Scientific Software Challenges and Community Responses

Daniel S. Katz

Division of Advanced Cyberinfrastructure (ACI)

dkatz@nsf.gov, d.katz@ieee.org, @danielskatz

Data Science Seminar
March 4, 2016, Indiana University

NSF Division of Advanced Cyberinfrastructure (ACI)

- My role is as a program officer in ACI, which:
  - Supports and coordinates the development, acquisition, and provision of state-of-the-art cyberinfrastructure resources, tools, and services
  - Supports forward-looking research and education to expand the future capabilities of cyberinfrastructure
  - Serves the growing community of scientists and engineers, across all disciplines, whose work relies on the power of advanced computation, data-handling, and networking

What is cyberinfrastructure?

"Cyberinfrastructure consists of
computing systems,
data storage systems,
advanced instruments and
data repositories,
visualization environments, and
people,
all linked together by
***software** and*
high performance networks,
to improve research productivity and
enable breakthroughs not otherwise possible."

-- Craig Stewart

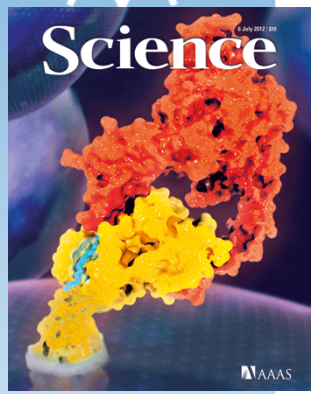
Infrastructure challenges for science (1)

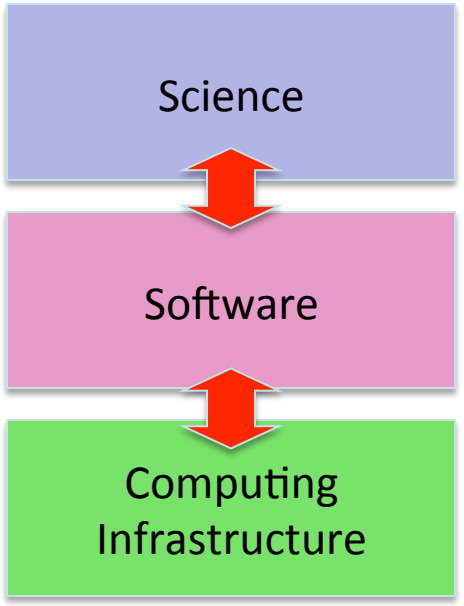
- Science
  - Larger teams, more disciplines, more countries
  - Coupling of simulation, experiment, observation
- Data
  - Size, complexity, rates all increasing rapidly
  - Need for interoperability (systems and policies)
- Systems
  - More cores, more architectures (GPUs), more memory hierarchy
  - Changing balances (latency vs bandwidth)
  - Changing limits (power, funds)
  - System architecture and business models changing (clouds)
  - Network capacity growing; increase networks -> increased security

Infrastructure challenges for science (2)

- Software
  - Multiphysics algorithms, frameworks
  - Data analysis and understanding
  - Programing models and abstractions for science, data, and hardware
  - Complexity
  - Productivity vs. efficiency
  - End-to-end performance needed across complex science driven workflows
  - V&V, reproducibility, fault tolerance
  - Collaboration with industry
- People
  - Education and training
  - Building a diverse workforce
  - Lifelong re-training
  - Career paths
  - Credit and attribution

