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, ESRC, NERC

Project funded by Jisc

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


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

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

Training

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

Outreach

Exploiting our platform to enable engagement, delivery & uptake

Bringing together the right people to understand and address topical issues

Policy

Exploiting our platform to enable engagement, delivery & uptake

Community

Exploiting our platform to enable engagement, delivery & uptake
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


- 92% Use software
- 69% Fundamental to results
- 56% Develop own software
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.

6%  9%  12%  18%  26%  50%  65%

Software in nature

“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
Software is important but often overlooked
Consequences of “incorrect” software can be large
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

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

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

“a homemade data-analysis program had flipped two columns”

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.
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
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>
<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>
“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
“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.”

Stoddin, 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”
## 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 (i.e. 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>
“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.”

One of the biggest challenges is educating our peers.
How can we get adoption of best practices at scale?
Introduction

Scientific software is a rapidly growing area of software development and is now an essential part of research in many fields. It is used to model complex systems, to analyze large datasets, and to support scientific computations. The quality of scientific software can significantly impact the accuracy and reliability of research findings. However, the development of scientific software often lacks the same level of rigor and discipline as other types of software development.

Best Practices for Scientific Computing

- **Documentation**: Ensure that all software is well-documented. This includes detailed documentation of the software's purpose, its inputs and outputs, and any limitations or assumptions.
- **Testing**: Implement comprehensive testing to ensure the software works as expected and to identify any bugs or issues.
- **Version Control**: Use version control systems to manage changes to the software and to track the evolution of the software over time.
- **Code Quality**: Follow coding best practices and use tools to maintain code quality. This includes using appropriate naming conventions, writing clear and readable code, and avoiding common pitfalls such as deadlocks or race conditions.
- **Security**: Address security concerns by implementing secure coding practices and regularly testing for vulnerabilities.
- **Data Management**: Ensure that data is properly managed and that data files are appropriately labeled and versioned. This includes using appropriate file formats and ensuring that data is properly backed up.
- **Collaboration**: Foster a collaborative environment where multiple developers can work together on the software. This includes using version control systems and tools to facilitate communication and collaboration.

Box 1. Summary of Best Practices

- **Documentation**: Ensure that all software is well-documented. This includes detailed documentation of the software's purpose, its inputs and outputs, and any limitations or assumptions.
- **Testing**: Implement comprehensive testing to ensure the software works as expected and to identify any bugs or issues.
- **Version Control**: Use version control systems to manage changes to the software and to track the evolution of the software over time.
- **Code Quality**: Follow coding best practices and use tools to maintain code quality. This includes using appropriate naming conventions, writing clear and readable code, and avoiding common pitfalls such as deadlocks or race conditions.
- **Security**: Address security concerns by implementing secure coding practices and regularly testing for vulnerabilities.
- **Data Management**: Ensure that data is properly managed and that data files are appropriately labeled and versioned. This includes using appropriate file formats and ensuring that data is properly backed up.
- **Collaboration**: Foster a collaborative environment where multiple developers can work together on the software. This includes using version control systems and tools to facilitate communication and collaboration.

Write Programs for People, Not Computers

Software engineers should strive to make code that is both maintainable and easy to read and understand. Other programmers and users should be able to read and understand the code without needing extensive documentation. This improves the efficiency of the development process and reduces the time required for maintenance and updates.
Good enough practices in scientific computing

Greg Wilson, Jennifer Bryan, Karen Cranston, Justin Kitses, Lex Nederbragt, Tracy K. Teo

1 Software Carpentry Foundation, Austin, Texas, United States of America, 2 7iBiS 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.
* gwilson@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 col-
leagues, 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 con-
der 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 5 years, and from a variety of other guides. Our recom-
mendations are aimed specifically at people who are new to research computing.

Box 1: Summary of Practices

1. Data Management
   - Sort the raw data.
   - Create data you wish to see in the world.
   - Create analysis-friendly data.
   - Record all the steps used to process data.
   - Anticipate the need to use multiple tables.
   - Submit data to a reputable DGO-testing repository so that others can access and cite it.

