it, including maintenance)
  - via Research Software Alliance (ReSA) and others
- Encourage use of software citation to aid developers
  - via FORCE11 Software Citation Implementation Working Group
- Build better career paths for developers
  - via Research Software Engineer (RSE) movement (<https://society-rse.org>, <http://us-rse.org>)
- Develop and use software best practices
  - via Project Carpentry, Incubators (e.g. ESIP, Apache)
- Join groups working on these
  - SSI, URSSI, NumFOCUS, CS&S

# Potential project-specific 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
    - Project 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**

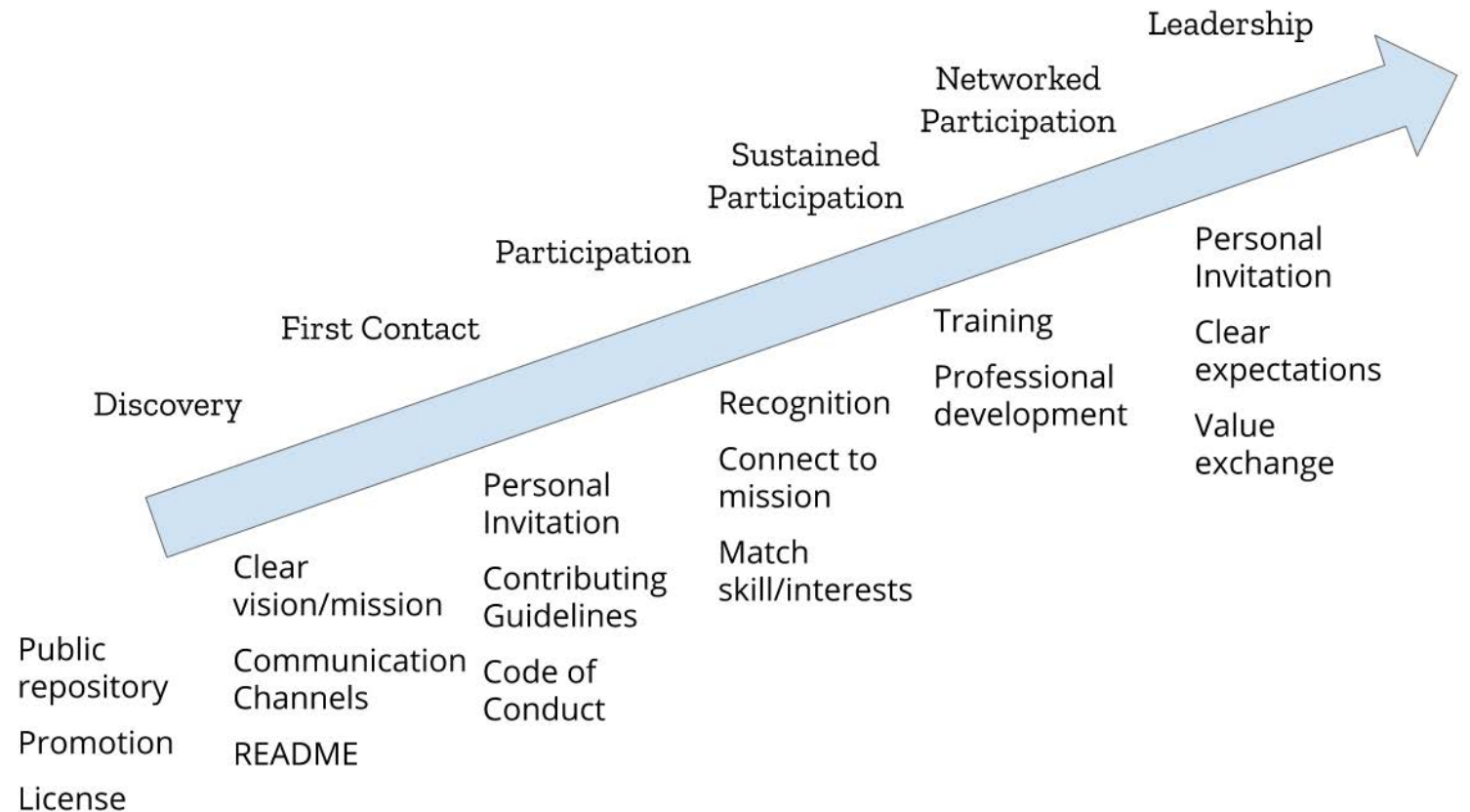
- Intrinsic motivation = self-fulfillment, altruism, satisfaction, accomplishment, pleasure of sharing, curiosity, real contribution to science
- Extrinsic motivation = job, rewards, recognition, influence, knowledge, relationships, community membership
- Support = ease, relevance, timeliness, value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



# 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, Rajiv Ramnath
- And to the BSSw project (<http://bssw.io>) for a fellowship to pursue some parts of the citation work
- More of my thinking
  - Blog: <http://danielskatzblog.wordpress.com>
  - Tweets: @danielskatz