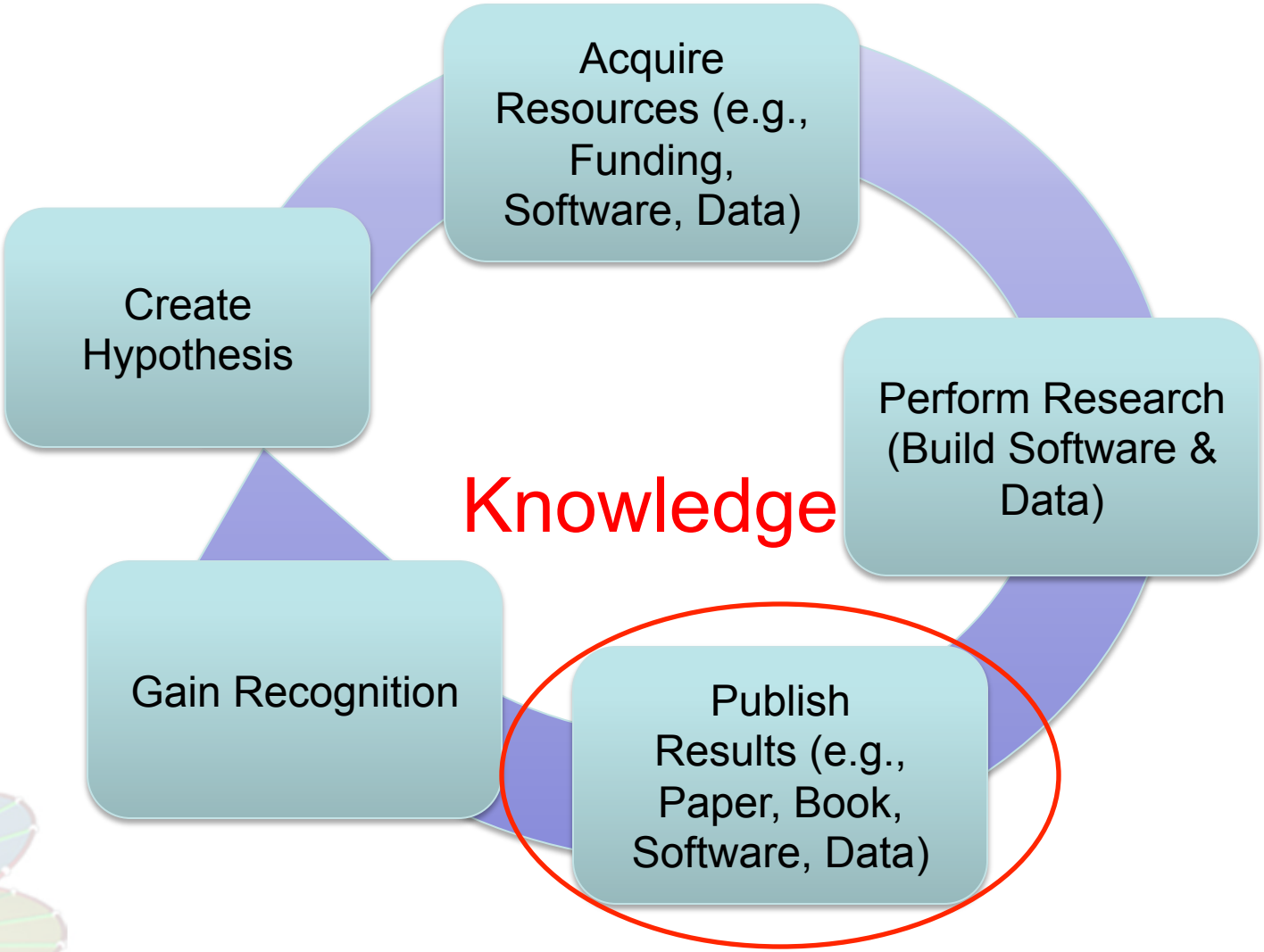
Software as infrastructure



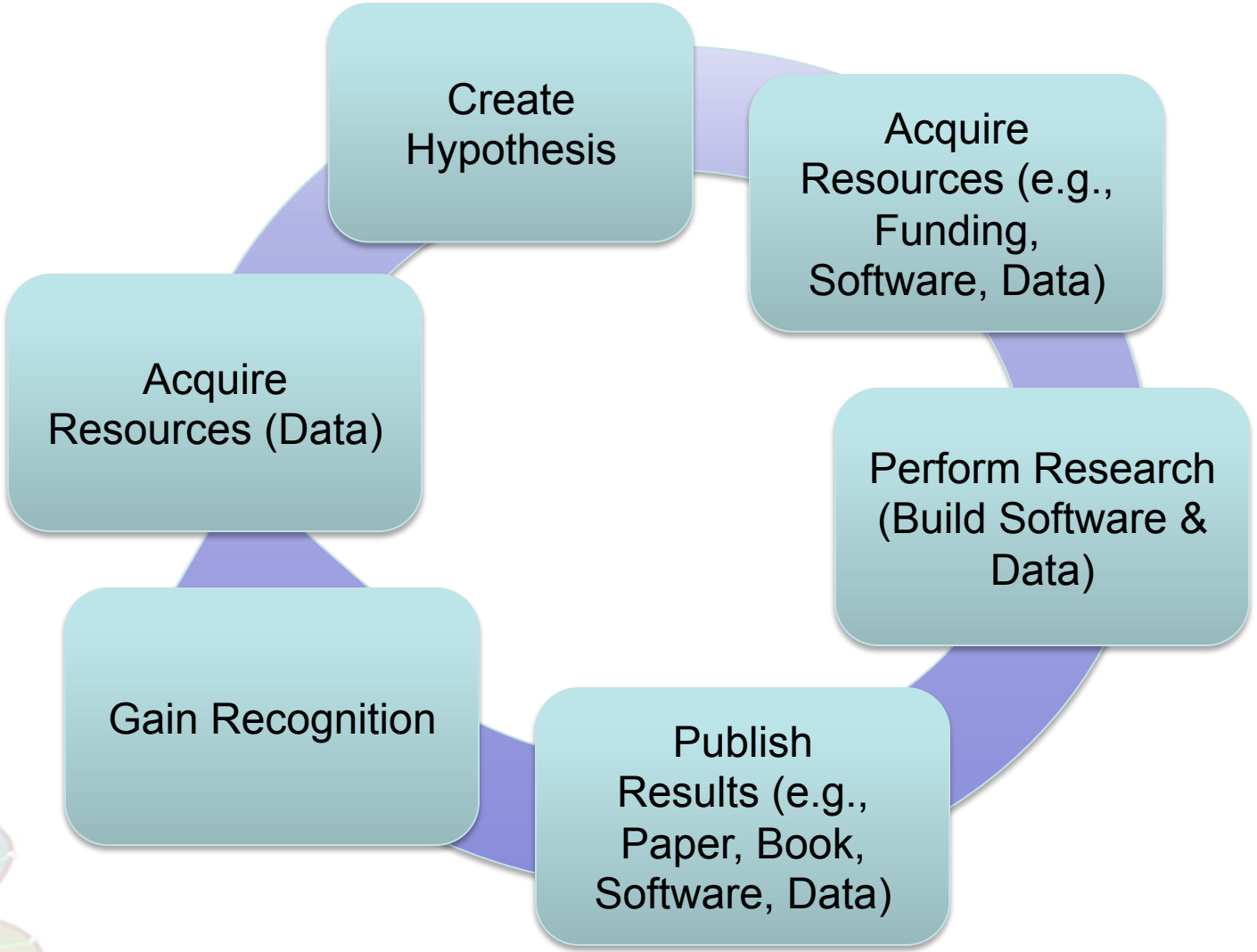


- Software (including services) essential for the bulk of science
  - About half the papers in recent issues of Science were software-intensive projects
  - Research becoming dependent upon advances in software
  - Significant software development being conducted across NSF: NEON, OOI, NEES, NCN, iPlant, etc
  - Wide range of software types: system, applications, modeling, gateways, analysis, algorithms, middleware, libraries
- Software is not a one-time effort, it must be sustained
  - Development, production, and **maintenance** are people intensive
  - Software life-times are long vs hardware
  - Software has under-appreciated value

