Cronfa - Swansea University Open Access Repository

_____

This is an author produced version of a paper published in:
Journal of Open Research Software

_____

Cronfa URL for this paper:

_____

Paper:

_____

ISSUES IN RESEARCH SOFTWARE

Reproducibility in Research: Systems, Infrastructure, Culture

Tom Crick[1], Benjamin A. Hall[2] and Samin Ishtiaq[3]

[1]Cardiff Metropolitan University, UK
[2]University of Cambridge, UK
[3]Microsoft Research Cambridge, UK
Corresponding author: Tom Crick (tcrick@cardiffmet.ac.uk)

The reproduction and replication of research results has become a major issue for a number of scientific disciplines. In computer science and related computational disciplines such as systems biology, the challenges closely revolve around the ability to implement (and exploit) novel algorithms and models. Taking a new approach from the literature and applying it to a new codebase frequently requires local knowledge missing from the published manuscripts and transient project websites. Alongside this issue, benchmarking, and the lack of open, transparent and fair benchmark sets present another barrier to the verification and validation of claimed results.

In this paper, we outline several recommendations to address these issues, driven by specific examples from a range of scientific domains. Based on these recommendations, we propose a high-level prototype open automated platform for scientific software development which effectively abstracts specific dependencies from the individual researcher and their workstation, allowing easy sharing and reproduction of results. This new e-infrastructure for reproducible computational science offers the potential to incentivise a culture change and drive the adoption of new techniques to improve the quality and efficiency – and thus reproducibility – of scientific exploration.

1 Introduction

Marc Andreessen (co-author of Mosaic, the first widely used web browser) boldly stated in 2011 that *"software is eating the world"* [1]. This is true: we live in a computational world, with our everyday communications, entertainment, shopping, banking, transportation, national security, etc, all heavily data-driven and largely overtaken by software.

Andreessen's statement is particularly true for science and engineering. A 2012 report by the UK's Royal Society stated that computational techniques have *"moved on from assisting scientists in doing science, to transforming both how science is done and what science is done"* [2]. Many of the examples discussed in this paper exploit a fundamental advantage of computer science and more generally, computational science: the unique ability for researchers to share the raw outputs of their work as software and datafiles. New experiments, simulations, models, benchmarks, even proofs increasingly cannot be done without software. This software does not consist of simple hack-together, use-once, throw-away scripts; research software repositories contain thousands, perhaps millions, of lines of code and they increasingly need to be actively supported and maintained. More importantly,

with reproducibility being a fundamental tenet of science, they need to be open and re-useable.

However, if we closely analyse the scientific literature related to software tools it often does not appear to be adhering to these rules [3, 4]. How many of them are open and available? How many explain their experimental methodologies, in particular the basis for their benchmarking? In particular, can we (re)build the code? [5] We, the authors, are perhaps as guilty as anyone in the past, where we have published papers [6, 7] with benchmarks and promises of code to be released in the near future which depreciate as you move onto the next project.

There are various reasons why the wider scientific community is in this position. We are currently undergoing significant changes to models of academic dissemination, especially considering the wider open research movement, with new models being proposed [8–10]. Now, numerous "high-impact" journals explicitly require that source code and data is made available online under some form of open source license, but there still exists large disciplinary gaps. While these initiatives are great, they are often optional, seem piecemeal, and do

little to enable the verification and validation of scientific results at a later stage. Even within the same field, there are different ideas of what defines reproducibility [11], as well as evidence of "overturn bias" – replications that overturn original results are much easier to publish than those that confirm original results [12].

Nevertheless, the reproduction and replication of reported scientific results has become a widely discussed topic within the scientific community [13–15]. Whilst the increasing number of retractions of studies across a variety of disciplines has drawn the focus of many commentators, automated systems, which allow easy reproduction of results, offer the potential to improve the efficiency of scientific exploration and drive the adoption of new techniques. However, just publishing (linked) scientific data is not enough to ensure the required reusability [16]. There exists a wider socio-cultural problem that pervades the scientific community, with estimates that as much as 50% of published studies, even those in top-tier academic journals, cannot be repeated with the same conclusions by an industrial lab [17, 18]. There are numerous non-technical impediments to making software maintainable and re-useable. The pressure to "make the discovery", publish quickly and move onto the next project disincentivises careful software curation and preservation. Releasing code prematurely is often seen to give your competitors an advantage, but we should be shining light into these "black boxes" [14]; in essence: better software, better research [19].

However, there is promising existing work in this area [20–23], with a variety of manifestos for reproducible research and community initiatives [24–29], top tips and "ten simple rules" [30–36], as well as analysis of the wider legal, professional, ethical and risk perspectives [37, 38]. Things can, should and need to be much better if we want to uphold and maintain the scientific tenets of openness and sharing. Building upon previous work [39, 40], we present a call to action, along with a set of recommendations which we hope will lead to better, more sustainable, more re-useable software, to move towards an imagined future practice and usage of scientific software development. We also propose a high-level specification for a service that would automate many of our recommendations.

2 We Need to Talk About Reproducibility
2.1 Can I Implement Your Algorithm?
Reproducibility is a fundamental tenet of high-quality research. Yet many descriptions of algorithms are too high-level, too obscure, too poorly-defined to allow an easy re-implementation by a third party. A step in the algorithm might say: *"We pick an element from the frontier set"* but which element do you pick? Will the first one do? Why will any element suffice? Sometimes the author would like to give more implementation detail but is constrained by an arbitrary page limit of a conference or journal paper. Sometimes the authors' description in-lines other algorithms or data structures that perhaps only that author is familiar with.

