Check for updates

RESEARCH ARTICLE

Platforms for publishing and archiving computer-aided research [version 1; peer review: 2 approved with reservations]

Konrad Hinsén^{1,2}¹Centre de Biophysique Moléculaire (UPR4301 CNRS), Rue Charles Sadron, Orléans, 45071, France²Synchrotron SOLEIL, Division Expériences, St Aubin, Gif sur Yvette, 91192, France

V1 First published: 24 Nov 2014, 3:289
<https://doi.org/10.12688/f1000research.5773.1>

Second version: 02 Mar 2015, 3:289
<https://doi.org/10.12688/f1000research.5773.2>

Latest published: 14 Jul 2015, 3:289
<https://doi.org/10.12688/f1000research.5773.3>

Abstract

Computational models and methods take an ever more important place in modern scientific research. At the same time, they are becoming ever more complex, to the point that many such models and methods can no longer be adequately described in the narrative of a traditional journal article. Often they exist only as part of scientific software tools, which causes two important problems: (1) software tools are much more complex than the models and methods they embed, making the latter unnecessarily difficult to understand, (2) software tools depend on minute details of the computing environment they were written for, making them difficult to deploy and often completely unusable after a few years. This article addresses the second problem, based on the experience gained from the development and use of a platform specifically designed to facilitate the integration of computational methods into the scientific record.

Keywords

Computational models, publishing, software tools, scientific record

Open Peer Review

Approval Status

	1	2	3
version 3			
(revision)			
14 Jul 2015			
version 2			
(revision)			
02 Mar 2015			
version 1			
24 Nov 2014			

1. **Neil Chue Hong** , University of Edinburgh, Edinburgh, UK
2. **Mercè Crosas**, Harvard University, Cambridge, USA
Vito D'Orazio, Harvard University, Cambridge, USA
3. **Carl Boettiger**, University of California, Santa Cruz, Santa Cruz, USA

Any reports and responses or comments on the article can be found at the end of the article.

Corresponding author: Konrad Hinsen (konrad.hinsen@cnrs-orleans.fr)

Competing interests: No competing interests were disclosed.

Grant information: The development of ActivePapers and the first research project in which it as applied were supported by the French "Agence Nationale de la Recherche" (Contract No. ANR-2010-COSI-001-01).

The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

Copyright: © 2014 Hinsen K. This is an open access article distributed under the terms of the [Creative Commons Attribution License](#), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited. Data associated with the article are available under the terms of the [Creative Commons Zero "No rights reserved" data waiver](#) (CC0 1.0 Public domain dedication).

How to cite this article: Hinsen K. **Platforms for publishing and archiving computer-aided research [version 1; peer review: 2 approved with reservations]** F1000Research 2014, **3**:289 <https://doi.org/10.12688/f1000research.5773.1>

First published: 24 Nov 2014, **3**:289 <https://doi.org/10.12688/f1000research.5773.1>

Introduction

In the course of a few decades, computers have become essential tools in scientific research and have profoundly changed the way scientists work with data and with theoretical models. Until now, these changes have had little impact on the scientific record, which still consists mainly of narratives published in articles that are limited in size and type of contents, and linked to each other through citations. Some particularly data-intensive fields of research also have their own digital repositories. An early example is the Protein Data Bank¹, which publishes and archives structures of biological macromolecules. However, for most domains of research, no such repositories exist, and datasets are mostly neither published nor archived.

While the technology used for publishing and archiving the scientific record has shifted from the printing press and libraries to PDF files and Web servers, the kind of information that is being stored has hardly changed. Some scientific journals offer the possibility of submitting “supplementary material” with articles, as a way to circumvent the habitual length restrictions, and for providing unprintable information such as videos. In principle, the data underlying an article can be published as supplementary material as well but this remains an exception and is in fact of little practical interest. The reasons are the various restrictions on file formats and file sizes imposed arbitrarily by different scientific journals, but also the often difficult access to these electronic resources, which usually requires a careful study of each journal’s Web site. Only the recent advent of Web repositories^{2–4} and peer-to-peer networks⁵ for scientific data has finally made the publication of scientific data accessible to any scientist willing to do so.

The increasing number of mistakes found in published scientific findings based on non-trivial computations^{6,7} has made evident the necessity of making computational science more transparent by publishing software and datasets along with any descriptions of the results obtained from them. While this is now technically possible, and initiatives have been started to create incentives for scientists to invest the additional effort required for making such material available⁸, much more work remains to be done to ensure that published software and datasets can actually be understood, verified, and reused by other scientists. This is particularly important because computational methods are becoming an essential aspect of all scientific research, including experimental and theoretical work in which computations are not the main focus of activity. It is therefore more appropriate to discuss these issues in the context of “computer-aided research” rather than the more narrow specialty called computational science.

The most current efforts in this direction (see e.g. Ref. 9–11) start from the *status quo* of computation in science and propose small-step improvements in order to facilitate adoption by the scientific community. The work presented in this article takes the opposite approach of starting from the requirements of the scientific record and exploring how software and electronic datasets need to be prepared in order to become useful parts of this record. Both approaches are complementary: while ease of adoption is important for rapid improvement, it is also important to have a clear idea of the goal that should ultimately be reached, in order to avoid getting stuck in technological dead-ends.

The main contribution made by this work are the following insights:

- The traditional distinction between “software” and “data” is not adapted to the needs of scientific communication. It should be replaced by a distinction between “computational tools” and “scientific contents”. Scientific contents are the information that is conserved permanently in the scientific record. It includes experimental data, theoretical models, and computational protocols.
- Theoretical models and computational protocols include algorithms, whose permanent conservation requires a precise and stable representation with well-defined semantics.
- Scientific contents consist of distinct information items linked by dependency and provenance information, which must be stored in the scientific record as well in order to ensure reproducibility. The same information can be used for attributing credit to everyone involved in producing the information.
- A proof-of-concept implementation shows that these goals are attainable with the existing technology.

These insights are the result of developing a new computational framework, called ActivePapers¹², and using it for several research projects in the field of biomolecular simulation. The ActivePapers framework is not the principal result of this work, and it will be described only in as much as its technical characteristics matter for the conclusions. In fact, one of the conclusions of this study is that the current ActivePapers framework does not satisfy all the requirements for integrating software and datasets into the scientific record. Nevertheless, ActivePapers is a useful tool in spite of its imperfections, and researchers interested in exploring the future of computational science are invited to use it and build on it.

The state of the art

Software and electronic datasets

The main consequence of the computerization of scientific research has been an enormous increase in the volume and complexity of scientific information. In the pre-computing era, experimental data rarely exceeded a couple of printed tables, and the description of experimental protocols and theoretical models rarely exceeded a few pages. This information could easily be recorded in text form, replicated on printed paper, and stored in libraries. Moreover, each such piece of information could be read, understood, and verified by an individual scientist with sufficient experience in the underlying domain of research.

Today’s computer-aided research is characterized by large datasets which can only be stored electronically, and processed by software that embodies theoretical models and methods. While there is no fundamental difference between a printed table and a computer file, or between an experimental protocol and a piece of software, there is a very important practical difference: the size and complexity of the electronic versions often puts them beyond the limit of what an individual scientist can understand or verify. Moreover, neither datasets nor software have traditionally been published along with the articles describing a scientific study, making verification strictly impossible even in cases of moderate size and complexity.