Computational science research



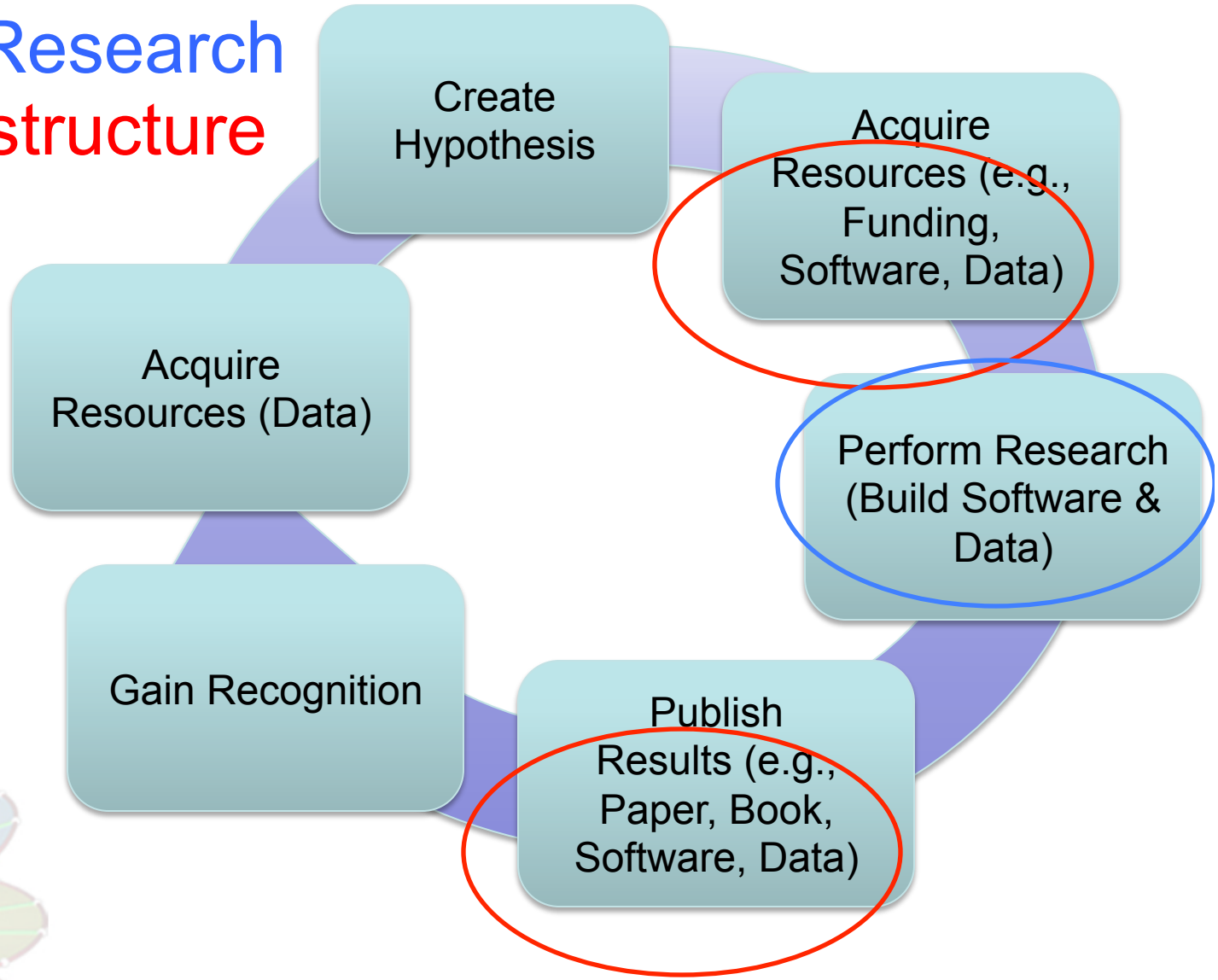
Data science research



Purposes of software in research

Research

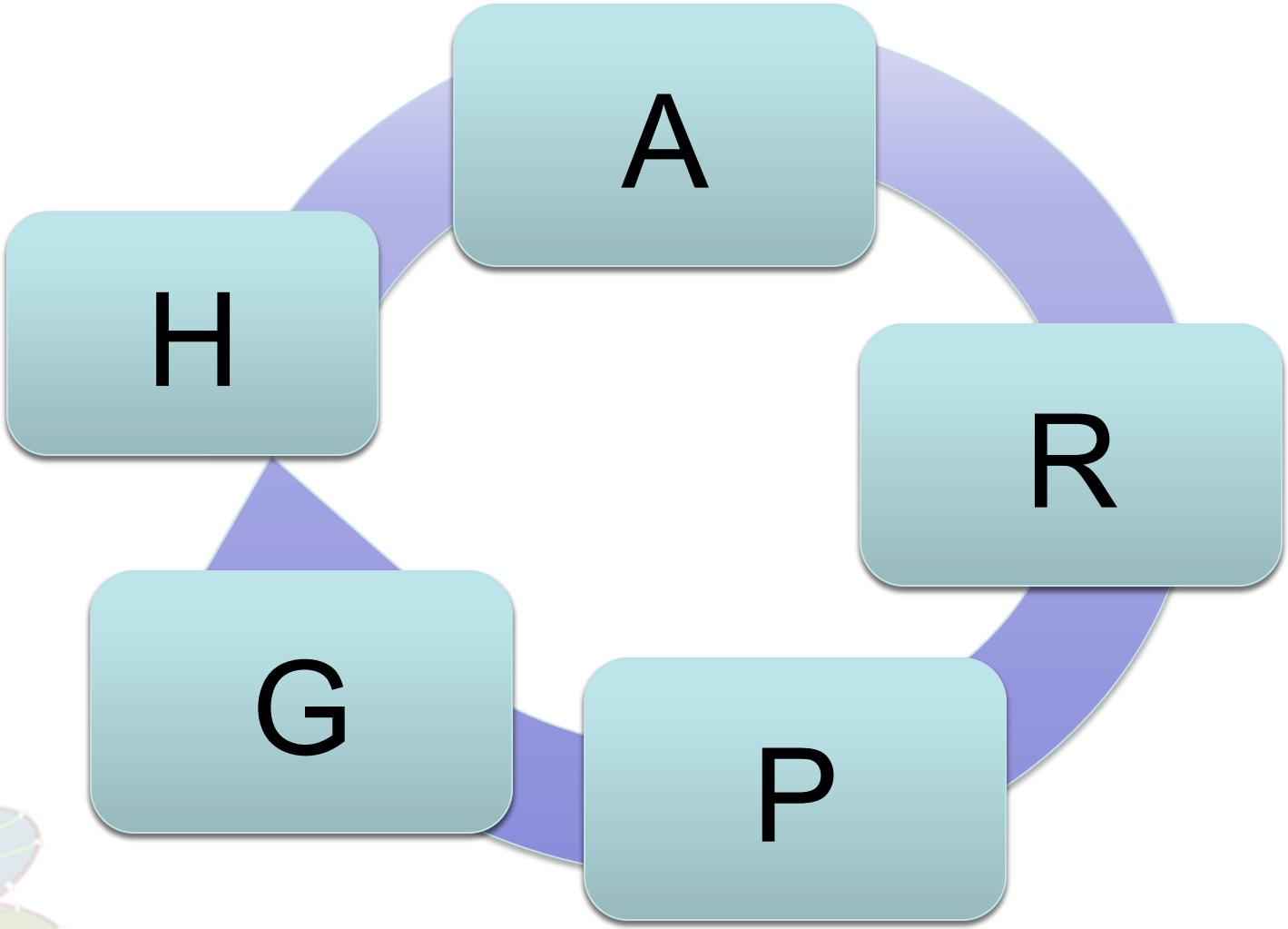
Infrastructure



Research software vs. infrastructure software

- Some software is intended for research
  - Funded by many parts of NSF, sometimes explicitly, often implicitly
  - Intended for immediate use by developer
  - Maybe archived for future use and reproducibility
- Other software is intended as infrastructure
  - Funded by many parts of NSF, often ACI, almost always explicitly
  - Intended for use by community
- Focus mostly on infrastructure software, but many issues cross between
  - Reproducibility causes the most overlap

Purposes of software in research



12 scientific software challenges

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

- **All are tied together**

Challenge: software engineering

- What software engineering practices (from theory, from industry, etc.) work in science?
- Issues:
  - Significant student and “early stage researcher” labor
  - Prevalence of idiosyncratic architectures needing out-of-the-mainstream skills
  - Turnover (students graduate, staff are hired away)
  - Software development best practices (e.g. Agile) not well understood or not easily transferable to the scientific environment
- Subject of ongoing discussion/debate
  - 3 workshops on Software Engineering for High Performance Computing in Computational Science and Engineering at SC13-SC15
  - Some community gathering in Computational Science & Engineering (CSE) Software Forum – https://cse-software.org
  - Barry Boehm: Balancing Agility and Discipline

Challenge: portability

- How do we write software to work on many platforms and in many situations?
- Many different views of the science world, and corresponding software
  - Files vs. databases
  - HPC vs. cloud vs. web
  - CPU vs. GPU
  - Fortran vs. Python
  - Computational science vs. data science
- Each set of decisions leads to particular software choices
- Not optimal, but necessary? Can we do better?

Challenge: training and education

- How to use software to teach science?