Until recently, reproducibility was only discussed at conferences and workshops convened explicitly for that purpose. This is changing, and a number of high-profile computer science venues such as the ACM SIGPLAN conferences POPL and PLDI now explicitly acknowledge the importance of reproducibility, promoting community-driven reviewing and validation of software artefacts.

Recommendation I: We recommend that a paper must describe the algorithm in such a way that it is implementable by any knowledgeable reader of that algorithm. The description is, of course, subjective, but to help encourage better descriptions, we also recommend that — in addition to having incentives to support sharing of computational artefacts — relevant scientific conferences develop special tracks for papers that re-implement past papers' algorithms, techniques or tools.

2.2 Set The Code Free
There can be no better proof of your algorithm working, than if you provide the source code of an implementation; software development is hard, but sharing and re-using code is relatively easy.

Many years ago, Richard Stallman (founder of the GNU Project and Free Software Foundation) postulated that all code would be free [41] and we would make our money by consulting on the code. As it turns out, this is now the case for a significant part of the computing industry. There are, of course, hard commercial pressures for keeping code closed-source. Even in the scientific domain, scientists and their collaborators may wish to hold onto their code as a competitive advantage, especially if there exists larger competitors who could use the available code to "reverse scoop" the inventors, charging into a promising new research area opened by the inventors.

Closed source is one thing; licenses that deny the user from viewing, modifying, or sharing the source are another thing. There are, however, even licences on widely adopted tools like Gaussian [42] (for computational chemistry) that prohibit even analysing software performance and behaviour. For example, a wide variety of licenses exist for molecular dynamics software, with different degrees of openness e.g. Gromacs uses the GNU Lesser General Public License (LGPL) [43], CHARMM and Desmond are Academic/Commercial software licences [44, 45], Amber and NAMD are custom open-like licences. Z3 is an example from the verification area: the code itself was only recently open sourced, but the previous MSR-LA license allowed the source code to be read, copied, forked for academic use, providing researchers in the field substantial flexibility [46].

Even ignoring licensing issues, sometimes the source is not made open because the author thinks that it is not quite finished. You should follow the "release early, release often" mantra, as well as releasing somewhere public like GitHub, where it is easy to share and fork. Your code is good enough [13].

Recommendation II: There is little doubt that, if scientific research wants to be open and free, then the code that underlies it too needs to be open and free. Code that is available for browsing, modifying, and forking, facilitates testing and comparison. We recommend that code be published under

an appropriate open source license [47]; while we defer legal discussion of the specifics of any particular licences, BSD and Apache are good, flexible ones.

2.3 Be A Better Academic Citizen

If you have the appropriate knowledge, skills and experience, you can create better software. We have seen the emergence of successful initiatives, such as the Software Sustainability Institute (http://www.software. ac.uk) and the UK Community of Research Software Engineer (http://www.rse.ac.uk), in cultivating world-class research through software, developing software skills and raising the profile of research software engineers.

Many scientists will not have had any formal, or even informal, training in scientific software development. Building upon the work of Software Carpentry (http:// software-carpentry.org) and Data Carpentry (http://www. datacarpentry.org), basic training in software engineering concepts like version control (git, mercurial), unit testing (tests written to exercise the smallest testable parts of a system, like a function exported from a module), regression testing (a test framework that ensures that previous results are maintained over the changes in the source code), build tools (Make, scons), etc, can help improve the quality of the software written enormously [48]. Interestingly, many of these concepts are taught to computer science undergraduates, but it could be argued that they are taught at the wrong time of their careers, without the experience of complex, long-running projects.

> **Recommendation III:** Software development skills should be regarded as fundamental literacies for scientists and engineers: we recommend that formal programming, data and computational skills are taught as core at undergraduate and postgraduate level.

2.4 The Lingua Franca of Computational Research

There is no other scientific or technical field where its participants can just make up a non-principled artefact like a programming language so easily. In a way, it shows how much of a "commons" computer science has become, that anyone can create a new programming language, API, framework or compiler. This clearly has its advantages and disadvantages.

High-level languages are generally more readable than their low-level relations. The "density" of a program is often seen to be a good thing, but it is not always the case that a shorter Haskell program (for example) is easier to maintain than a longer C++ one; what is important is the readability of the code itself. A good example here is from the world of automatic theorem proving: the SSReflect language is much more readable than the original, standard Coq language [49]. SSReflect uses mathematicians' vernacular for script commands, allows reproducibility of automatic proof-checking because parameters are named rather than numbered. Even though these proof scripts are really only ever going to be run by a machine, they seek to maintain the basic mathematical idea that a proof should be readable by another mathematician.

Many high-level programming languages impose constraints like types: that you can never add a number and a string is the most basic example, but ML's functors provide principled ways of plugging in components with their implementations completely hidden. Aggressive type checking avoids a subset of bugs which can arise due to incorrectly written functions e.g. well publicised problems with a NASA Mars orbiter (http://www.cnn. com/TECH/space/9909/30/mars.metric.02/). A further example is a pressure coupling bug (http://redmine. gromacs.org/issues/14) in Gromacs [43], which arose due to the inappropriate swapping of a pressure term with a stress tensor. A further extension of types, a concept called units of measure that is implemented in languages such as F#, can deal with these kinds of bugs at compile time. Similarly, problems found using in-house software for crystallography led to the retraction of five papers [50], due to a bug which inverted the phases.

> **Recommendation IV:** The use of a principled, high-level, typed programming language in which to write your software helps hugely with the maintainability, robustness and openness of the software produced. Even in web frontend work, you have choices: use Typescript or Flow rather than plain old Javascript; use Hack rather than PHP.

