



[www.software.ac.uk](http://www.software.ac.uk)

# Crash Course in Clean Code

8<sup>th</sup> December 2017, Open Science for Eco-Evo Research, Amsterdam  
Neil Chue Hong (@npch), Software Sustainability Institute  
ORCID: 0000-0002-8876-7606 | N.ChueHong@software.ac.uk

*Supported by*



*Project funding  
from*



Slides licensed under  
CC-BY where indicated:





# Who am I?

- Graduated in Physics
- Worked in technology transfer
- Developer and manager on open source scientific software projects
- Started the Software Sustainability Institute based on what I experienced



[www.software.ac.uk](http://www.software.ac.uk)

# Who are you?

- Your name
- Where you're from
- What you do in one (short) sentence
- One thing you'd like to learn from this workshop



[www.software.ac.uk](http://www.software.ac.uk)

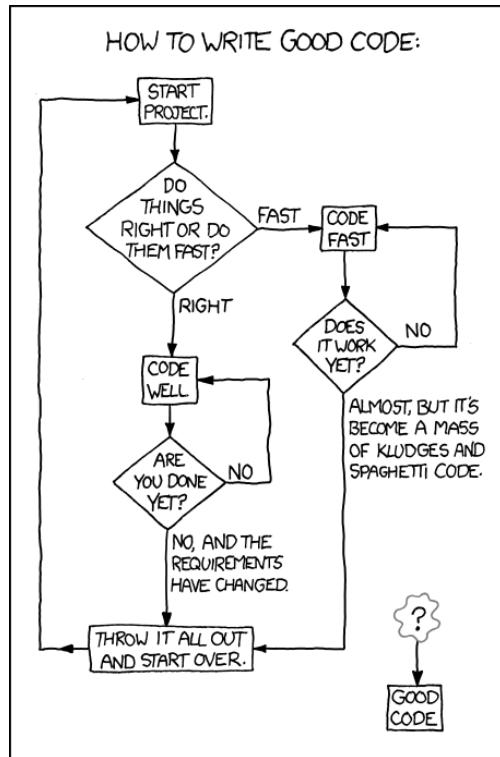
# About this workshop

- A “crash course”
  - It won’t teach you everything
  - But it will help you learn what you need to know
- Programming language agnostic
  - But we will be using Python for examples
    - Don’t worry!
- Mixture of discussion and exercises
  - Be prepared to talk to each other!

# Identifying good practice



[www.software.ac.uk](http://www.software.ac.uk)



- It's not always easy to understand how to write good code
- However you probably know more than you think

# Identifying good practice



[www.software.ac.uk](http://www.software.ac.uk)

- Split into pairs
- One person in the pair:
  - Suggests practices that improve the process of software development
  - Explains to their partner why this might be useful
- Swap and do the same for the other person
- We'll do this for five minutes each

# What's good practice?



[www.software.ac.uk](http://www.software.ac.uk)

- What things do you think are most important?
- Were there things that were unexpected?
- Are any things which are easier to implement than others?
- Do you think any things are more effective than others?



[www.software.ac.uk](http://www.software.ac.uk)

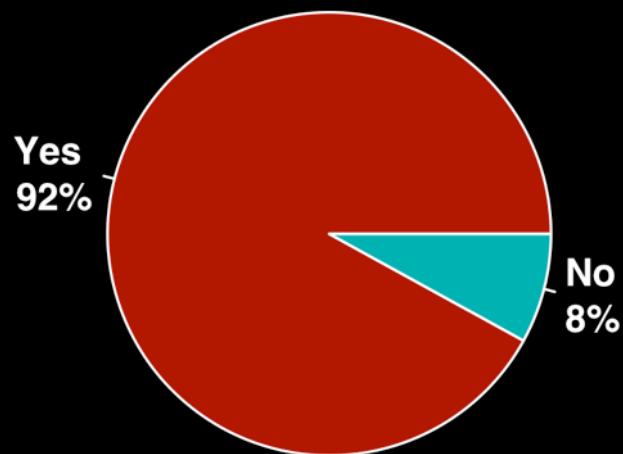
# Why is this important?

# Research depends on software

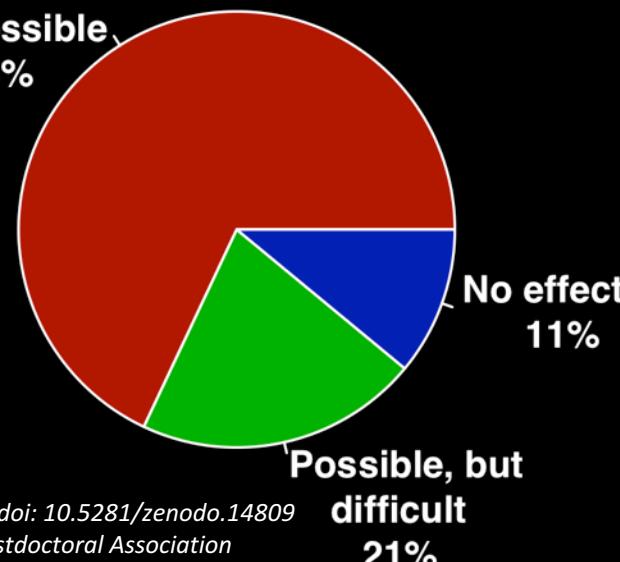


[www.software.ac.uk](http://www.software.ac.uk)

Do you use research software?



What would happen to your research without software



56%

Of UK researchers develop their own research software

71%

Of UK researchers have had no formal software development training

S.J. Hettrick, et al, "UK Research Software Survey 2014", Zenodo, 2014. doi: 10.5281/zenodo.14809

