More Than Top Down or Bottom Up:
Fostering Software Engineering Best Practice in Diverse Groups

https://doi.org/10.6084/m9.figshare.6731696
3rd July 2018, PASC2018, Basel
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.
Best practice is HARD

• 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

Outreach
Exploiting our platform to enable engagement, delivery & uptake

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

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

Community
Bringing together the right people to understand and address topical issues
About this talk

- Why software best practice is important
  - Ubiquity of software; Reproducibility crisis; Careers

- What’s already happening
  - Changes in Policy; Carpentries; Research Software Engineers; Community Guidelines

- What the next challenges are
  - Lack of resources; Belief in research commons
UK software survey 2014

S.J. Hettrick et al.
UK Research Software Survey 2014
Software in research papers

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

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
Can you spot the mistake?

“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

“Chang’s data are good... but the faulty software threw everything off”

“a homemade data-analysis program had flipped two columns”
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
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.”
### Table 1. Responses to emailed requests ($n = 180$)

<table>
<thead>
<tr>
<th>Type of response</th>
<th>Count</th>
<th>Percent, %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Did not share data or code:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Contact another person</td>
<td>20</td>
<td>11</td>
</tr>
<tr>
<td>Asked for reasons</td>
<td>20</td>
<td>11</td>
</tr>
<tr>
<td>Refusal to share</td>
<td>12</td>
<td>7</td>
</tr>
<tr>
<td>Directed back to supplement</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>Unfulfilled promise to follow up</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>Impossible to share</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Shared data and code</td>
<td>65</td>
<td>36</td>
</tr>
<tr>
<td>Email bounced</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>No response</td>
<td>46</td>
<td>26</td>
</tr>
</tbody>
</table>
Culture change is hard

“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:"

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.

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.

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.

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

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

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

Open source learning, “Train the trainers”
Introduction

Scientific computing has become increasingly important as data sizes and computational requirements continue to grow. This necessitates the use of software tools that can efficiently handle large datasets and complex calculations. Software tools are essential for researchers to perform their work effectively and accurately. They enable scientists to analyze data, simulate processes, and extract meaningful insights from experimental results.

Best Practices for Scientific Computing

- Write code for people, not computers.
- A program should not require its users to hold more than a handful of keys in memory at once.
- Make code style and formatting consistent.
- Make the computer report errors.
- Make the computer report warnings.
- Follow standard paths with frequent feedback and course correction.
- Use an on-the-fly text editor, and don’t edit anything that has been saved manually.
- Write code instead of editing it.
- Fix the bugs.
- Add assertions in programs to check their operation.
- Use an on-the-fly text editor.
- Use a symbolic debugger.
- Use a build tool to automate workflows.

Box 1. Summary of Best Practices

1. Write code for people, not computers.
2. A program should not require its users to hold more than a handful of keys in memory at once.
3. Make code style and formatting consistent.
4. Make the computer report errors.
5. Make the computer report warnings.
6. Follow standard paths with frequent feedback and course correction.
7. Use an on-the-fly text editor.
8. Write code instead of editing it.
9. Fix the bugs.
10. Add assertions in programs to check their operation.
11. Use an on-the-fly text editor.
12. Use a symbolic debugger.
13. Use a build tool to automate workflows.
Good enough practices in scientific computing

Greg Wilson*, Jennifer Bryan, Karen Cranston, Justin Kitzes, Lex Nederbragt, Tracy K. Teitz

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.
* provision@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, and from the experiences of the thousands of people who have taken part in Software Carpentry and Data Carpentry workshops in 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. Store 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-hosting 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 behaviour.
   i. Provide a simple example or test data set.
   j. Submit code to a reputable DOI-hosting repository.

3. Collaboration
   a. Create an overview of your project.
   b. Create a shared public “holy” list.
   c. Make the license explicit.
   d. Make the project visible.

4. Project Organization
   a. Place 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 notebooks 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 off the researcher’s working machine.
   f. Use a file named CHANGES.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.
   b. Write the manuscript in a plain text format that permits version control.

