Doing Science Properly in the Digital Age

Doing Science Properly in the Digital Age

Neil Chue Hong

Software Sustainability Institute & University of Edinburgh, JCMB, Mayfield Road, Edinburgh, EH9 3JZ.

Background

Science has changed dramatically over the last seventy years. The rise of computational simulation and data-intensive research – the third and fourth paradigms of science – to take their place alongside the established theory and experimental methods has ushered in a new digital age of science. This has led to pervasive use of software across most scientific disciplines, ranging from monolithic codes to web-based execution environments.

With this increase in computation and data come associated challenges. How do we deal with the increasing amounts of data? As the size and geographical distribution of collaborations increases, how do we ensure that the network does not become more fragile? Can we record all the metadata associated with each step of our research workflow? Most importantly, can we convince our peers that the outcomes of our research are useful?

In my work with the Software Sustainability Institute, I have had the opportunity to collaborate with both researchers developing and using software, as well as researchers studying those who develop and use software. This position paper examines some of the issues and approaches to dealing with scientific software as a ubiquitous part of research.

The challenge facing the new generations of researchers is clear: what does it mean to say that we are doing science properly in the digital age?

Reproducible Research

In the medical field of drug design, controlled trials have been used since James Lind's study on the prevention of scurvy in 1747. For decades "evidence based medicine" and randomised double-blind trials are accepted as the gold standard for medical research. And yet even in this field, studies have shown that reproducibility of published results is poor [Beg12] [Pri11].

The term *reproducible research* [Fom09] was coined by Jon Claerbout to refer to the idea that the ultimate product of research is the traditional paper output along with the full computational environment used to produce the results in the paper such as the code, data and methods necessary for reproduction of the results and building upon the research. Whilst great strides have been made to ensure that we can preserve the data sets along with the paper, less has been done in terms of understanding how to preserve the software and its related environment (with some notable exceptions [Yale10] [Dam11] [Now11]).

To paraphrase from Patrick Vandewalle, Jelena Kovacevic and Martin Vetterli's Reproducible Research website, reproducibility is important because:

- It helps the researcher to reproduce figures in revisions of a paper, and to recreate earlier results at a later stage;
- It helps other researchers working in the same or a related field to start from the current state of the art, instead of spending time trying to figure out what was exactly done in a certain paper;
- It highly simplifies the task of comparing a new method to existing methods. Results can be compared more easily, and one is also sure that the implementation is the correct one.

What this ultimately means is that it makes you able to trust the research.

This is not an easy task. [Peng11] suggests a reproducibility spectrum going from publication only, through publication + code, publication + code + data, publication + linked and executable code and data, to the gold

standard of full replication. However it is not always easy to identify how to achieve each step because of the nature of software.

In general, when data is deposited in a repository, that specifies its granularity – the dataset can be considered as a unique object. It may consist of a collection of pieces of data that have distinct characteristics (e.g. an album is well-defined as a collection of songs) but importantly there is a point in time at which the distinction is made. Software has become harder to define, in particular as software has made the leap from a single machine to a distributed system. What is the "code" here: source code, binaries, workflows, manuals, web services? And how much should we consider in terms of dependencies such as operating systems and hardware characteristics?

One effective approach is to provide tooling that makes it easier to meet requirements placed by funders (e.g. the DMPOnline tool from the Digital Curation Centre for data management plans) and use this to engender culture change by then offering training to improve the long term preservation and curation: also an issue for software.

Reuse and Reward – Encouraging Software Specialists

Another way of doing science properly in the digital age is to do it with the right people. Just as any member of a typical research team has specialist skills, so research using software has specialist roles [How11].

Whilst there is a wide spectrum (discussed more fully in a separate position paper on the Research Software Engineer submitted to Digital Research 2012), at the two ends of this spectrum we see:

- the "Researcher-Developer" who wants to be judged on their scientific output, is a researcher at heart, however develops a lot of code due to the nature of their research. They would like recognition comparable to a research paper for a software implementation taken up by others.
- the "Research Software Engineer" comes from a research background but is also a skilled software developer, and relishes challenge of not just developing code to solve a problem but doing it well. They want to be recognised for producing tools which others rely on for research.

Nevertheless, the current academic reward structures emphasise publications as the primary research output, and peer review promulgates this.