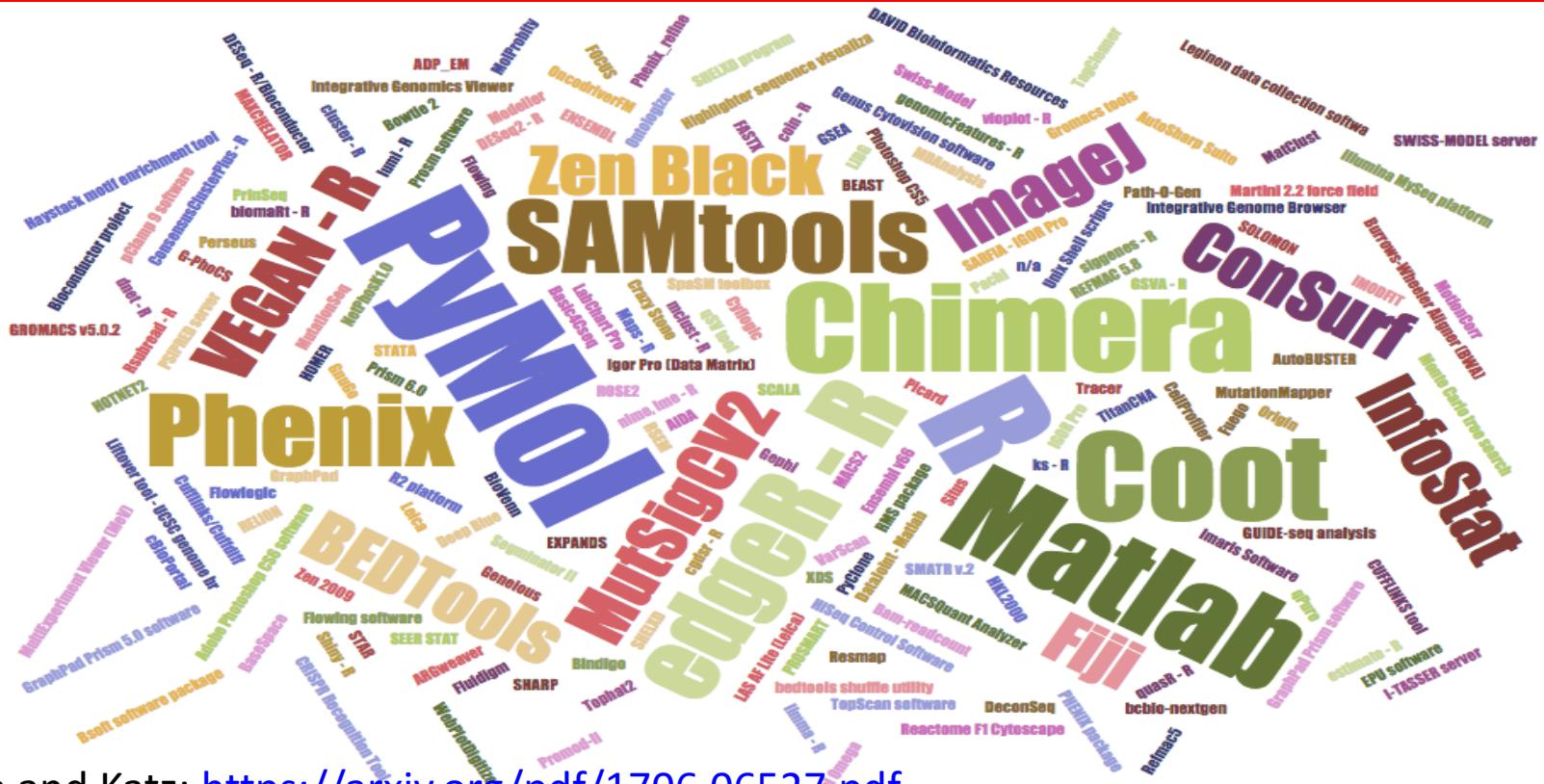
U. Nangia and D. S. Katz, "Track 1 Paper: Surveying the U.S. National Postdoctoral Association Regarding Software Use and Training in Research," Zenodo, 2017. doi: 10.5281/zenodo.814102



# Software in nature



[www.software.ac.uk](http://www.software.ac.uk)





# “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.



www.software.ac.uk

# When things go wrong

1010101 0							
1010101 0							

64 bit floating point number

16 bit signed integer

“nozzle deflections were commanded by the ... software on the basis of data transmitted by the active Inertial Reference System (SRI 2) ... Part of these data ... did not contain proper flight data, but ... was interpreted as flight data.”

“SRI 2 did not send correct attitude data ... due to a software exception”

**“software exception was caused during execution of a data conversion from 64-bit floating point to 16-bit signed integer value”**

ARIANE 5 Flight 501 Failure, Report by the Inquiry Board, 19 July 1996

[http://www.esa.int/For\\_Media/Press\\_Releases/Ariane\\_501 - Presentation of Inquiry Board report](http://www.esa.int/For_Media/Press_Releases/Ariane_501 - Presentation of Inquiry Board report)  
Cost of this bug: US\$370 million



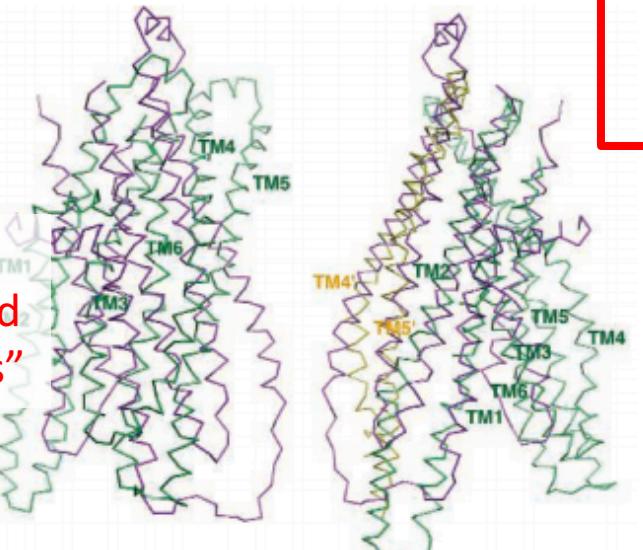
[http://www.esa.int/spaceinimages/Images/2009/09/Explosion\\_of\\_first\\_Ariane\\_5\\_flight\\_June\\_4\\_1996](http://www.esa.int/spaceinimages/Images/2009/09/Explosion_of_first_Ariane_5_flight_June_4_1996)

# 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," says Chang. "A researcher published a paper in *Nature* that described 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



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 milestones are often attributed to software because they are large, unwieldy, and notoriously difficult to work with. The materials needed for x-ray crystallography, Rees says, are among the most expensive in science'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... but the faulty software threw everything off"**

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.



www.software.ac.uk

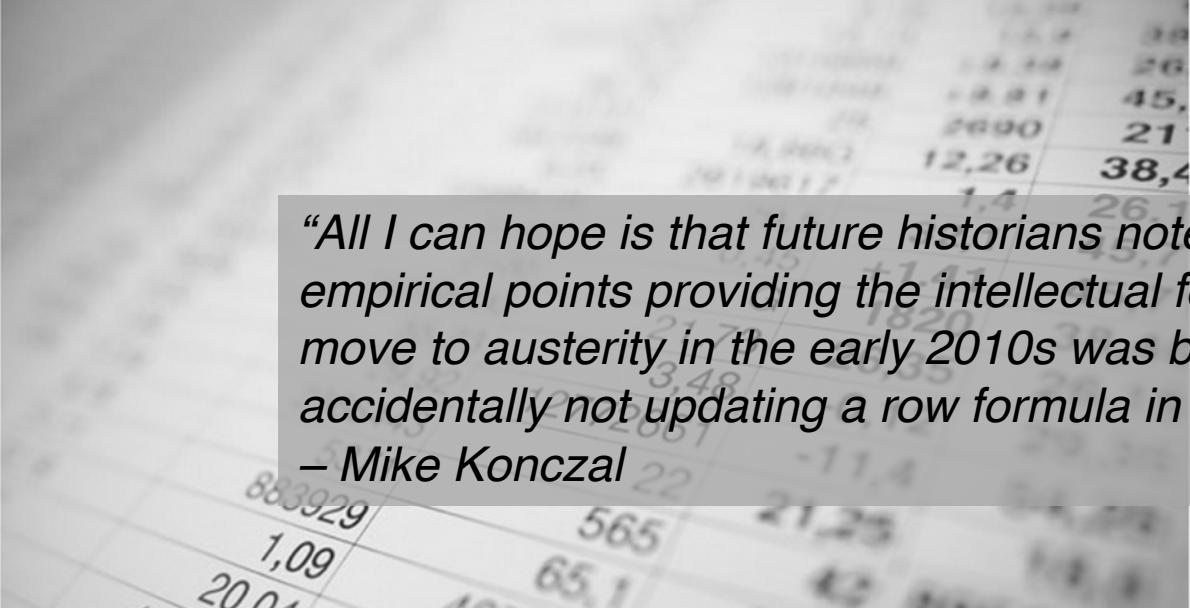
[http://science.sciencemag.org/  
content/314/5807/1856.full](http://science.sciencemag.org/content/314/5807/1856.full)

