Preserving and Sharing Software for Transparent and Reproducible Research: A Review

Fernando Rios
Data Managment Services,
Johns Hopkins University
rios@jhu.edu

August 18, 2016
v1.4

Abstract

Safeguarding research outputs so that they remain accessible over time is of great interest to researchers and funders as it enables research to be independently verified, reproduced, and reused. The last 10 to 15 years have seen a great deal of effort on the part of research communities, funders, academic libraries, and private industry towards ensuring that research outputs, especially data, are preserved. However, the same level of attention has not been given to addressing the problem as it relates to computer software. This article reviews preservation approaches, metadata, publishing, and legal challenges related to software produced as part of the research process. Because it shares many of the same issues, work surrounding the preservation of historically or culturally important software such as video games is discussed where appropriate.

1 Introduction

The problem of managing the torrent of valuable knowledge generated by the vast number of researchers, inventors, and innovators has been, and continues to be, a daunting task for those charged with doing so. In the digital domain, this torrent is not only composed of "data" (e.g., results of experiments, musical recordings, images, etc.), but also the set of computer instructions needed to decode, interpret, and add value to the data, i.e., "software". The need to safeguard these kinds of information and make them available to others, both in the present and the future, in an organized way has been recognized almost since the beginning of the digital revolution. Despite this, it has been only in the last 10 to 15 years that the issue has gained wider exposure outside the library and museum community, thanks, in part, to organizations such as the National Science Foundation (NSF) via the so-called Atkins report [1] and JISC (formerly the Joint Information Systems Committee) which put forth the call for a Digital Curation Centre in 2003 [2]. As a result of such efforts, the last decade or so has finally seen a critical mass of those recognizing the importance of managing and preserving digital information in a structured way. This has kickstarted the growth of organizations to serve the varied needs of those tasked with, and benefiting from the preservation digital knowledge. These include the Digital Curation Centre in 2004 (`http://www.dcc.ac.uk`) and its associated conference series (initiated in 2005), the International Conference on Digital Preservation (iPRES) series which began in 2004 (`http://www.ipres-conference.org`), and the Research Data Alliance, founded in 2013 (`https://rd-alliance.org`), among others. The recent book by Ray [3] and a SPEC Kit 334 Research Data Management Services published by the Association of Re-search Libraries [4] provide excellent surveys of the current state of data management strategies and service provision from the point of view of academic libraries.

Although the past decade has seen a great deal of effort expended by the library and research communities in developing methodologies and workflows for managing, sharing and preserving the data portion of knowledge, software has not received the same level of attention (as evidenced by noting the relative paucity of software-related topics in the above mentioned conferences).

This work briefly reviews software preservation and sharing within an academic context and aims to present, in summary form, what has been done, the challenges involved, the current state-of-the art, trends and unsolved problems. The emphasis is on research software, that is, software used or produced as part of the academic research process. However, software such as video games or historical software, is also discussed where appropriate as preservation work conducted in that context informs research software as well. The intended audience consists of library researchers, data management specialists, and those who are invested in helping researchers manage, preserve, and share their software.

2 How is Software Different From Data?

On the surface, data and software are both digital objects which can exist in a variety of prescribed formats and can have a particular set of information that describes them. For instance, an image can exist as a file on a hard disk and can be described by the number of bytes it occupies on disk, a name, a descriptor that identifies the format its encoded in etc. Likewise, a piece of software, be it in executable or source code form, can have a similar set of attributes. This suggests that at least on a basic level, the methods, tools, and techniques currently used in the curation and management of data can also be applied to software. In fact, developing common approaches to handling digital objects in general has been the central aim of the digital curation field since its inception.

First of all, what is software? The National Institute of Standards and Technology (NIST) defines computer software somewhat recursively as “computer programs and associated data that may be dynamically written or modified during execution” [5]. The Institute of Electrical and Electronics Engineers (IEEE) Standards Dictionary Online gives at least 13 different definitions of software found in various IEEE standards publications [6] where distinctions are made between different levels of granularity. For instance, one definition of software in the IEEE dictionary is: “Computer programs, procedures, and possibly associated documentation and data pertaining to the operation of a computer system.” A computer program is then defined as "a combination of computer instructions and data definitions that enable computer hardware to perform computational or control functions." However, taking a broader view, a software product is defined as "The complete set of computer programs, procedures, and associated documentation and data designated for delivery to a user." A particularly succinct definition which is suitable in this discussion is given by Chun [7]: software is "a set of instructions that direct a computer to do a specific task."

With these definitions, software can be thought of as a list of commands that causes a computer to behave in a certain way. On the other hand, data are simply static facts or measurements. Therefore, preserving software is, in essence, a task in preserving this set of behaviors. As an example in the cultural domain, consider the task of ensuring that a video game remains playable in its original form. This task boils down to preserving the set of behaviors so that, given certain inputs, the outputs remain true to some benchmark over time. In the context of research software, the general goal is the same: the reproduction of a computation at some future time, relative to some benchmark. It is important to note that for research software, reproduction may not be the only goal. Other goals may include modification and re-use, achieving legal compliance and accountability, creating heritage value, and enabling continued access to data and services [8]. Nevertheless, reproducibility can be thought of as a precursor to these.

Matthews et al. [9] break down software preservation into four major aspects: storage (backups, format migration, replication), retrieval (the ability to search and retrieve), reconstruction (the ability to re-install or re-build the software so it functions close to the original), and replay (the ability to execute the software so it functions close to the original). Software is different than data in the last two aspects since "...we are more interested in what software does rather than what software is" [9]. Matthews et al. introduce the concepts of "performance" (i.e., the behaviors) of software and the "adequacy" of its preservation. The notion of capturing the performance of software such that a certain level of adequacy is met relative to some set of significant properties is really the crux of the technical side of software preservation. As will be seen in the following sections, actually addressing this is a formidable challenge.

3 Research vs. Cultural Heritage Software

Even though the focus of this discussion is on research software, historically significant software such as video games or other forms of software-based art share many of the same issues.

Work involving the history of computing and other culturally significant software has taken place since at least the mid 1980's to early 1990's [10]. Recently, there have been several successful projects to study the issues surrounding video game preservation. A whitepaper by the Game Preservation Special Interest Group of the International Game Developers Association [11] as well as the Virtual Worlds project report [12] both give a thorough introduction to the salient issues regarding video game preservation including why games are worth preserving, obstacles (technical, social, legal) and how they might be overcome, and a survey of game preservation activities in the community. Many of the obstacles such as media decay, obsolescence, metadata, and legal issues are shared by research software as well. A notable preservation project is the Internet Archive's Software Collection (`https://archive.org/details/software`). In addition to housing historical software, the collection also holds video games which can be played directly in the browser, thereby removing major hurdle in accessing old software (the availability of the original hardware). In the academic library world, a nascent project is the Software Preservation Network (SPN). The SPN has received funding from the Institute of Museum and Library Services (IMLS) to conduct research into software preservation geared at archivists, curators, librarians and other information professionals for the purpose of supporting the long-term access of born-digital content [13].

Although the problems associated with preserving cultural heritage and research software are similar, the goals, and therefore the approaches taken, may vary. For example, in order to preserve a video game, it may be very important that the "feel" of the game be captured so as to recreate the original experience. In this case, a certain level of error may be acceptable (e.g., small graphical glitches) if the overall experience is retained. Furthermore, it may be of great interest to obtain original user manuals and promotional materials in order to give context to the work. On the other hand, for a protein modeling code, the exact method of interaction may not matter as long as the result is "correct". Correctness could mean agreement with previous results for instance. When correctness and overall experience are articulated for a specific case, they contribute to the definition of the performance of the software (see section 2).

Another notable difference is that unlike video games, research software may change rapidly due to the experimental nature of research. In this case, preservation activities are often framed in the context of transparency, accountability, reproducibility, and sharing capability [8, 14–16]. This has a significant impact on the types of information that need to be captured. For instance, examining the source code to scientific software may be of critical importance to verify its correct functioning while for a video game, examining the source it may not be as important.

4 Towards Research Software Preservation

4.1 Motivation

Software preservation has been in the mind of archivists and librarians for some time. However, the focus was not on research software but on the historical significance of software. This view meant that software was seen more like a museum piece to be brought out, examined, and placed back in storage [17–19]. From this perspective, the most obvious approach to preservation is to keep the original hardware (or some form of it) operational. In this area, libraries, industry, government, and other organizations have much experience. For example, there are a multitude of computer museums housing historical hardware and software such as the Computer History Museum in Mountain View, California, and the National Museum of Computing in Bletchley Park (see `https://en.wikipedia.org/wiki/Computer_museum` for a large list).