2.5 Lineage (or: "Standing On The Shoulders Of Giants")

Research software is not just software – it is the instantiation of novel algorithms and data structures (or at least novel applications of data structures). Thus, lineage is important:

> **Recommendation V:** Code should always include links to papers publishing key algorithms and the code should include explicit relationships to other projects on the repository (i.e. *Project B* was branched from *Project A*). This ensures that both the researchers and software developers working upstream of the current project are properly credited, encouraging future sharing and development. Remember, the people who did the research are not necessarily the same people as the developers and maintainers of the software, so it is important to reward both appropriately with citations: take note of the FORCE11 Software Citation Principles [51].

2.6 YMMV

The tweet in **Figure 1** is satirical but worryingly true, highlighting the perils of reproducible research. Often, the tool that the paper describes does not exist for download. Or runs only on one particular bespoke platform. Or might run for the author, for a while, but will 'bit-rot' so quickly that even the author cannot compile it the following year. Computational reproducibility would appear to be more straightforward than replicating physical experiments, but the complex and rapidly changing nature of computer systems and environments that are being used across different disciplines makes being able to reproduce and extend such work a serious challenge [52].

Figure 1: `#overlyhonestmethods` on Twitter by @ianholmes [*source*: https://twitter.com/ianholmes/status/288689712636493824].

> **Recommendation VI:** You must provide the source code of the tool, but also with details of precisely *how* you built and wrote the software. For example:
>
> · You should provide the compiler and build toolchain;
> · You should provide build tools (e.g. Makefiles/Ant/etc) and comprehensive build instructions;
> · You should list or link to all non-standard packages and libraries that you use;
> · You should note the specifics of the hardware and OS used.
>
> This may appear to be significant extra overhead for researchers, but GitHub APIs, continuous integration servers, virtual machines and cloud environments can make it easier; see Section 3 for more on this.

2.7 Data Representations and Formats

We often do not, and should not, care how things are stored on disk, what their precise representations are. A common, constrained, standard representation is however good for passing tests or models around between different tools. A properly described representation, like the SMT-LIB format (http://smt-lib.org) for Satisfiability Modulo Theory (SMT) solvers, where both the syntax and semantics are well understood, hugely aids developing tools, techniques and benchmarks.

Another example, from biology, is that of the standard representation of qualitative networks and Boolean networks [53, 54]. These networks can be expressed in SMV format, but this would mean that standard qualitative/Boolean network behaviours have to be hard-coded for each variable, introducing the possibility for errors. In the BioModelAnalyzer tool [55], the JSON contains *only* the modifiable parameters limiting the possibility for error; the SBML-Qual standard achieves a similar goal for logical models [56].

> **Recommendation VII:** Avoid creating new representations when common formats already exist. Use existing extensible internationally

standardised representations and formats to facilitate sharing and re-use.

2.8 World Records

The benchmarks the tool describes are fashioned only for this instance of this time. They might claim to be from the Microsoft Windows device driver set, but the reality is that they are stripped down versions of the originals. Stripped down so much as to be useless to anyone but the author vs. the referee. It is worse than that really: enough benchmarks are included to beat other tools. The comparisons are never fair (especially for comparisons against your tool). If every paper has to be novel, then every benchmark, too, will be novel; there is no monotonic, historical truth in new, synthetically-crafted benchmarks. It is as if, in order to beat Usain Bolt's 100m world record time, you make him race overweight and out of season, with a winter overcoat and the wrong sized shoes. Given this setup, you could surely hope to beat his 9.58s time on a shorter length track.

> **Recommendation VIII:** Benchmarks should be public. They should allow anyone to contribute, implying that the tests are in a standard format. Further, these benchmarks must be heavily curated. Every test/assertion should be justified. Papers should be penalised if they do not use these public benchmarks. While there are some domains in which it may not be immediately possible to share full benchmarks sets, this should be the exception (with justification) rather than the norm.

A good example of some of these points is the RCSB Protein Data Bank (http://www.pdb.org) and Systems Biology Markup Language [56]. The software ones we know of, the SMT Competition (http://smtcomp.sourceforge.net/2014/), SV-COMP (http://sv-comp.sosy-lab.org/2015/) and Termination Problems Data Base (http://termination-portal.org/wiki/TPDB) are on that journey. Such repositories would allow the tests to be taken and easily analysed by any competitor tool. Some communities go further; the Critical assessment of methods of protein structure prediction and prediction of interactions (CASP and CAPRI) [57, 58] communities present a single-blind test of protein folding and docking algorithms annually, allowing open competition on a level playing field. Similarly the DREAM challenges (http://dreamchallenges.org/) attempt to address large scale problems through open competition.

2.9 Test It To See

Some models may be chaotic and influenced by floating-point errors (e.g. molecular dynamics), further frustrating testing. For example: Sidekick is an automated tool for building molecular models and performing simulations [59]. Each system is simulated from an different initial random seed, and under most circumstances this is the only difference expected between replicas. However, on a mixed cluster with both AMD and Intel microprocessors on the nodes, the difference in architecture was found to

alter the number of water molecules added to each system by one. This meant that the same simulation performed on different architectures would diverge. Similarly, in a different simulation engine, different neighbour searching strategies gave divergent simulations due to the differing order in which forces were summed.

A further example is the handling of pseudo-random number generation in Avida [60], an open source scientific software platform for conducting and analysing experiments with self-replicating and evolving computer programs. While it may initially appear attractive to develop bespoke random number generators within a system for consistency or performance across platforms, this invariably adds complexity to your system and may inhibit sharing and reproducibility.

> **Recommendation IX:** Despite these challenges to testing, unshared code is ultimately untestable. Testing new complex scientific software is difficult – until the software is complete, unit tests may not be available. You should aim to re-use modules or repos (git submodules) from publicly-shared code; a corollary of Linus's Law ("given enough eyeballs, all bugs are shallow") might be that shared code is inherently more test-able.