# FAQ: Reinhart, Rogoff, and the Excel Error That Changed History

By Peter Coy  | April 18, 2013



[www.software.ac.uk](http://www.software.ac.uk)



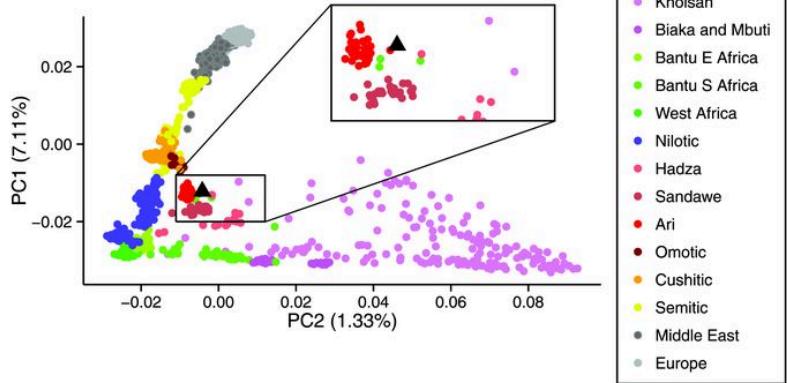
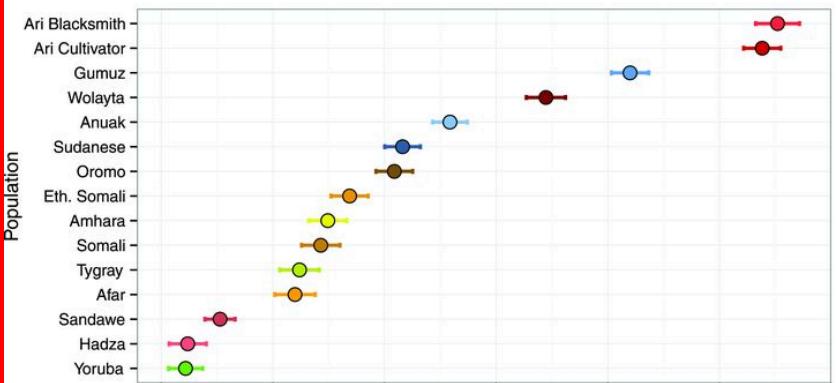
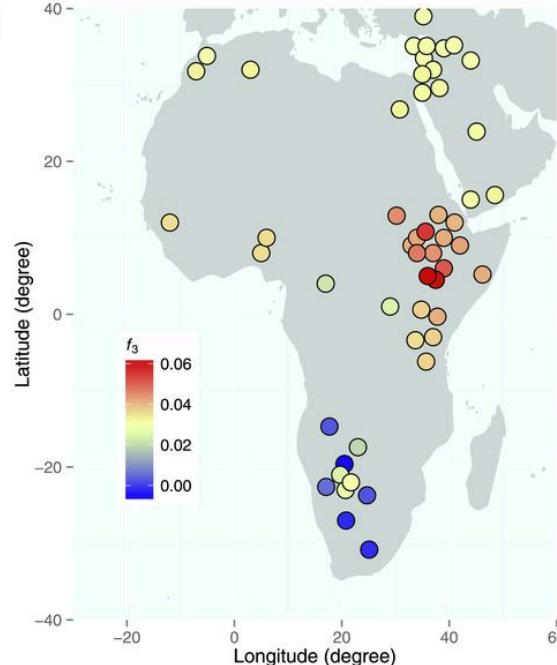
*“All I can hope is that future historians note that one of the core empirical points providing the intellectual foundation for the global move to austerity in the early 2010s was based on someone accidentally not updating a row formula in Excel”*

— Mike Konczal

Harvard University economists Carmen Reinhart and Kenneth Rogoff have acknowledged making a spreadsheet calculation mistake in a 2010 research paper, “[Growth in a Time of Debt](#)” (PDF), which has been widely cited to justify budget-

<https://www.bloomberg.com/news/articles/2013-04-18/faq-reinhart-rogoff-and-the-excel-error-that-changed-history>

Photograph by Bloomberg

**A****B****C**

[www.software.ac.uk](http://www.software.ac.uk)



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

The results presented in the Report “Ancient Ethiopian genome reveals extensive Eurasian admixture throughout the African continent” were affected by a bioinformatics error



[www.software.ac.uk](http://www.software.ac.uk)

If software is so  
important, why  
do researchers  
find it so difficult?