A strong motivating case for preserving software is retaining the ability to access old or obsolete file formats. This application is of interest not only to libraries and museums but also commercial entities as it enables continued access to the company's completed works. In fact, some large corporations have become adept at keeping critical but obsolete hardware and software running for decades [19]. In this use case, the study of the software itself is not the purpose. Instead, it is a tool that enables the examination of data and workflows which have previously been completed. In the libraries and museums world, this may have scholarly ends while in a commercial setting, the motivation may be monetary in nature (e.g., saving time by reusing existing work). In these scenarios, the software may also often be treated as a piece to be brought out, used as needed, and returned to storage. However, the scenarios straddle the treatment of software as an artifact with the final motivating case which is the reproduction and reuse of workflows (especially for research) in which the software is a part.

While the "museum" approach works (in principle) for historical software or when the software is a tool to be used to examine old files, it is not as useful for research software. The reason is that the frameworks under which the aforementioned preservation cases function may not lend themselves to solving issues surrounding transparency, accountability, reproducibility, and especially sharing. For instance, a software catalog which exists only on physical media makes it difficult for others to reproduce work. Nevertheless, there are general commonalities in preservation which can be broadly placed into two main groups: migrating the software to a new environment (if possible), and simulating the original environment to various degrees of fidelity (sometimes called emulation). See section 5 for a more in-depth review.

4.2 Challenges

The complex and dynamic nature of software compared to data makes it difficult to identify what to preserve and when [20]. Should it be captured at the source-code stage at the executable stage or both? Should the entire software stack be preserved or only the relevant parts? How will the choice affect the reproducibility and reuse of the software? There are currently no well-defined answers to these questions. Perhaps there cannot be well-defined answers due to the wide ranging needs of different software projects and limitations on what resources are available for preservation. For instance, an important piece of code may need to be emulated. However for more transient software, it may be that preserving the software interactions and workflow parameters is more important than the software itself [20]. In addition, there are questions related to making the business case for preservation, including to determining the appropriate preservation approach, and to what level context and other tacit information should be preserved in order to balance preservation requirements with costs [21].

It is widely recognized that adequately capturing software dependencies is an important hurdle that must be addressed regardless of the preservation approach [16, 22–24]. In particular, Thain et al. [24] consider this to be the fundamental challenge in software preservation. Because of the large demands placed on the researcher to address this problem, automating the capture of dependencies (software, data, parameters) and documenting their interactions is critical as this captures important provenance information [14, 24–26].

Other technical challenges are also significant. How can migration of software code be carried out in a robust and transparent way? Software engineering principles provide guidance but there remains the challenge of educating researchers on best practices. Perhaps a more significant challenge is getting researchers and funding agencies to recognize the value in adhering to some minimum level of software engineering principles in the first place, so that proper resources are allocated at the start of a project. In terms of the emulation, many of the significant technical hurdles are generally considered to have been overcome, on small scales at least [22, 24]. However there remain major challenges in terms of the implementation and sustainability of an emulation-oriented software preservation infrastructure at larger scales. These include the need to reduce the costs of creating images, developing a sustainable business model for the maintenance and delivery of emulated software, standardization of emulation systems/formats, maintenance of the emulation software, and the need to work with software rights-holders to ensure the that proprietary software can be preserved [27].

Looking at the sharing and dissemination aspect, describing software for the purpose of discovery and understanding its capabilities is another area fraught with difficulty [22, 28]. For instance, consider the wide range of situations where research software is used, as well as the types of software used, and it becomes immediately clear that for any successful preservation activity, adequately describing the relevant parts of the software is critical. After all, if the software elements cannot be adequately described, what hope is there to document the interrelationships between it and other parts of the research workflow? Without these descriptions, the usefulness of preservation is diminished as it is difficult or impossible to find a particular software artifact and understand its capabilities. The salient question here is determining what parts of software should be identified and what an adequate description consists of. This should be addressed at the level of the research community as needs will vary between communities. A related problem concerns standardizing this metadata (or at least creating a crosswalk) to enable interoperability between the descriptions of each community to increase findability and machine readability.

In supporting the preservation of software, there are several important challenges faced by repositories and software libraries. A highly pertinent one is determining how long to preserve software for. The answer is highly debated and depends on one's perspective. From the cultural heritage standpoint, the answer is ideally "forever". However, research software has a potentially shorter life. The data repositories figshare and Zenodo make guarantees that the software will remain for at least 10 or 20 years respectively (as indicated in their FAQs). However in conversations with various researchers, some consider the actual useful lifetime of much research software to be much shorter, on the order of 5 years before it is superseded. Another challenge is the selection problem, i.e., determining what to preserve in the first place, given limited resources. Another barrier is the treatment of software as data. Many data repositories (e.g., CISRO Data Access Portal, `https://data.csiro.au`, Dataverse, `http://dataverse.org`, and many institutional repositories) already accept software but they treat the deposit in the same way as data, ignoring software-specific information such as the programming language used, the intended operating environment, or dependency information. Perhaps the most difficult challenge faced by repositories is determining who will pay for the infrastructure to support long-term archiving. Currently, several well-known repositories make their services freely available to researchers and funding comes from a variety of public and private sources. For instance, Dataverse is funded by Harvard with additional support from the Alfred P. Sloan Foundation, National Science Foundation (NSF), and Helmsley Charitable Trust while Zenodo is supported by CERN with funding from the European Commission. However, the sustainability of these approaches to making the preservation infrastructure sustainable remains in question. See [29, 30] for a more detailed discussion of this issue.

Academic libraries also face challenges. Although many institutional data repositories already accept software, there has been little activity in ensuring that the software remains executable in the future. Challenges include creating infrastructure for enabling that capability (including technical aspects like emulated or virtualized environments and training of personnel), and dedicating time to develop best practices for creating software that is better engineered to last. In terms of meeting funding mandates (as has been the case with data), although few concrete instances of requirements for software management plans and code sharing have materialized, such requirement may eventually arise from the major funders in the US. Therefore libraries dedicated to providing data management planning services may also need to consider software management planning.

Finally, legal issues related to the use, archiving, distribution, and reuse of research software and data are also significant challenges [14, 27, 31]. These may include things like copyrighted works or the use of licenses that do not allow the distribution or copying of the software. The use of commercial software limits the reuse potential of an algorithm or process while software created by researchers and released under non-open licenses makes it difficult for others to improve on it.

How some of these challenges are being addressed is the subject of the remaining sections in this report.

5 Preservation Approaches

5.1 Overview

Discussion regarding software preservation has taken place mainly in two contexts. The first is related to maintaining access to digital material of historical significance such as historically important software or software-based art [19, 28, 32–34]. The other is concerned with the reproduction, transparency, and sharing of the research workflow. This involves not only the preservation of software but also of other information which gives context to the work, including data collection approaches, the intention behind observations or other actions, tacit knowledge, and the provenance of the research [8, 9, 14, 22, 24, 35–37].

Depending on the context, the details of preservation may vary. However, in both cases, the two most often discussed classes of preservation approaches are: migration and emulation [27, 38]. Here, migration is taken to mean enabling the software to have the capability of being compiled and/or executed in an environment different than the one it was originally compiled/executed on. This may or may not require re-compiling the code using tools appropriate for the new environment, modification of the source code, updating or replacing dependent libraries, or updating associated resources (e.g., sounds or images, configuration files). Emulation is taken to mean simulating the original compilation and/or execution environment within a different environment. This may include simulating both hardware and software components. In this report the term emulation used very loosely as a catch-all which includes any kind of approach which is capable of capturing all or part of a hardware or software environment for use in a potentially different environment. This includes true emulation, virtualization, and containerization. See section 5.2.

The line between emulation and migration blurs when the original environment is captured imperfectly (as is often the case). In this case, detailed knowledge about the software (e.g., dependencies) is required in order to be able to reproduce the desired performance. Therefore, a migration of the software may be needed to address any differences in functionality between the original and the simulated environment. This may include modifying pieces of code to conform to the new environment or reconstructing parts of the original environment using functionality provided by the new environment.

