www.software.ac.uk

Software sustainability - lessons learned from different disciplines

https://doi.org/10.6084/m9.figshare.6935840

8th August 2018, Best Practices for HPC Software Developers Webinar Series

Neil Chue Hong (@npch), Software Sustainability Institute

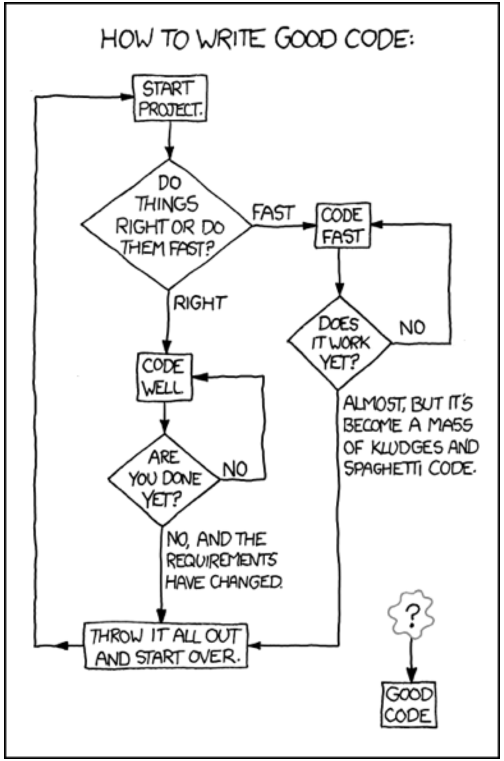
ORCID: 0000-0002-8876-7606 | N.ChueHong@software.ac.uk

Supported by BBSRC EPSRC Pioneering research ESRC Economic & Social Research Council

Project funding from Jisc NERC Science of the Environment

Best practice is HARD

www.software.ac.uk



https://xkcd.com/844/ – "Good Code" by Randall Munroe

- It's not easy to understand how to produce good software
- One size doesn't always fit all
- Best practice requires more than just one person "buying in" to become widely adopted

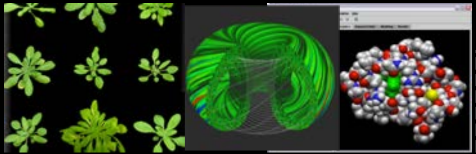
A national facility for cultivating better, more sustainable, research software to enable world-class research

- Software reaches boundaries in its development cycle that prevent improvement, growth and adoption
- Providing the expertise and services needed to negotiate to the next stage
- Developing the policy and tools to support the community developing and using research software

Supported by EPSRC Grant EP/H043160/1
+ EPSRC/ESRC/BBSRC grant EP/N006410/1

Software

Helping the community to develop software that meets the needs of reliable, reproducible, and reusable research



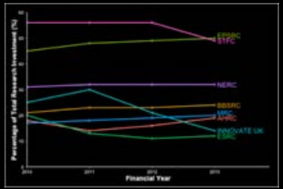
Training

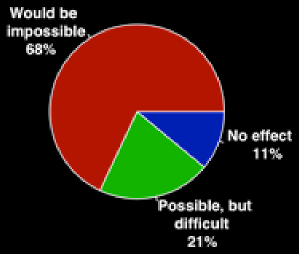
Delivering essential software skills to researchers via CDTs, institutions & doctoral schools

Outreach

Exploiting our platform to enable engagement, delivery & uptake

Collecting evidence on the community's software use & sharing with stakeholders





Policy

Bringing together the right people to understand and address topical issues

Community

About this talk

- Software is ubiquitous, fundamental and diverse
- Consequences of “incorrect” software can be large
- Is the issue a cultural one?
- How can we get adoption of best practices at scale?
- It’s impossible to do this on your own

No-one intentionally writes unsustainable software

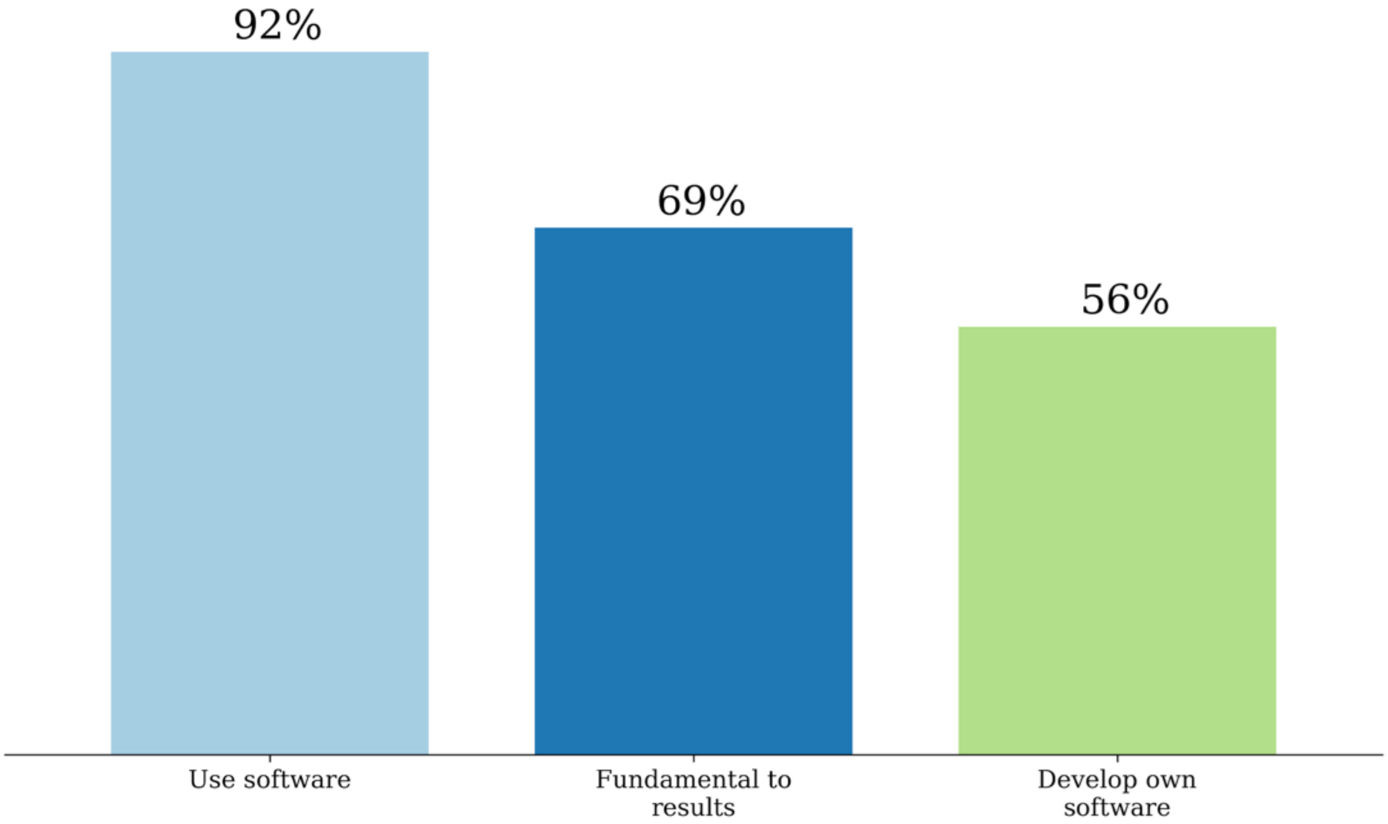
"Sustainable" software

- What is the definition of software sustainability?
  - *"will continue to be available in the future, on new platforms, meeting new needs"* - Katz
  - *"fulfils its intent over time"* – Lago
- Tension between fulfilling authors purpose vs others' potential needs

Use of software in research is ubiquitous, fundamental + diverse

UK software survey 2014

S.J. Hettrick et al,
UK Research Software Survey 2014
DOI:10.5281/zenodo.1183562.