# 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%

# It's still all about reputation



www.software.ac.uk

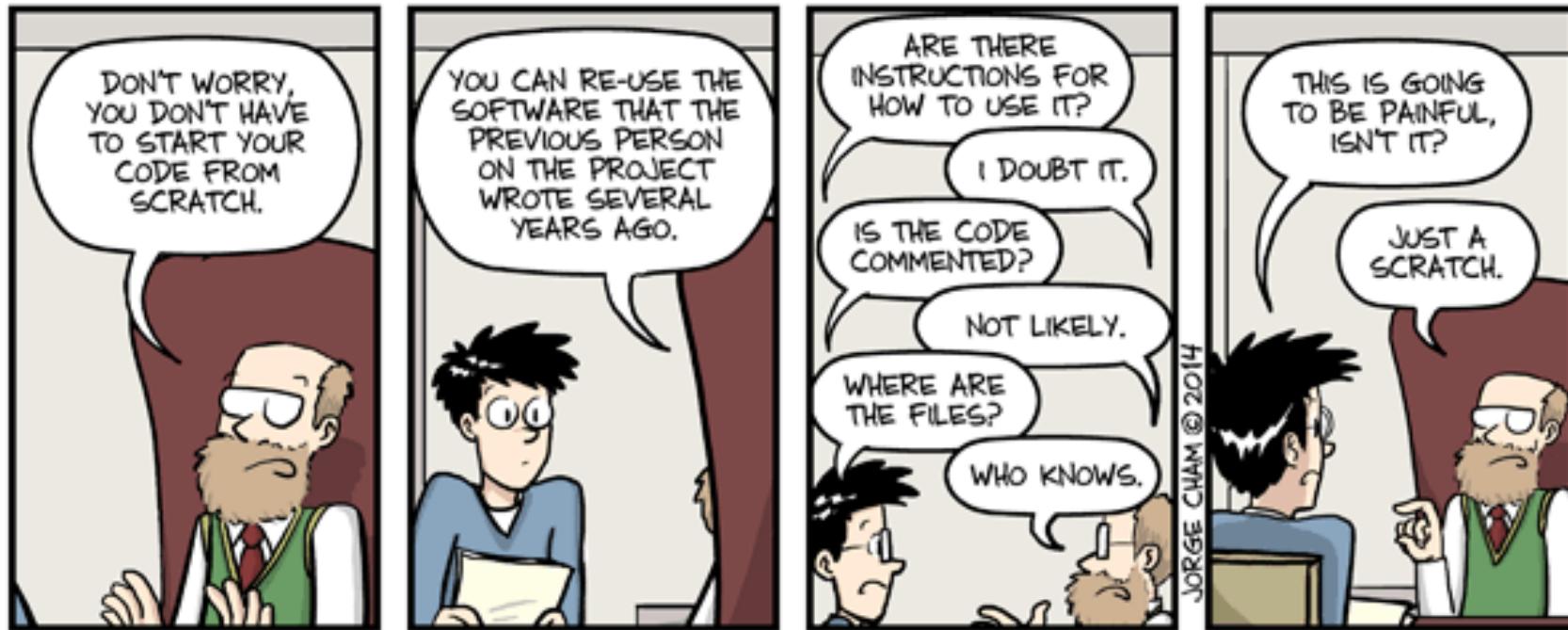
“This particular project was something I wrote a couple years ago to help me out with a workflow... I'd put it up on Github, so that others could potentially use it or use the code. So I went to see what people were saying about this project. It seemed like I'd done something fundamentally wrong, so 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>



www.software.ac.uk

# Legacy code

Piled Higher and Deeper" by Jorge Cham  
[www.phdcomics.com](http://www.phdcomics.com)



Everything is legacy code – just wait a few months!

<http://phdcomics.com/comics.php?c=1689> Software Sustainability Institute



www.software.ac.uk

# Croucher's Law

*I can be an idiot and will make mistakes*

*You are no different!*

Corollary: Even when you're  
an expert programmer



[http://mikecroucher.github.io/MLPM\\_talk/](http://mikecroucher.github.io/MLPM_talk/)

## Community Page

## Best Practices for Scientific Computing

Greg Wilson<sup>1\*</sup>, D. A. Arulah<sup>2</sup>, C. Titus Brown<sup>3</sup>, Neil P. Chue Hong<sup>4</sup>, Matt Davis<sup>5</sup>, Richard T. Guy<sup>6\*</sup>, Steven H. D. Haddock<sup>7</sup>, Kathryn D. Huff<sup>8</sup>, Ian M. Mitchell<sup>9</sup>, Mark D. Plumbe<sup>10</sup>, Ben Waugh<sup>11</sup>, Ethan P. White<sup>12</sup>, Paul Wilson<sup>13</sup>

**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 effectively. As a result, there are many bad habits and practices that could 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 large amounts of data that are generated in single research projects, comparing disparate datasets to assess synthetic predictions, 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 scientific software is as important as physical 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 Journal of Human Genetics*.

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 should practice that will 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.

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 within scientific computing and in the software industry. For instance [17,18], reports from many other groups [19–23], guidelines for commercial and open source software development [24,27], 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 than can be spent focusing on the underlying scientific questions.

Our practices are summarized in Box 1; for the benefit of those who do not have time to read the entire article. For brevity of space, we did not discuss the equally important (but independent) issues of reproducible research, publication and curation 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.

**Attribution:** Wilson G, Arulah DA, Brown CT, Chue Hong NP, Davis M, et al. 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

**Copyright:** © 2014 Wilson et al. This is an open-access article distributed under the terms of the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.

**Funding:** Neil Chue Hong was supported by the UK Engineering and Physical Sciences Research Council (EPSRC) through grant EP/I000640/1 for the Software Sustainability Institute. Ian M. Mitchell was supported by NSERC Discovery Grant #RGP2011. Mark Plumbe was supported by EPSRC through a Leadership Fellowship (EP/L000200). Greg Wilson was supported by a Microsoft Research PhD fellowship. Ethan P. 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 (SFP-2011-12).

**Competing interests:** The lead author (GWW) is involved in a pilot study of code review in scientific computing with PLOS Computational Biology. \* E-mail: gwillson@softwarecarpentry.org

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

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

## 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 regenerate anything that needs to be regenerated.

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 once, so if a program is too complex, it "chunks" 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 conduct 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 pascal\_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 to find the data they need [37]. 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 second paradigm shift in 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 should 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.



www.software.ac.uk



www.software.ac.uk

## Good Enough Practices in Scientific Computing

Greg Wilson<sup>1,4\*</sup>, Jennifer Bryan<sup>2,4</sup>, Karen Cranston<sup>3,†</sup>, Justin Kitzes<sup>4,‡</sup>, Lex Nederbragt<sup>5,‡</sup>, Tracy K. Teal<sup>6,‡</sup>  
1 Software Carpentry Foundation / gwilson@software-carpentry.org  
2 University of British Columbia / jenny@stat.ubc.ca  
3 Duke University / karen.cranston@duke.edu  
4 University of California, Berkeley / jkitzes@berkeley.edu  
5 University of Oslo / lex.nederbragt@ibv.uio.no  
6 Data Carpentry / tkteal@datacarpentry.org

‡ These authors contributed equally to this work.

\* E-mail: Corresponding gwilson@software-carpentry.org

## Abstract

We present a set of computing tools and techniques that every researcher can and should adopt. 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 six years, and from a variety of other guides. Our recommendations are aimed specifically at people who are new to research computing.

## 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, they take days or weeks to do things that could be done in minutes or hours, are often unable to reproduce their own work (much less the work of others), and have no idea how reliable their computational results are.

This paper presents a set of good computing practices that every researcher can adopt regardless of their current level of technical 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.

## Introduction

Two years ago a group of researchers involved in Software Carpentry<sup>1</sup> and Data Carpentry<sup>2</sup> wrote a paper called “Best Practices for Scientific Computing” [1]. It was well received, but many novices found its litany of tools and techniques intimidating. Also, by definition, the “best” are a small minority. What practices are comfortably within reach for the “rest”?