2. Software
   - Place a brief explanatory comment at the start of every program.
   - Decompose programs into functions.
   - Be ruthless about eliminating duplication.
   - Always search for well-maintained software libraries that do what you need.
   - Test libraries before relying on them.
   - Give functions and variables meaningful names.
   - Make deprecation and requirements explicit.
   - Do not commit and subsequent versions of code to control a program’s behavior.
   - Provide a simple example or test data set.
   - Submit code to a reputable DGO-testing repository.

3. Collaboration
   - Create a shared public “code” list.
   - Make the license explicit.
   - Make the project citable.

4. Project Organization
   - Put each project in its own directory, which is named after the project.
   - Put test documents associated with the project in the data directory.
   - Put raw data and metadata in a data directory, and files generated during cleanup and
     analysis in a results directory.
   - Put project source code in the src directory.
   - Put external scripts, or compiled programs in the bin directory.
   - Name all files to reflect their content or function.

5. Keeping Track of Changes
   - Back up (almost) everything created by a human being as soon as it is created.
   - Keep changes small.
   - Store changes frequently.
   - Create, maintain, and use a checklist for going and sharing changes to the project.
   - Store each project in a folder that is mirrored off the researcher’s working machine.
   - Use a file named .GITHUB to record changes, and
   - Copy the entire project whenever a significant change has been made, or
   - Use a version control system to manage changes

6. Manuscripts
   - Write manuscripts using online tools with rich formatting, change tracking, and
     reference management, or
   - Write the manuscript in a plain text format that permits version control.
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”
Growth of RSE Community

RSE Groups

- Alan Turing Institute
- University of Bath
- University of Birmingham
- University of Bristol
- University of Cambridge
- Culham Centre for Fusion En...
- Imperial College London
- Francis Crick Institute
- University of Leeds
- University of Leicester
- The University of Manchester
- Newcastle University
- The University of Sheffield
- University of Southampton
- University College London
- EPCC
- Daresbury Laboratory
- Oxford eResearch Centre
- Software Engineering Suppo...

1211 members
29 April 2018

Slide courtesy of Simon Hettrick
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
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?
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

<table>
<thead>
<tr>
<th>Change Management</th>
<th>Recommendation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>EAM.2: The most important information describing how to contribute to development are stored in a central location. (from application class 1)</td>
</tr>
<tr>
<td></td>
<td>EAM.5: Known bugs, important unresolved tasks and ideas are at least noted in bullet point form and stored centrally. (from application class 1)</td>
</tr>
<tr>
<td></td>
<td>EAM.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)</td>
</tr>
<tr>
<td></td>
<td>EAM.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)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Reasoning and further advice</th>
</tr>
</thead>
<tbody>
<tr>
<td>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.</td>
</tr>
<tr>
<td>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:</td>
</tr>
</tbody>
</table>
Community standards

- CLARIAH (Arts and Humanities): https://github.com/CLARIAH/software-quality-guidelines
- ELIXIR (Life Sciences): 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

Software Sustainability Institute
Research Software Workflow

develop → share → preserve

describe

Developed and versioned using code repository

Published via code repository or website

Deposited in digital repository with paper / for preservation Made citable

Software Sustainability Institute
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 $1.0 \times 10^{-20}$. 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$^{+130}_{-90}$ Mpc corresponding to a redshift $z = 0.09^{+0.06}_{-0.04}$. In the source frame, the initial black hole masses are $36^{+6}_{-5} M_\odot$ and $29^{+7}_{-6} M_\odot$, and the final black hole mass is $62^{+11}_{-9} M_\odot$, with $3.0^{+0.5}_{-0.6} M_\odot c^2$ 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
Adoption of best practice comes from making it easy and useful for everyone.
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


Software Sustainability Institute
Examples of CoPs

Sheffield Astrophysics Code Review Club
https://ourcodingclub.github.io/

https://cookbook.carpentries.org/

https://collections.plos.org/ten-simple-rules

Software Sustainability Institute
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.

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

Software Sustainability Institute

Piled Higher and Deeper” by Jorge Cham
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

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

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

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