5.2 Emulation

The term "emulation" warrants a further explanation to differentiate it from "virtualization". Although the end result may appear to be the same to the end user (a simulation of the original environment), the two terms refer to different things; According to Rosenthal [27], an emulator translates machine instructions such that it becomes possible to execute the unmodified software on a computer (the host) with a different set of machine instructions than the original, while in virtualization, the software can only be executed if the host uses same instruction set as the original. However, as Rosenthal notes, this difference is not absolute. Here, another term lumped into emulation is containerization. Unlike a virtualized system which partitions physical hardware so that multiple operating systems can share hardware resources while existing independently without interfering with each other, a container partitions the resources of an operating system so that they can be used by multiple pieces of software, each existing independently without interfering with other containers. Although there are use cases in which one might be preferred over the other, both virtualization and containerization can be used to capture and possibly move a software artifact and its dependencies to another environment without (or minimal) modification and reconfiguration. As stated in section 5.1, the term emulation will be used to refer to concept of capturing and replaying the original environment, be it via true emulation, virtualization, containerization, or some combination thereof, except where noted.

Emulation presents a tantalizingly attractive option since, in principle, it can be used to preserve nearly any software without the need to understand how it works, as long as the underlying operating environment can be adequately captured. Capturing the environment only needs to be done once, after which any similar software can be preserved with comparatively less effort. Rothenberg [39, 40] optimistically presented emulation as a silver bullet to the problem of software preservation with the near complete dismissal of other approaches. However this view was quickly challenged as emulation has some significant drawbacks, such as the technical complexity involved in creating and deploying emulators, the sheer number of different systems and configurations that must be considered for emulation, and the fact than an emulator is also software which itself must be preserved [41, 42]. Additionally, blindly using emulation may result in the loss of context. For example, the significance of external information (software dependencies, usage information) may be lost [10].

Despite its inability to completely solve all software preservation problems, emulation has gained much more traction in comparison to migration because regardless of its drawbacks, it is often the only realistic way to conduct preservation, as evidenced by the recent and rapid increase in practical applications [43]. Rosenthal [27] provides a much more in-depth discussion of emulation than allowed by the space here.

There are several notable examples of projects which have taken the emulation approach to preserving software. To enable the rendering of digital objects, the National Library of the Netherlands and the National Archive of the Netherlands developed a modular emulator which is built specifically for digital curation [44]. More recently, there appears to be a shift towards the cloud. Apart from the work by the Internet Archive mentioned in Section 3, the Olive Project (`https://olivearchive.org`) is an initiative aiming to "...establish a robust ecosystem for long-term preservation of software, games, and other executable content." The goal is to encapsulate an execution environment and deliver it across the internet so it can be used directly from a web browser [45]. Although the archive currently stores a limited selection of historical software (which, for legal reasons, is not publicly accessible), future use cases include the preservation of academic work. Billing itself as an "emulation-as-a-service" platform, the bwFLA project (`http://bw-fla.uni-freiburg.de`) has attracted interest for use in the preservation of historical software [46].

On the research software side, virtualization and containerization are beginning to find use for enabling the sharing and reproducibility of research. Chameleon (`https://www.chameleoncloud.org`) and Jetstream (`http://jetstream-cloud.org`) are two examples of scientific cloud services which can package up environments for sharing, aimed at computer scientists and the broader long-tail research communities respectively. The DASPOS project (`https://daspos.crc.nd.edu`) and CERNs OpenData (`http://opendata.cern.ch/?ln=en`) aim to address the capture and preservation of research environments with solutions specific to the high-energy physics community. To this end, DASPOS held a workshop in May 2016 that primarily explored containers as a means to enable reproducible software (`https://osf.io/view/containerstrategies`) Finally, some general-purpose research software repositories such as ResearchCompendia (`http://researchcompendia.org`) and Recomputation.org (`http://recomputation.org`) also have a stated goal of being able to reproduce research by making use of virtualization technology which can be executed in the cloud. However prototypes with this functionality have not yet emerged.

5.3 Migration

Within the research software development context, the idea of migration as a preservation approach is usually presented in the form of recommendations to adopt software engineering best practices such as version control, proper documentation, openness, and workflow tracking [14, 15]. These best practices are meant to enable others to reproduce the research (including software) in a new environment, which is essentially a large part of what migration aims to accomplish. However, there is a recognized need to balance these recommendations with the burden placed on already busy researchers [47].

Migration can be seen as enabling software reuse and is tied to the broader notion of research reproducibility. By capturing information surrounding the purpose and use of software (rather than just the code itself), the migration process becomes more tractable since intent is made explicit rather than having to be deciphered from potentially incorrect code. One way to enable reproducible computational research is to capture the workflow. Although there are several established scientific workflow management tools such as Taverna [48], VisTrails [49], and Kepler [50], there has been a recent trend towards tools which automate the capture of the parameters and dependencies associated with software experiments for the purpose of reproducing those experiments in environments different than the original [24, 25, 51, 52]. In particular, ReproZip [52] is notable because it provides options to run experiments within virtual machines or containers, in addition to using VisTrails to examine and execute the workflow. Although a step in the right direction, these software-specific tools are not currently mature enough for widespread use as they only work with certain operating systems or programming languages.

One might wonder under what circumstances migration is preferable to emulation. Thain et al. [24] provide some insight. They characterize the problem as a choice between “capturing the mess” (emulation) and “encouraging cleanliness” (migration). While emulation is currently the more popular of the two approaches, Thain et al. presented it as ineffective for long-term preservation because of its inefficiencies in storage and execution, its rigid composition (i.e., distinguishing environment components and the software itself is difficult), and the imprecise capture of external dependencies (e.g., a file on a web server). Therefore, if any of those issues are of concern, emulation may not be effective. On the other hand, if cleanliness is encouraged before and during the software development process, migrating may become a feasible approach since all the pieces required to reconstruct the software in the new environment will have already been documented, making them easier to capture either manually or automatically (as in Thain et al.). Finally, cleanliness has the additional benefit of promoting better reuse through good software engineering practices.

5.4 Other Approaches

Migration and emulation are not the only approaches. The Software Preservation Benefits framework [8] is a report from the Software Sustainability institute which aims to make the case for research software preservation to developer groups and funding bodies. The framework includes a variety of “centric” preservation activities. These are techno-centric (preserving the original hardware and software), process-centric (keeping the knowledge about the software ‘alive’, e.g., an open development model), knowledge-centric (storing the software to resurrect it in the future), data-centric (software is kept in the same state via emulation), functionality-centric (update software to maintain functionality). Under this view, migration is said to be functionality-centric while emulation can be classified as data-centric. The process-centric approach can arguably be included as part of migration since an active development community will preserve the software by migrating it to new or updated environments (in principle). Except for the process-centric approach, all the other approaches can also apply to the historical preservation of cultural heritage software.

6 Describing and Disseminating Research Software

6.1 Overview

Preserving research software for preservation's sake is usually not the end goal. Instead, it is so it may be examined, reused, and repurposed by others. To enable this, there needs to be mechanisms that enable describing, sharing, and receiving credit for software. This section discusses recent developments in this area. However, it must be noted that unlike the technical aspects of emulation and migration discussed in the previous section, work in regards to the description and dissemination of software is much less mature. Nevertheless, there has been recent momentum in this regard.

6.2 Software Description and Attribution

The purpose software metadata and citation standards is to enable its unambiguous identification and to communicate an understanding of its capabilities in order to improve reproducibility, reuse, and attribution [47, 53]. Out of all the problems associated with preserving scientific software, developing metadata and citation standards is perhaps the low-hanging fruit, due to the parallels with data citation and attribution. However, a more fundamental problem is determining what should be identified or cited in the first place.

Since different communities may have different needs, determining the level of granularity at which to identify and describe the software is problematic [53, 54]. For instance, should identifiers be assigned down to the level of individual libraries or is a high-level description sufficient? Furthermore, due to the evolving nature of software, determining which software object to identify can also be problematic. For instance which version should be referenced? Or should such a specific reference to a version be made at all? What is the minimal level of information that is required in order to enable reproducibility? González-Beltrán et al. [55] assert that a barrier to reproducible papers is not necessarily a lack of effort on the part of researchers. Instead, it is a lack of understanding of what kind of information should be included in the first place. Gent et al. [54] and Chue Hong [47] provide guidelines to address these challenges in the form of levels of detail.

Gent et al. [54] break down the notion of software into increasingly specific entities starting with the general concept of the software product at the top, followed by a version, a variant, and instance with various kinds of relationships between them (see the reference for further detail). On the other hand, Chue Hong [47] outlines six levels of minimal information required for software discovery and reuse. Level zero is the theoretical minimum amount of information required for reproducibility (though its not necessarily a useful amount). Each subsequent level requires an increasing amount of information related to licensing, availability, quality, support, and incentives which enable developers and users to be rewarded for reuse [47]. The last category, incentives, is perhaps one of the most difficult non-technical challenges in creating reusable and citable software. This is due to the publication-oriented nature of the academic world in which good software is not rewarded [56].