<sup>1</sup><http://software-carpentry.org/>

## 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

# 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/](https://tools.ipol.im/wiki/ref/software_guidelines/)
- ELIXIR (Life Sciences):  
<https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5490478/>
- *Your community here?*



www.software.ac.uk

Good enough  
practices to please  
your future self



# Good Enough Practices - Data



- Data – play FAIR:
  - Save and backup raw data
  - Create analysis-friendly data
  - Record your processing steps
  - Anticipate the need to use multiple tables, and use a unique identifier for each record
  - Submit data to a repository and get a DOI

# Good Enough Practices - Software



- Use version control
- Document for your future self
- Learn to be modular
- Make it accessible in the future; choose a license
- Get a colleague to try using it
- Ask a collaborator to contribute and give feedback

# Good Enough Practices - Software



- Use version control
- Document for your future self
- Learn to be modular
- Make it accessible in the future; choose a license
- Get a colleague to try using it
- Ask a collaborator to contribute and give feedback



[www.software.ac.uk](http://www.software.ac.uk)

# Get some training



[admin@software-carpentry.org](mailto:admin@software-carpentry.org)

*Teach basic lab skills for scientific computing so that researchers can do more in less time and with less pain.*

*Open source learning, that can be tailored to disciplines.  
“Train the trainers”: building a capable base of instructors.*



[admin@datacarpentry.org](mailto:admin@datacarpentry.org)

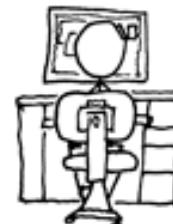
*Teach basic concepts, skills and tools for working more effectively with data. Workshops are designed for people with little to no prior computational experience.*



# What we're trying to avoid



www.software.ac.uk



<https://xkcd.com/292/>



# Code review –

*or how we avoid  
being eaten by  
raptors*

# Most efficient way to find bugs



[www.software.ac.uk](http://www.software.ac.uk)

- “Rigorous inspections can remove up to 90 percent of errors from a software product...”
- “...before the first test case is run”
  - Fact 37, Glass, R. Facts and Fallacies of Software engineering, Addison Wesley, 1992. ISBN-10: 0321117425. ISBN-13: 978-0321117427.
  - Fagan, M.E. “Design and Code inspections to reduce errors in program development”, IBM Systems Journal 15(3), pp182–211, NN 1975. doi:10.1147/sj.153.0182.

# Most efficient way to find bugs



- First review and first hour matter most
  - Cohen, J. “Best Kept Secrets of Peer Code Review”, Smart Bear Inc, 2006. ISBN-10: 1599160676. ISBN-13: 978-1599160672.
- Not just for bugs, but alternatives, improvements and two-way knowledge transfer
- Doesn't have to be “heavyweight” / difficult



# Things to look for

- Is the program broken up into easily understood components?
- Are the components loosely coupled?
- Are names consistent, distinctive and meaningful?
- Is code style and formatting consistent?
- Are constants defined exactly once?
- Does the program have duplicated code?
- Does the program reuse existing libraries?
- Does the documentation describe dependencies and versions?



# Things to look for

- Are component interfaces documented?
- Do comments document reasons why code has been implemented as it is, rather than the implementation itself?
- Is documentation embedded in the program?
- Is there user documentation which describes how to use the program?
- Does the program use assertions to check validity of inputs and outputs?
- Does the program have automated tests?

# But remember to “play nice”



www.software.ac.uk

YOUR CODE LOOKS LIKE  
SONG LYRICS WRITTEN  
USING ONLY THE STUFF  
THAT COMES AFTER THE  
QUESTION MARK IN A URL.

SORRY.



IT'S LIKE A JSON  
TABLE OF MODEL  
NUMBERS FOR  
FLASHLIGHTS  
WITH "TACTICAL"  
IN THEIR NAMES.



LIKE YOU READ TURING'S  
1936 PAPER ON COMPUTING  
AND A PAGE OF JAVASCRIPT  
EXAMPLE CODE AND GUessed  
AT EVERYTHING IN BETWEEN.



IT'S LIKE A LEET-SPEAK TRANSLATION  
OF A MANIFESTO BY A SURVIVALIST CULT  
LEADER WHO'S FOR SOME REASON  
OBSESSED WITH MEMORY ALLOCATION.

I CAN GET SOMEONE  
ELSE TO REVIEW MY CODE.

NOT MORE THAN  
ONCE, I BET.



From <https://xkcd.com/1833/>



# Code review exercise

- In small groups:
  - Take a look at “program.py”
  - Spend 30 minutes reviewing it
  - Write up issues you find
- Then we'll:
  - Discuss these issues
  - Prioritise them



www.software.ac.uk

# Learning to love version control



www.software.ac.uk

# This is version control



<http://phdcomics.com/comics/archive.php?comicid=1531>

**Piled Higher and Deeper" by Jorge Cham**  
[www.phdcomics.com](http://www.phdcomics.com)



www.software.ac.uk

# This is also version control

A STORY TOLD IN FILE NAMES:				
Filename	Date Modified	Size	Type	
data_2010.05.28_test.dat	3:37 PM 5/28/2010	420 KB	DAT file	
data_2010.05.28_re-test.dat	4:29 PM 5/28/2010	421 KB	DAT file	
data_2010.05.28_re-re-test.dat	5:43 PM 5/28/2010	420 KB	DAT file	
data_2010.05.28_calibrate.dat	7:17 PM 5/28/2010	1,256 KB	DAT file	
data_2010.05.28_huh???.dat	7:20 PM 5/28/2010	30 KB	DAT file	
data_2010.05.28_WTF.dat	9:58 PM 5/28/2010	30 KB	DAT file	
data_2010.05.29_aaarrgh.dat	12:37 AM 5/29/2010	30 KB	DAT file	
data_2010.05.29_#\$@*!!..dat	2:40 AM 5/29/2010	0 KB	DAT file	
data_2010.05.29_crap.dat	3:22 AM 5/29/2010	437 KB	DAT file	
data_2010.05.29_notbad.dat	4:16 AM 5/29/2010	670 KB	DAT file	
data_2010.05.29_woohoo!.dat	4:47 AM 5/29/2010	1,349 KB	DAT file	
data_2010.05.29_USETHISONE.dat	5:08 AM 5/29/2010	2,894 KB	DAT file	
analysis_graphs.xls	7:13 AM 5/29/2010	455 KB	XLS file	
ThesisOutline.doc	7:26 AM 5/29/2010	38 KB	DOC file	
Notes_Meeting_with_ProfSmith.txt	11:38 AM 5/29/2010	1,673 KB	TXT file	
JUNK...	2:45 PM 5/29/2010		Folder	
data_2010.05.30_startingover.dat	8:37 AM 5/30/2010	420 KB	DAT file	

Type: Ph.D Thesis Modified: too many times Copyright: Jorge Cham www.phdcomics.com

- Everyone's done this at some point in their life
- Are you still doing it now?