This situation has led to numerous mistakes in published results based on computation, of which the identified and publicized cases^{6,7} are only the tip of the iceberg. In fact, these cases plus our daily experience with buggy computer software in other aspects of life should make us consider any computational result in science suspect, unless clear evidence for verification and quality assurance is provided by the authors. This evidence includes software testing, formal proof of correctness, comparison of the outcomes of independent computational studies, and proof of rigor in the non-automatic aspects of the application of computational protocols. However, the most basic requirement for building confidence in computational results is total transparency: the publication of all datasets and software used during a scientific study. This is the main goal of the Reproducible Research movement, which has been gaining traction over the last years^{13–15}. The transition to more trustworthy computer-aided research requires changes of both technological and social nature. The computational tools that scientists use must make replication not only possible, but straightforward. The maintainers of the scientific record must integrate electronic datasets and software into their archiving and publication process and reject submissions based on computational work that is not made fully transparent. But most of all, scientists must adopt a much more critical attitude towards computational results. The current tacit convention in computational science is that published results are assumed correct unless there is clear evidence suggesting a mistake. As C.A.R. Hoare famously said¹⁶, “There are two ways of constructing a software design: One way is to make it so simple that there are obviously no deficiencies, and the other way is to make it so complicated that there are no obvious deficiencies”. This observation applies equally to computational science, where the vast majority of published results have no obvious deficiencies, but are obtained using software that is much too complicated for anyone to be certain about its correctness.

The scientific record

The term “scientific record” refers to the totality of published scientific findings in history. It started to become organized in 1665 with the creation of the first scientific journal, the *Philosophical Transactions of the Royal Society*. A scientific journal publishes articles, which are narratives explaining the motivation for a specific study, the methods being applied, the observations made, and the conclusions drawn. The exact observations are provided in the form of tables and figures. To this day, scientific journals are the most visible part of the scientific record, although digital repositories have become an important second pillar in several domains of research. These repositories store datasets that are too big to be included as tables or figures. In addition, repositories facilitate the reuse of data in other studies, including the application of data mining techniques in meta-analysis studies.

One of the main characteristics of the scientific record is its permanence. Once an article is published in a journal, or a dataset in a database, a permanent reference is attached to it, and the publisher or database accepts the moral obligation to make the information accessible through this reference for as long as possible. Traditionally, this reference is a citation taking a more or less standardized format. With the transition to electronic publishing, the role of the permanent reference is fulfilled by a Digital Object

Identifier (DOI)¹⁷, which is defined by the international standard ISO 26324:2012. For published articles, permanence also applies to the contents: once published, they cannot change. Mistakes detected after publishing can be corrected only by publishing a separate short article called “erratum”. The situation is less clear for repositories, some of which, for example the Protein Data Bank (PDB)¹, do correct formal mistakes in their electronic records, but do not allow changes that would modify their scientific interpretation. More recently, the idea of versioning has been applied by some publishers, e.g. F1000Research: a DOI refers permanently to a specific article, but multiple versions of this article remain accessible permanently in order to document the evolution of the publication.

The permanence of the scientific record applies only to the preservation of the original expression of each information item, but not to its semantics. A published article can become unintelligible because of changes in terminology and in the scientists’ education. A modern physicist would not recognize the theory of classical mechanics in Newton’s “*Philosophiæ Naturalis Principia Mathematica*”¹⁸ without prior training in Latin and in the history of science. The longevity of electronic datasets relies on a careful documentation of the data models and data formats being used, and on proper curation of submissions by the database managers to ensure adherence to these formats. Scientific journals generally use the PDF/A format for their articles, which is a variant of the popular PDF format designed specifically for long-term archiving, defined by the international standard ISO 19005:2005. The latest generation of uncured Web repositories^{2–4} allows any computer file to enter the scientific record without even requiring a definition of the data format. It is to be expected that much of the information in these repositories will quickly become unusable.

Software presents a particular challenge because formal data models for executable computer code are rare and in particular non-existent for the programming languages commonly used in computational science today. This is one of the key problems addressed in this work. Many practitioners consider the idea of preserving scientific software for many years unrealistic, and some even argue that it is unnecessary because computational methods change so rapidly that their long-term conservation is of no interest. The latter argument is manifestly not valid in general. As an example, the DSSP method for defining secondary-structure elements in proteins¹⁹ was published more than thirty years ago and is still widely used today. The example of DSSP is also interesting in that the most widely used software implementing DSSP today²⁰ does not in fact implement the exact method published in the original paper. The differences are not documented anywhere at this time, and scientific papers using the modern software systematically cite the original paper without further comment. It can thus be assumed that most DSSP users are not aware of the fact that they are using a modified method. If the original method had been published in executable form and preserved until today, such discrepancies could have been avoided.

Replication and reproduction of scientific results

One of the cornerstones of scientific research is the *reproducibility* of scientific findings: in principle, anyone applying a published

research protocol to a sufficiently similar object of study should obtain very similar results. The notion of reproducibility is necessarily imprecise. In performing experiments, the samples and environments are never exactly the same. Moreover, the description of an experimental protocol is never fully complete, because the experimenters cannot know with certainty which parameters need to be recorded. The level of similarity required in the experimental setup and in the results for the latter to be considered a successful reproduction thus varies considerably across domains of research. A reproduction attempt, whether successful or not, always yields new scientific knowledge, because it explores the impact of variations in the protocols, environments, and samples.

In computational science, the protocols, input data, and computational environment can in principle be recorded exactly, being digital information. A sufficiently complete recording of this information makes a computational study *replicable*: another scientist can re-run the exact same computation and obtain exactly identical results. Reproducibility, like in experimental science, refers to the less precise idea of re-doing computations based on the description of the methods in the published narrative, which is supposed to describe the “important” aspects (e.g. a numerical algorithm) but not details generally considered to be irrelevant (e.g. compiler and library version numbers).

Replicability and reproducibility in computational science play different roles in the scientific process. Replicability is part of quality assurance. If scientist B can replicate scientist A’s computation, this shows that A has provided a sufficiently precise and complete record of the original work. This is far from trivial because a precise and complete description of a computational study is a dataset that is both large and complex. Moreover, it is often difficult to obtain in today’s computing environments. Reproducibility plays the same role as in other branches of science: it establishes which aspects of a computational protocol are important for reaching specific conclusions.

However, replicability and reproducibility are not completely independent. For all but the simplest computational methods, reproducibility requires replicability. If a reproduction attempt leads to significantly different results, the cause of the differences must be explored. This is in practice only possible if the original results can at least be replicated. Otherwise, the most probable explanation for the difference is that the original study was insufficiently documented, and is thus of very limited value. But exploring the differences requires more than replicability: the original computational protocol must also be understandable by the scientist who sets out to explore the failure of a reproduction attempt. The ultimate problem is that computational methods have become very complex. A summary in the narrative is not sufficiently detailed to allow reproduction. On the other hand, the actual executable computer code is precise and complete, but even more complex than the method because it also needs to take into account complex technical issues such as performance and resource management. I have outlined solutions to this problem in Ref. 21.