  - Lecturers need to know what's available and what researchers actually use (see dissemination)
  - Developers need to consider spectrum from advanced researchers to novice when designing interfaces
- How to add software training to science/CS curriculum?
  - Formal?
    - May not fit – e.g. Chem PhD students take ~4 courses during their PhD work
    - Course are full of material – if software is added, what is removed?
  - Informal?
    - Software & Data Carpentry – a nice model, but only trains the people who know they need training

Challenge: incentives, citation/ credit models, and metrics (1)

- Hypothesis: gaining credit for software encourages people to produce and share it
- What to measure
  - How many downloads? (easiest to measure, least value)
  - How many contributors?
  - How many uses? (maybe the value/effort?)
  - How many papers cite it?
  - How many papers that cite it are cited? (hardest to measure, most value)
- Alternatively, can measure
  1. Downloads
  2. Installation/Build
  3. Use/Run
  4. Impact

Challenge: incentives, citation/ credit models, and metrics (2)

1. Downloads
   - From web/repository logs/stats
2. Installation/Build
   - Add “phone home” mechanism at build level
   - e.g., add ‘curl URL’ in makefile
3. Use/Run
   - 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

Challenge: incentives, citation/ credit models, and metrics (3)

4. Impact
   - Project CRediT (and Mozilla contributorship badges)
     - Record contributorship roles
   - Force 11 Software Citation Working Group (and merged WSSSPE software credit WG)
     - Define software citation principles
   - Codemeta
     - Minimal metadata schemas for science software
   - Lots of research projects (including my own university research on transitive credit - http://doi.org/10.5334/jors.by)
   - Altmetrics – not citations, but other structured measures of discussion (tweets, blogs, etc.)
   - ImpactStory
     - Measure research impact: reads, citations, tweets, etc.
   - Depsy (roughly ImpactStory specifically for software)
     - Measure software impact: downloads; software reuse (if one package is reused in another package), literature mentions (citations of a software package), and social mentions (tweets, etc.)