Piled Higher and Deeper" by Jorge Cham  
[www.phdcomics.com](http://phdcomics.com)

<http://phdcomics.com/comics/archive.php?comicid=1323>



www.software.ac.uk

# This is not version control

- Change-bound
  - Google Drive: 100 revisions
- Time-bound
  - Google Drive: 30 days
  - DropBox: Last 30 days, 1 year (by subscription)
- What if multiple changes to multiple files need to be treated as a single change?
- How/where to record why changes were made?



# Version control / revision control / source code repository



www.software.ac.uk

- Unlimited changes
- Unlimited time
- Bound only by available storage
- Treat changes to multiple files as a single change
- Record/view who changed what files, when and why
  - And, for ASCII files, what they changed
- Tag specific versions e.g. “RELEASE-1.0”, “CHEP-2017”
- Retrieve any previous version by version number or tag
- Collaborate on shared files without losing work
- “a lab notebook for code and documents”





# Which version to choose?

## *System*

- Client-Server
  - Subversion (SVN)
  - CVS
- Distributed
  - Git
  - Mercurial

## *Hosting Platform*

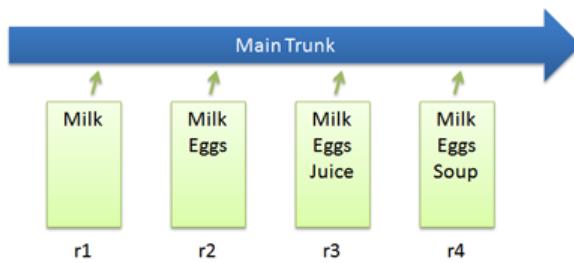
- Local
  - SVN
  - Gitlab
- External
  - GitHub
  - BitBucket



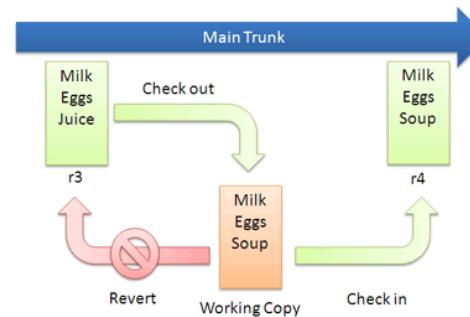
www.software.ac.uk

# Using version control

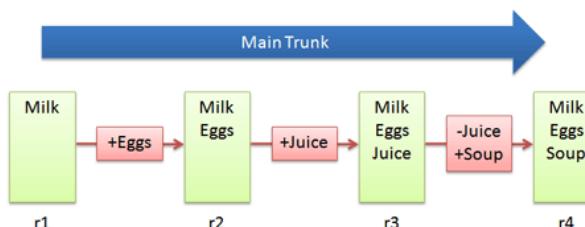
## Basic Checkins



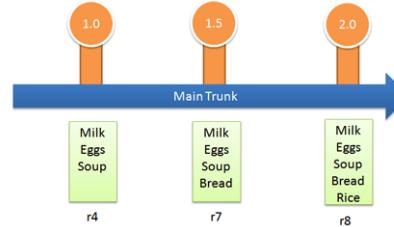
## Checkout and Edit



## Basic Diffs



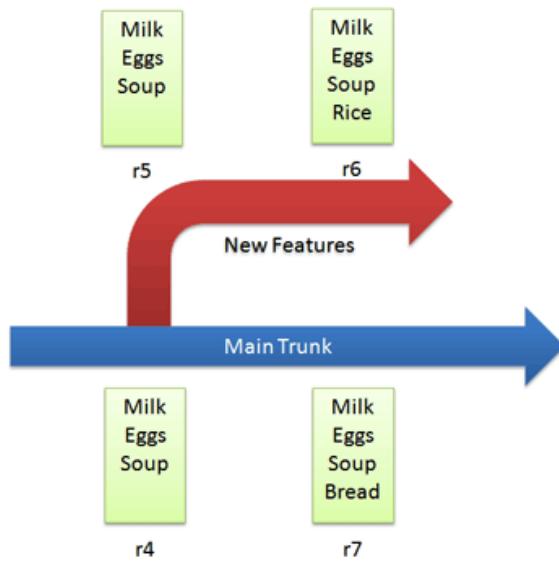
## Tagging



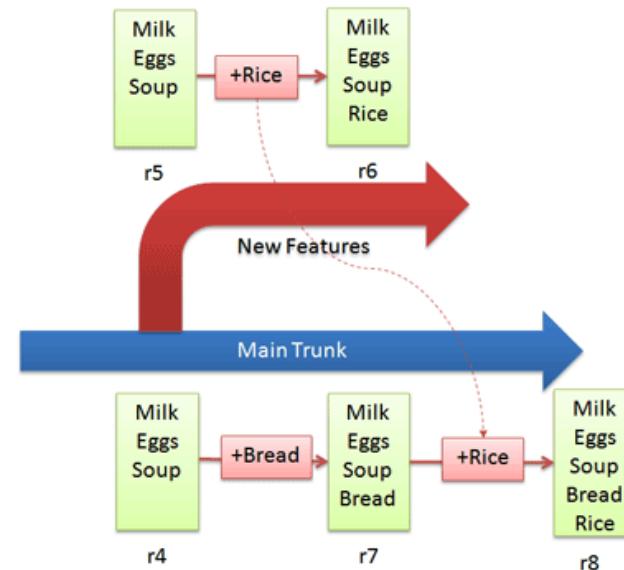


# Using version control

## Branching



## Merging





www.software.ac.uk

# Record who, what, when & why

-  Update badges in README  
robintw committed on GitHub on 23 Feb ✓  
  
 Merge pull request #177 from softwaresaved/recipy\_test ...  
robintw committed on GitHub on 23 Feb
- Commits on Feb 20, 2017
  -  Added links from docs/TestFramework.md to PackageVersionFailures.md a...  
mikej888 committed on 20 Feb  
...
  -  Added binaryornot to setup.py removing need for pip install -r requir...  
mikej888 committed on 20 Feb
  -  .travis.yml and appveyor.yml explicitly invoke pip install -r require...  
mikej888 committed on 20 Feb
  -  Merge branch 'master' into recipy\_test  
mikej888 committed on 20 Feb
- Commits on Feb 7, 2017
  -  Merge pull request #173 from chiffa/tinyDB\_correction ...  
robintw committed on GitHub on 7 Feb
- Commits on Feb 6, 2017
  -  Moved version detection. Closes recipy/recipy #128 ...  
chiffa committed on GitHub on 6 Feb



www.software.ac.uk

# Record/view what was changed

.travis.yml and appveyor.yml explicitly invoke pip install -r require...

...ments.txt so that some required recipy packages e.g. binaryornot, which are not specified in setup.py, are installed.

master (#177)

mikej888 committed on 20 Feb 1 parent cdf62af commit 35fff953e7a62ffe9b9e6e4b8925f308d8ddfb8b

Showing 2 changed files with 6 additions and 0 deletions.