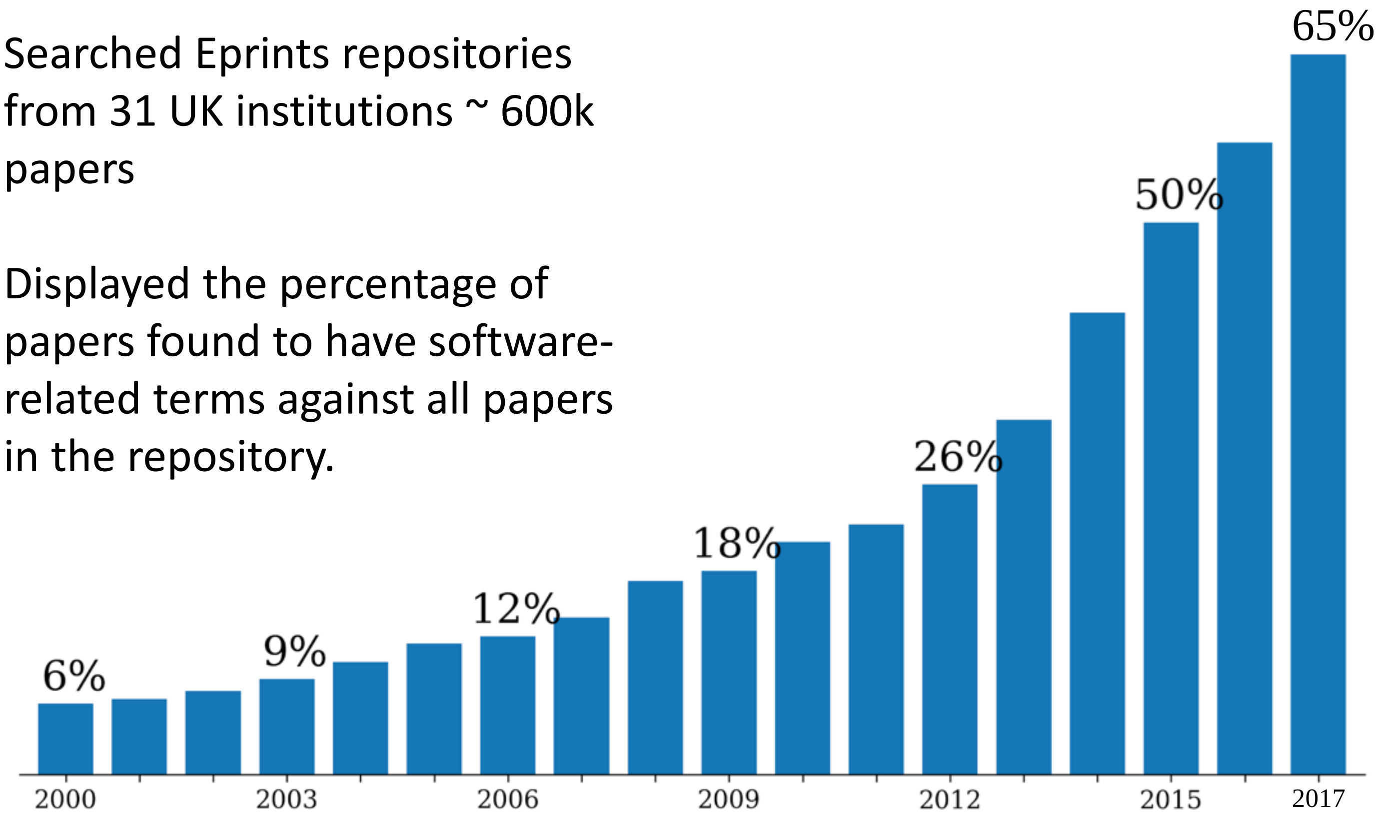


Software in research papers

www.software.ac.uk

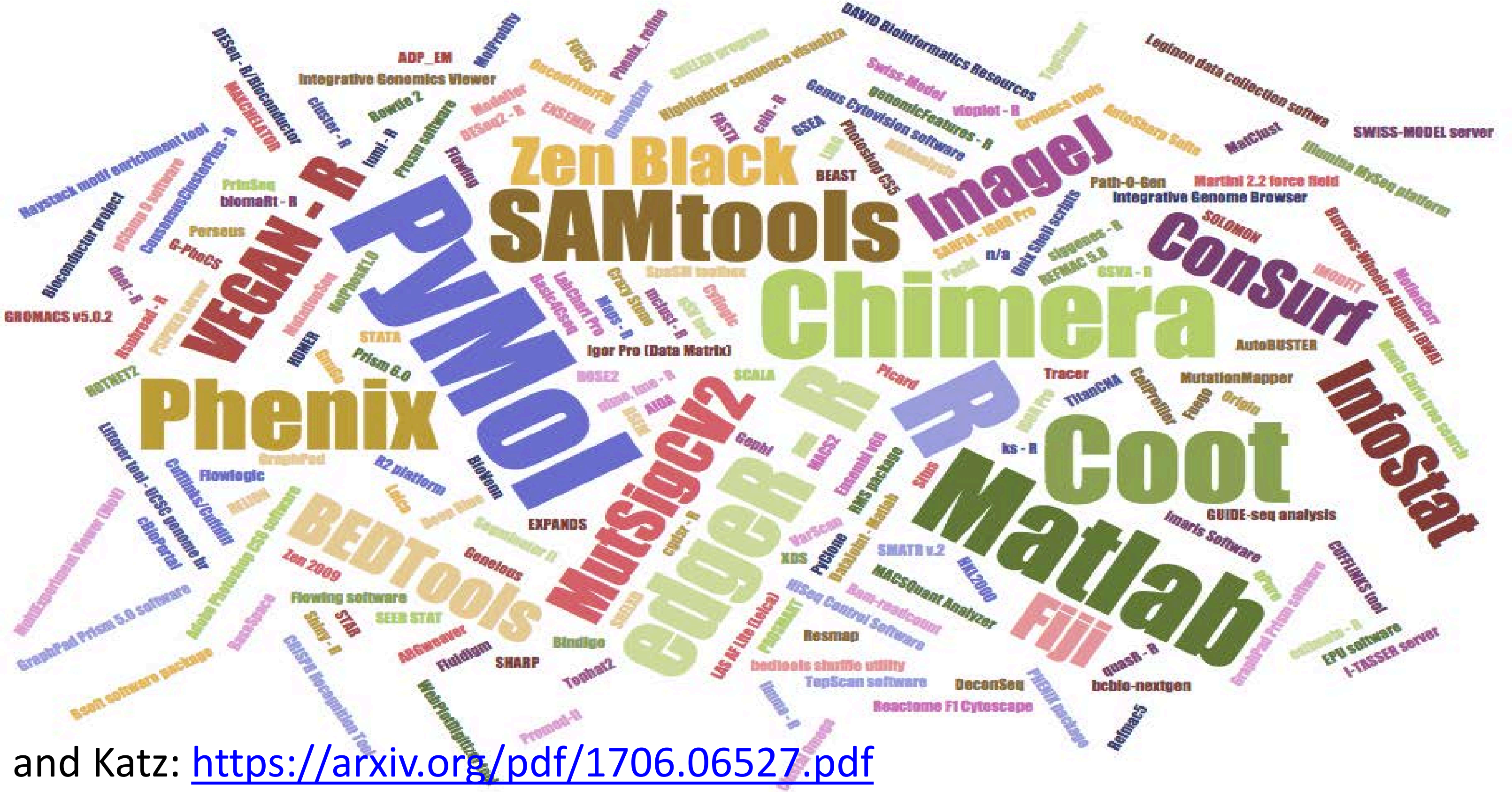
Searched Eprints repositories from 31 UK institutions ~ 600k papers

Displayed the percentage of papers found to have software-related terms against all papers in the repository.



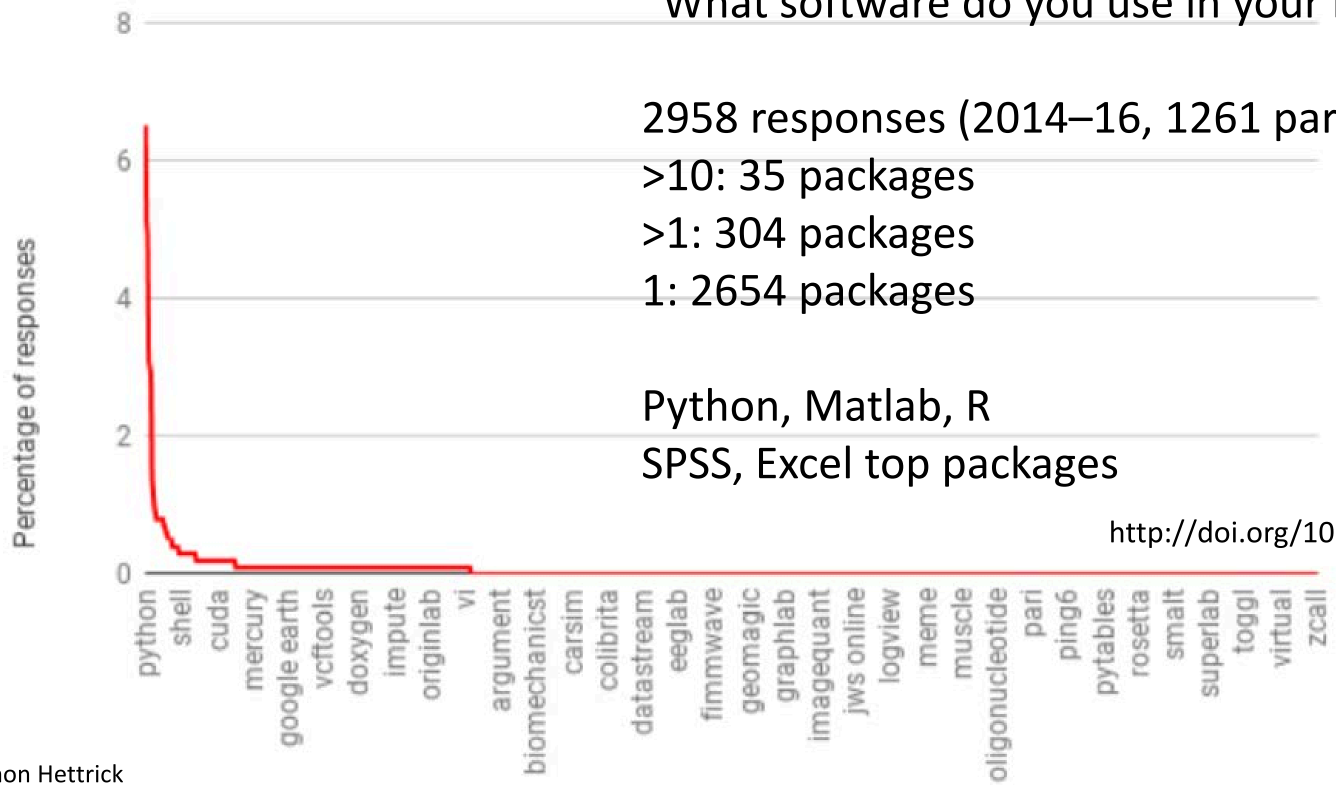
Software in nature

www.software.ac.uk



Nangia and Katz: https://arxiv.org/pdf/1706.06527.pdf

Long, long, long tail



“What software do you use in your research?”