Different parts of the scientific process impose different criteria for replicability and reproducibility. Conducting a study requires only

short-time local replicability for quality assurance, i.e. the authors must be able to re-run their own computation in their own computational environment. Collaboration requires short-time non-local replicability, because co-workers usually have somewhat different computational environments. Pre- or post-publication peer review requires more stringent short-time non-local replicability, because reviewers are likely to have significantly different computing environments at their disposal. Peer review also requires a minimal level of reproducibility in that reviewers must at least be convinced that they *could* reproduce the findings if they tried, even though for lack of time they rarely do. Publication and archiving as part of the scientific record add the requirements of long-term reproducibility and thus long-term replicability. All archived data must remain usable *and* understandable for as long as the study remains of scientific interest, which is usually a few decades.

Tools for replicable and reproducible research

Since replicability is a purely technical aspect of computer-aided research, it can in principle be guaranteed by software tools. Many ongoing research and development projects aim to create such tools, although for pragmatic reasons the goal is rarely complete replicability, but rather some weaker requirement that is easier to achieve with existing technology. Reproducibility implies human understanding and thus cannot be ensured or even verified automatically. Nevertheless, software tools can help in the process of documenting computational methods in a way that is both replicable and understandable. In the following, I will summarize the currently most popular approaches.

An approach that has already been widely adopted in domains of research that make heavy use of computation is the conservation of datasets in digital repositories issuing permanent references. A scientific narrative is separately published in a journal, and uses the permanent reference to establish a link with the data. This link between narrative and datasets can be tightened to the point where articles no longer contain any data in the form of tables, but only permanent references to repository entries. A good example for this strategy in chemistry is described in Ref. 22. In that particular work, the software does not enter the scientific record at all, and appears only in metadata stored along with the computational results, where it identifies the software packages, version numbers, and other details of the computational environment. Other initiatives (see e.g. 23, 24) advocate storing a snapshot of the program source code in a digital repository as a dataset, in order to provide at least an archive of the exact version of the software that was used, even if it is difficult or impossible to re-run that software later. This ambivalent attitude towards software stems from the recognition of its fundamental importance on one hand and from the practical impossibility to fully integrate today’s scientific software into the scientific record on the other hand.

The *computational notebook* approach¹¹, pioneered by Mathematica²⁵ and recently popularized by the Jupyter project (formerly known as the IPython notebook)²⁶, builds on earlier developments in literate programming²⁷, which have also been applied to computational science directly^{28,29}. It aims to integrate computational methods expressed as working code with input/output data and the scientific narrative. It permits a seamless transition from interactive

exploratory work to a documented computational method that can be shared and published. Compared to traditional scripts, computational notebooks represent an important advance in improving reproducibility through improving human understanding. However, none of the existing notebook implementations improve on scripts in replicability, which remains local and short-term. Like a script, a notebook depends on the computational environment in which it was generated. This environment is neither conserved nor even documented in the notebook. A few years later, a notebook still provides a human-readable and rather detailed description of the method, but re-running it is likely to be difficult or impossible. Moreover, the notebook approach does not take into account datasets, unless they are small enough to be included as literal data into the notebook itself. All other data is accessed by usually nonpermanent references such as filenames or Universal Resource Locators (URLs).

Similar remarks apply to workflow management systems such as Kepler³⁰ or VisTrails³¹. In fact, workflows, scripts, and notebooks all refer to the same basic concept: the outer algorithmic layer that defines a specific computational study in terms of more generic components. The differences lie in the user interface and in the kind of components that can be used (libraries, executables, Web services, etc.). Some workflow managers can archive these components partially, and also some kinds of datasets, but such support is neither complete nor exhaustive, because the technology on which today's scientific software is built does not allow this.

The most comprehensive approach to archiving scientific software in an executable form is based on virtual machine technology^{32–34}. The authors of a computational study produce a virtual machine image that contains their complete computational environment, starting with the operating system, in addition to the problem-specific data and workflows. The resulting archives are in general too big to be archived in today's general-purpose digital repositories. Moreover, it is not possible to refer to or re-use individual pieces of software or data inside a virtual machine image, nor is it straightforward in general to analyze the software or data except by the tools explicitly provided by the authors. But most importantly, the longevity of archived virtual machine images is uncertain. Executing such an image requires complex and sophisticated software, which for the moment is produced and maintained by non-scientific organizations for reasons completely unrelated to science. Once technological progress makes these efforts obsolete, it must be expected that computations archived as virtual machine images will become unusable.

A conclusion that can be drawn from the approaches summarized above is that today's computational scientists cannot publish their work in a form that is at the same time executable, understandable by human readers, and reusable. Of the three basic kinds of information in computational science, software, data, and narrative, it is clearly the software that is at the root of the difficulties.

Platforms and contents

For understanding the difficulties caused by software, and thus to identify the possible solutions, it is useful to introduce the concept of a *platform* for scientific computing. A platform defines the interface between a computational infrastructure and computational contents. In particular, the platform defines the exact data formats

that the contents must respect, and specifies how each data item will be interpreted. For example, the MP3 standard defines a platform for handling music in computers. It defines a file format for storing sound samples, and defines how an MP3 player interprets the data in such files. Any MP3 player is an implementation of the MP3 platform. Any MP3 file is a piece of contents for the MP3 platform. In general, a platform can be more complex and define formats for many different kinds of data. There are also customizable platforms that define some basic features of their contents but also a mechanism for adding more specific features. The best-known example is the XML platform, which allows working with generic structured text data, including the definition of more specific subformats through DTDs or schemas.

For a piece of software, the platform required to run it varies considerably as a function of how the software is presented. A compiled executable for the Microsoft Windows platform has very specific requirements concerning the instruction set of the real or virtual processor used to run it, but also concerning how the operating system services are accessed. In addition, its correct function may depend on specific versions of specific dynamically loadable libraries. Although most aspects of the Microsoft Windows platform are documented to some degree, there is no comprehensive documentation for the total, and its complexity makes it unlikely that such a documentation will ever be produced. In practice, only a test run can establish whether a given program works on a given machine. For software published in source code form, the requirements are very different but equally complex. Typical dependencies include a compiler or interpreter for a specific programming language, specific versions of specific libraries, and sometimes even specific files being accessible in specific locations. None of these details are documented comprehensively, which is why installation and deployment of software is very difficult. Even standardized programming languages are not defined with precise semantics, a situation that has already caused many serious problems (see e.g. Ref. 35 for the languages C and C++). This lack of a precisely defined and stable platform for executable code is also the root cause of non-replicability in computer-aided research.

Of the various attempts to remedy this situation, the best known and most successful one is the Java Virtual Machine (JVM)³⁶, originally defined as a support for running software written in the Java language. Today the JVM hosts a variety of languages and ensures a high level of interoperability between them. The goal of the JVM developers was to enable the distribution of executable code via the Web, which users could run in their browsers without any prior installation or configuration. This goal has overall been reached successfully, and with remarkable stability: Java code written in 1995 can still be run without modification. There are only two aspects in which the JVM platform failed to attain universal portability: (1) interfacing to certain operating-system services, such as user interface layers or concurrency management, and (2) floating-point computations. The latter failure is due to a deliberate decision to give up the precise initial specification of the JVM in favor of a less rigid one that leaves more room for performance optimizations. It is still possible to use the original precise floating-point semantics, but in practice this feature is hardly used because most computer users give a higher priority to performance than to replicability.