Major funders have begun to take notice of the need to adequately describe and share software and they have committed funds to help their communities tackle some of the associated problems. JISC in the UK and the NIH in the US have both funded projects which aim to address the need to describe software in a specific community: biomedicine. The JISC-funded Software Ontology (SWO) [57] takes a decidedly software-user viewpoint in determining the significant properties of software necessary to reproduce research results. This is unsurprising since the goal of the SWO is "...the description of resources used in storing, managing and analyzing data." Note the lack of emphasis on sharing and dissemination in this case. In the US, the NIH has recognized a need for an index or database of research software which enables the reproducibility of research [16]. As part of this "software discovery index", a set of priorities were developed (including a recommendation for minimal metadata) which paralleled the SWO [58]. Along the same lines as the SWO, the NSF-funded OntoSoft project [59] produced an ontology to describe scientific software, geared towards capturing information relevant to how scientists might share and reuse software. The ontology was implemented in publicly accessible OntoSoft "portals" (community-specific sites) which are metadata repositories that capture researcher-provided descriptions of software.

Although these and other organizations and repositories are starting to investigate software metadata, there is the potential for incompatibilities due to differing requirements. The CodeMeta project (`https://github.com/codemeta/codemeta`), which is a collaboration between Mozilla Science Lab, GitHub (`https://github.com`), figshare (`https://figshare.com`), Zenodo (`https://zenodo.org`), and various researchers at several academic institutions is attempting to address this problem by creating a crosswalk for software metadata which can be used to map between the various metadata schemas and descriptions of software such as those from Zenodo, OntoSoft, SWO and many others.

For cases such as the preservation of software within emulated environments, the metadata landscape is even less clear. The Jetstream project indicates on its home page (`http://jetstream-cloud.org`) that its virtual machines will have DOIs attached to them. However, as of this writing, no details are given as to what kinds of metadata should be attached to these objects. Nevertheless, there are efforts are underway to extend metadata standards such as PREMIS to describe these emulated environments [27].

Giving credit to those that develop software is an area that has received attention from the research community. In terms of citation standards for software, Niemeyer et al. [60] outline the main challenges:

- identifying necessary metadata associated with software for citation,
- standardizing proper formats for citing software in publications,
- establishing mechanisms for software to cite other software (i.e., dependencies),
- developing infrastructure to support indexing of software citations along with (or complementary to) the existing publication citation ecosystem,
- determining standard practices for peer review of software, and
- increasing cultural acceptance of the concept of software as a digital product.

Another challenge is the need to adopt different crediting schemes for software, instead of using static publication-style schemes as these may produce undesirable effects [61]. For example, a fixed author list, may create a disincentive for others to contribute to improving the software because they will not receive credit, thereby increasing the tendency to "fork" code [62]. Another example is the inability of a publication-style citation to give credit to those that contributed indirectly, e.g., by having their code integrated into a piece of software via a library or code snippet. Refer to the work by Howison and Bullard and others for more detail surrounding these issues [61–63].

Depsy [64] is attempting to address the problem of giving credit to software authors when there are many contributions to a project over time by assigning fractional credit based on various factors. Furthermore if an author's work is a dependency of a different piece of software, that author will receive credit. This notion is referred to as transitive credit [65]. The idea of roles to identify different kinds of contributions to publications has been developed by Project CRediT [66], which has roles associated with software contributions. The related PaperBadger project from Mozilla Science Lab (`https://badges.mozillascience.org`) is an experimental service which issues badges corresponding to different roles (e.g., software) linked to specific publications and individuals.

The need to move beyond discussion into action has recently given rise to several workshops which aim to surface challenges and produce recommendations in regards to citation and credit. For example, in 2015, the NSF, in conjunction with the Alfred P. Sloan Foundation held a workshop on software citation [56]. UC Davis also held a similar workshop that year (Software for Science - Getting Credit for Code, October, 2015. `http://icis.ucdavis.edu/?p=765`). The Software Sustainability Institute also held a Software Credit Workshop in October of 2015 (`http://www.software.ac.uk/software-credit`), and credit and citation were topics in the Third Workshop on Sustainable Software for Science: Practice and Experiences (WSSSPE3) [67]. In February 2016, the Laura and John Arnold Foundation held the Arnold III workshop on transparency and reproducibility in modeling and code. (`http://www.aaas.org/event/iii-arnold-workshop-modeling-and-code`). The FORCE11 Software Citation Working group (`https://www.force11.org/group/software-citation-working-group`) has mainly focused on citation practices and their recommendations were presented at the FORCE2016 conference. The software citation principles they produced will serve as guidance for software citation and include a set of use cases and an actionable set of recommendations for researchers to use for citing software. (`https://www.force11.org/meetings/force2016/program/agenda/propelled-force-force11-working-groups`). The Software Sustainability Institute is also tackling these problems as evidenced by the wealth of resources on their website which discuss a variety of topics pertaining to describing, preserving, and citing software, e.g., [8, 22, 47, 53].

6.3 Repositories and Software Libraries

The preservation and sharing of software requires a trusted and persistent location for the software to reside. Of course, what qualifies as trusted and "persistent" can depend on the needs of a particular community For instance, many researchers (and some journals such as

SoftwareX, `http://www.journals.elsevier.com/softwarex`) consider source code repositories such as GitHub to be acceptable places to archive and share code. If long-tern preservation is not required, this may be acceptable. However, these types of code repositories are not intended as long-term archives [68]. For instance, they do not generally make guarantees about the length of preservation or fixity. Furthermore, the metadata attached to code in a code repository may be inadequate for machine readability and attribution purposes.

Archival repositories such as Zenodo, figshare, and the Open Science Framework (OSF, `http://osf.io`) have recognized the popularity of code repositories like GitHub while acknowledging their shortcomings. To enable code archiving in the most frictionless manner possible, both Zenodo and OSF are capable of pulling code directly from GitHub and other services. Zenodo goes further by automatically pulling in code and assigning it a DOI whenever a release is tagged in GitHub, therefore allowing for unambiguous identification of a particular release which makes it easier to cite. Additionally, metadata in these repositories is more structured compared to source code repositories, enabling machine readability. A particularly ambitious project is the Software Heritage archive (`https://www.softwareheritage.org`), launched in June 2016. This project aims to preserve and share all publicly available source code from repositories such as GitHub. Although in its early stages, it has already archived 2,788,611,328 source code files[1]. Although it aims to support software in all its forms, the Software Heritage archive aims to support scientific reproducibility by ensuring the availability and traceability of source code [69].

Apart from general code repositories, community-specific software libraries are another way to enable access to software. In terms of software-based art, the Rhizome (`http://rhizome.org`) is an archive of digital works and has investigated emulation for preservation [31]. In the cultural heritage space, there are many computer history museums that strive to make old software available. A significant effort in preserving software of cultural importance is taking place as part of the UNESCO PERSIST project [70]. In the research space, there are also many examples such as the Astrophysics Source Code Library (`http://ascl.net`) and the NIHs proposed software discovery index [16]. For reference purposes, other efforts, apart from the ones already presented in section 5.2, include an effort between Notre Dame University and the Center for Open Science to develop a reproducible software engineering environment [70], and the previously mentioned OntoSoft Project which allows for the storage and discovery of re-search software knowledge via general or community-specific portals [59].

7 Publishing

As mentioned in section 4, an important motivating factor for preserving research software is transparency and accountability. These factors are linked to what is widely considered the most important research product: publications. Although reproducibility is a cornerstone of scientific inquiry, it has not often been emphasized in the publication process for various reasons such as article length limits, and a focus on "what did you learn" over "how did you get there." The means that much research is not reproducible or verifiable with only the text as a guide

[1]as of August 18, 2016

[14]. However, there has been a growing movement to encourage researchers to share their code. In support of reproducibility, transparency, and credit, journals such as Computer Physics Communications, the F1000 Research journals, the American Astrophysical Society, and Nature Methods have introduced software-related guidelines, with varying levels of detail. To address the general notion of research transparency the Transparency and Openness Promotion (TOP) guidelines [71] have been proposed as a common base for journals to draft their own community-specific guidelines. The journal Science recently adopted the TOP guidelines (effective January, 2017) where publication in the journal is contingent on the sharing of any code used in the paper [72].

Although software papers (papers which describe scientific software in the context of a study) are commonly accepted in many journals, software-specific journals such as SoftwareX, the Journal of Open Research Software (JORS), and the Journal of Open Source Software (JOSS) are fully focused on software as a research output so that it can be elevated to the same status (or at least cited in the same way, as JOSS is meant to advocate) as a traditional publication. Of course, these efforts also require community support for their success.