2958 responses (2014–16, 1261 participants)
>10: 35 packages
>1: 304 packages
1: 2654 packages

Python, Matlab, R
SPSS, Excel top packages

http://doi.org/10.5281/zenodo.60276

Slide courtesy of Simon Hettrick

www.software.ac.uk

Software is important but often overlooked

Consequences of “incorrect” software can be large

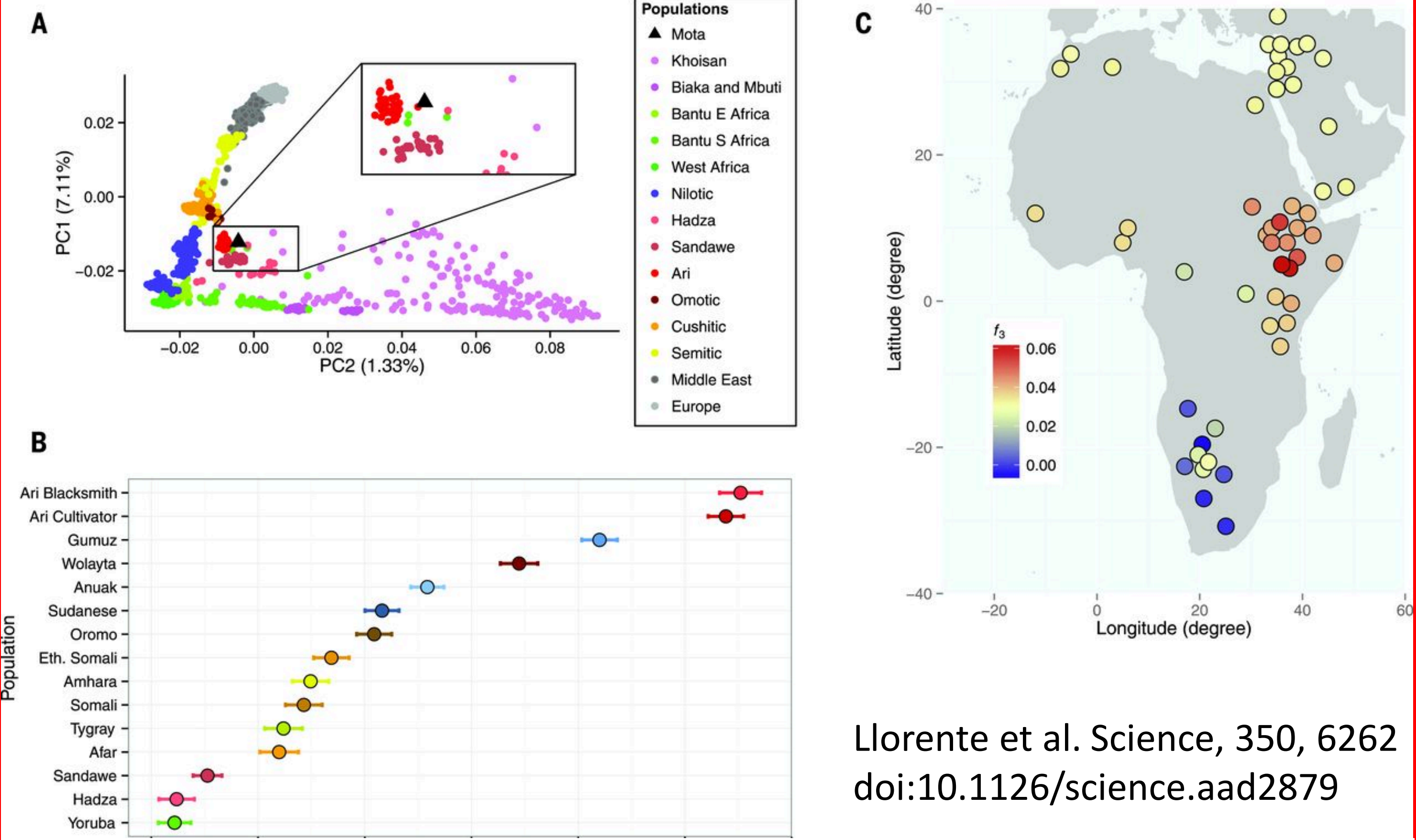
	B	C	I	J	K	L	M
2			Real GDP growth				
3			Debt/GDP				
4	Country	Coverage	30 or less	30 to 60	60 to 90	90 or above	30 or less
26			3.7	3.0	3.5	1.7	5.5
27	Minimum		1.6	0.3	1.3	-1.8	0.8
28	Maximum		5.4	4.9	10.2	3.6	13.3
29							
30	US	1946-2009	n.a.	3.4	3.3	-2.0	n.a.
31	UK	1946-2009	n.a.	2.4	2.5	2.4	n.a.
32	Sweden	1946-2009	3.6	2.9	2.7	n.a.	6.3
33	Spain	1946-2009	1.5	3.4	4.2	n.a.	9.9
34	Portugal	1952-2009	4.8	2.5	0.3	n.a.	7.9
35	New Zealand	1948-2009	2.5	2.9	3.9	-7.9	2.6
36	Netherlands	1956-2009	4.1	2.7	1.1	n.a.	6.4
37	Norway	1947-2009	3.4	5.1	n.a.	n.a.	5.4
38	Japan	1946-2009	7.0	4.0	1.0	0.7	7.0
39	Italy	1951-2009	5.4	2.1	1.8	1.0	5.6
40	Ireland	1948-2009	4.4	4.5	4.0	2.4	2.9
41	Greece	1970-2009	4.0	0.3	2.7	2.9	13.3
42	Germany	1946-2009	3.9	0.9	n.a.	n.a.	3.2
43	France	1949-2009	4.9	2.7	3.0	n.a.	5.2
44	Finland	1946-2009	3.8	2.4	5.5	n.a.	7.0
45	Denmark	1950-2009	3.5	1.7	2.4	n.a.	5.6
46	Canada	1951-2009	1.9	3.6	4.1	n.a.	2.2
47	Belgium	1947-2009	n.a.	4.2	3.1	2.6	n.a.
48	Austria	1948-2009	5.2	3.3	-3.8	n.a.	5.7
49	Australia	1951-2009	3.2	4.9	4.0	n.a.	5.9
50							
51			4.1	2.8	2.	=AVERAGE(L30:L44)	

Can you spot the mistake?

www.software.ac.uk

"All I can hope is that future historians note that one of the core empirical points providing the intellectual foundation for the global move to austerity in the early 2010s was based on someone accidentally not updating a row formula in Excel" – Mike Konczal

- Reinhart, Carmen M.; Rogoff, Kenneth S. (2010). "Growth in a Time of Debt". American Economic Review. 100 (2): 573–78. doi:10.1257/aer.100.2.573
- https://qz.com/75035/fixing-this-excel-error-transforms-high-debt-countries-from-recession-to-growth/
- http://www.nytimes.com/2013/04/26/opinion/debt-growth-and-the-austerity-debate.html
- https://www.bloomberg.com/news/articles/2013-04-18/faq-reinhart-rogoff-and-the-excel-error-that-changed-history



Llorente et al. Science, 350, 6262
doi:10.1126/science.aad2879

www.software.ac.uk

The results presented in the Report “Ancient Ethiopian genome reveals extensive Eurasian admixture throughout the African continent“ were affected by a bioinformatics error https://www.nature.com/news/error-found-in-study-of-first-ancient-african-genome-1.19258

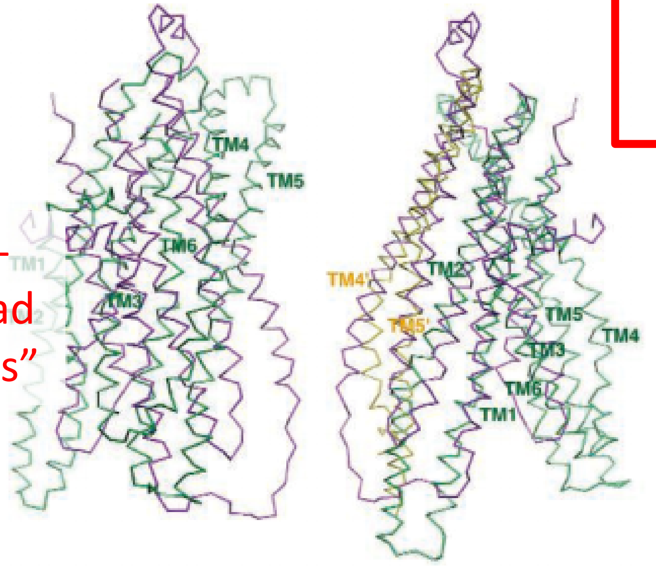
SCIENTIFIC PUBLISHING

A Scientist's Nightmare: Software Problem Leads to Five Retractions

Until recently, Geoffrey Chang's career was on a trajectory most young scientists only dream about. In 1999, at the age of 28, the protein crystallographer landed a faculty position at the prestigious Scripps Research Institute in San Diego, California. The next year, in a ceremony at the White House, Chang received a Presidential Early Career Award for Scientists and Engineers, the country's highest honor for young researchers. His lab generated a stream of high-profile papers detailing the molecular structures of important proteins embedded in cell membranes.