The reasons for the JVMs success in establishing a stable software platform are various, and to a significant degree due to the interplay of the commercial strategies of the major companies in the computing market. Among the technical reasons, the main one is the choice of a data model for executable code that is situated at a higher level of abstraction than machine code, but at a lower level than typical programming languages. Machine code evolves rapidly because of progress in processor design, and programming languages evolve, somewhat less rapidly, because of advances in software engineering. Stability can only be found in between these two extremes. Other stable software platforms have adopted the same fundamental approach. In particular, the ECMA standard CLI³⁷ can be considered a more modern implementation of the basic JVM idea. Google's much more recent Portable Native Client (PNaCL) platform³⁸ chose a more low-level code representation defined by the LLVM project³⁹ and a less precisely defined computational environment, in order to facilitate the adaptation of software written in traditional programming languages. It is too early to say if this approach will turn out to be successful.

In summary, the JVM experience proves that a significant aspect of the software portability problem can be solved: it is possible to define a stable platform for executable code. The difficulties encountered with the portability of JVM code can be avoided by limiting oneself to the important subset of pure computations, i.e. software that transforms input data into output data but does not interact with its environment in any other way. This observation is important because the scientific aspects of software are always pure computations.

ActivePapers

The goal of the ActivePapers research and development project is to define a platform for publishing and archiving computer-aided research. Such a platform should ideally meet all of the following requirements:

- A published electronic dataset, in the following called an ActivePaper, should contain all the data, code, and narrative related to a research project, with internal links among all the pieces of information that indicate dependencies and provenance.
- An ActivePaper should be able to refer to data items in previously published ActivePapers. Such references should allow both re-use and attribution of scientific credit.
- An ActivePaper should support large datasets by ensuring compact storage and high-performance data access.
- The representation of executable code inside an ActivePaper should be well-defined, stable, and sufficiently simple to allow implementation on future computing systems with minimal effort. The execution of any piece of code from an ActivePaper should always produce exactly the same results at the bit level.
- Any code stored in an ActivePaper should be safe to execute, i.e. it should not be able to cause any harm to the computing environment it is executed on.
- An ActivePaper should contain metadata for provenance tracking and reproducibility.

It is important to note that it is *not* required that all software used for a computational study be stored in ActivePapers. On the contrary, it is to be expected that important software tools remain forever outside of the ActivePaper universe and work on ActivePapers as data. This includes everything requiring user interaction, from authoring tools to data visualization programs, and also highly machine-specific software such as batch execution managers. It is also possible to write external code accelerators that take code from an ActivePaper and execute it after optimization and/or parallelization, guaranteeing identical results. While the current state of the art does not provide techniques for making such code accelerators both general and efficient, it is possible and even straightforward to write problem-specific code accelerators, which are simply efficient reimplementations (in a language like Fortran or C) of algorithms stored in an ActivePaper, with the equivalence of the results verified by extensive tests.

The ActivePapers JVM edition

The original ActivePapers architecture⁴⁰, which was subsequently implemented in the "ActivePapers JVM edition", was a proof-of-concept design intended to show that it is possible with existing technology to meet all these requirements. The key design and implementation choices were

- An ActivePaper is a file in HDF5 format⁴¹. The HDF5 format ensures flexibility, compactness, and high-performance data access.
- HDF5 dataset attributes are used to store metadata, including a dataflow graph that records provenance.
- Any data item inside a published ActivePaper can be referenced by the combination of the ActivePaper's DOI and the HDF5 path to the dataset.
- Executable code is stored as JVM bytecode. Any other code representation, in particular human-readable source code in any language, is admissible if a compiler or interpreter exists in the form of JVM bytecode.
- The JVM security model is used to prevent executable code in an ActivePaper from accessing any data outside of the ActivePapers platform. This ensures both security (the user's computing environment is protected) and the absence of unrecorded dependencies.
- Individual programs inside an ActivePaper can be declared as data importers, in which case they have unrestricted read access to anything, including local files and network resources. They share the write restrictions of all other code, meaning that they cannot modify anything outside of the ActivePapers platform. Moreover, they are never run automatically, but only on explicit user request.

An implementation of the original ActivePapers platform is available from the ActivePapers Web site¹². Its only dependencies are (1) a Java Virtual Machine implementation, (2) the HDF5 library, and (3) JHDF5⁴², a Java interface to the HDF5 library. The ActivePapers software provides a command-line interface for creating ActivePapers, inspecting their contents and metadata, and for running the

embedded executable code. This is clearly a minimal working environment. For production use by a wide community of computational scientists, many convenience functions would have to be added: a code and data editor, data visualization, data management, etc.

An important design decision is related to the management of the metadata that tracks dependencies and provenance. The ActivePapers platform creates and updates this metadata automatically during program execution. From the user's point of view, an ActivePaper is a collection of datasets and programs, of which the latter can be run individually just like traditional executables or scripts. The ActivePapers platform tracks all data accesses from programs and generates the dependency graph from them. When a program is re-run, typically after modification, all the datasets it generated earlier are deleted automatically. Moreover, when a program reads data generated by another program which has been modified since it was last run, the modified program is re-run automatically to ensure coherence of all data. This automatic dependency handling has worked well in practice. It is the exact opposite of the approach taken by automation tools such as *make*⁴³, which execute programs according to a manually prepared definition of the dependencies between their results.

The main difficulty with the original ActivePapers platform is the lack of scientific software compatible with its constraints. All code running inside of the ActivePapers platform must exist as JVM bytecode. All code storing data in an ActivePaper must use the HDF5 library. All code that falls into both categories, which includes in particular the workflow of a specific scientific project, must exist as JVM bytecode accessing the HDF5 library. There is almost no publicly available code that meets these requirements, mostly due to the lack of popularity of the JVM in scientific computing.

The ActivePapers Python edition

In order to gain experience with the ActivePapers approach in practice, a second implementation was developed for the Scientific Python ecosystem⁴⁴. Its dependencies are the Python language⁴⁵, the HDF5 library, the *h5py* library⁶⁰ for interfacing HDF5 to Python, and the *NumPy* library⁴⁶ which is a dependency of *h5py*. For the Python edition of ActivePapers, all executable code must exist in the form of Python scripts, which access the datasets through the *h5py* library. Libraries that contain compiled code, which are very common, cannot be placed inside an ActivePaper, but can be declared as an external dependency. This effectively means that the platform required for using an ActivePaper with such a dependency includes that library in addition to the packages listed above. Adding external dependencies is clearly not desirable from a replicability point of view, but it provides a short-term workaround to the fundamental problem that most scientific software is not ready for long-term replicability.

The Scientific Python ecosystem provides a large choice of libraries that can be used within these constraints, and the Python language is already very popular for scientific computing, making the ActivePapers Python edition a good vehicle for testing the ActivePapers approach on real research projects. On the other hand, the Python edition cannot fulfill all the requirements listed above. In

particular, the Python language lacks sufficiently strong security mechanisms to implement a useful level of user protection. A more subtle problem is the stability of the platform itself. The Python language has no formal specification and in fact evolves together with its principal implementation. The scientific libraries, in particular *NumPy*, also evolve rather rapidly, with only moderate efforts to maintain compatibility with older versions. The ActivePapers platform records the version of all libraries that were used in the preparation of an ActivePaper, but the long-time usability of these versions is questionable, as in general only the current versions can be expected to work in current computing environments.

The Python edition of the ActivePapers platform has been used for several research projects in the field of biomolecular simulation, some of which have already been published^{47–49}. Each publication has one or more ActivePaper files deposited as supplementary material, but all the files are also available in digital repositories with DOIs. Among the published ActivePapers, there are software libraries^{50,51}, a database of protein structures⁵², and combinations of datasets and code that document computational studies^{53–55}. Additional published ActivePapers contain obsolete versions of the *pyMosaic* library^{56–58}. These files remain permanently available because other ActivePapers depend on them. They also remain usable for as long as the underlying platform remains compatible.