This highlights the current issues surrounding software specialists: across the spectrum, they are not being rewarded for doing good work, and thus their career progression is less substantial as well. Research communities risk losing knowledge as people migrate to more rewarding jobs in industry or other professions.

Work has been done in [How11] to understand the incentive structures used in different areas of science. There has also been work done that shows that open data sharing leads to improved academic recognition through traditional citation mechanism [Pio07] [Hen11] though as yet the studies have not been reproduced for open software – something which the SSI seeks to do in the future building on the authors previous work [Chu11] which established five tenets of research software:

1. Open science is a fundamental requirement for the overall improvement and achievement of scientific research.
2. Open science is built on the tenets of reuse, repurposing, reproducibility and reward.
3. Software has become the third pillar of research, supporting theory and experiment.
4. Current mechanisms for measuring impact do not allow the impact of software to be properly tracked in the research community.
5. We must establish a framework for understanding the impact of software that both recognises and rewards software producers, software users and software contributors; and encourages the sharing and reuse of software to achieve maximum research impact.

Ultimately, a change in culture is required so that software is considered as a first-level research output, something recognised by the Science Code Manifesto [SCM]:

"Software is an essential research product, and the effort to produce, maintain, adapt, and curate code must be recognized. Software stands among other vital scientific contributions besides published papers. "

Better science through superior software

A final way to do science properly in the digital age is to consider how best practice from other areas might fit the needs of the researcher. In particular can we improve the quality, efficiency and reproducibility of scientific software by adhering to software engineering best practice?

There have been many studies on the effectiveness of software engineering techniques, for instance on the evolution of software development models [Boe06]. Relatively few ([Kel07] [Han09] [Sle11] [Wil09]) have investigated the relationship between and use of software engineering and scientific software.

A particular issue in comparing "traditional" commercial software development and "conventional" scientific software development is that many software engineering practices expect known requirements and customer-focussed timescales. In contrast, cutting-edge research by definition is exploratory and experimental in nature, making it harder – though not impossible – to define tests and milestones. Combined with approaches to teaching that tend to emphasise learning a language rather than teaching researchers why programming in particular ways helps their research, and it is no surprise that the experience and skills across the UK research community vary widely.

In contrast, the pedagogical approach taken by Software Carpentry [Wil09] has proved effective [Ara12] in teaching the basic principles of software engineering *as they apply* to researchers. This in turn means they are in a better position to develop and use software effectively. As C Titus Brown says: "Better science through superior software".

Conclusions

It can be seen from this position paper that *doing science properly in the digital age* is no different from doing science properly in any age. The key goal is to enable yourself and others to place *trust* in the outputs of your research.

To enable this trust I propose three pillars:

- Reproducible research: making it easy for people to record and validate their experiments and results, and for readers to reproduce results;
- Recognition of software as a research output: making it possible for skilled software specialists to progress in a career in research;
- Software skills: a proper foundation in the fundamentals of good software engineering as they apply in research should form part of every researcher's basic training.

With these in place we can prepare to tackle the next network age, where the boundaries of research collaboration are radically redrawn.

Acknowledgements

This paper would not have been possible without the input of and information from my colleagues at the Software Sustainability Institute, in particular Rob Baxter, Steve Crouch, David De Roure, Carole Goble, Mike Jackson; and also discussions with Nick Barnes, David Gavaghan, Brian Hole, Brian Matthews, Cameron Neylon, Mark Plumbley, Tom Pollard, Jennifer Schopf, and Greg Wilson. The Software Sustainability Institute is supported by the EPSRC under grant EP/H043160/1.

References

[Ara12] Jorge Aranda, Software Carpentry Assessment Report (2012). Retrieved from http://software-carpentry.org/blog/wp-content/uploads/2012/07/aranda-assessment-2012-07.pdf on 27th July 2012.

[Beg12] C. Glenn Begley & Lee M. Ellis, Drug development: Raise standards for preclinical cancer research. Nature 483, 531–533 (2012). doi:10.1038/483531aS

[Boe06] Barry Boehm, A View of 20th and 21st Century Software Engineering, Proceedings of the 28th international conference on Software engineering, pp. 12 – 29. 2006. doi: 10.1145/1134285.1134288

[Chu11] Neil P. Chue Hong, Software Impact – the differences from datasets. Beyond Impact (2011). Retrieved from http://beyond-impact.org/?p=175 on 27th July 2012.