Then the dream turned into a nightmare. In September, Swiss researchers published a paper in *Nature* that cast serious doubt on a protein structure Chang's group had described in a 2001 *Science* paper. When he investigated, Chang was horrified to discover that a homemade data-analysis program had flipped two columns of data, inverting the electron-density map from which his team had derived the final protein structure. Unfortunately, his group had used the program to analyze data for

2001 *Science* paper, which described the structure of a protein called MsbA, isolated from the bacterium *Escherichia coli*. MsbA belongs to a huge and ancient family of molecules that use energy from adenosine triphosphate to transport molecules across cell membranes. These so-called ABC transporters perform many



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

Sciences and a 2005 *Science* paper, described EmrE, a different type of transporter protein.

Crystallizing and obtaining structures of five membrane proteins in just over 5 years was an incredible feat, says Chang's former postdoc adviser Douglas Rees of the California Institute of Technology in Pasadena. Such proteins are a challenge for crystallographers because they are large, unwieldy, and notoriously difficult to coax into the crystals needed for x-ray crystallography. Rees says determination was at the root of Chang's success: "He has an incredible drive and work ethic. He really pushed the field in the sense of getting things to crystallize that no one else had been able to do." Chang's data are good, Rees says, but the faulty software threw everything off.

Ironically, another former postdoc in Rees's lab, Kaspar Locher, exposed the mistake. In the 14 September issue of *Nature*, Locher, now at the Swiss Federal Institute of Technology in Zurich, described the structure of an ABC transporter called Sav1866 from *Staphylococcus aureus*. The structure was dramatically—and unexpectedly—different from that of MsbA. After pulling up Sav1866 and Chang's MsbA from *S. typhimurium* on a computer screen, Locher says he realized in minutes that the MsbA structure was inverted. Interpreting the "hand" of a molecule is always a challenge for crystallographers,

"Chang's data are good... but the faulty software threw everything off"

"a homemade data-analysis program had flipped two columns"

www.software.ac.uk

http://science.sciencemag.org/content/314/5807/1856.full

www.software.ac.uk

Mistakes in software erode trust in research and researchers

Is the issue a cultural one?

Sharing is key to reproducibility

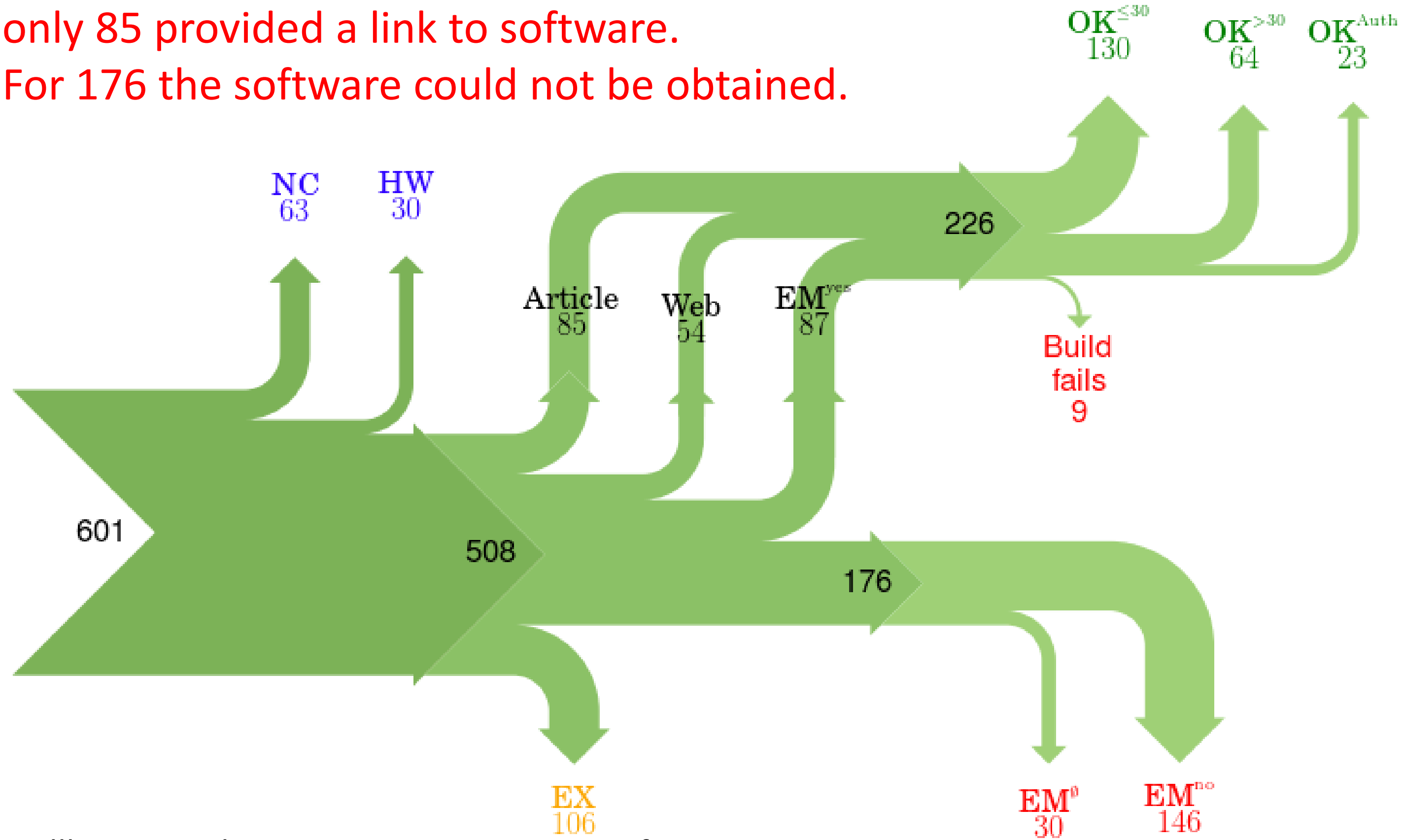
- Improves transparency
- Improves understanding
- Elimination of errors
- Encourages collaboration
- Easier on-ramping
- Improves trust

“Deep intellectual contributions now encoded only in software" – Stodden

“Scholarship is the full software environment, code and data, that produced the result” – Claerbout

FAIR Software?

Of 601 papers in ACM Computer Science journals and proceedings, only 85 provided a link to software.
For 176 the software could not be obtained.



Collberg, Proebsting, Warren, University of Arizona TR 14-04, 2015
http://reproducibility.cs.arizona.edu/v2/RepeatabilityTR.pdf

www.software.ac.uk

Culture change is hard

www.software.ac.uk

Science AAAS

In 2011 Science changed its editorial policies:

"We require that all computer code used for modeling and/or data analysis that is not commercially available be deposited in a publicly accessible repository upon publication."

"After publication, all reasonable requests for data, code, or materials must be fulfilled."

Stodden, Seiler, Ma. An empirical analysis of journal policy effectiveness for computational reproducibility
https://doi.org/10.1073/pnas.1708290115

Culture change is hard

www.software.ac.uk

Science AAAS

Table 1. Responses to emailed requests (n = 180)

Type of response	Count	Percent, %
Did not share data or code:		
Contact another person	20	11
Asked for reasons	20	11
Refusal to share	12	7
Directed back to supplement	6	3
Unfulfilled promise to follow up	5	3
Impossible to share	3	2
Shared data and code	65	36
Email bounced	3	2
No response	46	26

Stodden, Seiler, Ma. An empirical analysis of journal policy effectiveness for computational reproducibility
https://doi.org/10.1073/pnas.1708290115

Culture change is hard

Science
AAAS

"There appeared to be some confusion among authors, some of whom seemed to be unaware of Science's data and code sharing requirement. We can most easily demonstrate this with some anonymized author responses that highlight some of the barriers to sharing they perceived:"

Stodden, Seiler, Ma. An empirical analysis of journal policy effectiveness for computational reproducibility
https://doi.org/10.1073/pnas.1708290115

Culture change is hard

Science AAAS

"Normally we do not provide this kind of information to people we do not know. It might be that you want to check the data analysis, and that might be of some use to us, but only if you publish your findings while properly referring to us."

"Thank you for your interest in our paper. For the [redacted] calculations I used my own code, and there is no public version of this code, which could be downloaded. Since this code is not very user-friendly and is under constant development I prefer not to share this code."

"I have to say that this is a very unusual request without any explanation! Please ask your supervisor to send me an email with a detailed, and I mean detailed, explanation."

"When you approach a PI for the source codes and raw data, you better explain who you are, whom you work for, why you need the data and what you are going to do with it."

Stodden, Seiler, Ma. An empirical analysis of journal policy effectiveness for computational reproducibility
https://doi.org/10.1073/pnas.1708290115

“Perceived” importance

- “It has been said … that writing a large piece of software is akin to building infrastructure such as a telescope rather than a creditable scientific contribution…”
- “software development [is] often discounted in the scientific community, and programming is treated as **something to spend as little time on as possible**”
- “Serious scientists are **not expected to carefully test code**, let alone **document** it, in the same way they are trained to properly use other tools or document their experiments”
  - Stodden, V., Bailey, D. H., Borwein, J., LeVeque, R.J., Rider, W., Stein, W. “Setting the Default to Reproducible Reproducibility in Computational and Experimental Mathematics”, ICERM 2013, February 2013.