One problem encountered in the course of these research projects is the relatively low size limit that today's digital repositories impose on archived files. *Zenodo*⁴ provides the most generous limit of 2 GB per file. However, the input data for one study⁵⁵ contains ten Molecular Dynamics (MD) trajectories for lysozyme in solvent, and requires 10 GB of storage even in compressed form. Since these data were not essential for the subsequent analysis step, which requires only the rigid-body motion of the protein, they were removed from the published files. The alternative would have been to publish each MD trajectory separately as an ActivePaper, and use DOI-based references in the analysis step to refer to this data. Because such references are nearly transparent to the user (the dependencies are downloaded automatically when needed), file size limits apply in practice only to individual HDF5 datasets.

ActivePapers proposes another mechanism to reduce file sizes: the deletion of recomputable datasets. Any dataset that was generated by a program stored in an ActivePaper can be replaced by a dummy dataset that retains only the dependency metadata. The full dataset can be re-computed on demand, or automatically when another program tries to read it. Recomputation consists in rerunning the program that generated the dataset initially. This mechanism makes sense only if the replicability of a dataset is guaranteed. In practice, this applies to any program that does not use floating-point operations. The latter are insufficiently specified in most of today's programming languages, including Python, and therefore floating-point computations can produce different results when the same program is run on two different computers.

Future developments

Experience with the two current implementations of the ActivePapers idea has shown that all of the requirements defined at the outset

can be fulfilled and that the approach works well in practice. In particular, the ActivePapers project has shown that installation-free software deployment and long-time software conservation are possible, contrary to a common belief in the scientific computing community. As mentioned earlier, ActivePapers can achieve these goals because the science part of software takes the form of pure computations, which are possible identically in all computational environments.

The existence of two distinct ActivePapers platforms is an historical accident that is clearly not desirable. The envisaged solution is a split of the ActivePapers platform into two parts: a data publishing system, which defines the HDF5 conventions for ActivePapers, in particular the metadata, and a code execution system that defines how specific datasets in ActivePapers are interpreted as executable code. Only the second part would differ between the current two implementations, and its separation also opens the way for additional execution systems for other code representations. There is in fact no fully satisfying code representation for scientific computations at this time, which is a strong argument for flexibility in the platform definition.

One line of future development is an integration of a narrative into the computational methods stored in an ActivePaper. Work on integrating the ActivePapers Python edition with the Jupyter project²⁶ (formerly the IPython notebook) is underway. Unfortunately, non-fundamental technical issues make this a non-trivial project: the various components (HDF5, Python, Jupyter) have different and conflicting requirements and restrictions concerning concurrency. Aside from these software engineering issues, the main question to be solved is how to reconcile the interactivity of the notebook approach with the permanence requirements of the scientific record. The coherence of code and results in a notebook is guaranteed only if it has been executed linearly from start to end. Any interactive manipulation results in general in a non-replicable state. Two solutions are currently explored. The first solution marks notebooks as non-replicable except when executed linearly. No ActivePaper containing such non-replicable notebooks should be accepted by a digital repository. The second solution is to record all interactive code execution in a log, which can then be replayed. After a complete linear execution of the notebook code, the log of interactive executions is then deleted.

Another direction for future developments explores how to provide a realistic transition from today's scientific computing environments to future ones that take into account the needs for publishing and archiving computations. One important advantage of the ActivePapers approach in this context is that the minimal requirements for adopting it are modest: any software tool that can work with the ActivePapers file format, which is HDF5 plus a small set of conventions, can read and write publishable datasets. With a very small additional effort, software tools can be adapted to handle ActivePapers metadata and thus ensure dependency and provenance tracking. None of this requires that the software live inside the ActivePapers platform. The challenge for future ActivePapers developments is to facilitate the transition of computational methods from subroutines

hidden inside software tools to precise specifications that become part of the scientific record.

Conclusion

As I have pointed out earlier²¹, today's scientific software fulfills two distinct roles: it is a tool that permits doing computations, but also the only precise and complete description of the models and methods applied in these computations. This situation is the result of the growing complexity of computational methods in science, which make the documentation of these methods in the traditional narrative of a journal article impossible. Understanding and evaluating computational science requires both the possibility to read the source code of all software and to run it on suitable input data. A useful documentation of computational science in the scientific record thus requires archiving all software parts that have an influence on computational results, in a form that can be both inspected and executed, for as long as the study remains relevant for science, which is typically several decades.

The ActivePapers project has shown that these goals are achievable in principle using existing technology. It has defined two variants of a platform that gives computational methods the status of publishable content with well-defined data formats that guarantee long-term replicability. However, it has also shown that the vast majority of today's scientific software is not easily integrated into such a platform. The main reason is that most of the computing technology used by scientists was developed outside of scientific research, for domains of application where replicability is not important.

A key ingredient in the transition from the current state of the art, in which scientific software cannot be fully archived in the scientific record, is a clear distinction between scientific models and methods on one hand, and software tools on the other hand. It is only the models and methods that need to be archived, but not the tools. The long-term usability of the models and methods is guaranteed by a complete and precise specification of their data formats, rather than by a preservation of the tools that work on them. Computational tools must in fact evolve with the progress of technology in order to remain useful to the communities that develop and apply them⁵⁹. This distinction is completely analogous to how other digital content is handled. We archive articles in PDF format, movies in MPEG3 format, or protein structures in mmCIF format because these formats are well documented and allow anyone, at any point in time, to interpret the archived contents, even if today's software tools are no longer usable because of the inherent instability of computational environments.

What distinguishes computational models and methods from articles, movies, and protein structures is their algorithmic nature, which makes them look like "software" rather than "data". However, this distinction between software and data, although deeply ingrained in the habits of computational scientists, is not fundamental: software is just a specific kind of data, defined by the existence of some mechanism to execute it. It is very common to use software that treats other software as data, e.g. compilers, interpreters,

workflow managers, debuggers, etc. In the context of scientific communication, we should treat software exactly like other kinds of data. The fundamental distinction is not “software” vs. “data”, but “computational tool” vs. “scientific content”.

Publishing and archiving scientific results has always involved an additional effort compared to keeping personal records. Experimentalists do not publish their raw lab notebooks with the user manual of their scientific equipment as an appendix. Theoreticians do not submit scans of their hand-written notes for publication. Publication always implies presenting the work that has been done and its results in a form that is understandable to and usable by other scientists. The same principle applies to computational science, whose practitioners need to be prepared to invest additional

effort to make their libraries, programs, and scripts suitable for publishing.

Competing interests

No competing interests were disclosed.

Grant information

The development of ActivePapers and the first research project in which it as applied were supported by the French “Agence Nationale de la Recherche” (Contract No. ANR-2010-COSI-001-01).