Challenge: intellectual property

- Producing software to be shared, even just for reproducibility, at a university means understanding university IP policies
- At many universities
  - Publications are a mark of scientific output and prestige
  - Software is something of potential monetary value
  - If someone is going to make money off work done by the university, the university should get a cut; this leads to policies that either make open source difficult or require non-commercial use
  - If there is no money to be made, the university is happy with open source, but mostly doesn't help to make it happen; either the creators must state that the work cannot be commercialized, or again, a license that forbids commercial use is required
- Situation is slowly changing as software is seen as a scientific output
- If a university grant/agreement says that open source software will be an product, the university will allow it even if they don't otherwise
- Note: the goal of scientific software projects should be science impact, not open source for its own sake

Challenge: publication and peer review (1)

- How should software be published?
- Software papers
  - Papers about software
  - Easily fits journal model
  - List: http://www.software.ac.uk/resources/guides/which-journals-should-i-publish-my-software
- Actual software
  - Claerbout via Donoho (1995): “an article about computational result is advertising, not scholarship. The actual scholarship is the full software environment, code and data, that produced the result.”
  - Code as a research object
    - Mozilla Science Lab project – https://www.mozillascience.org/code-as-a-research-object-a-new-project
    - GitHub/Zenodo linkage: Making your code citable – https://guides.github.com/activities/citable-code/
    - Archives a version of software, requires small amount of metadata, produces a DOI

Challenge: publication and peer review (2)

- How should software be peer-reviewed?
  - What to check? Code, environment, performance, tests, results?
  - Issues with hard-to-access resources (e.g., HPC, big data sets)
  - How many reviews are needed?
- Journals each have different ideas, no real convergence/standards yet
- Nothing in place yet for non-journal publications
  - Should review be done by science communities?
  - Or by publication sites? (e.g. Zenodo)

Challenge: software communities and sociology

- Goal: Engagement in software communities
- Engagement: meaningful and valuable actions that produce a measurable result
- Engagement = 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

Challenge: sustainability and funding models

- Sustainability: the capacity to endure
  - For software, sustainability means software will continue to be available in the future, on new platforms, meeting new needs
- Software often funded (explicitly as infrastructure or 'on the sly' in research) by 3-5 year grants
  - But many software projects are much longer
- In commercial software, more users means more revenue, more ability to support users, add new features, etc.
  - In research software, more users means more time needed to support users, less time to add features, etc.
- Researchers want to work with their peers, including those in other countries, but funding agencies are generally set up nationally and struggle to fund international collaborations
- Can we make funding decisions based on future impacts?
  - At least, can we make funding decisions based on metrics from the recent past?
- Need to examine new delivery/sustainability models
  - e.g., platforms, software as a service, VMs/containers

Challenge: career paths

- Are there appropriate career paths
  - for scientists who also develop some software?
  - for software developers?
- In both cases, university structure & academic culture rewards publications, not software
- For software developers, industry offers more financial rewards, cohorts (others with similar problems and solutions) and promotion opportunities
- Permanent postdocs are not happy
- Potential solutions
  - Research Software Engineer (RSE) track in UK (w/ EPSRC support)
  - University centers, e.g. TACC, NCSA, RENCI, Notre Dame CRC
  - Moore/Sloan Data Science program

Challenge: software dissemination, catalogs, search, and review (1)

- Should software be in a catalog?
- Advantages
  - Funders want to have a list of the products that they support, tied to the projects that they funded
  - Users want to know what software is likely to be around for a while (who funds it, how big is the user and developer community, how often is it updated, what others use it for, what they think, etc.)
  - Developers want others to be able to easily find (and then both use and cite) their software
- How
  - Need unique identifiers for each object
  - Need catalogs and indices to store identifiers, metadata, pointers to code
  - Need people/services/interfaces to make this work

Challenge: software dissemination, catalogs, search, and review (2)

- Examples:
  - Manually curated catalogs
    - NSF SI2 – http://bit.ly/sw-ci
    - DOE X-Stack – https://xstackwiki.modelado.org/X-Stack_Project_Products
    - Astronomy Source Code Library (ASCL) – http://ascl.net/
    - Can list anything, needs human effort
    - How to maintain catalogs over time is unclear; most catalogs seem to eventually stop being maintained, e.g. freecode
  - Semi-automatically curated catalogs
    - DARPA Open Catalog – http://opencatalog.darpa.mil
    - NASA – http://code.nasa.gov
    - Uses DOAP JSON file in GitHub repo, needs less human effort, still has maintenance issue, though smaller
  - hubZERO – Collaboration environment that includes software publishing
    - Uses DOIs for software
    - Can run code in hub environment, review, rate, …