2.10 Welcome to Web 2.0

Virtual machines (VMs) in the cloud also make the testing of scaling properties more simple. If you have a tool that you claim is more efficient, you could put together a cluster of slow nodes in the cloud to demonstrate how well the software scales for parallel calculations. Cloud computing is cheap, and getting cheaper. Algorithms that used to require massive HPC resources can now be run cheaply by bidding on the VM spot market. The web is a great leveller: use and share workflows and web services [61, 62].

> **Recommendation X:** The web and the cloud really do open up a whole new way of working. Even small, seemingly trivial features like putting up a web interface to your tool and its tests will allow users who are not able to install necessary dependencies to explore the running of the tool [63]. Ultimately, this can lead to making an "executable paper" appear on the Internet. The interactive *Try F#* (http://www.tryfsharp.org/Learn) and Z3 tutorials (http://rise4fun.com/Z3/tutorial/guide) are a great start that begin to expose what can be done in this area.

3 A Model for Reproducible Research Software

Some of our Recommendations, such as *"Be A Better Person"* or *"The Lingua Franca"*, are abstract, airy-fairy, pie-in-the-sky even. However, most of them can be concretely realised by a service for reproducibility. This service provides a concrete implementation of free source code (*"Set The Code Free"*) that depends on other free source code (*"Lineage"*) building (*"YMMV", "Welcome to Web 2.0"*) and running tests contributed in public (*"Data Representations", "World Records"*) in a completely reproducible regime.

The service we describe here can be seen as a specification. We have not built it, but many services like travis-ci or Azure VSTS provide some of the mechanical parts of it. A service for reproducibility is intended to play three important roles; it should:

i) Demonstrate that a piece of code can be compiled, run and behaves as described, without manual intervention from the developer;
ii) Store and link specific artefacts with their linked publications or other publicly-accessible datasets;
iii) Allow new benchmarks to be added, by users other than the developer, to widen the testing and identify potential bugs.

The whole premise of our previous paper [40] is that *algorithms* (and their *implementations*) and *models* (sometimes also called *benchmarks*) are inextricably linked. Algorithms are designed for certain types of models; models, though created to mimic some physical reality, also serve to express the current known algorithms. An integrated autonomous open cloud-based service can make this link explicit.

By developing a cloud-based, centralised service, which performs automated code compilation, testing and benchmarking (with associated auditing), we will link together published implementations of algorithms and input models. This will allow the prototype to link together software and data repositories, toolchains, workflows and outputs, providing a seamless automated infrastructure for the verification and validation of scientific models and in particular, performance benchmarks. The program of work will lead the cultural shift in both the short and long-term to move to a world in which computational reproducibility helps researchers achieve their goals, rather than being perceived as an overhead.

A system as described here has several up-front benefits: it links research papers more closely to their outputs, making external validation easier and allows interested users to explore unaddressed sets of models. Critically, it helps researchers across computational science to be more productive, rather than reproducibility being an overhead on top of their day-to-day work. In the same way that tools such as GitHub make collaborating easier while simultaneously allowing effortless sharing, we envisage our system being similarly usable for sharing and testing algorithms and their implementations, software, models and benchmarks online.

Suppose you have come up with a better algorithm to deal with some standard problem. You write up the paper on the algorithm, and you also push an implementation of your algorithm to the our cloud environment's section on this standard problem. The effect of pushing your implementation is to register your program as a possible competitor in this standard problem competition. There exist several dozen widely-agreed tests on this problem already on our cloud environment's database. Maybe, after some negotiation due to your novel approach to this standard problem, you add some of your own tests to the database too.

Pushing your code activates the environment's continuous integration system. The cloud pulls in all
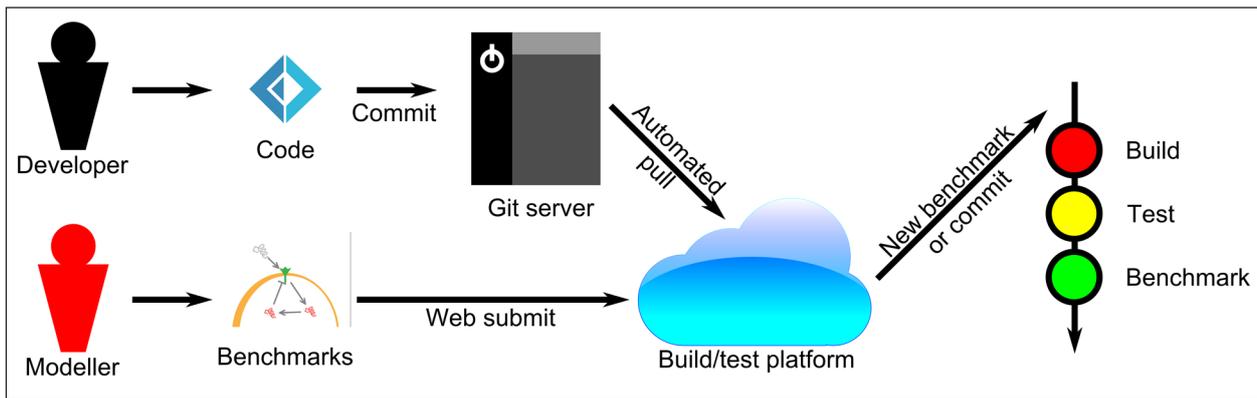
Figure 2: Proposed reproducibility service workflow.

the dependencies your code needs, on the platforms you specify, and runs all the benchmarks. This happens every time you push. It also happens every time one of your dependencies (a library, a firmware upgrade for your platform, a new API) changes too. This system (presented in **Figure 2**) would integrate with publicly available source code repositories, automates the build, testing and benchmarking of algorithms and benchmarks. It would allow testing models against competing algorithms, and the addition of new models to the test suite (either manually or from existing online repositories).