Barriers to Data and Code Sharing in Computational Science

Survey of Machine Learning Community, NIPS (Stodden, 2010):

Code		Data
77%	Time to document and clean up	54%
52%	Dealing with questions from users	34%
44%	Not receiving attribution	42%
40%	Possibility of patents	-
34%	Legal Barriers (ie. copyright)	41%
-	Time to verify release with admin	38%
30%	Potential loss of future publications	35%
30%	Competitors may get an advantage	33%
20%	Web/disk space limitations	29%

Victoria Stodden, AMP 2011 http://www.stodden.net/AMP2011/, Special Issue Reproducible Research Computing in Science and Engineering July/August 2012, 14(4)

Howison and Herbsleb (2013) "Incentives and Integration In Scientific Software Production" CSCW 2013.

It’s still all about reputation

“This particular project was something I wrote a couple years ago to help me out with a workflow… I’d put it up on Github, so that others could potentially use it or use the code. So I went to see what people were saying about this project. It seemed liked I’d done something fundamentally wrong, so stupid that it flabbergasts someone... So of course I start sobbing. Then I see these people’s follower count, and I sob harder. I can’t help but think of potential future employers that are no longer potential.”

http://www.software.ac.uk/blog/2013-01-25-haters-gonna-hate-why-you-shouldnt-be-ashamed-releasing-your-code

One of the biggest challenges is educating our peers

How can we get adoption of best practices at scale?

OPEN ACCESS Freely available online

PLOS | BIOLOGY

Community Page

Best Practices for Scientific Computing

Greg Wilson[1]*, D. A. Aruliah[2], C. Titus Brown[3], Neil P. Chue Hong[4], Matt Davis[5], Richard T. Guy[6¤], Steven H. D. Haddock[7], Kathryn D. Huff[8], Ian M. Mitchell[9], Mark D. Plumbley[10], Ben Waugh[11], Ethan P. White[12], Paul Wilson[13]

1 Mozilla Foundation, Toronto, Ontario, Canada, 2 University of Ontario Institute of Technology, Oshawa, Ontario, Canada, 3 Michigan State University, East Lansing, Michigan, United States of America, 4 Software Sustainability Institute, Edinburgh, United Kingdom, 5 Space Telescope Science Institute, Baltimore, Maryland, United States of America, 6 University of Toronto, Toronto, Ontario, Canada, 7 Monterey Bay Aquarium Research Institute, Moss Landing, California, United States of America, 8 University of California Berkeley, Berkeley, California, United States of America, 9 University of British Columbia, Vancouver, British Columbia, Canada, 10 Queen Mary University of London, London, United Kingdom, 11 University College London, London, United Kingdom, 12 Utah State University, Logan, Utah, United States of America, 13 University of Wisconsin, Madison, Wisconsin, United States of America

Introduction

Scientists spend an increasing amount of time building and using software. However, most scientists are never taught how to do this efficiently. As a result, many are unaware of tools and practices that would allow them to write more reliable and maintainable code with less effort. We describe a set of best practices for scientific software development that have solid foundations in research and experience, and that improve scientists' productivity and the reliability of their software.

Software is as important to modern scientific research as telescopes and test tubes. From groups that work exclusively on computational problems, to traditional laboratory and field scientists, more and more of the daily operation of science revolves around developing new algorithms, managing and analyzing the large amounts of data that are generated in single research projects, combining disparate datasets to assess synthetic problems, and other computational tasks.

Scientists typically develop their own software for these purposes because doing so requires substantial domain-specific knowledge. As a result, recent studies have found that scientists typically spend 30% or more of their time developing software [1,2]. However, 90% or more of them are primarily self-taught [1,2], and therefore lack exposure to basic software development practices such as writing maintainable code, using version control and issue trackers, code reviews, unit testing, and task automation.

We believe that software is just another kind of experimental apparatus [3] and should be built, checked, and used as carefully as any physical apparatus. However, while most scientists are careful to validate their laboratory and field equipment, most do not know how reliable their software is [4,5]. This can lead to serious errors impacting the central conclusions of published research [6]: recent high-profile retractions, technical comments, and corrections because of errors in computational methods include papers in *Science* [7,8], *PNAS* [9], the *Journal of Molecular Biology* [10], *Ecology Letters* [11,12], the *Journal of Mammalogy* [13], *Journal of the American College of Cardiology* [14], *Hypertension* [15], and *The American Economic Review* [16].

In addition, because software is often used for more than a single project, and is often reused by other scientists, computing errors can have disproportionate impacts on the scientific process. This type of cascading impact caused several prominent retractions when an error from another group's code was not discovered until after publication [6]. As with bench experiments, not everything must be done to the most exacting standards; however, scientists need to be aware of best practices both to improve their own approaches and for reviewing computational work by others.

This paper describes a set of practices that are easy to adopt and have proven effective in many research settings. Our recommendations are based on several decades of collective experience both building scientific software and teaching computing to scientists [17,18], reports from many other groups [19–25], guidelines for commercial and open source software development [26,27], and on empirical studies of scientific computing [28–31] and software development in general (summarized in [32]). None of these practices will guarantee efficient, error-free software development, but used in concert they will reduce the number of errors in scientific software, make it easier to reuse, and save the authors of the software time and effort that can used for focusing on the underlying scientific questions.

Our practices are summarized in Box 1; labels in the main text such as "(1a)" refer to items in that summary. For reasons of space, we do not discuss the equally important (but independent) issues of reproducible research, publication and citation of code and data, and open science. We do believe, however, that all of these will be much easier to implement if scientists have the skills we describe.

The Community Page is a forum for organizations and societies to highlight their efforts to enhance the dissemination and value of scientific knowledge.

Citation: Wilson G, Aruliah DA, Brown CT, Chue Hong NP, Davis M, et al. (2014) Best Practices for Scientific Computing. PLoS Biol 12(1): e1001745. doi:10.1371/journal.pbio.1001745

Academic Editor: Jonathan A. Eisen, University of California Davis, United States of America

Published January 7, 2014

Funding: Neil Chue Hong was supported by the UK Engineering and Physical Sciences Research Council (EPSRC) Grant EP/H043160/1 for the UK Software Sustainability Institute. Ian M. Mitchell was supported by NSERC Discovery Grant #298211. Mark Plumbley was supported by EPSRC through a Leadership Fellowship (EP/G007144/1) and a grant (EP/H043101/1) for SoundSoftware.ac.uk. Ethan White was supported by a CAREER grant from the US National Science Foundation (DEB 0953694). Greg Wilson was supported by a grant from the Sloan Foundation. The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

Competing Interests: The lead author (GVW) is involved in a pilot study of code review in scientific computing with *PLOS Computational Biology*.

* E-mail: gvwilson@software-carpentry.org

¤ **Current address:** Microsoft, Inc., Seattle, Washington, United States of America

Box 1. Summary of Best Practices

1. Write programs for people, not computers.
   (a) A program should not require its readers to hold more than a handful of facts in memory at once.
   (b) Make names consistent, distinctive, and meaningful.
   (c) Make code style and formatting consistent.
2. Let the computer do the work.
   (a) Make the computer repeat tasks.
   (b) Save recent commands in a file for re-use.
   (c) Use a build tool to automate workflows.
3. Make incremental changes.
   (a) Work in small steps with frequent feedback and course correction.
   (b) Use a version control system.
   (c) Put everything that has been created manually in version control.
4. Don't repeat yourself (or others).
   (a) Every piece of data must have a single authoritative representation in the system.
   (b) Modularize code rather than copying and pasting.
   (c) Re-use code instead of rewriting it.
5. Plan for mistakes.
   (a) Add assertions to programs to check their operation.
   (b) Use an off-the-shelf unit testing library.
   (c) Turn bugs into test cases.
   (d) Use a symbolic debugger.
6. Optimize software only after it works correctly.
   (a) Use a profiler to identify bottlenecks.
   (b) Write code in the highest-level language possible.
7. Document design and purpose, not mechanics.
   (a) Document interfaces and reasons, not implementations.
   (b) Refactor code in preference to explaining how it works.
   (c) Embed the documentation for a piece of software in that software.
8. Collaborate.
   (a) Use pre-merge code reviews.
   (b) Use pair programming when bringing someone new up to speed and when tackling particularly tricky problems.
   (c) Use an issue tracking tool.

Write Programs for People, Not Computers

Scientists writing software need to write code that both executes correctly and can be easily read and understood by other programmers (especially the author's future self). If software cannot be easily read and understood, it is much more difficult to know that it is actually doing what it is intended to do. To be productive, software developers must therefore take several aspects of human cognition into account: in particular, that human working memory is limited, human pattern matching abilities are finely tuned, and human attention span is short [33–37].

First, ***a program should not require its readers to hold more than a handful of facts in memory at once (1a)***. Human working memory can hold only a handful of items at a time, where each item is either a single fact or a "chunk" aggregating several facts [33,34], so programs should limit the total number of items to be remembered to accomplish a task. The primary way to accomplish this is to break programs up into easily understood functions, each of which conducts a single, easily understood, task. This serves to make each piece of the program easier to understand in the same way that breaking up a scientific paper using sections and paragraphs makes it easier to read.