Challenge: software dissemination, catalogs, search, and review (3)

- Internet catalogs have been replaced by index/search
- Is there an ideal solution for software?
- Maybe an Open Software Index
  - Like Google, but for software
- Based on DOIs of published software
- With sufficient metadata (see incentives, citation/credit models, and metrics)
- And using DOAP files
- But not just in one repository
- With a curation & peer review process (see publication and peer review)
- Maybe with flags/badges
  - Automated tests to avoid bit rot
  - For peer reviews
  - For user feedback
  - For funders

Challenge: multidisciplinary science

- Producing software requires skills in multiple disciplines: science subject area, computational/ data science, software engineering, computer science
- Different disciplines come with different practices, different jargons, different methods, different reward structures, different priorities
- Skills can exist in one or more person(s)
  - If more than one, traditional team challenges apply
    - And all members want to be partners, not clients
  - If one, add career path challenge
  - In both cases, academic departments generally reward work within silos, not across them

Challenge: reproducibility (1)

- Problems:
  - Scientific culture: reproducibility is important at a high level, but not in specific cases
    - Like Mark Twain's definition of classic books: those that people praise but don't read
    - No incentives or practices that translate the high level concept of reproducibility into actions that support actual reproducibility
  - Reproducibility can be hard due to a unique situation
    - Data can be taken with a unique instrument, transient data
    - Unique computer system was used
  - Given limited resources, reproducibility is less important than new research
    - A computer run that took months is unlikely to be repeated, because generating a new result is seen as a better use of the computing resources than reproducing the old result

Challenge: reproducibility (2)

- Solutions, starting with the easy parts
  - Require (some) reviewers of publications to actually reproduce the results
    - Examples: Transaction of Mathematical Software (TOMS) Replicated computational results (RCR) review process, artifact evaluation (AE) in recent software engineering conferences (http://www.artifact-eval.org)
  - Funders support reproducibility and require reproducibility statements in project final reports
  - Faculty require students to reproduce work of other students or work in papers
- Harder parts – unique situations
  - Can we isolate the unique parts and reproduce the parts around them?
  - Or use thought experiments?
- Think about benefits & costs, then decide

Challenges redux

- Still lots of challenges
- Most are somewhat tied together, as can be seen in the software cycle
- Ideally, progress in one helps in others, as long as all are considered
  - Some danger of making choices in one that hurt others
- Many groups and communities interested
  - See next slide
- Good progress underway for many challenges
- Overall, the future is bright

Active groups

- Public and private funding agencies
  - e.g. NSF, NIH, DOE, RCUK, Sloan, Moore, Helmsley, Wellcome
- Companies/Projects
  - e.g. GitHub, Zenodo, figshare, arXiv, DataCite, CrossRef, VIVO, ORCID
- Community practitioners, advocates, and researchers
  - e.g. WSSSPE, RDA, Force11
- Publishers/indexers
  - e.g. Scopus, Web of Science, Google?, MS?

NSF View: Software as Infrastructure

SSE, SSI, other NSF programs such as ABI

Create and maintain a software ecosystem providing new **capabilities** that advance and accelerate scientific inquiry at unprecedented complexity and scale

Most NSF programs, including CDS&E & XPS

Support the foundational **research** necessary to continue to efficiently advance scientific software

S2I2

Enable transformative, interdisciplinary, collaborative, **science and engineering** research and education through the use of advanced software and services

Transform practice through new policies for software, addressing challenges of academic culture, open dissemination and use, reproducibility and trust, curation, sustainability, governance, citation, stewardship, and attribution of software authorship

Partner with community, ACI & NSF programs & policies

Develop a next generation diverse workforce of scientists and engineers equipped with essential skills to use and develop software, with software and services used in both the research and **education** process