If we are truly serious about addressing the systemic socio-technical issues in scientific disciplines that are underpinned by leveraging software and computational techniques, then the proposal above would bring together almost all of the points we have discussed in this paper to provide an open research infrastructure for all. There are already several web services that already aim to do many of these things [22, 64], so a service that can integrate most if not all of these features is possible. Such a service would then allow algorithms and models to evolve together, and be reproducible from the outset. Something more open and complete, and stamped with the authority of the major domain conferences/journals/national academies, would mean that your code would never 'bit-rot', and no one would have problems reproducing the implementation of your published algorithm.

4 Next Steps

Following the proposal of such a system, the question becomes: *how do we encourage widespread uptake, or even standardisation?* Such a service would appear to be non-trivial, given the large numbers of tools and workflows that could potentially require to be supported by the service. After such a service has been implemented, how do we ensure it is *useful* and *usable* for researchers. Furthermore, how do we make it sustainable?

The benefits to the wider computational research community from a cultural change to favour reproducibility are clear and as such we should aim through software e-infrastructure and sharable, community curated research workflows to mitigate these costs. Furthermore, we can reasonably expect the distinct needs of specific research communities to evolve over time, and initial implementations of the platform may require refinement in response to user feedback (supporting the critical cultural change by

improving the efficiency of researchers). As such, if the wider research community is to move to requiring reproducibility, it seems most reasonable that this is staggered over a number of years to allow for both of these elements to develop, until eventually all researchers are required to use the service.

The key question for different research communities then becomes: *how to initialise this change?* Such a requirement creates a set of new costs to researchers, both in terms of time spent ensuring that their tools work on the centralised system (in addition to their local implementation), but also potentially in terms of equipment (in terms of running the system). Such costs may be easier to bear for some groups compared to others, especially those with large research groups who can more easily distribute the tasks, and it is important that the service does not present a barrier to early career researchers and those with efficient budgets (this type of cost analysis is not unique to reproducibility efforts – it has been estimated that a shift to becoming exclusively open access for a journal may lead to a ten-fold increase in computer science publication costs [65]).

Nevertheless, this proposed new e-infrastructure could have a profound impact on the way that computational science is performed, repositioning the role of models, algorithms and benchmarks and accelerating the research cycle, perhaps truly enabling a "fourth paradigm" of data intensive scientific discovery [66]. Ultimately though, continuing with an honest and open discussion of what reproducibility means for the wider science research community is important: we all need to explicitly confirm that this is worthwhile and commit to addressing it, or don't bother doing it at all.

4.1 A Note on Re-Writing the WSSSPE Paper

Many of the ideas, comments — even attitudes — in this paper come from the authors' experience in programming, programming languages, software. We have started from the Marc Andreessen comment that opens this paper. In editing this paper from its original WSSSPE workshop form, we realised that one assumption that seems to run through the manuscript is that the behaviours we think are good are in fact those that can be enforced in software. Take mutability of variables in programming as an example. Mutability increases the scope for bugs, so modern programming languages like OCaml or C++14 enforce immutability at the language or library level. But in fact

immutability leads very naturally to state-less or de novo build environments, and so to the guideline that "software must be compilable with de novo continuous integration". And, similarly, so does the issue of openly publishing your toolchain: it too must be compilable in a from-scratch build environment to be of use to anyone else.

Competing Interests
The authors have no competing interests to declare.