Second, scientists should ***make names consistent, distinctive, and meaningful (1b)***. For example, using non-descriptive names, like a and foo, or names that are very similar, like results and results2, is likely to cause confusion.

Third, scientists should ***make code style and formatting consistent (1c)***. If different parts of a scientific paper used different formatting and capitalization, it would make that paper more difficult to read. Likewise, if different parts of a program are indented differently, or if programmers mix CamelCaseNaming and pothole_case_naming, code takes longer to read and readers make more mistakes [35,36].

Let the Computer Do the Work

Science often involves repetition of computational tasks such as processing large numbers of data files in the same way or regenerating figures each time new data are added to an existing analysis. Computers were invented to do these kinds of repetitive tasks but, even today, many scientists type the same commands in over and over again or click the same buttons repeatedly [17]. In addition to wasting time, sooner or later even the most careful researcher will lose focus while doing this and make mistakes.

Scientists should therefore ***make the computer repeat tasks (2a)*** and ***save recent commands in a file for re-use (2b)***. For example, most command-line tools have a "history" option that lets users display and re-execute recent commands, with minor edits to filenames or parameters. This is often cited as one reason command-line interfaces remain popular [38,39]: "do this again" saves time and reduces errors.

A file containing commands for an interactive system is often called a *script*, though there is real no difference between this and a program. When these scripts are repeatedly used in the same way, or in combination, a workflow management tool can be used. The paradigmatic example is compiling and linking programs in languages such as Fortran, C++, Java, and C# [40]. The most widely used tool for this task is probably Make (http://www.gnu.org/software/make), although many alternatives are now available [41]. All of these allow people to express dependencies between files, i.e., to say that if A or B has changed, then C needs to be updated using a specific set of commands. These tools have been successfully adopted for scientific workflows as well [42].

To avoid errors and inefficiencies from repeating commands manually, we recommend that scientists ***use a build tool to automate workflows (2c)***, e.g., specify the ways in which intermediate data files and final results depend on each other, and on the programs that create them, so that a single command will regenerate anything that needs to be regenerated.

Best Practices for Scientific Computing: https://doi.org/10.1371/journal.pbio.1001745

PERSPECTIVE

Good enough practices in scientific computing

Greg Wilson[1☯ *], **Jennifer Bryan**[2☯], **Karen Cranston**[3☯], **Justin Kitzes**[4☯], **Lex Nederbragt**[5☯], **Tracy K. Teal**[6☯]

1 Software Carpentry Foundation, Austin, Texas, United States of America, 2 RStudio and Department of Statistics, University of British Columbia, Vancouver, British Columbia, Canada, 3 Department of Biology, Duke University, Durham, North Carolina, United States of America, 4 Energy and Resources Group, University of California, Berkeley, Berkeley, California, United States of America, 5 Centre for Ecological and Evolutionary Synthesis, University of Oslo, Oslo, Norway, 6 Data Carpentry, Davis, California, United States of America

☯ These authors contributed equally to this work.
* gvwilson@software-carpentry.org

Author summary

Computers are now essential in all branches of science, but most researchers are never taught the equivalent of basic lab skills for research computing. As a result, data can get lost, analyses can take much longer than necessary, and researchers are limited in how effectively they can work with software and data. Computing workflows need to follow the same practices as lab projects and notebooks, with organized data, documented steps, and the project structured for reproducibility, but researchers new to computing often don't know where to start. This paper presents a set of good computing practices that every researcher can adopt, regardless of their current level of computational skill. These practices, which encompass data management, programming, collaborating with colleagues, organizing projects, tracking work, and writing manuscripts, are drawn from a wide variety of published sources from our daily lives and from our work with volunteer organizations that have delivered workshops to over 11,000 people since 2010.

Overview

We present a set of computing tools and techniques that every researcher can and should consider adopting. These recommendations synthesize inspiration from our own work, from the experiences of the thousands of people who have taken part in Software Carpentry and Data Carpentry workshops over the past 6 years, and from a variety of other guides. Our recommendations are aimed specifically at people who are new to research computing.

Box 1: Summary of Practices

1. Data Management
   a) Save the raw data.
   b) Create the data you wish to see in the world.
   c) Create analysis-friendly data.
   d) Record all the steps used to process data.
   e) Anticipate the need to use multiple tables.
   f) Submit data to a reputable DOI-issuing repository so that others can access and cite it.
2. Software
   a) Place a brief explanatory comment at the start of every program.
   b) Decompose programs into functions.
   c) Be ruthless about eliminating duplication.
   d) Always search for well-maintained software libraries that do what you need.
   e) Test libraries before relying on them.
   f) Give functions and variables meaningful names.
   g) Make dependencies and requirements explicit.
   h) Do not comment and uncomment sections of code to control a program's behavior.
   i) Provide a simple example or test data set.
   j) Submit code to a reputable DOI-issuing repository.
3. Collaboration
   a) Create an overview of your project.
   b) Create a shared public "to-do" list.
   c) Make the license explicit.
   d) Make the project citable.
4. Project Organization
   a) Put each project in its own directory, which is named after the project.
   b) Put text documents associated with the project in the `doc` directory.
   c) Put raw data and metadata in a `data` directory, and files generated during cleanup and analysis in a `results` directory.
   d) Put project source code in the `src` directory.
   e) Put external scripts, or compiled programs in the `bin` directory.
   f) Name all files to reflect their content or function.
5. Keeping Track of Changes
   a) Back up (almost) everything created by a human being as soon as it is created.
   b) Keep changes small.
   c) Share changes frequently.
   d) Create, maintain, and use a checklist for saving and sharing changes to the project.
   e) Store each project in a folder that is mirrored off the researcher's working machine.
   f) Use a file called `CHANGELOG.txt` to record changes, and
   g) Copy the entire project whenever a significant change has been made, OR
   h) Use a version control system to manage changes
6. Manuscripts
   a) Write manuscripts using online tools with rich formatting, change tracking, and reference management, OR
   b) Write the manuscript in a plain text format that permits version control

www.software.ac.uk

Good Enough Practices in Scientific Computing: https://doi.org/10.1371/journal.pcbi.1005510

Foundational skills for researchers

www.software.ac.uk

THE CARPENTRIES

software carpentry

Basic lab skills for scientific computing; researchers can do more in less time and with less pain.

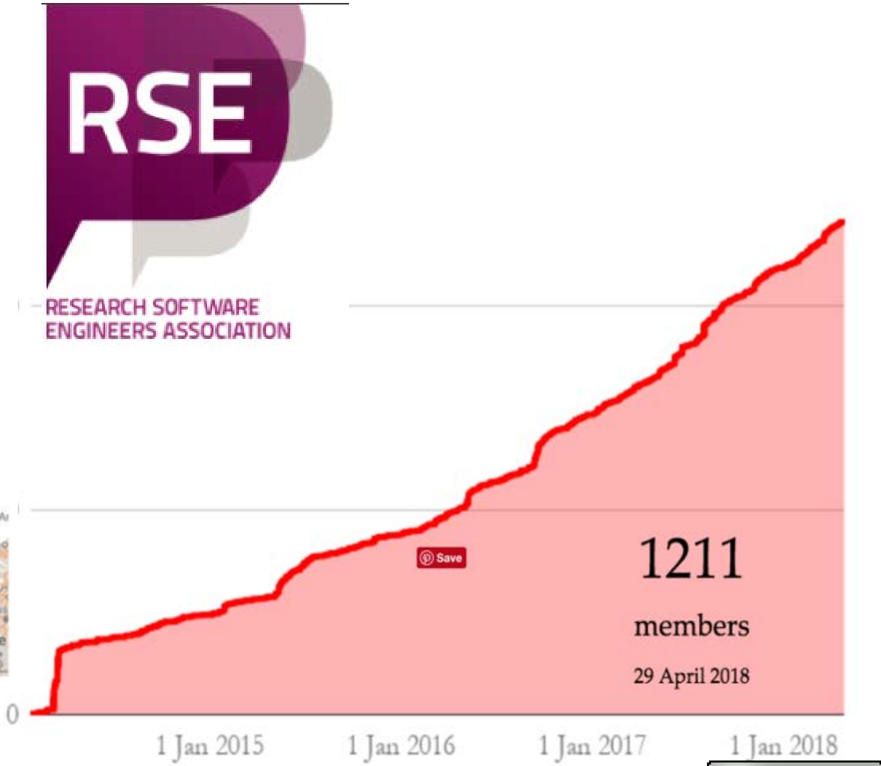
DATA CARPENTRY

Basic concepts, skills and tools for working more effectively with data.

Open source learning, "Train the trainers"

Growth of RSE Community

www.software.ac.uk



Slide courtesy of Simon Hettrick

RSEs are worldwide

- Wants to work in a research environment
- Wants to advance research
- Wants to develop software

Country	Gender	PhDs	Background	Reason to work
Canada	n/a	45%	IT	n/a
Germany	83% male	48%	Physics	Research environment
Netherlands	63% male	56%	Comp. sci.	N/A
UK	84% male	67%	Comp. sci./physics	Research environment
USA	82% male	60%	Comp. sci.	Advance research
South Africa	92% male	68%	Physics	Research environment

github.com/softwaresaved/international-survey/tree/master/analysis

Guides are popular

- Software Evaluation Guide (over 65k unique visits)
- Choosing a repository for your software project (over 50k unique visits)
- How to cite and describe software (over 25k unique visits)
- Developing maintainable software (over 25k unique visits)
- In which journals should I publish my software (over 22k unique visits)

Online sustainability evaluation

The following evaluation is a short, free, online version of the full sustainability evaluation that the Institute can perform for your project.