[Dam11] I. Damnjanovic, L. A. Figueira, C. Cannam, and M. D. Plumbley, "SoundSoftware.ac.uk Survey Report," http://code.soundsoftware.ac.uk/documents/17, 2011.

[Fom09] Sergey Fomel and Jon Claerbout, "Guest Editors' Introduction: Reproducible Research," Computing in Science and Engineering, vol. 11, no. 1, pp. 5–7, Jan./Feb. 2009, doi:10.1109/MCSE.2009.14

[Han09] Jo Erskine Hannay, Hans Petter Langtangen, Carolyn MacLeod, Dietmar Pfahl, Janice Singer, and Greg Wilson. How Do Scientists Develop and Use Scientific Software? In Second International Workshop on Software Engineering for Computational Science and Engineering, 2009.

[Hen11] E. A. Henneken and A. Accomazzi, "Linking to Data - Effect on Citation Rates in Astronomy," PLoS Biology, no. May, p. 4, 2011.

[How11] James Howison and James D Herbsleb. "Scientific software production and collaboration." Source 09 (2011) : 513–522. DOI: 10.1145/1958824.1958904

[Kel07] D.F. Kelly, "A Software Chasm: Software Engineering and Scientific Computing," IEEE Software, vol. 24, no. 6, 2007, pp. 118–120. DOI: 10.1109/MS.2007.155

[Now11] Piotr Nowakowski, Eryk Ciepiela, Daniel Harężlak, Joanna Kocot, Marek Kasztelnik, Tomasz Bartyński, Jan Meizner, Grzegorz Dyk, Maciej Malawski, The Collage Authoring Environment, Procedia Computer Science, Volume 4, 2011, Pages 608-617, ISSN 1877-0509, 10.1016/j.procs.2011.04.064.

[Peng11] Roger D Peng, Reproducible Research in Computational Science, Science 334, 1226 (2011). DOI: 10.1126/science.1213847

[Piw07] H. A. Piwowar, R. S. Day, and D. B. Fridsma, "Sharing Detailed Research Data Is Associated with Increased Citation Rate," PLoS ONE, vol. 2, no. 3, p. 5, 2007.

[Pri11] Florian Prinz1, Thomas Schlange2 & Khusru Asadullah3, Believe it or not: how much can we rely on published data on potential drug targets?, Nature Reviews Drug Discovery 10, 712 (September 2011) | doi:10.1038/nrd3439-c1

[SCM] Science Code Manifesto discussion. Retrieved from http://sciencecodemanifesto.org/discussion on 27th July 2012.

[Sle11] Magnus Thorstein Sletholt, Jo Erskine Hannay, Dietmar Pfahl, Hans Petter Langtangen, "What Do We Know about Scientific Software Development's Agile Practices?," Computing in Science and Engineering, vol. 14, no. 2, pp. 24-37, March-April 2012, doi:10.1109/MCSE.2011.113

[Sto09] V. Stodden, The Legal Framework for Reproducible Scientific Research: Licensing and Copyright, Computing in Science and Engineering, vol. 11, no. 1, pp. 35-40, Jan./Feb. 2009, doi:10.1109/MCSE.2009.19

[Mor12] A. Morin, J. Urban, P. D. Adams, I. Foster, A. Sali, D. Baker, and P. Sliz, Shining Light into Black Boxes, Science 336 (6078), 159-160 (2012). DOI:10.1126/science.1218263

[Wil06] Greg Wilson. Software Carpentry: Getting Scientists to Write Better Code by Making Them More Productive. Computing in Science & Engineering, November-December 2006.

[Wil09] Greg Wilson and Andrew Lumsdaine. Software Engineering and Computational Science. Computing in Science & Engineering, 11(6):12–13, 2009. DOI:10.1109/MCSE.2009.206

[Yale10] Yale Law School Roundtable on Data and Code Sharing, "Reproducible Research," Computing in Science and Engineering, vol. 12, no. 5, pp. 8-13, Sep./Oct. 2010, doi:10.1109/MCSE.2010.113