The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript

References

- Berman H, Henrick K, Nakamura H: **Announcing the worldwide Protein Data Bank.** *Nature Struct Biol.* 2003; **10**(12): 980.
[PubMed Abstract](#) | [Publisher Full Text](#)
- Dryad. 2014.
[Reference Source](#)
- figshare. 2014.
[Reference Source](#)
- Zenodo. 2014.
[Reference Source](#)
- Academic torrents. 2014.
[Reference Source](#)
- Merali Z: **Computational science: ...Error.** *Nature.* 2010; **467**(7317): 775–777.
[PubMed Abstract](#) | [Publisher Full Text](#)
- Joppa LN, McInerney G, Harper R, et al.: **Computational science. Troubling trends in scientific software use.** *Science.* 2013; **340**(6134): 814–815.
[PubMed Abstract](#) | [Publisher Full Text](#)
- Priem J, Taraborelli D, Groth P, et al.: **altmetrics: a manifesto.** 2014.
[Reference Source](#)
- Stodden V, Miguel S: **Best Practices for Computational Science: Software Infrastructure and Environments for Reproducible and Extensible Research.** *Soc Sci Res Network.* 2013; 2322276.
[Reference Source](#)
- Stodden V, Leisch F, Peng RD: **Implementing Reproducible Research.** Chapman and Hall/CRC. 2014.
[Reference Source](#)
- Shen H: **Interactive notebooks: Sharing the code.** *Nature.* 2014; **515**(7525): 151–152.
[PubMed Abstract](#) | [Publisher Full Text](#)
- Hinsen K: **ActivePapers - computational science made reproducible and publishable.** 2014.
[Reference Source](#)
- Peng RD: **Reproducible research in computational science.** *Science.* 2011; **334**(6060): 1226–1227.
[PubMed Abstract](#) | [Publisher Full Text](#) | [Free Full Text](#)
- Donoho DL: **An invitation to reproducible computational research.** *Biostatistics.* 2010; **11**(3): 385–388.
[PubMed Abstract](#) | [Publisher Full Text](#)
- Stodden V: **Reproducible research: Tools and strategies for scientific computing.** *Comput Sci Eng.* 2012; **14**(4): 11–12.
[Publisher Full Text](#)
- Hoare CAR: **The emperor's old clothes.** *Communications of the ACM.* 1981; **24**(2): 75–83.
[Publisher Full Text](#)
- The DOI system. 2014.
[Reference Source](#)
- Newton I: **Philosophiae Naturalis Principia Mathematica.** *R Soc.* 1686.
[Reference Source](#)
- Kabsch W, Sander C: **Dictionary of protein secondary structure: pattern recognition of hydrogen-bonded and geometrical features.** *Biopolymers.* 1983; **22**(12): 2577–2637.
[PubMed Abstract](#) | [Publisher Full Text](#)
- Hekkelman M: **DSSP 2.1.0.**
[Reference Source](#)
- Hinsen K: **Computational science: shifting the focus from tools to models.** [v2; ref status: indexed, <http://f1000r.es/3p2>]. *F1000Res.* 2014; **3**: 101.
[PubMed Abstract](#) | [Publisher Full Text](#) | [Free Full Text](#)
- Harvey MJ, Mason NJ, Rzepa HS: **Digital data repositories in chemistry and their integration with journals and electronic notebooks.** *J Chem Inf Model.* 2014; **54**(10): 2627–2635.
[PubMed Abstract](#) | [Publisher Full Text](#)
- exec&share. 2014.
[Reference Source](#)
- Mozilla Science Lab, Github, and figshare. **Code as a research object.** 2014.
[Reference Source](#)
- Wolfram Research, Inc. **Mathematica 2.0.** 1991.
- Project Jupyter. 2014.
[Reference Source](#)
- Knuth DE: **Literate programming.** *The Computer journal.* 1984; **27**(2): 97–111.
[Reference Source](#)
- Schulte E, Davison D: **Active documents with orgmode.** *Comput Sci Eng.* 2011; **13**(3): 66–73.
[Publisher Full Text](#) | [Free Full Text](#)
- Xie Y: **Dynamic Documents with R and knitr.** Chapman & Hall. 2013.
- The Kepler Project.
[Reference Source](#)
- VisTrails.
[Reference Source](#)
- Van Gorp P, Grefen P: **Supporting the internet-based evaluation of research software with cloud infrastructure.** *Softw Syst Model.* 2012; **11**(1): 11–28.
[Publisher Full Text](#)
- Gent I, Kothhoff L: **recomputation.org home page.** 2014.
[Reference Source](#)
- Boettiger C: **An introduction to Docker for reproducible research, with examples from the R environment.** *arXiv.org.* 2014.
[Reference Source](#)
- Regehr J: **A guide to undefined behavior in C and C++.**
[Reference Source](#)
- Lindholm T, Yellin F: **The Java Virtual Machine Specification.** Prentice Hall. 1999.
[Reference Source](#)
- ECMA Standard 335: **Common Language Infrastructure CLI.**
[Reference Source](#)
- Portable Native client: **The “pinnacle” of speed, security, and portability.** 2014.
[Reference Source](#)
- Lattner C, Adve V: **LLVM: A compilation framework for lifelong program analysis and transformation.** In *Proceedings of the 2004 International Symposium on Code Generation and Optimization*, San Jose, CA, USA, 2004; 75–88.
[Reference Source](#)
- Hinsen K: **A data and code model for reproducible research and executable papers.** *Pro Comput Sci.* 2011; **4**: 579–588.
[Publisher Full Text](#)
- The HDF Group. **Hierarchical data format version 5.**
[Reference Source](#)
- Scientific IT Services of ETH Zürich. **JHDF5, HDF5 for Java.**
[Reference Source](#)
- Wikipedia. **Make (software)** — Wikipedia, the free encyclopedia. 2014.
[Reference Source](#)
- Millman KJ, Aivazis M: **Python for scientists and engineers.** *Computing in*

- Science Engineering*. 2011; **13**(2): 9–12.
[Publisher Full Text](#)
45. **Python Software Foundation**. The Python language. 2014.
[Reference Source](#)
 46. **NumPy development team**. NumPy. 2014.
[Reference Source](#)
 47. Hinsén K, Hu S, Kneller GR, *et al.*: **A comparison of reduced coordinate sets for describing protein structure**. *J Chem Phys*. 2013; **139**(12): 124115.
[PubMed Abstract](#) | [Publisher Full Text](#)
 48. Chevrot G, Hinsén K, Kneller GR: **Model-free simulation approach to molecular diffusion tensors: Water**. *figshare*. 2013.
[PubMed Abstract](#) | [Publisher Full Text](#)
 49. Hinsén K: **MOSAIC: a data model and file formats for molecular simulations**. *J Chem Inf Model*. 2014; **54**(1): 131–137.
[PubMed Abstract](#) | [Publisher Full Text](#)
 50. Hinsén K: **ImmutablePy 0.1 in ActivePapers format**. *figshare*. 2013.
[Reference Source](#)
 51. Hinsén K: **pyMosaic 0.3.1**. *Zenodo*. 2014.
[Reference Source](#)
 52. Hinsén K: **ASTRAL-SCOPE subset 2.04 in ActivePapers format**. *Zenodo*. 2014.
[Reference Source](#)
 53. Hinsén K, Shuangwei Hu, Kneller GR, *et al.*: **A comparison of reduced coordinate sets for describing protein structure**. *figshare*. 2013.
[Reference Source](#)
 54. Chevrot G, Hinsén K, Kneller GR: **Model-free simulation approach to molecular diffusion tensors: Lysozyme**. *figshare*. 2013.
[Reference Source](#)
 55. Chevrot G, Hinsén K, Kneller GR: **Model-free simulation approach to molecular diffusion tensors: Lysozyme**. *figshare*. 2013.
[Reference Source](#)
 56. Hinsén K: **pyMosaic 0.1.1 in ActivePapers format**. *figshare*. 2013.
[Reference Source](#)
 57. Hinsén K: **pyMosaic 0.2.0**. *Zenodo*. 2014.
[Reference Source](#)
 58. Hinsén K: **pyMosaic 0.3.0**. *Zenodo*. 2014.
[Reference Source](#)
 59. Katz DS, Allen G, Chue Hong N, *et al.*: **First workshop on sustainable software for science: Practice and experiences (WSSSPE): submission and peer-review process, and results**. *arXiv.org*, abs/1311.3523, 2013.
[Reference Source](#)
 60. Drummond C: **Replicability is not reproducibility: nor is it good science**. 2009.
[Reference Source](#)

Open Peer Review

Current Peer Review Status: ? ?