It takes about 15 minutes to complete the questionnaire, which gives you the opportunity to review the main issues that affect the sustainability of your software. At the end of the evaluation, a report will be generated and emailed to you with sustainability advice that is tailored to your project.

All questions are mandatory and need to be completed before you can progress through the evaluation.

But on their own, they don't ensure adoption of good practice

https://doi.org/10.6084/m9.figshare.5331703.v2

Carina Haupt and Tobias Schlauch

German Aerospace Center (DLR)

Numbers

- More than 8000 employees
- ~20% of DLR employees involved in software development

→ DLR is one of the biggest „software houses“ in Germany

Characteristics

- Variety of
  - Fields
  - Maturity
  - Software technologies
  - Team sizes
- "Developers" often do not have any training in software development

Goal: Improve sustainability and quality of software products

How to teach them software engineering?

DLR

https://doi.org/10.6084/m9.figshare.5331703.v2

Carina Haupt and Tobias Schlauch

Software Engineering Initiative of DLR

Software Engineering Initiative of DLR

Guidelines	Trainings	Knowledge Provision	Collaboration	Experience Exchange

https://doi.org/10.6084/m9.figshare.5331703.v2

Carina Haupt and Tobias Schlauch

Software Engineering Guidelines

Guidelines support developers to self-assess their software concerning good development practices.

- Joint development with focus on **good practices**, **tools**, and **essential documentation**
- **Three maturity level** available as **checklists in different formats** to ease practical usage

Checklists for different maturity levels

Change Management

Recommendation	Comment	Status
EÄM.2: The most important information describing how to contribute to development are stored in a central location. *(from application class 1)*	Build steps are missing	todo
EÄM.5: Known bugs, important unresolved tasks and ideas are at least noted in bullet point form and stored centrally. *(from application class 1)*		ok
EÄM.7: A repository is set up in a version control system. The repository is adequately structured and ideally contains all artifacts for building a usable software version and for testing it. *(from application class 1)*		ok
EÄM.8: Every change of the repository ideally serves a specific purpose, contains an understandable description and leaves the software in a consistent, working state. *(from application class 1)*		ok

Reasoning and further advice

The repository is the central entry point for development. All main artifacts are stored in a safe way and are available at a single location. Each change is comprehensible and can be traced back to the originator. In addition, the version control system ensures the consistency of all changes.

The repository directory structure should be aligned with established conventions. References are usually the version control system, the build tool (see the Automation and Dependency Management section) or the community of the used programming language or framework. Two examples:

Community standards

- ESIP (Earth Sciences): https://esipfed.github.io/Software-Assessment-Guidelines/
- CLARIAH (Arts and Humanities): https://github.com/CLARIAH/software-quality-guidelines
- IPOL (Image Processing): https://tools.ipol.im/wiki/ref/software_guidelines/
- ELIXIR (Life Sciences): https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5490478/

Software Development Best Practices for Life Sciences

www.software.ac.uk

- Goal:
  - Define procedures to improve quality and sustainability of software development that could be adopted by ELIXIR and other biomedical Research Infrastructures
- Series of workshops (lightning talks, facilitated sessions)
  - Agree practices
  - Build community
  - Create policy
  - Develop guidance
- Outputs
  - "Four simple recommendations" paper
  - "Top 10 Metrics" paper
  - Training course (in development)
  - Endorsement by community
  - Repo: https://github.com/SoftDev4Research/

1. Develop publicly accessible open source code from day one
2. Make software easy to discover by providing software metadata via a popular community registry
3. Adopt a license and comply with the licence of third-party dependencies
4. Have a clear and transparent contribution, governance and communication processes

Research Software Workflow

www.software.ac.uk

describe

GitLab
GitHub
Bitbucket

<sword />
zenodo
DRYAD
figshare
credit for all your research

develop → share → preserve

Developed and versioned using code repository

Published via code repository or website

Deposited in digital repository with paper / for preservation
Made citable

LIGO Example

www.software.ac.uk

PRL **116,** 061102 (2016) Selected for a Viewpoint in *Physics* PHYSICAL REVIEW LETTERS week ending 12 FEBRUARY 2016

Observation of Gravitational Waves from a Binary Black Hole Merger

B. P. Abbott *et al.**
(LIGO Scientific Collaboration and Virgo Collaboration)
(Received 21 January 2016; published 11 February 2016)

On September 14, 2015 at 09:50:45 UTC the two detectors of the Laser Interferometer Gravitational-Wave Observatory simultaneously observed a transient gravitational-wave signal. The signal sweeps upwards in frequency from 35 to 250 Hz with a peak gravitational-wave strain of 1.0×10^{-21}. It matches the waveform predicted by general relativity for the inspiral and merger of a pair of black holes and the ringdown of the resulting single black hole. The signal was observed with a matched-filter signal-to-noise ratio of 24 and a false alarm rate estimated to be less than 1 event per 203 000 years, equivalent to a significance greater than 5.1σ. The source lies at a luminosity distance of 410^{+160}_{-180} Mpc corresponding to a redshift $z = 0.09^{+0.03}_{-0.04}$. In the source frame, the initial black hole masses are $36^{+5}_{-4} M_\odot$ and $29^{+4}_{-4} M_\odot$, and the final black hole mass is $62^{+4}_{-4} M_\odot$, with $3.0^{+0.5}_{-0.5} M_\odot c^2$ radiated in gravitational waves. All uncertainties define 90% credible intervals. These observations demonstrate the existence of binary stellar-mass black hole systems. This is the first direct detection of gravitational waves and the first observation of a binary black hole merger.

DOI: 10.1103/PhysRevLett.116.061102

I. INTRODUCTION

In 1916, the year after the final formulation of the field equations of general relativity, Albert Einstein predicted the existence of gravitational waves. He found that the linearized weak-field equations had wave solutions: transverse waves of spatial strain that travel at the speed of light, generated by time variations of the mass quadrupole

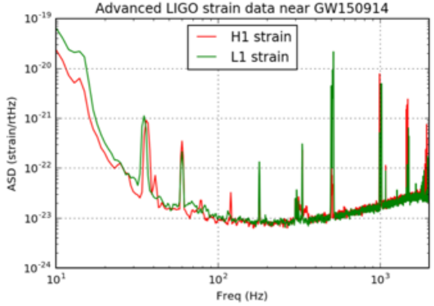
The discovery of the binary pulsar system PSR B1913+16 by Hulse and Taylor [20] and subsequent observations of its energy loss by Taylor and Weisberg [21] demonstrated the existence of gravitational waves. This discovery, along with emerging astrophysical understanding [22], led to the recognition that direct observations of the amplitude and phase of gravitational waves would enable

There's a signal in these data! For the moment, let's ignore that, and assume it's all noise.

In [7]:
```python
# number of sample for the fast fourier transform:
NFFT = 1*fs
fmin = 10
fmax = 2000
Pxx_H1, freqs = mlab.psd(strain_H1, Fs = fs, NFFT = NFFT)
Pxx_L1, freqs = mlab.psd(strain_L1, Fs = fs, NFFT = NFFT)

# We will use interpolations of the ASDs computed above for whitening:
psd_H1 = interp1d(freqs, Pxx_H1)
psd_L1 = interp1d(freqs, Pxx_L1)

# plot the ASDs:
plt.figure()
plt.loglog(freqs, np.sqrt(Pxx_H1),'r',label='H1 strain')
plt.loglog(freqs, np.sqrt(Pxx_L1),'g',label='L1 strain')
plt.axis([fmin, fmax, 1e-24, 1e-19])
plt.grid('on')
plt.ylabel('ASD (strain/rtHz)')
plt.xlabel('Freq (Hz)')
plt.legend(loc='upper center')
plt.title('Advanced LIGO strain data near GW150914')
plt.savefig('GW150914_ASDs.png')
```



NOTE that we only plot the data between fmin = 10 Hz and fmax = 2000 Hz.

Adoption of best practice comes from making it easy and useful for everyone

www.software.ac.uk

www.software.ac.uk

It’s impossible to do this on your own

Communities of Practice

- Domain: A domain of knowledge creates common ground, inspires members to participate, guides their learning and gives meaning to their actions
- Community: The notion of a community creates the social fabric for that learning. A strong community fosters interactions and encourages a willingness to share ideas.
- Practice: While the domain provides the general area of interest for the community, the practice is the specific focus around which the community develops, shares and maintains its core of knowledge.

Wenger, Etienne; McDermott, Richard; Snyder, William M. (2002). Cultivating Communities of Practice (Hardcover). Harvard Business Press; 1 edition. ISBN 978-1-57851-330-7.

Cultivating successful CoPs

- Design the community to evolve naturally
- Create opportunities for open dialog within and with outside perspectives
- Welcome and allow different levels of participation
- Develop both public and private community spaces
- Focus on the value of the community
- Combine familiarity and excitement
- Find and nurture a regular rhythm for the community

Wenger, Etienne; McDermott, Richard; Snyder, William M. (2002). Cultivating Communities of Practice (Hardcover). Harvard Business Press; 1 edition. ISBN 978-1-57851-330-7.