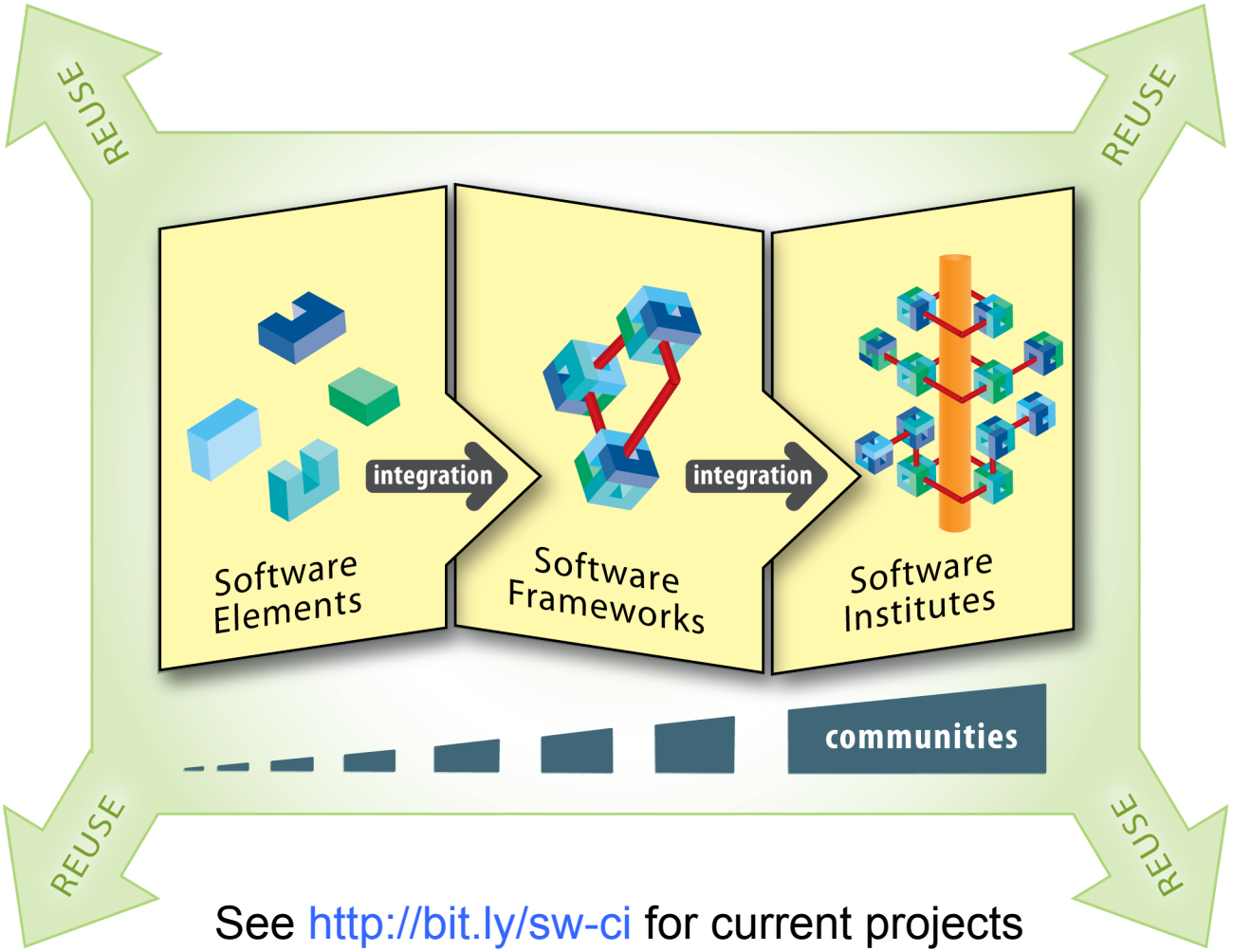
Partner with other ACI (particularly LWD) & NSF programs

NSF SI^2 program for software as infrastructure

6 rounds of funding, 80 SSEs

5 rounds of funding, 50 SSIs, current round under review

2 rounds of funding, 14 S^{2}I^2 conceptualizations, Institutes under review



See http://bit.ly/sw-ci for current projects

Resources

- More from me (http://danielskatz.org)
  - Blog: http://danielskatzblog.wordpress.com
  - Talks/slides: http://www.slideshare.net/danielskatz/presentations
- Community
  - WSSSPE: http://wssspe.researchcomputing.org.uk
  - UK Software Sustainability Institute: http://software.ac.uk
  - Force11: http://force11.org
  - Mozilla Science Lab: https://www.mozillascience.org
- Other events
  - Computational Science & Engineering Software Sustainability and Productivity Challenges (CSESSP Challenges): https://www.nitrd.gov/csessp/
  - Software and Data Citation Workshop Report: https://softwaredatacitation.org/

Acknowledgements

- Other current and former program officers in NSF SI2 program, particularly Gabrielle Allen and Rajiv Ramnath
- C. Titus Brown, Kyle Chard, Neil Chue Hong, Ian Foster, Melissa Haendel, Bill Miller, Arfon Smith

Final question

What will **you** do to make things better?

As a user, developer, member of your institution, and member of the scientific community

Thank you!

Daniel S. Katz
Division of Advanced Cyberinfrastructure (ACI)
dkatz@nsf.gov, d.katz@ieee.org, @danielskatz

Data Science Seminar
March 4, 2016, Indiana University