A logistically difficult aspect of publication in terms of code is the review process. Even with code sharing requirements, performing the most basic of checks (e.g., actually verifying that the submitted code reproduces the work) is a challenge for reviewers since there is a wide variety of different ways software can be developed and packaged. Furthermore, the quality of the installation scripts or instructions can also vary widely. Since reviewers will likely not have access to the original build and execution environment, the ease with which software can be tested is critical to enable software reviews at scale. Since having to overcome these hurdles can quickly eat away at a reviewers limited time, not addressing them makes software reviews untenable. Therefore, an approach which enables transparency, accountability, and reproducibility while making every effort to address the barriers for reviewers becomes very important. In computer science, the concept of artifact evaluation (AE) [73] has been considered for conference papers (the preferred place to publish in that field). Submitting ones code to AE is voluntary (though can be incentivized, possibly via a cash reward as in [73]) and broadly consists of providing standardized packaging guidelines to authors, after which the authors submit the package for testing to reviewers (the package is not publicly distributed, assuaging concerns of having work misappropriated by others). The choice whether to disclose the result of evaluation is left to the authors. Perhaps the lessons learned in the AE experiment in [73] can be applied more broadly to give some structure to the review of research software. Once the effort of creating a software packages and submitting it to AE has been carried out, much of the hard work in creating a quality preservable and shareable artifact will have been done. However, it remains to be determined whether research communities are open to investing the time in doing so.

8 Intellectual Property

Intellectual property is perhaps the most challenging aspect of preserving software [10, 12, 46]. It must be considered for any kind of preservation or sharing activity. A notable example where this issue is manifested is in the use of emulation, especially by institutions. Because an adequate legal framework for this use case has not been established, there has been a reluctance to adopt emulation due to potential problems with copyright (if it is even possible to identify the copyright holder), and license agreements which do not take into account the preservation use case [27]. To begin addressing the issue, the Software Preservation Network's proposal to the Institute for Museum and Library Sciences emphasized the need for "articulating a business model and organizational structure through which software licensing and legitimate, ongoing non-profit re-use can operate harmoniously" [13]. However, their work in that regard is currently ongoing.

In terms of research transparency, the use of proprietary software poses problems in cases where the sharing of software is a requirement for funding or publication. Because of this, care must be taken when drafting policy around sharing requirements in order to acknowledge the fact that not all code can be shared. For instance, in drafting their adoption of the TOP guidelines, Science included a clause to allow researchers to not share their software if there are "truly exceptional circumstances, such as special-purpose code that is proprietary or platform-dependent" [72].

Solutions to the intellectual property issues are few and far between, however some have been proposed [31] though they remain mostly untested. The simplest solution is to not make the software publicly available, as was the case in the AE experiment [73]. This approach is also used by the Olive Archive project where access to archived virtual machines is restricted to research collaborators. However, if the goal is to enable others to reuse and repurpose software, restricting access to comply with licensing issues is obviously not a useful solution.

When the rights to the software are owned by the researcher who created them, the problem becomes more tractable. To lower intellectual property-related barriers related to research reproducibility and reuse, the Reproducible Research Standard (RRS), proposed by Stodden [74], gives a set of licenses that can be applied to the selection and arrangement of data (the data itself is not copyrightable), media, and code. The main advantage to this approach is that it frees the researcher from having to select from a wide range of licenses each time scholarship is released.

Because of the significant hurdles presented by copyright and associated issues, there is no single solution that can address them all. However, at least from the point of view of publishing reproducible research, some journals, e.g., Nature Methods, F1000Research, and the journals of the American Astronomical Society to name a few, are addressing the issue by now requiring or strongly urging authors to submit source code, and requiring or encouraging using an open source license [75–77].

Although small steps have been made, addressing intellectual property issues will likely remain the most challenging aspect of software preservation and sharing for the foreseeable future.

9 Final Words

This report summarizes the current state of software preservation and sharing, with emphasis on work and challenges associated with research software. In this area, preservation is associated with the ability to verify and reproduce scientific results as well as the ability for others to be able to build on and reuse software that has been preserved. The state-of-the art for a wide variety of aspects of software preservation is summarized. These aspects include recent work surrounding the technical side of preservation (e.g., emulation), work around describing, sharing, and citing software. In addition issues around publishing and intellectual property are outlined.

As a result of the combined work of several communities, some efforts are beginning to yield results. Although progress has been made on several fronts, work related to software citation and attribution has produced actionable results (see the software citation principles produced by the FORCE11 software citation working group) as has work on descriptions of software (see the OntoSoft ontology and associated portals). Promising work is being undertaken by the NIH which is in the process of addressing many of the issue surrounding the reproducibility of research (including software sharing) via initiatives such as the cloud commons model of sustainable infrastructure and the initial efforts in investigating a software discovery index.

Although there is much work being done in terms of preserving software, much more remains to be accomplished. Mirroring the data realm, work in some aspects of preserving software (e.g. technical ones) are more mature than others. However, it is the non-technical aspects that are the most challenging. In order to make a stronger value proposition for preserving software, there needs to be a change in cultural attitude by research communities, promotion committees, funders, and publishers to treat software as a valuable first-class scholarly product. Without this, software risks becoming a tool to be discarded on the path to knowledge creation, resulting in lost knowledge and dooming those that follow to needlessly repeat work resulting in wasted resources. Despite their relative infancy, the efforts mentioned in this report have shown promise but a sustained effort, much like has been done with data, is needed to maintain the momentum to make change possible.

References

[1] D. Atkins, *Revolutionizing science and engineering through cyberinfrastructure: Report of the National Science Foundation blue-ribbon advisory panel on cyberinfrastructure*, 2003. [Online]. Available: `https://arizona.openrepository.com/arizona/handle/10150/106224` (visited on Jan. 13, 2016) (cit. on p. 1).

[2] C. Rusbridge, P. Burnhill, S. Ross, P. Buneman, D. Giaretta, L. Lyon, and M. Atkinson, "The digital curation centre: A vision for digital curation," in *Local to Global Data Interoperability – Challenges and Technologies*, IEEE, 2005, pp. 31–41. [Online]. Available: `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1612461` (visited on Jan. 19, 2016) (cit. on p. 1).

[3] J. M. Ray, *Research data management: Practical strategies for information professionals.* Purdue University Press, 2014 (cit. on p. 2).

[4] D. Fearon, B. Gunia, S. Lake, B. E. Pralle, and A. L. Sallans, *SPEC Kit 334: Research data management services*, en. Association of Research Libraries, Jul. 2013. [Online]. Available: `http://publications.arl.org/Research-Data-Management-Services-SPEC-Kit-334/` (visited on Aug. 20, 2013) (cit. on p. 2).

[5] R. Kissel, “Glossary of key information security terms,” National Institute of Standards and Technology, Tech. Rep. NIST IR 7298r2, May 2013. [Online]. Available: `http://nvlpubs.nist.gov/nistpubs/ir/2013/NIST.IR.7298r2.pdf` (visited on Jan. 20, 2016) (cit. on p. 2).

[6] IEEE, *IEEE standards dictionary online*, Accessed Jan 19, 2016. [Online]. Available: `http://ieeexplore.ieee.org/xpls/dictionary.jsp` (visited on Jan. 19, 2016) (cit. on p. 2).

[7] W. H. K. Chun, “On software, or the persistence of visual knowledge,” *Grey Room*, vol. 18, pp. 26–51, 2004. [Online]. Available: `http://www.jedbrubaker.com/wp-content/uploads/2009/12/annot-test.pdf` (visited on Nov. 9, 2015) (cit. on p. 3).

[8] N. Chue Hong, S. Crouch, S. Hettrick, T. Parkinson, and M. Shreeve, “Software preservation benefits framework,” *Software Sustainability Institute Technical Report*, 2010 (cit. on pp. 3, 4, 7, 10, 13).

[9] B. Matthews, A. Shaon, J. Bicarregui, and C. Jones, “A framework for software preservation,” *International Journal of Digital Curation*, vol. 5, no. 1, pp. 91–105, 2010. [Online]. Available: `http://ijdc.net/index.php/ijdc/article/view/148` (visited on Nov. 9, 2015) (cit. on pp. 3, 7).

[10] H. Lowood, “The lures of software preservation,” in *Preserving.exe: Toward a national strategy for software preservation*, T. Owens, Ed., Washington, D.C.: National Digital Information Infrastructure and Preservation Program at the Library of Congress, 2013, pp. 4–11. [Online]. Available: `http://hdl.loc.gov/loc.gdc/lcpub.2013655114.1` (cit. on pp. 3, 8, 15).