Examples of CoPs

www.software.ac.uk

http://melbourne.resbaz.edu.au/HackyHour

https://ourcodingclub.github.io/

Sheffield Astrophysics Code Review Club
https://www.software.ac.uk/index.php/blog/2018-05-18-code-review-academia

WORKING TOWARDS SUSTAINABLE SOFTWARE FOR SCIENCE: PRACTICE AND EXPERIENCES

mozilla Science Lab

THE CARPENTRIES
https://cookbook.carpentries.org/

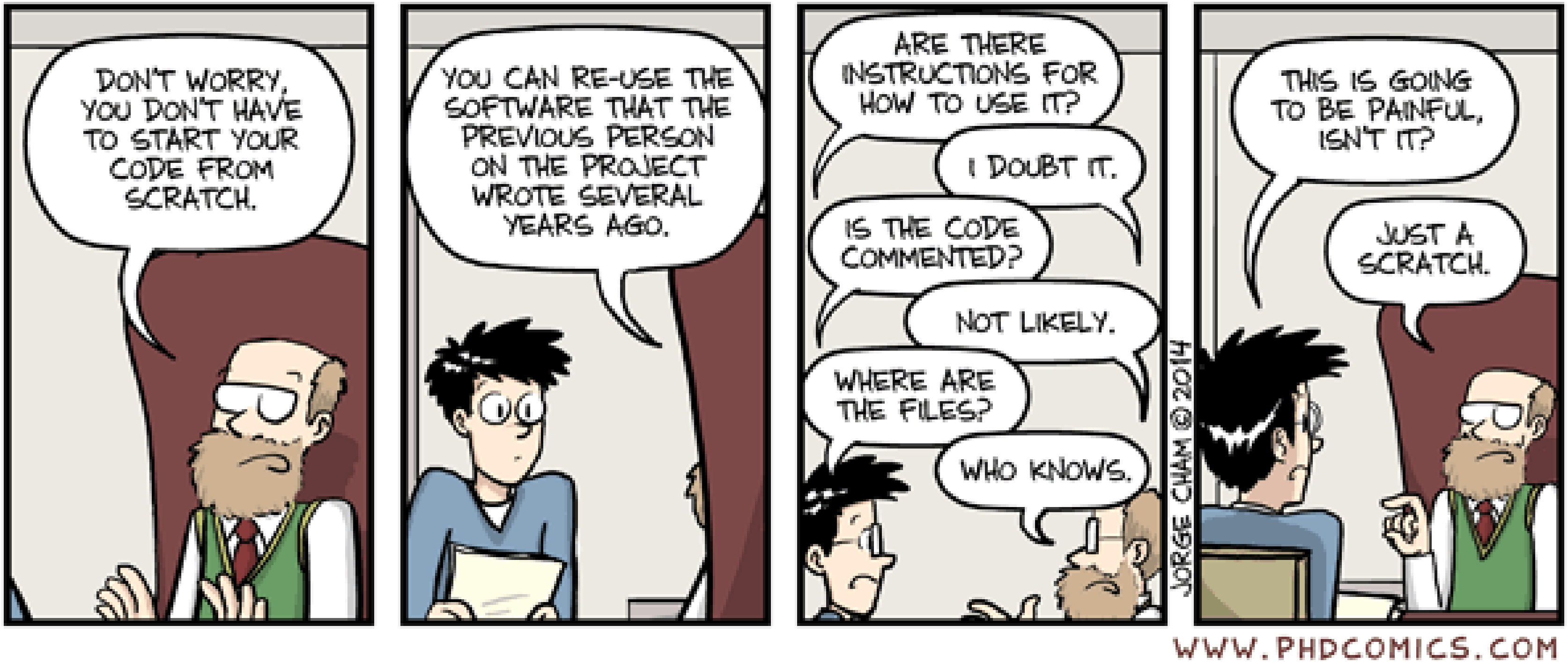
better scientific software

rOpenSci

10 SIMPLE RULES
http://collections.plos.org/ten-simple-rules

This is still going to be painful

www.software.ac.uk



"Piled Higher and Deeper" by Jorge Cham
www.phdcomics.com

http://phdcomics.com/comics.php?f=1689 – used with permission from author

Summary

- Unsustainable code isn't intentional, it comes from the tension between solving a task quickly for yourself versus solving it well for others
- Challenge is that research does not incentivize good practice, even when the stakes are high
  - Though many are changing this
- Success has come from supporting formation of communities of practice, and sharing materials
  - But this takes effort and goodwill

Find out more about the SSI

www.software.ac.uk

- Community Engagement (Lead: Shoaib Sufi)
  - Fellowship Programme & Events and Workshops
- Consultancy (Lead: Steve Crouch)
  - Open Call for Projects / Collaborations
  - Online Software Evaluation & Software Management Planning
- Policy and Publicity (Lead: Simon Hettrick)
  - Case Studies / Policy Campaigns
  - Software and Research Blog
- Training (Lead: Aleksandra Nenadic)
  - Software Carpentry and Data Carpentry
  - Guides and Top Tips
- Journal of Open Research Software (Editor: Neil Chue Hong)

Collaboration between universities of Edinburgh, Manchester, Oxford and Southampton
Supported by EPSRC Grant EP/H043160/1 + EPSRC/ESRC/BBSRC grant EP/N006410/1

Acknowledgements

www.software.ac.uk

The SSI team/*alumni*:

- Aleksandra Nenadic
- *Aleksandra Pawlik*
- *Alexander Hay*
- *Arno Proeme*
- Carole Goble
- Claire Wyatt
- Clem Hadfield
- Dave De Roure
- *Devasena Prasad*
- Giacomo Peru
- Graeme Smith
- *Iain Emsley*
- James Graham
- John Robinson
- Les Carr
- *Malcolm Atkinson*
- *Malcolm Illingworth*
- Mario Antonioletti
- Mark Parsons
- Mike Jackson
- Olivier Philippe
- *Priyanka Singh*
- Raniere Silva
- *Rob Baxter*
- *Robin Wilson*
- Shoaib Sufi
- Simon Hettrick
- Stephen Crouch
- *Tim Parkinson*
- *Toni Collis*
- *Plus the SSI Fellows and RSE community*

Scientific software:

- Dan Katz
- Heather Piowowar
- James Howison
- Jeff Carver
- Jennifer Schopf
- Kaitlin Thaney
- Martin Fenner
- Victoria Stodden
- WSSSPE community

Software/Data Carpentry

- Greg Wilson
- Jonah Duckles
- Tracy Teal
- Instructor Community

Supported by EPSRC Grant EP/H043160/1 + EPSRC/ESRC/BBSRC grant EP/N006410/1

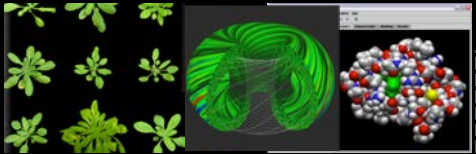
A national facility for cultivating better, more sustainable, research software to enable world-class research

- Software reaches boundaries in its development cycle that prevent improvement, growth and adoption
- Providing the expertise and services needed to negotiate to the next stage
- Developing the policy and tools to support the community developing and using research software

Supported by EPSRC Grant EP/H043160/1
+ EPSRC/ESRC/BBSRC grant EP/N006410/1

Software

Helping the community to develop software that meets the needs of reliable, reproducible, and reusable research



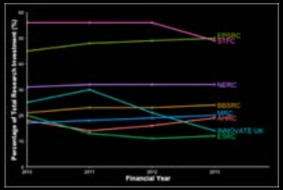
Training

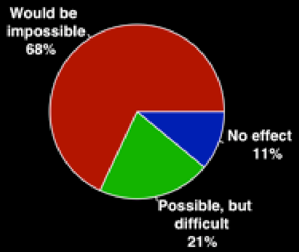
Delivering essential software skills to researchers via CDTs, institutions & doctoral schools

Outreach

Exploiting our platform to enable engagement, delivery & uptake

Collecting evidence on the community's software use & sharing with stakeholders



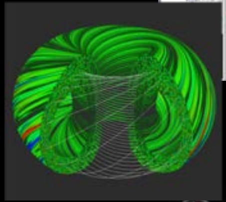


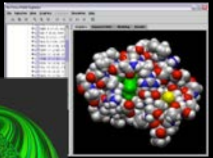
Policy

Bringing together the right people to understand and address topical issues

Community

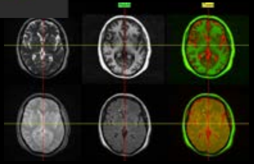
Software





Consultancy

50+ projects



Advice

130+ evaluations
4 surgeries

Training

Courses

35+ UK SWC workshops
1000+ learners

Guides

80+ guides
50,000 readers

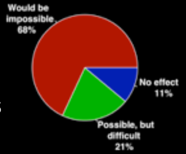
Outreach

Website & blog

150+ contributed articles
20,000 unique visitors per month
3,000 Twitter followers

Research

740 researchers
50,000 grants analysed



Campaigns

300+ RSEs engaged

BETTER SOFTWARE BETTER RESEARCH

2100 signatures

13 issues highlighted

Policy

Journal of open research software

Workshops

20+ workshops organised

Fellowship

61 domain ambassadors

Community