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

- jobs.ac.uk
- 10,000 jobs assessed
- ~400 related to software development (4%)
A new name is born

Software Sustainability Institute
Growth of RSE Community
RSEs are worldwide

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

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

[github.com/softwaresaved/international-survey/tree/master/analysis](https://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)

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

<table>
<thead>
<tr>
<th>Change Management</th>
<th>Recommendation</th>
<th>Comment</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>EAM.2</td>
<td>The most important information describing how to contribute to development are stored in a central location. (from application class 1)</td>
<td>Build steps are missing</td>
<td>todo</td>
</tr>
<tr>
<td>EAM.3</td>
<td>Known bugs, important unresolved tasks and ideas are at least noted in bullet point form and stored centrally. (from application class 1)</td>
<td></td>
<td>ok</td>
</tr>
<tr>
<td>EAM.7</td>
<td>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)</td>
<td></td>
<td>ok</td>
</tr>
<tr>
<td>EAM.8</td>
<td>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)</td>
<td></td>
<td>ok</td>
</tr>
</tbody>
</table>

### 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

- ELIXIR (Life Sciences): [https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5490478/](https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5490478/)
Software Development Best Practices for Life Sciences

• **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

develop ➔ share ➔ preserve

Developed and versioned using code repository
Published via code repository or website
Deposited in digital repository with paper / for preservation

Software Sustainability Institute
LIGO Example

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 $1.0 \times 10^{-22}$. 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 200,000 years, equivalent to a significance greater than 5.9 sigma. The source lies at a luminosity distance of $410^{+100}_{-150}$ Mpc corresponding to a redshift $z = 0.09^{+0.03}_{-0.02}$. In the source frame, the initial black hole masses are $36^{+5}_{-5} M_\odot$ and $29^{+4}_{-4} M_\odot$, and the final black hole mass is $62^{+5}_{-5} M_\odot$, with $3.0^{+0.1}_{-0.1} M_\odot$ 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...
“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”
## Barriers to Data and Code Sharing in Computational Science

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

<table>
<thead>
<tr>
<th>Code</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>77%</td>
<td>Time to document and clean up</td>
</tr>
<tr>
<td>52%</td>
<td>Dealing with questions from users</td>
</tr>
<tr>
<td>44%</td>
<td>Not receiving attribution</td>
</tr>
<tr>
<td>40%</td>
<td>Possibility of patents</td>
</tr>
<tr>
<td>34%</td>
<td>Legal Barriers (ie. copyright)</td>
</tr>
<tr>
<td>-</td>
<td>Time to verify release with admin</td>
</tr>
<tr>
<td>30%</td>
<td>Potential loss of future publications</td>
</tr>
<tr>
<td>30%</td>
<td>Competitors may get an advantage</td>
</tr>
<tr>
<td>20%</td>
<td>Web/disk space limitations</td>
</tr>
</tbody>
</table>
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
“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.”

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.

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

Examples of CoPs

- http://melbourne.resbaz.edu.au/HackyHour
- https://ourcodingclub.github.io/
- https://cookbook.carpentries.org/
- http://collections.plos.org/ten-simple-rules
Summary

• Increasing amounts of research software, increasing understanding of good practice
• Challenge is that research does not incentivize good practice
  ▪ So bottom-up training and top-down policies don’t work on their own
• Success has come from supporting formation of communities of practice, and sharing materials
  ▪ But this takes effort and goodwill
This is still going to be painful

Don't worry, you don't have to start your code from scratch.

You can re-use the software that the previous person on the project wrote several years ago.

Are there instructions for how to use it? I doubt it.

Is the code commented? Not likely.

Where are the files? Who knows.

This is going to be painful, isn't it?

Just a scratch.


Software Sustainability Institute
Find out more about the SSI

- 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

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

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

Software Sustainability Institute
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

Bringing together the right people to understand and address topical issues

Policy

Exploiting our platform to enable engagement, delivery & uptake

Possible, but difficult 21%