[11] D. Monnens, A. Armstrong, J. Ruggill, K. S. McAllister, Z. Vowell, R. Donahue, H. Lowood, International Game Developers Association, and Game Preservation Special Interest Group, *Before it’s too late: A digital game preservation white paper*, English. 2009, ISBN: 978-0-557-05322-3. [Online]. Available: `https://apps.lis.illinois.edu/wiki/download/attachments/3736446/IGDA_Game_Preservation_SIG_-_Before_It’s_Too_Late_-_A_Digital_Game_Preservation_White_Paper.pdf` (visited on Jan. 20, 2016) (cit. on p. 3).

[12] J. McDonough, R. Olendorf, M. Kirschenbaum, K. M. Kraus, D. Reside, R. Donahue, A. Phelps, C. Egert, H. Lowood, S. Rojo, *et al.*, *Preserving virtual worlds final report*, Accessed Nov 12, 2015, 2010. [Online]. Available: `https://www.ideals.illinois.edu/handle/2142/17097` (cit. on pp. 3, 15).

[13] Z. Vowell, J. Meyerson, and C. Ovalle, *Software Preservation Network project grant application to the Institute of Museum and Library Services National Leadership Grants for Libraries program*, 2015. [Online]. Available: `https://www.imls.gov/sites/default/files/proposal_narritive_lg-73-15-0133_cal_poly_corporation.pdf` (visited on Dec. 20, 2015) (cit. on pp. 4, 16).

[14] V. Stodden, "Best practices for computational science: Software infrastructure and environments for reproducible and extensible research," en, *Journal of Open Research Software*, vol. 2, no. 1, e21, 2014. DOI: `10.5334/jors.ay`. [Online]. Available: `http://openresearchsoftware.metajnl.com/articles/10.5334/jors.ay/` (visited on Nov. 9, 2015) (cit. on pp. 4, 5, 7, 9, 15).

[15] W. C. Lenhardt, "Data management lifecycle and software lifecycle management in the context of conducting science," en, *Journal of Open Research Software*, vol. 2, no. 1, e15, 2014, ISSN: 2049-9647. DOI: `10.5334/jors.ax`. [Online]. Available: `http://openresearchsoftware.metajnl.com/articles/10.5334/jors.ax/` (visited on Nov. 9, 2015) (cit. on pp. 4, 9).

[16] V. Bonazzi, P. Bourne, S. Brenner, R. Brown, I. Chandramouliswaran, J. Couch, S. Davis, L. Derr, A. Dhar, L. Dunlap, K. Eliceiri, L. Finnegan, I. Fore, M. Haendel, M. Hammitzsch, T. Huyen, D. S. Katz, M. Kellen, D. Kennedy, J. Larkin, J. Lin, P. Lyster, R. Margolis, G. Marth, M. Martone, M. McLennan, M. Morgan, F. Ouellette, V. Pai, A. Prlic, W. Schroeder, M. Sherman, H. Sofia, J. Taylor, K. Thaney, C. Wellington, and O. White, *Software discovery index workshop report*, Mar. 2015. [Online]. Available: `https://nciphub.org/resources/885` (visited on Jan. 22, 2016) (cit. on pp. 4, 5, 12, 14).

[17] D. Swade, "The problems of software conservation," *Resurrection - The Bulletin of the Computer Conservation Society*, no. 7, 1993. [Online]. Available: `http://www.computerconservationsocie` `org/resurrection/res07.htm` (visited on Oct. 1, 2015) (cit. on p. 4).

[18] ——, "Preserving software in an object-oriented culture," in *History and Electronic Artefacts*, H. Edward, Ed., Oxford: Clarendon Press, 1998, pp. 195–206 (cit. on p. 4).

[19] J. G. Zabolitzky, "Preserving software: Why and how," *Iterations: An Interdisciplinary Journal of Software History*, vol. 1, no. 13, pp. 1–8, 2002. [Online]. Available: `http://www.cbi.umn.edu/iterations/zabolitzky.pdf` (visited on Oct. 6, 2015) (cit. on pp. 4, 7).

[20] N. Chue Hong, "Digital preservation and curation: The danger of overlooking software," in *The Preservation of Complex Objects*, J. Delve, D. Anderson, M. Dobreva, D. Baker, C. Billenness, and L. Konstantelos, Eds., vol. 1, The University of Portsmouth, 2012, pp. 25–35, ISBN: 978-1-86137-630-5 (cit. on p. 5).

[21] B. Matthews, S. Crompton, C. Jones, and S. Lambert, "Towards the preservation of the scientific memory," *International Journal of Digital Curation*, vol. 10, no. 1, Feb. 2015, 0000, ISSN: 1746-8256. DOI: `10.2218/ijdc.v10i1.361`. [Online]. Available: `http://ijdc.net/index.php/ijdc/article/view/361` (visited on Feb. 4, 2016) (cit. on p. 5).

[22] N. Chue Hong, "Dealing with software: The research data issues," Edinburgh, United Kingdom: Figshare, 2014. DOI: `10.6084/m9.figshare.1150298` (cit. on pp. 5–7, 13).

[23] K. Rechert, D. von Suchodoletz, and I. Valizada, "Future-proof preservation of complex software environments," in *Proceedings of the 9th International Conference on Preservation of Digital Objects*, R. Moore, K. Ashley, and S. Ross, Eds., Toronto, October 1 - 5, 2012, pp. 180–183. (visited on Nov. 9, 2015) (cit. on p. 5).

[24] D. Thain, P. Ivie, and H. Meng, "Techniques for preserving scientific software executions: Preserve the mess or encourage cleanliness?" In *Proceedings of the 12th International Conference on Digital Preservation (iPRES)*, Nov. 2015. [Online]. Available: `http://dx.doi.org/doi:10.7274/R0CZ353M` (visited on Nov. 9, 2015) (cit. on pp. 5–7, 10).

[25] T. McPhillips, T. Song, T. Kolisnik, S. Aulenbach, K. Belhajjame, R. K. Bocinsky, Y. Cao, J. Cheney, F. Chirigati, S. Dey, J. Freire, C. Jones, J. Hanken, K. W. Kintigh, T. A. Kohler, D. Koop, J. A. Macklin, P. Missier, M. Schildhauer, C. Schwalm, Y. Wei, M. Bieda, and B. Ludascher, "Yesworkflow: A user-oriented, language-independent tool for recovering workflow information from scripts," *International Journal of Digital Curation*, vol. 10, no. 1, Feb. 2015, ISSN: 1746-8256. DOI: `10.2218/ijdc.v10i1.370`. [Online]. Available: `http://ijdc.net/index.php/ijdc/article/view/370` (visited on Feb. 5, 2016) (cit. on pp. 5, 10).

[26] B. Ludascher, I. Altintas, S. Bowers, J. Cummings, T. Critchlow, E. Deelman, D. D. Roure, J. Freire, C. Goble, M. Jones, S. Klasky, T. McPhillips, N. Podhorszki, C. Silva, I. Taylor, and M. Vouk, "Scientific process automation and workflow management," in *Scientific data management: Challenges, technology, and deployment*, Boca Raton: CRC Press, 2010 (cit. on p. 5).

[27] D. S. Rosenthal, "Emulation & virtualization as preservation strategies," 2015, Accessed Nov 12, 2015. [Online]. Available: `https://mellon.org/Rosenthal-Emulation-2015` (visited on Feb. 1, 2016) (cit. on pp. 6–9, 12, 16).

[28] H. Lowood, *Playing history with games: Steps towards historical archives of computer games.* 00023, 2004. [Online]. Available: `http://aic.stanford.edu/sg/emg/library/pdf/lowood/Lowood-EMG2004.pdf` (cit. on pp. 6, 7).

[29] P. E. Bourne, J. R. Lorsch, and E. D. Green, "Perspective: Sustaining the big-data ecosystem," *Nature*, vol. 527, no. 7576, S16–S17, 2015 (cit. on p. 6).

[30] F. Berman and V. Cerf, "Who Will Pay for Public Access to Research Data?" *Science*, vol. 341, pp. 616–617, 2013 (cit. on p. 6).

[31] M. McKeehan, *Intellectual Property Rights Issues for Software Emulation: An Interview with Euan Cochrane, Zach Vowell, and Jessica Meyerson*, eng, webpage, Jan. 2016. [Online]. Available: `http://blogs.loc.gov/digitalpreservation/2016/01/intellectual-property-rights-issues-for-software-emulation-an-interview-with-euan-cochrane-zach-vowell-and-jessica-meyerson/` (visited on Mar. 2, 2016) (cit. on pp. 7, 14, 16).

[32] J. Newman, *Best before: Videogames, supersession and obsolescence.* Routledge, 2012 (cit. on p. 7).