Unified Split

View ▾

3 .travis.yml

```
@@ -31,6 +31,9 @@ install:  
 31   31     - conda install flask  
 32   32     # Install recipy.  
 33   33     - python setup.py install  
 34 + # Install requirements not specified in setup.py but specified in  
 35 + # requirements.txt  
 36 + - pip install -r requirements.txt  
 37   # Install behave for unit tests  
 38   - pip install -e . behave  
 39   # Install py.test explicitly else
```

Updated content to reflect Bath boot camp instructors and syllabus

2013-07-bath (#116)

mikej888 committed on 22 May 2013 1 parent 6425908 commit 757eaa2ebcf19c44da59776a66bf4b956b7d79f

Showing 2 changed files with 0 additions and 0 deletions.

Unfiled Split

View ▾

3 appveyor.yml

```
@@ -29,6 +29,9 @@ install:  
 29   29     - conda install flask  
 30   30     # Install recipy.  
 31   31     - python setup.py install  
 32 + # Install requirements not specified in  
 33 + # requirements.txt  
 34 + - pip install -r requirements.txt  
 35
```

BIN Conclusion.ppt

Binary file not shown.

BIN Welcome.ppt

Binary file not shown.

# Everything created manually

- Source code
  - C, C++, Fortran, Java, Python, R, ...
  - Shell scripts
  - LaTeX
- Word, Powerpoint, Visio
- Build scripts e.g. Makefiles, ANT files
- Configuration files
- Meta-data for images/audio
  - Store images/audio in an archive

# The feeling of (version) control



- “Your laptop's just been stolen”
  - “No problem, I've pushed regularly to a remote repository”
- “Can I have the code you used to create the data that you graphed in your conference paper?”
  - “Let me check the logs then get the version with that tag”
- “You deleted my analysis section from our paper”
  - “Go and get the previous version from the repository then!”
- “Why did they rewrite my function?”
  - “Look at the commit messages and see why they did it”



# Choosing a license

- Don't overthink it
  - Commercial
    - Retain complete control
  - Open Source – Permissive (MIT, BSD, Apache)
    - Make available as widely as possible
  - Open Source – Copyleft (GPL, LGPL)
    - Ensure that changes are contributed back



# Choosing a license

- Think about who you want to use your software and how they will interact with you
- Non-judgemental Guidance
  - <https://choosealicense.com/>
- Software Licenses in plain English
  - <https://tldrlegal.com/>



www.software.ac.uk



Connecting Research  
and Researchers

EDIT YOUR RECORD

ABOUT ORCID

CONTACT US

HELP

## Neil P. Chue Hong

### Biography

#### ORCID ID

<https://orcid.org/0000-0002-8876-7606>

Print view

#### Other IDs

Scopus Author ID: 8558277300

Neil is the founding Director of the Software Sustainability Institute. Graduating with an MPhys in Computational Physics from the University of Edinburgh, he began his career at EPCC, becoming Project Manager there in 2003. During this time he led the Data Access and Integration projects (OGSA-DAI and DAIT), and collaborated in many e-Science projects, including the EU FP6 NextGRID project.

He is presently responsible for representing both the Institute and UK researchers at a national and international level. Within the organisation, he oversees operations, contributes to policy development, develops and manages collaborations, and acts as the principal liaison with stakeholders. His current research is in cloud computing, computational chemistry, software ecosystems, and digital repositories.

▼ Education (1)

↑ Sort

University of Edinburgh: Edinburgh, Edinburgh, United Kingdom

1994-09 to 1999-07 | MPhys (hons) Computational Physics (Physics and Astronomy)





www.software.ac.uk

# Software in Zenodo

zenodo

Search

Upload Communities

All versions

Found 15181 results.

Access Right

- Open (288413)
- Closed (18929)
- Restricted (459)
- Embargoed (277)

File Type

- Png (124044)
- Pdf (79327)
- Jpg (44651)
- Zip (28919)
- Hdf5 (12139)
- Xml (5660)
- Docx (2877)
- Gz (1860)
- Xlsx (1038)
- Md (1021)

Keywords

- Taxonomy (160396)
- Animalia (153609)

Sort by: Most recent asc. View

December 6, 2017 (v1.0.0) Software Open Access

Tracking Smallholder Farm Households: EPAR Project #356 [Stata .do files]

Evans School Policy Analysis & Research Group (EPAR);

Initial release of code to reproduce analyses from EPAR Project #354, Tracking Smallholder Farm Households: <https://evans.uw.edu/policy-impact/epar/research/tracking-smallholder-farm-households>

Uploaded on December 6, 2017

September 19, 2017 (0.6.0) Software Open Access

PyDL

Weaver, Benjamin Alan;

Library of IDL astronomy routines converted to Python.

Uploaded on December 6, 2017

December 6, 2017 (v2017.12.6) Software Open Access

materialsproject/pymatgen: v2017.12.6

Shyue Ping Ong; gmatteo; Michiel van Setten; Will Richards; Xiaohui Qu; Joseph Montoya; Anubhav Jain; Kiran Mathew; Geoffroy Hautier; Stephen Dacek; Bharat Medasani; cedergroupclusters; Richard Tran; Shyam Dwaraknath; Michael; Sai Jayaraman; Matthew Horton; Nils Zimmermann; Germain Salvato Vallverdu; yanikou19; Guido Petretto; fraricci; Dan Gunter; Zhi Deng; Tess; Oskar Weser; Saurabh Bajaj; Alireza Faghani; ndardenne; Bruno Rousseau;

Support for HDF5 output for VolumetricData (CHGCAR, LOCPOT, etc.). Support for Crystal Orbital Hamilton Populations (COHPs) (@marcoesters) REST interface for Pourbaix data Support for optical property parsing in Vasprun. Improvements to LammpsData Misc bug fixes.

Uploaded on December 6, 2017

109 more version(s) exist for this record

- Zenodo is a digital repository
- It allows you to deposit software easily

# Deposit software in Zenodo



www.software.ac.uk

- If your software is in GitHub, it's easy to put in Zenodo:
  - <https://guides.github.com/activities/citable-code/>
- Gives your code a Digital Object Identifier (DOI)
- Preserves a particular version
  - E.g. used for a paper, major release



www.software.ac.uk

Three months  
from now,  
you will thank  
yourself!