References
1. **Andreessen, M** "Why Software Is Eating The World," *The Wall Street Journal*, August 2011. Available online: http://online.wsj.com/news/articles/SB1000142405 3111903480904576512250915629460.
2. **Royal Society** 2012 "Science as an open enterprise," Available from: https://royalsociety.org/policy/projects/science-public-enterprise/report/.
3. **Editorial** 2011 "Devil in the details," *Nature*, 470(7334): 305–306, DOI: https://doi.org/10.1038/470305b
4. **Alberts, B, Cicerone, R J, Fienberg, S E, Kamb, A, McNutt, M, Nerem, R M, Schekman, R, Shiffrin, R, Stodden, V, Suresh, S, Zuber, M T, Kline Pope, B** and **Jamieson, K** 2015 "Self-correction in science at work," *Science*, 348(6242): 1420–1422. DOI: https://doi.org/10.1126/science.aab3847
5. **Collberg, C** and **Proebsting, T A** 2016 "Repeatability in Computer Systems Research," *Communications of the ACM*, 59(3): 62–69. DOI: https://doi.org/10.1145/2812803
6. **Crick, T, De Vos, M, Brain, M** and **Fitch, J** 2009 "Generating Optimal Code using Answer Set Programming." In: *Proceedings of 10th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'09)*, of *Lecture Notes in Computer Science*, 5753: 554–559, Springer. DOI: https://doi.org/10.1007/978-3-642-04238-6_57
7. **Berdine, J, Cook, B** and **Ishtiaq, S** 2011 "SLAyer: Memory Safety for Systems-Level Code," In: *Proceedings of the 23rd International Conference on Computer Aided Verification (CAV 2011)*, of *Lecture Notes in Computer Science*, 6806: 178–183, Springer. DOI: https://doi.org/10.1007/978-3-642-22110-1_15
8. **De Roure, D** "Replacing the Paper: The Twelve Rs of the e-Research Record." Available from: http://www.scilogs.com/eresearch/replacing-the-paper-the-twelve-rs-of-the-e-research-record/, November 2011.
9. **Stodden, V, Guo, P** and **Ma, Z** 2013 "Toward Reproducible Computational Research: An Empirical Analysis of Data and Code Policy Adoption by Journals," *PLoS ONE*, 8(6). DOI: https://doi.org/10.1371/journal.pone.0067111
10. **Fursin, G** and **Dubach, C** 2014 "Community-Driven Reviewing and Validation of Publications," In: *Proceedings of the 1st ACM SIGPLAN Workshop on Reproducible Research Methodologies and New Publication Models in Computer Engineering (TRUST'14)*, pp. 1–4, ACM Press. DOI: https://doi.org/10.1145/2618137.2618142
11. **National Academies of Sciences, Engineering, and Medicine** 2016 *Statistical Challenges in Assessing and Fostering the Reproducibility of Scientific Results: Summary of a Workshop*. The National Academies Press.
12. **Galiani, S, Gertler, P** and **Romero, M** "Incentives for Replication in Economics," *Tech. rep.*, National Bureau of Economic Research, July 2017. NBER Working Paper No. 23576.
13. **Barnes, N** 2010 "Publish your computer code: it is good enough," *Nature*, 467(753). DOI: https://doi.org/10.1038/467753a
14. **Morin, A, Urban, J, Adams, P D, Foster, I, Sali, A, Baker, D** and **Sliz, P** 2012 "Shining Light into Black Boxes," *Science*, 336(6078): 159–160. DOI: https://doi.org/10.1126/science.1218263
15. **Joppa, L N, McInerny, G, Harper, R, Salido, L, Takeda, K, O'Hara, K, Gavaghan, D** and **Emmott, S** 2013 "Troubling Trends in Scientific Software Use," *Science*, 340(6134): 814–815. DOI: https://doi.org/10.1126/science.1231535
16. **Bechhofer, S, Buchan, I, De Roure, D, Missier, P, Ainsworth, J, Bhagata, J, Couch, P, Cruickshank, D, Delderfield, M, Dunlop, I, Gamble, M, Michaelides, D, Owen, S, Newman, D, Sufi, S** and **Goble, C** 2013 "Why linked data is not enough for scientists," *Future Generation Computer Systems*, 29(2): 599–611. DOI: https://doi.org/10.1016/j.future.2011.08.004
17. **Osherovich, L** 2011 "Hedging against academic risk," *Science-Business eXchange*, 4(15).
18. **Hesman Saey, T** 2015 "Repeat Performance: Too many studies, when replicated, fail to pass muster," *Science News*, 187(2): 21–26. DOI: https://doi.org/10.1002/scin.2015.187002014
19. **Goble, C** 2014 "Better Software, Better Research," *IEEE Internet Computing*, 18(5): 4–8. DOI: https://doi.org/10.1109/MIC.2014.88
20. **Chirigati, F, Troyer, M, Shasha, D** and **Freire, J** 2013 "A Computational Reproducibility Benchmark," *IEEE Data Engineering Bulletin*, 36(4): 54–59.
21. **Stodden, V** and **Miguez, S** 2014 "Best Practices for Computational Science: Software Infrastructure and Environments for Reproducible and Extensible Research," *Journal of Open Research Software*, 2(1): 1–6. DOI: https://doi.org/10.5334/jors.ay
22. **Stodden, V, Miguez, S** and **Seiler, J** 2015 "ResearchCompendia.org: Cyberinfrastructure for Reproducibility and Collaboration in Computational Science," *Computing in Science & Engineering*, 17(12). DOI: https://doi.org/10.1109/MCSE.2015.18
23. **Stodden, V, McNutt, M, Bailey, D H, Deelman, E, Gil, Y, Hanson, B, Heroux, M A, Ioannidis, J P** and **Taufer, M** 2016 "Enhancing reproducibility for computational methods," *Science*, 354(6317): 1240–1241. DOI: https://doi.org/10.1126/science.aah6168
24. **Fomel, S** and **Claerbout, J F** 2008 "Reproducible Research," *Computing in Science & Engineering*, 11(1).
25. **"Reproducible Research"** 2010 *Computing in Science & Engineering*, 12(5): 8–13. DOI: https://doi.org/10.1109/MCSE.2010.113
26. **Gent, I P** "The Recomputation Manifesto." Available from: http://arxiv.org/abs/1304.3674, April 2013.

27. **Fursin, G, Miceli, R, Lokhmotov, A, Gerndt, M, Baboulin, M, Malony, A D, Chamski, Z, Novillo, D** and **Del Vento, D** 2014 "Collective mind: Towards practical and collaborative auto-tuning," *Scientific Programming*, 22(4): 309–329. DOI: https://doi.org/10.1155/2014/797348

28. **Bailey, D, Borwein, J** and **Stodden, V** 2013 "Set the Default to "Open"," *Notices of the AMS*.

29. **James, D, Wilkins-Diehr, N, Stodden, V, Colbry, D, Rosales, C, Fahey, M R, Shi, J, da Silva, R F, Lee, K, Roskies, R, Loewe, L, Lindsey, S, Kooper, R, Barba, L, Bailey, D H, Borwein, J M, Corcho, Ó, Deelman, E, Dietze, M C, Gilbert, B, Harkes, J, Keele, S, Kumar, P, Lee, J, Linke, E, Marciano, R, Marini, L, Mattmann, C, Mattson, D, McHenry, K, McLay, R T, Miguez, S, Minsker, B S, Pérez-Hernández, M S, Ryan, D, Rynge, M, Pérez, I S, Satyanarayanan, M, Clair, G S, Webster, K, Hovig, E, Katz, D S, Kay, S, Sandve, G K, Skinner, D, Allen, G, Cazes, J, Cho, K W, Fonseca, J, Hwang, L, Koesterke, L, Patel, P, Pouchard, L, Seidel, E** and **Suriarachchi, I** 2014 "Standing Together for Reproducibility in Large-Scale Computing: Report on reproducibility@XSEDE," *Tech. rep.*, XSEDE.