[33] E. Kaltman, N. Wardrip-Fruin, H. Lowood, and C. Caldwell, "A Unified Approach to Preserving Cultural Software Objects and their Development Histories," 2014. [Online]. Available: `http://escholarship.org/uc/item/0wg4w6b9.pdf` (visited on Nov. 9, 2015) (cit. on p. 7).

[34] L. Shustek, "What should we collect to preserve the history of software?" *IEEE Annals of the History of Computing*, no. 4, pp. 112–110, 2006. [Online]. Available: `http://www.computer.org/web/csdl/index/-/csdl/mags/an/2006/04/man2006040112.html` (visited on Nov. 9, 2015) (cit. on p. 7).

[35] J. Freire, P. Bonnet, and D. Shasha, "Computational reproducibility: State-of-the-art, challenges, and database research opportunities," in *Proceedings of the 2012 ACM SIGMOD international conference on management of data*, ACM, 2012, pp. 593–596 (cit. on p. 7).

[36] J. Freire, C. Silva, S. Callahan, E. Santos, C. Scheidegger, and H. Vo, *Managing rapidly-evolving scientific workflows*, 2006. [Online]. Available: `http://dx.doi.org/10.1007/11890850_2` (cit. on p. 7).

[37] V. Stodden, F. Leisch, and R. Peng, *Implementing Reproducible Research*, ser. Chapman & Hall/CRC The R Series. CRC Press, 2014 (cit. on p. 7).

[38] J. Rothenberg, *Avoiding technological quicksand: Finding a viable technical foundation for digital preservation: A report to the Council on Library and Information Resources.* Washington, DC: Council on Library and Information Resources, 1999, ISBN: 978-1-887334-63-1 (cit. on p. 7).

[39] ——, "Ensuring the Longevity of Digital Documents," *Scientific American*, vol. 272, no. 1, pp. 42–47, 1995 (cit. on p. 8).

[40] ——, "Ensuring the Longevity of Digital Information," Tech. Rep., 1999. [Online]. Available: `http://www.clir.org/pubs/archives/ensuring.pdf.` (visited on Nov. 9, 2015) (cit. on p. 8).

[41] S. Granger, "Emulation as a digital preservation strategy," *D-Lib Magazine*, vol. 6, no. 10, 2000. DOI: `10.1045/october2000-granger` (cit. on p. 8).

[42] D. Bearman, "Reality and Chimeras in the Preservation of Electronic Records," *D-Lib Magazine*, vol. 5, no. 4, 1999, 00115, ISSN: 1082-9873. DOI: `10.1045/april99-bearman` (cit. on p. 8).

[43] P. Wheatley, *Themes of 2015*, 2016. [Online]. Available: `https://digipresnews.wordpress.com/2016/02/10/digital-preservation-news-themes-of-2015/` (visited on Feb. 10, 2016) (cit. on p. 9).

[44] J. van der Hoeven, B. Lohman, and R. Verdegem, "Emulation for digital preservation in practice: The results," *The International Journal of Digital Curation*, vol. 2, no. 2, pp. 123–132, 2007 (cit. on p. 9).

[45] M. Satyanarayanan, G. St. Clair, B. Gilbert, Y. Abe, J. Harkes, D. Ryan, E. Linke, and K. Webster, "One-click time travel," Technical report, Computer Science, Carnegie Mellon University, Tech. Rep. CMU-CS-15-115, 2015. [Online]. Available: `http://reports-archive.adm.cs.cmu.edu/anon/anon/usr/ftp/usr0/ftp/2015/CMU-CS-15-115.pdf` (visited on Feb. 10, 2016) (cit. on p. 9).

[46] E. Cochrane, *Emulation as a service (EaaS) at Yale University Library*, 2014. [Online]. Available: `http://blogs.loc.gov/digitalpreservation/2014/08/emulation-as-a-service-eaas-at-yale-university-library/` (visited on Oct. 30, 2015) (cit. on pp. 9, 15).

[47] N. Chue Hong, *Minimal information for reusable scientific software*, 2014. [Online]. Available: `http://dx.doi.org/10.6084/m9.figshare.1112528` (visited on Feb. 5, 2016) (cit. on pp. 9, 11, 13).

[48] K. Wolstencroft, R. Haines, D. Fellows, A. Williams, D. Withers, S. Owen, S. Soiland-Reyes, I. Dunlop, A. Nenadic, P. Fisher, J. Bhagat, K. Belhajjame, F. Bacall, A. Hardisty, A. Nieva de la Hidalga, M. P. Balcazar Vargas, S. Sufi, and C. Goble, "The Taverna workflow suite: Designing and executing workflows of Web Services on the desktop, web or in the cloud," *Nucleic Acids Research*, vol. 41, no. W1, W557–W561, 2013, ISSN: 0305-1048, 1362-4962. DOI: `10.1093/nar/gkt328`. [Online]. Available: `http://nar.oxfordjournals.org/lookup/doi/10.1093/nar/gkt328` (visited on Mar. 2, 2016) (cit. on p. 10).

[49] S. P. Callahan, J. Freire, E. Santos, C. E. Scheidegger, C. T. Silva, and H. T. Vo, "VisTrails: Visualization meets data management," in *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, New York, NY, USA: ACM, 2006, pp. 745–747, ISBN: 1-59593-434-0. DOI: `10.1145/1142473.1142574`. [Online]. Available: `http://doi.acm.org/10.1145/1142473.1142574` (visited on Mar. 2, 2016) (cit. on p. 10).

[50] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludascher, and S. Mock, "Kepler: An extensible system for design and execution of scientific workflows," in *16th International Conference on Scientific and Statistical Database Management, 2004. Proceedings*, 2004, pp. 423–424. DOI: `10.1109/SSDM.2004.1311241` (cit. on p. 10).

[51] L. Murta, V. Braganholo, F. Chirigati, D. Koop, and J. Freire, "Noworkflow: Capturing and analyzing provenance of scripts," in *Provenance and Annotation of Data and Processes*, ser. Lecture Notes in Computer Science 8628, B. Ludascher and B. Plale, Eds., Springer International Publishing, 2014, pp. 71–83. [Online]. Available: `http://link.springer.com/chapter/10.1007/978-3-319-16462-5_6` (visited on Mar. 2, 2016) (cit. on p. 10).

[52] F. Chirigati, D. Shasha, and J. Freire, "ReproZip: Using Provenance to Support Computational Reproducibility," in *5th USENIX Workshop on the Theory and Practice of Provenance*, Berkeley, CA: USENIX, 2013. [Online]. Available: `https://www.usenix.org/conference/tapp13/reprozip-using-provenance-support-computational-reproducibilitythe` (cit. on p. 10).

[53] M. Jackson, *How to cite and describe software*, Accessed Dec. 18, 2015, 2012. [Online]. Available: `http://software.ac.uk/how-cite-and-describe-software` (visited on Dec. 18, 2015) (cit. on pp. 11, 13).

[54] I. Gent, C. Jones, and B. Matthews, "Guidelines for Persistently Identifying Software using DataCite - A JISC Research Data Spring Project," Tech. Rep., 2015. [Online]. Available: `http://rrr.cs.st-andrews.ac.uk/wp-content/uploads/2015/10/guidelines-software-identification.pdf` (visited on Dec. 11, 2015) (cit. on p. 11).

[55] A. González-Beltrán, P. Li, J. Zhao, M. S. Avila-Garcia, M. Roos, M. Thompson, E. van der Horst, R. Kaliyaperumal, R. Luo, T.-L. Lee, T.-w. Lam, S. C. Edmunds, S.-A. Sansone, and P. Rocca-Serra, "From peer-reviewed to peer-reproduced in scholarly publishing: The complementary roles of data models and workflows in bioinformatics," *PLoS ONE*, vol. 10, no. 7, e0127612, 2015. DOI: `10.1371/journal.pone.0127612`. [Online]. Available: `http://dx.doi.org/10.1371/journal.pone.0127612` (visited on Feb. 3, 2016) (cit. on p. 11).

[56] S. Ahalt, T. Carsey, A. Couch, R. Hooper, R. Idaszak, M. B. Jones, J. Lin, and E. Robinson, "NSF Workshop on Supporting Scientific Discovery through Norms and Practices for Software and Data Citation and Attribution," Arlington, Virginia, Tech. Rep., 2015. [Online]. Available: `https://softwaredatacitation.org/Pages/home.aspx` (visited on Dec. 16, 2015) (cit. on pp. 11, 13).