Version 1

Reviewer Report 26 January 2015

<https://doi.org/10.5256/f1000research.6172.r6988>

© 2015 Crosas M et al. This is an open access peer review report distributed under the terms of the [Creative Commons Attribution License](#), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Mercè Crosas

Institute for Quantitative Social Sciences, Harvard University, Cambridge, MA, USA

Vito D'Orazio

Harvard University, Cambridge, USA

Summary of the Paper

This article summarizes arguments in support of reproducibility for scientific research, specifically with respect to computational science. It raises important issues about scholarly communication and reproducibility of previous research work, and it presents ActivePapers as a solution to many problems associated with reproducibility.

Reaction

Upon reading the article, our reaction is twofold. First, we do not understand the author's primary purpose. Is this a review article or a research article? Is this an article summarizing arguments in support of reproducibility? Is it making the case for ActivePapers? Second, regardless of the primary purpose, we believe the work misses key existing technologies and practices.

Structural Issues

Is this a review article or a research article? If this is a review article, replicability and reproducibility should be discussed in more detail, and ActivePapers in less. If this is a research article, we suggest less review of the ActivePapers platform, and more on its contribution and the ways it addresses problems that other systems do not.

The introduction states, *"the work presented in this article takes the opposite approach of starting from the requirements of the scientific record and exploring how software and electronic datasets need to be prepared in order to become useful parts of this record."* What are the requirements of the scientific record? They seem scattered throughout the next few sections, such as *"the most basic requirement for building confidence in computational results is total transparency"* (p. 3). The philosophy of science literature might be helpful in summarizing the basic requirements.

Based on our reading, we do not think the main contributions of this paper are the bulleted items

in the introduction, but rather the ideal platform requirements on page six. These logically flow from a discussion of the requirements of the scientific record followed by a discussion of tools for reproducible and replicable research.

On Data Reuse

There are preservation practices and tools, which follow the approach widely used in Libraries, that help to make a dataset reusable in the longer term:

First, researchers are encouraged to use and share datasets in formats commonly used by their discipline, and when possible, formats that do not depend on proprietary software. When the data file format depends on a specific software (or is not considered a preservation format), there exist software tools that convert the proprietary format to a preservation format. For example, for tabular data in SPSS and STATA format which depend on a specific statistical package, the files can be converted to a plain text (tab-delimited or CSV file) plus a metadata file (in XML or JSON format) that contains information about the columns in the original tabular data file. Information is not lost and it can be re-combine to generate an SPSS or STATA file.

This preservation feature can be found in public data repositories such as the Dataverse repository software (dataverse.org). There are software projects that focus on other similar automated preservation tools to re-format data files into preservation formats and provide additional preservation metadata. One example is Archivematica.

These preservation tools are important for data repositories if they want to make their data accessible and reusable in the future, when the original software might be obsolete.

On Methods/Code/Software

One approach on sharing the code used to model/analyze a research work is by using an open-source language like R, where the models and packages are shared and disseminated through a common R package repository (CRAN). Some R packages have explored solutions for reproducibility by tracking the detailed information on what model/code ran and the computing configuration (see <http://arxiv.org/pdf/1501.02284.pdf>).

Also, there are on-going efforts in reproducibility that support hosting code and provide executable functionality to run in the cloud (<http://researchcompendia.org/>).

Minor Issues

The word *conservation* should be replaced with *preservation*. Conservation generally implies the careful maintenance of a finite resource, while preservation implies the protection of a thing in order to keep that thing as is.

We recommend reviewing the work and guidelines provided by the Data Citation Principles: <https://www.force11.org/datacitation> which can apply to software as well, and should help towards giving due importance to software and providing formal long term access and reuse, if the principles are followed.

In the conclusion: *"the main reason is that most of the computing technology used by scientists was developed outside of scientific research, for domains of application where replicability is not important."* Is this statement supported?

Indexability

This article requires considerable work to be indexed. First, clarify the type of article that is being written, and restructure accordingly. Second, the author misses some key existing technologies and practices that should be discussed.

Competing Interests: No competing interests were disclosed.

We confirm that we have read this submission and believe that we have an appropriate level of expertise to confirm that it is of an acceptable scientific standard, however we have significant reservations, as outlined above.

Author Response 24 Feb 2015

Konrad Hinsén, Centre de Biophysique Moléculaire (CNRS), France

The article has been completely restructured in order to make its primary goal stand out: report the lessons learned from developing and using a platform for publishing and archiving computational science. The revised article starts with an analysis of the needs of the scientific record, deduces technical requirements for a platform designed to meet these needs, and then describes the concrete technical choices made in the two implementations of ActivePapers, followed by a report of the lessons learned from its use. The review of existing technology and the description of the ActivePapers platform are still quite long, but inevitable because I am not aware of any other work I could refer the reader to for this essential background information. Even for fundamental issues such as the requirements of the scientific record, I did not find any publication discussing the specific problems of software and datasets. Software in science is almost always discussed exclusively from the point of view of its utility, ignoring its role as an encoding of scientific knowledge.

I fully agree with the reviewers' point of view on data reuse and how to improve it, but I don't see much adoption of these techniques in my scientific environment. Scientists prefer the simplicity of just publishing the files they have on their Web site or on a no-questions-asked digital repository. It's encouraging to see that other domains have succeeded in establishing better habits.

I was not aware of the work by Becker et al. on the tools switchr and GRANbase, which are based on very similar ideas as ActivePapers. A short comparison has been added to the revised article. I was aware of Research Compendia, which however does not address software preservation or even execution in its current version. I do mention the two cloud-hosting sites with on-line execution that I know of: Exec&Share (<http://www.execandshare.org>) and Elsevier's Collage system (<http://collage.elsevier.com>).

The term "conservation" has been replaced by "preservation" in the revision, and the related issues of software preservation and software deployment are now the common thread around which the presentation is organized. Existing work on the preservation of electronic artifacts has been integrated into the discussion.

Competing Interests: No competing interests were disclosed.

Reviewer Report 09 December 2014

<https://doi.org/10.5256/f1000research.6172.r6774>

© 2014 Chue Hong N. This is an open access peer review report distributed under the terms of the [Creative Commons Attribution License](#), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Neil Chue Hong

Software Sustainability Institute, University of Edinburgh, Edinburgh, UK

I, Neil Chue Hong, have reviewed this research article following the principles set out in the Open Science Peer Review Oath v1 (DOI: [10.12688/f1000research.5686.1](https://doi.org/10.12688/f1000research.5686.1)).