30. **Prlić, A** and **Procter, J B** 2012 "Ten Simple Rules for the Open Development of Scientific Software," *PLoS Computational Biology*, 8(12): e1002802. DOI: https://doi.org/10.1371/journal.pcbi.1002802

31. **Masum, H, Rao, A, Good, B M, Todd, M H, Edwards, A M, Chan, L, Bunin, B A, Su, A I, Thomas, Z** and **Bourne, P E** 2013 "Ten Simple Rules for Cultivating Open Science and Collaborative R&D," *PLoS Computational Biology*, 9(9): e1003244. DOI: https://doi.org/10.1371/journal.pcbi.1003244

32. **Sandve, G, Nekrutenko, A, Taylor, J** and **Hovig, E** 2013 "Ten Simple Rules for Reproducible Computational Research," *PLoS Computational Biology*, 9(10): e1003285. DOI: https://doi.org/10.1371/journal.pcbi.1003285

33. **Osborne, J M, Bernabeu, M O, Bruna, M, Calderhead, B, Cooper, J, Dalchau, N, Dunn, S-J, Fletcher, A G, Freeman, R, Groen, D, Knapp, B, McInerny, G J, Mirams, G R, Pitt-Francis, J, Sengupta, B, Wright, D W, Yates, C A, Gavaghan, D J, Emmott, S** and **Deane, C** 2013 "Ten Simple Rules for Effective Computational Research," *PLoS Computational Biology*, 10(3): e1003506. DOI: https://doi.org/10.1371/journal.pcbi.1003506

34. **Goodman, A, Pepe, A, Blocker, A W, Borgman, C L, Cranmer, K, Crosas, M, Di Stefano, R, Gil, Y, Groth, P, Hedstrom, M, Hogg, D W, Kashyap, V, Mahabal, A, Siemiginowska, A** and **Slavkovic, A** 2014 "Ten Simple Rules for the Care and Feeding of Scientific Data," *PLoS Computational Biology*, 10(4): e1003542. DOI: https://doi.org/10.1371/journal.pcbi.1003542

35. **Chue Hong, N P, Crick, T, Gent, I P, Kotthoff, L** and **Takeda, K** 2015 "Top Tips to Make Your Research Irreproducible." Available from: http://arxiv.org/abs/1504.00062.

36. **List, M, Ebert, P** and **Albrecht, F** 2017 "Ten Simple Rules for Developing Usable Software in Computational Biology," *PLoS Computational Biology*, 13(1): e1005265. DOI: https://doi.org/10.1371/journal.pcbi.1005265

37. **Stodden, V** 2008 "The Legal Framework for Reproducible Scientific Research: Licensing and Copyright," *Computing in Science & Engineering*, 11(1).

38. **Haas, C N** 2016 "Reproducible Risk Assessment," *Risk Analysis*, 6(10): 1829–1833. DOI: https://doi.org/10.1111/risa.12730

39. **Crick, T, Hall, B A** and **Ishtiaq, S** 2014 ""Can I Implement Your Algorithm?": A Model for Reproducible Research Software," In: *2nd International Workshop on Sustainable Software for Science: Practice and Experiences (WSSSPE2)*.

40. **Crick, T, Hall, B A, Ishtiaq, S** and **Takeda, K** 2014 ""Share and Enjoy": Publishing Useful (and Usable) Scientific Models," In: *Proceedings of the 7th IEEE/ACM International Conference on Utility and Cloud Computing*, pp. 957–961.

41. **Stallman, R M** 2010 *Free Software Free Society: Selected Essays of Richard M. Stallman*. Free Software Foundation.

42. **Giles, J** 2004 "Software company bans competitive users," *Nature*, 429(6989). DOI: https://doi.org/10.1038/429231a

43. **Hess, B, Kutzner, C, van der Spoel, D** and **Lindahl, E** 2008 "GROMACS 4: Algorithms for Highly Efficient, Load-Balanced, and Scalable Molecular Simulation," *Journal of Chemical Theory and Computation*, 4(3): 435–447. DOI: https://doi.org/10.1021/ct700301q

44. **Brooks, B R, Brooks, C L, Mackerell, A D, Nilsson, L, Petrella, R J, Roux, B, Won, Y, Archontis, G, Bartels, C, Boresch, S, Caflisch, A, Caves, L, Cui, Q, Dinner, A R, Feig, M, Fischer, S, Gao, J, Hodoscek, M, Im, W, Kuczera, K, Lazaridis, T, Ma, J, Ovchinnikov, V, Paci, E, Pastor, R W, Post, C B, Pu, J Z, Schaefer, M, Tidor, B, Venable, R M, Woodcock, H L, Wu, X, Yang, W, York, D M** and **Karplus, M** 2009 "CHARMM: The biomolecular simulation program," *Journal of Computational Chemistry*, 30(10): 1545–1614. DOI: https://doi.org/10.1002/jcc.21287

45. **Bowers, K J, Chow, E, Xu, H, Dror, R O, Eastwood, M P, Gregersen, B A, Klepeis, J L, Kolossvary, I, Moraes, M A, Sacerdoti, F D, Salmon, J K, Shan, Y** and **Shaw, D E** 2006 "Scalable algorithms for molecular dynamics simulations on commodity clusters," In: *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, IEEE Press. DOI: https://doi.org/10.1109/SC.2006.54

46. **de Moura, L** 2012 "Releasing the Z3 source code." Available online: http://leodemoura.github.io/blog/2012/10/02/open-z3.html.