# Publish a software paper



www.software.ac.uk

- Publish a software paper:

<http://bit.ly/softwarejournals>

- Methods in Ecology and Evolution
- Ecography (software notes)
- Journal of Open Research Software
- Journal of Open Source Software

# Literate Programming



- Traditional papers are just advertisements
  - A literate computing document is the research
- The technology is out there
  - [Jupyter notebooks](#)
  - [Mathematica](#)
  - [R Markdown](#)
  - [knitR](#)
  - [MATLAB Live scripts](#)



[www.software.ac.uk](http://www.software.ac.uk)

# LIGO Example

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  $1.0 \times 10^{-21}$ . 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  $5.1\sigma$ . 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

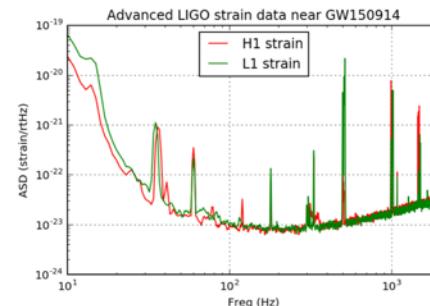
### I. INTRODUCTION

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]: # 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 = interpld(freqs, Pxx_H1)  
psd_L1 = interpld(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  $f_{\text{min}} = 10$  Hz and  $f_{\text{max}} = 2000$  Hz.



[www.software.ac.uk](http://www.software.ac.uk)

# Better program.py

- Jupyter notebook version



[www.software.ac.uk](http://www.software.ac.uk)

Don't be afraid to  
publish your code

# Research Software Workflow



[www.software.ac.uk](http://www.software.ac.uk)

→ **describe** →



GitLab



GitHub



Bitbucket



zenodo



**develop → share → preserve**

Developed and  
versioned using  
code repository

Published via  
code repository  
or website

Deposited in  
digital repository  
with paper /  
for preservation

# Find out more about the SSI



[www.software.ac.uk](http://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)



MANCHESTER  
1824



UNIVERSITY OF  
Southampton

EPSRC  
Pioneering research  
and skills

BBSRC

Jisc

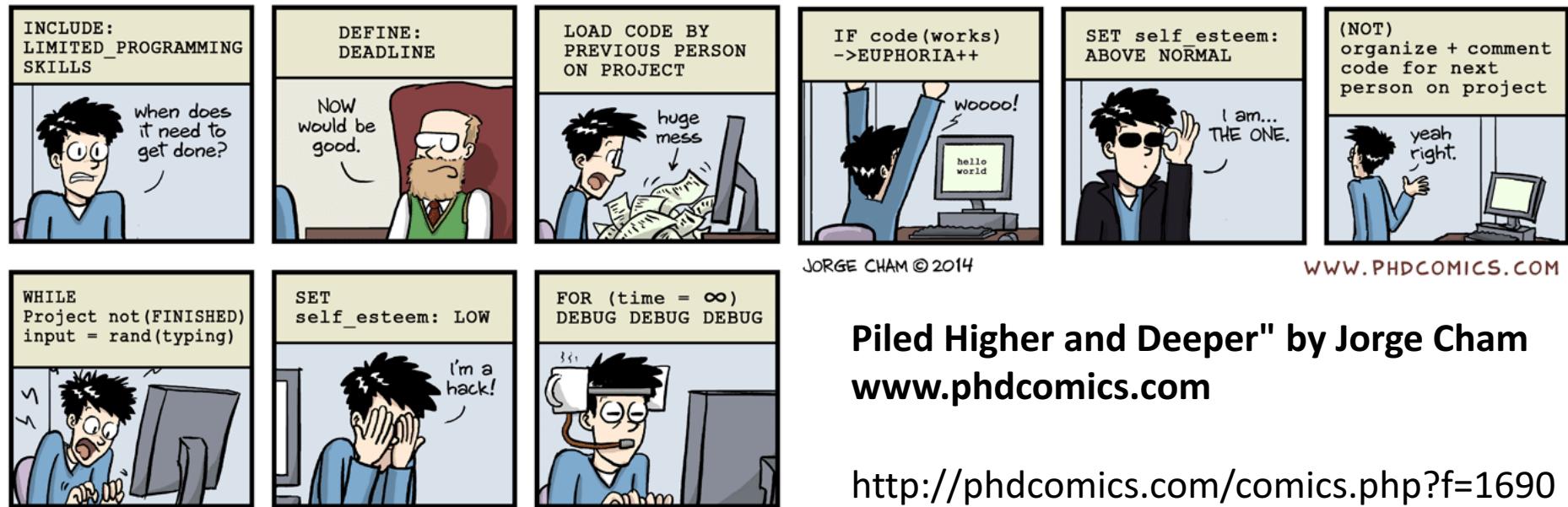


# This doesn't have to be you!



www.software.ac.uk

## PROGRAMMING FOR NON-PROGRAMMERS



**Piled Higher and Deeper" by Jorge Cham**  
[www.phdcomics.com](http://phdcomics.com)



[www.software.ac.uk](http://www.software.ac.uk)

# Acknowledgements

## The SSI team/alumni:

- Mario Antonioletti
- 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*

## Scientific software:

- Dan Katz
- Heather Piwowar
- 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



Pioneering research



# Software Sustainability Institute



[www.software.ac.uk](http://www.software.ac.uk)

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

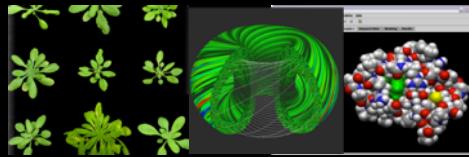
- Software reaches boundaries in its development cycle that prevent improvement, growth and adoption
- Providing the expertise and services needed to negotiate to the next stage
- Developing the policy and tools to support the community developing and using research software



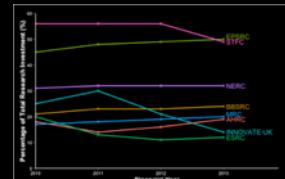
Supported by EPSRC Grant EP/H043160/1  
+ EPSRC/ESRC/BBSRC grant EP/N006410/1

# Software

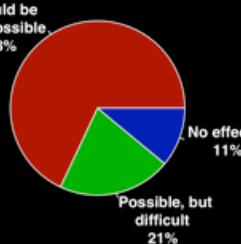
Helping the community to develop software that meets the needs of reliable, reproducible, and reusable research



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



# Policy



# Outreach

Exploiting our platform to enable engagement, delivery & uptake

Bringing together the right people to understand and address topical issues

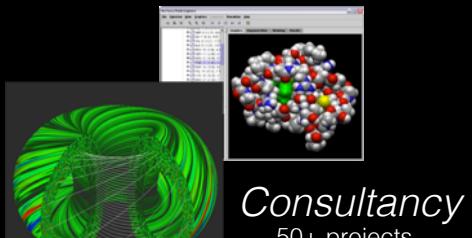


# Community

# Software

## Advice

# Training



Courses  
35+ UK SWC workshops  
1000+ learners



# Outreach

## Website & blog

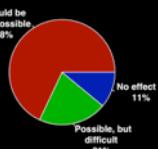
150+ contributed articles  
20,000 unique visitors per month  
3,000 Twitter followers

Guides  
80+ guides  
50,000 readers



## Research

740 researchers  
50,000 grants analysed



**BETTER SOFTWARE BETTER RESEARCH**

300+ RSEs engaged 2100 signatures

## Campaigns



13 issues highlighted



## Workshops



# Policy

# Community