This article by Konrad Hinsén discusses the very important issue of how we capture the detail of the dependencies and environment surrounding the software tools that we use, such that we can guarantee that the tools may be used in the future to replicate research and reuse the tools. Overall, it is a comprehensive summary of most of the area.

However as it stands, I believe that the article could be significantly improved by deciding whether the paper is to be a fully comprehensive "state of the art" summary, looking at the specific advantages and disadvantages of each approach; or one which sets out a shorter summary of the state of the art (as is present in the article in this version), and then goes on to describe the lessons learned from the ActivePapers work, in which case I would suggest a change in title to reflect the emphasis on ActivePapers as a primary example of a platform for publishing and archiving computer aided research. Therefore I am marking this as "Approved with reservations" as it requires a structural change, rather than because I believe the work contained in it is not technically sound.

In both scenarios for how the article could be rewritten, I believe the article would benefit from the use of more devices to highlight key points and comparisons, for instance by use of tables to compare the effect of different types of platforms on the ability to define environmental dependencies, or linkage with input data.

There is one important area of research that should be covered in the discussion of the state of the art - that around "significant properties" of software. In particular, the work of Brian Matthews at STFC in this area has previously considered the issue of capturing and prioritising details of both the environment, and how a computational tool is expected to function, thus forming a theoretical basis for describing the way that many current implementations ranging from dependency managers like Maven, through configuration management tools like Docker, Vagrant and Conda, to virtualisation. See:

1. http://www.jisc.ac.uk/media/documents/programmes/preservation/spsoftware_report_redacted.pdf
2. <http://ijdc.net/index.php/ijdc/article/view/148>

In terms of other areas where I felt that additional discussion would have provoked more debate, these would be around the trade-offs surround floating point operations, a discussion of other bytecode platforms, and around the long lifetime of successful pieces of software, in particular

around trust and how it is mechanically/technically checked.

This last point is illustrated in this example from random sampling:

<https://cryptogenomicon.wordpress.com/2014/10/13/cryptic-genetic-variation-in-software-hunting-a-buffered-41-year-old-bug/>

As minor points that I believe would improve the papers I would suggest the following:

- The term "Web repositories" for platforms like FigShare and Zenodo is not commonly used - indeed, it is more commonly used to refer to repositories of web pages. I would suggest the more commonly used "digital repositories" term, or perhaps "web accessible third party digital repositories"?
- On page 3, it would be useful to describe what makes "Web repositories" better. I would suggest it is cost (most are free for openly licensed deposits) and the ability to generate citable DOI s
- On page 3, when talking about the versioning used by *F1000Research*, it should be clarified that "a DOI refers permanently to a specific version of an article"
- On page 3, the author's example of the DSSP method still being widely used today could be seen to contradict the earlier arguments surrounding the inability of code to preserved effectively.
- Whilst I agree that as stated on Page 5 "This lack of a precisely defined and stable platform for executable code is also the root cause of non-replicability in computer-aided research" I feel that the author could discuss the tradeoffs (mostly optimisation and performance based) in more detail and put across their opinion of which are most important.
- On page 6, I had a little difficulty with the statement that "the scientific aspects of software are always pure computation". I think that I understand what the author means, but as written it makes me want to identify a counter example. In fairness, I haven't been able to find one yet.
- On page 7, I note that all the examples of ActivePapers I could find on FigShare have the author as a co-author on them. This means it is unclear whether ActivePapers Python edition is indeed suitable for the wide variety of scientific research areas, as it is unclear whether they represent a representative selection of the use cases for ActivePapers.
- I believe that the author could do more to support their statement that the fundamental distinction is between "computational tool" and "scientific content". This could probably be done by making it clearer how current platforms do or don't support this conceptual split, and whether those that do support the split lead to a more accurate ability to replicate research at a later date.

As a final note, I would say that it is the tacit convention in *all* science that published results are assumed correct unless there is clear evidence to suggest otherwise.

I believe that with some structural changes to give it a clear narrative emphasis, and better figures to present the information that this research article would provide significant information to the

community in this area.

Competing Interests: The author and I have both participated in the "Code as a Research Object" community which is referenced in this article. I am a co-organiser of the WSSSPE community which is mentioned in the article.

I confirm that I have read this submission and believe that I have an appropriate level of expertise to confirm that it is of an acceptable scientific standard, however I have significant reservations, as outlined above.

Author Response 24 Feb 2015

Konrad Hinsen, Centre de Biophysique Moléculaire (CNRS), France

My goal was not to write a review article on the topic of computational reproducibility, but to present the conclusions from developing ActivePapers and applying it in real-life research projects. The article has been restructured accordingly. The review of prior work has been shortened and reframed in the context of the ActivePapers project. The "lessons learned" from ActivePapers are presented in more detail.

I was not aware of the work of Brian Matthews on software preservation in the context of scientific computing. This is indeed very relevant and is mentioned in the revision. I have also expanded the discussion of bytecode platforms and floating-point operations. The discussion of bytecode platform also addresses the issue of trust and security.

The minor points have all been addressed in the revised manuscript.

Concerning the example of DSSP, it is the method but not the original code that is still widely used. Few users of today's code know that it differs from the originally published method. I do not know if the original code actually implemented the method described in the paper. It is no longer available and I have never seen it.

I have expanded my statement that "the scientific aspects of software are always pure computation" into a more detailed paragraph, and referred to it in several places in the article. I believe that this point is very important but often overlooked. The structure of an ActivePaper, combining software and the data it works on, makes it clear that only pure computations can be replicable. I then found out that this is common knowledge in other fields, such as programming language theory.

I have also expanded the discussion of the factors that have until now prevented the establishment of a stable platform for scientific computing. While performance is often quoted as an important factor, as the reviewer remarks, I do not think this can be backed up by much evidence. To the best of my knowledge, no attempt has been made to design and implement a high-performance yet rigorously defined platform, so it cannot be claimed that this is impossible. The problem is rather that for economical reasons, progress in computing happens as a sequence of small, localized changes: a revision of a language, then a new processor generation, a library update, etc. Each local change must work

correctly and efficiently with the existing ecosystem of computing technology. A stable platform definition requires coordinated changes in several technological layers, which is difficult to achieve.

It is well possible that the only currently published ActivePapers are those that I cite, and for which I am a co-author. I know about a few other groups experimenting with ActivePapers, but they have not published any results in this form yet. However, I do not make the claim that ActivePapers is in its current form a good solution for all branches of computational science. Like any other tool, it was written with specific techniques and workflows in mind. Like for any other tool, only long-time experience will show how universally applicable it is. Moreover, the difficulty of integrating legacy software is a real problem for adoption. This article doesn't pretend to do more than present the lessons learned from the applications that are cited.

I cannot support my opinion about the distinction between tools and scientific content by any empirical evidence, because to the best of my knowledge no existing framework or tool chain for scientific computing supports the implementation of such an approach.

I agree that the tacit assumption of correctness of published results is applied everywhere in science. The particularity of computational science is that this is usually impossible to verify, both for the authors of a scientific study and for their readers.

Competing Interests: No competing interests were disclosed.

The benefits of publishing with F1000Research:

- Your article is published within days, with no editorial bias
- You can publish traditional articles, null/negative results, case reports, data notes and more
- The peer review process is transparent and collaborative
- Your article is indexed in PubMed after passing peer review
- Dedicated customer support at every stage

For pre-submission enquiries, contact research@f1000.com

F1000Research