47. **"Open Source Licenses"** http://opensource.org/licenses.

48. **Wilson, G** 2006 "Software carpentry: Getting scientists to write better code by making them more productive," *Computing in Science & Engineering*, 8(6). DOI: https://doi.org/10.1109/MCSE.2006.122

49. **Gonthier, G, Ziliani, B, Nanevski, A** and **Dreyer, D** 2013 "How to make ad hoc proof automation less ad hoc," *Journal of Functional Programming*, 23(4): 357–401. DOI: https://doi.org/10.1017/S0956796813000051

50. **Miller, G** 2006 "A Scientist's Nightmare: Software Problem Leads to Five Retractions," *Science*, 314(5807): 1856–1857. DOI: https://doi.org/10.1126/science.314.5807.1856

51. **Smith, A M, Katz, D S, Niemeyer, K E** and **the FORCE11 Software Citation Working Group** 2016 "Software Citation Principles," *PeerJ Computer Science*, 2(e86).

52. **Boettiger, C** 2015 "An introduction to Docker for reproducible research," *ACM SIGOPS Operating Systems Review*, 49(1): 71–79. Special Issue on Repeatability and Sharing of Experimental Artifacts. DOI: https://doi.org/10.1145/2723872.2723882

53. **Kauffman, S A** 1969 "Metabolic stability and epigenesis in randomly constructed genetic nets," *Journal of Theoretical Biology*, 22(3): 437–67. DOI: https://doi.org/10.1016/0022-5193(69)90015-0

54. **Schaub, M A, Henzinger, T A** and **Fisher, J** 2007 "Qualitative networks: a symbolic approach to analyze biological signaling networks," *BMC Systems Biology*, 1: 4. DOI: https://doi.org/10.1186/1752-0509-1-4

55. **Benque, D, Bourton, S, Cockerton, C, Cook, B, Fisher, J, Ishtiaq, S, Piterman, N, Taylor, A** and **Vardi, M Y** 2012 "BMA: visual tool for modeling and analyzing biological networks," In: *Proceedings of the 24th International Conference on Computer Aided Verification (CAV 2012), of Lecture Notes in Computer Science*, 7358: 686–692, Springer. DOI: https://doi.org/10.1007/978-3-642-31424-7_50

56. **Chaouiya, C, Berenguier, D, Keating, S M, Naldi, A, van Iersel, M P, Rodriguez, N, Drager, A, Buchel, F, Cokelaer, T, Kowal, B, Wicks, B, Goncalves, E, Dorier, J, Page, M, Monteiro, P T, von Kamp, A, Xenarios, I, de Jong, H, Hucka, M, Klamt, S, Thieffry, D, Le Novere, N, Saez-Rodriguez, J** and **Helikar, T** 2013 "SBML qualitative models: a model representation format and infrastructure to foster interactions between qualitative modelling formalisms and tools," *BMC Systems Biology*, 7.

57. **Moult, J, Fidelis, K, Kryshtafovych, A, Schwede, T** and **Tramontano, A** 2014 "Critical assessment of methods of protein structure prediction (CASP)—round x," *Proteins: Structure, Function, and Bioinformatics*, 82: 1–6. DOI: https://doi.org/10.1002/prot.24452

58. **Lensink, M F, Velankar, S** and **Wodak, S J** 2017 "Modeling proteinprotein and proteinpeptide complexes: Capri 6th edition," *Proteins: Structure, Function, and Bioinformatics*, 85(3): 359–377.

59. **Hall, B A, Halim, K B A, Buyan, A, Emmanouil, B** and **Sansom, M S P** 2014 "Sidekick for membrane simulations: Automated ensemble molecular dynamics simulations of transmembrane helices," *Journal of Chemical Theory and Computation*, 10(5): 2165–2175. DOI: https://doi.org/10.1021/ct500003g

60. **Ofria, C** and **Wilke, C O** 2004 "Avida: A Software Platform for Research in Computational Evolutionary Biology," *Artificial Life*, 10(2): 191–229. DOI: https://doi.org/10.1162/106454604773563612

61. **Crick, T, Dunning, P, Kim, H** and **Padget, J** 2009 "Engineering Design Optimization using Services and Workflows," *Philosophical Transactions of the Royal Society A*, 367(1898): 2741–2751.

62. **Olabarriaga, S, Pierantoni, G, Taffoni, G, Sciacca, E, Jaghoori, M, Korkhov, V, Castelli, G, Vuerli, C, Becciani, U, Carley, E** and **Bentley, B** 2014 "Scientific Workflow Management – For Whom?," in *Proceedings of 10th IEEE International Conference on e-Science (e-Science 2014)*, 298–305, IEEE Press. DOI: https://doi.org/10.1109/eScience.2014.8

63. **Hall, B A, Jackson, E, Hajnal, A** and **Fisher, J** 2014 "Logic programming to predict cell fate patterns and retrodict genotypes in organogenesis," *Journal of The Royal Society Interface*, 11(98). DOI: https://doi.org/10.1098/rsif.2014.0245

64. **Rollins, N D, Barton, C M, Bergin, S, Janssen, M A** and **Lee, A** 2014 "A Computational Model Library for publishing model documentation and code," *Environmental Modelling & Software*, 61: 59–64. DOI: https://doi.org/10.1016/j.envsoft.2014.06.022

65. **Vardi, M Y** 2014 "Openism, IPism, Fundamentalism, and Pragmatism," *Communications of the ACM*, 57(8). DOI: https://doi.org/10.1145/2632265

66. **Hey, T, Tansley, S** and **Tolle, K** (eds.) 2009 *The Fourth Paradigm: Data-Intensive Scientific Discovery*. Microsoft Research.