[57] J. Malone, A. Brown, A. L. Lister, J. Ison, D. Hull, H. Parkinson, and R. Stevens, "The Software Ontology (SWO): A resource for reproducibility in biomedical data analysis, curation and digital preservation," *Journal of biomedical semantics*, vol. 5, no. 1, p. 25, 2014. [Online]. Available: `http://www.biomedcentral.com/content/pdf/2041-1480-5-25.pdf` (visited on Jan. 6, 2016) (cit. on p. 12).

[58] J. Malone, *It needs to be better than Google: A response to the NIH Software Discovery Index Report*, 2014. [Online]. Available: `http://drjamesmalone.blogspot.com/2014/10/it-needs-to-be-better-than-google.html` (visited on Mar. 2, 2016) (cit. on p. 12).

[59] Y. Gil, V. Ratnakar, and D. Garijo, "OntoSoft: Capturing scientific software metadata," en, in *Proceedings of the Eighth ACM International Conference on Knowledge Capture, Palisades, NY*, ACM Press, 2015, pp. 1–4, ISBN: 978-1-4503-3849-3. DOI: `10.1145/2815833.2816955`. [Online]. Available: `http://dl.acm.org/citation.cfm?doid=2815833.2816955` (visited on Apr. 12, 2016) (cit. on pp. 12, 14).

[60] K. E. Niemeyer, A. M. Smith, and D. S. Katz, "The challenge and promise of software citation for credit, identification, discovery, and reuse," *ArXiv preprint arXiv:1601.04734*, 2016. [Online]. Available: `http://arxiv.org/abs/1601.04734` (visited on Mar. 2, 2016) (cit. on p. 12).

[61] J. Howison and J. Bullard, "Software in the scientific literature: Problems with seeing, finding, and using software mentioned in the biology literature," en, *Journal of the Association for Information Science and Technology*, 2015, ISSN: 2330-1643. DOI: `10.1002/asi.23538`. [Online]. Available: `http://onlinelibrary.wiley.com/doi/10.1002/asi.23538/abstract` (visited on Mar. 3, 2016) (cit. on p. 13).

[62] J. Howison and J. D. Herbsleb, "Incentives and integration in scientific software production," in *Proceedings of the 2013 Conference on Computer Supported Cooperative Work*, ser. CSCW '13, New York, NY, USA: ACM, 2013, pp. 459–470. [Online]. Available: `http://doi.acm.org/10.1145/2441776.2441828` (visited on Mar. 3, 2016) (cit. on p. 13).

[63] ——, "Scientific software production: Incentives and collaboration," in *Proceedings of the ACM 2011 Conference on Computer Supported Cooperative Work*, ser. CSCW '11, New York, NY, USA: ACM, 2011, pp. 513–522. DOI: 10.1145/1958824.1958904. [Online]. Available: http://doi.acm.org/10.1145/1958824.1958904 (visited on Mar. 3, 2016) (cit. on p. 13).

[64] H. Piwowar and J. Priem, *Depsy: Valuing the software that powers science*, Accessed Jan. 22, 2016, 2016. [Online]. Available: https://github.com/Impactstory/depsy-research/blob/master/introducing_depsy.md (visited on Mar. 3, 2016) (cit. on p. 13).

[65] D. Katz, "Transitive credit as a means to address social and technological concerns stemming from citation and attribution of digital products," en, *Journal of Open Research Software*, vol. 2, no. 1, 2014, ISSN: 2049-9647. DOI: 10.5334/jors.be. [Online]. Available: http://openresearchsoftware.metajnl.com/articles/10.5334/jors.be/ (visited on Mar. 3, 2016) (cit. on p. 13).

[66] A. Brand, L. Allen, M. Altman, M. Hlava, and J. Scott, "Beyond authorship: Attribution, contribution, collaboration, and credit," en, *Learned Publishing*, vol. 28, no. 2, pp. 151–155, 2015, ISSN: 09531513, 17414857. DOI: 10.1087/20150211. [Online]. Available: http://doi.wiley.com/10.1087/20150211 (visited on Jan. 6, 2016) (cit. on p. 13).

[67] D. S. Katz, S.-C. T. Choi, K. E. Niemeyer, J. Hetherington, F. Löffler, D. Gunter, R. Idaszak, S. R. Brandt, M. A. Miller, S. Gesing, N. D. Jones, N. Weber, S. Marru, G. Allen, B. Penzenstadler, C. C. Venters, E. Davis, L. Hwang, I. Todorov, A. Patra, and M. de Val-Borro, "Report on the third workshop on sustainable software for science: Practice and experiences (WSSSPE3)," *ArXiv:1602.02296 [cs]*, 2016, arXiv: 1602.02296. [Online]. Available: http://arxiv.org/abs/1602.02296 (visited on Mar. 3, 2016) (cit. on p. 13).

[68] M. Fenner, *Software Citation Workflows*, 2015. [Online]. Available: https://blog.datacite.org/software-citation-workflows/ (visited on Dec. 5, 2015) (cit. on p. 14).

[69] Software Heritage Project, *Mission - an essential infrastructure for science*, 2016. [Online]. Available: https://www.softwareheritage.org/mission/science (visited on Aug. 18, 2016) (cit. on p. 14).

[70] Center for Open Science, *The Center for Open Science and the University of Notre Dame partner to advance technologies that support long-term solutions for open science initiatives*, 2016. [Online]. Available: https://cos.io/pr/2016-01-26/ (visited on Jan. 27, 2016) (cit. on p. 14).

[71] B. A. Nosek, G. Alter, G. C. Banks, D. Borsboom, S. D. Bowman, S. J. Breckler, S. Buck, C. D. Chambers, G. Chin, G. Christensen, M. Contestabile, A. Dafoe, E. Eich, J. Freese, R. Glennerster, D. Goroff, D. P. Green, B. Hesse, M. Humphreys, J. Ishiyama, D. Karlan, A. Kraut, A. Lupia, P. Mabry, T. Madon, N. Malhotra, E. Mayo-Wilson, M. McNutt, E. Miguel, E. L. Paluck, U. Simonsohn, C. Soderberg, B. A. Spellman, J. Turitto, G. VandenBos, S. Vazire, E. J. Wagenmakers, R. Wilson, and T. Yarkoni, "Promoting an open research culture," en, *Science*, vol. 348, no. 6242, pp. 1422–1425, 2015. DOI: 10.1126/science.aab2374. [Online]. Available: http://www.sciencemag.org/content/348/6242/1422 (visited on Dec. 9, 2015) (cit. on p. 15).

[72] M. McNutt, “Taking up TOP,” en, *Science*, vol. 352, no. 6290, pp. 1147–1147, 2016, ISSN: 0036-8075, 1095-9203. DOI: `10.1126/science.aag2359`. [Online]. Available: `http://www.sciencemag.org/cgi/doi/10.1126/science.aag2359` (visited on Jun. 6, 2016) (cit. on pp. 15, 16).

[73] S. Krishnamurthi, “Artifact evaluation for software conferences,” *ACM SIGPLAN Notices*, vol. 48, no. 4S, pp. 17–21, 2013 (cit. on pp. 15, 16).

[74] V. Stodden, “The legal framework for reproducible scientific research: Licensing and copyright,” *Computing in Science & Engineering*, vol. 11, no. 1, pp. 35–40, 2009. (visited on Mar. 2, 2016) (cit. on p. 16).

[75] D. Evanko, *Guidelines for algorithms and software in Nature Methods : Methagora*, 00000, 2014. [Online]. Available: `http://blogs.nature.com/methagora/2014/02/guidelines-for-algorithms-and-software-in-nature-methods.html` (visited on Jun. 7, 2016) (cit. on p. 16).

[76] T. Ingraham, *Software article & channelupdates*, 2015. [Online]. Available: `http://blog.f1000research.com/2015/11/05/software-article-channel-updates/` (visited on Apr. 25, 2016) (cit. on p. 16).

[77] A. A. Society, *Policy Statement on Software*, 00000, Jan. 2016. [Online]. Available: `http://journals.aas.org/policy/software.html` (visited on Jan. 2, 2016) (cit. on p. 16).

Changes

Ver	Date	Changes	Name
1.0	2016-06-10	Initial release	FR
1.1	2016-07-19	Change title and lightly revise abstract to emphasize reproducible research	FR
1.2	2016-08-01	Light revision to 4.1 to clarify the accessing old files use case	FR
1.3	2016-08-08	Migrate to LaTeX. Small clarification in last paragraph of 4.1, second to last paragraph of 4.2, clarification in 5.3 regarding maturity of software-specific tools, clarified the final paragraph of 6.1, corrected some typos	FR
1.4	2016-08-18	Correct Neil Chue Hong citations, add Software Heritage project	FR