Stimulating Reproducible Software Artifacts

Luis Oliveira, David Wilkinson, Daniel Mossé, and Bruce R. Childers
School of Computing and Information, Department of Computer Science, University of Pittsburgh
loliveira,dwilk,mosse,childers@pitt.edu

ABSTRACT

Concerns about software-based science reproducibility are ubiquitous throughout the research community as nearly every field of science depends on experiments using computational artifacts. Recently, federally funded research has been mandated releasing data artifacts, but not software and workflow artifacts. Whenever accessible, software artifacts are typically hard to install/use, and experiment workflows are poorly documented and/or ad hoc. This increases the difficulty of replicating results and reusing computational artifacts, slowing down scientific progress. Several software systems have emerged to address this challenge. In this paper, we describe an approach to evaluate these software systems and to determine well they meet needs to replicate and reuse software, data, and workflow artifacts. As an example of the approach, we evaluate our own Occam system.

CCS CONCEPTS

• **General and reference → Experimentation**; **Evaluation**; *Experimentation*; Evaluation; • **Software and its engineering → Software libraries and repositories**; *Software libraries and repositories*; *Open source model*.

KEYWORDS

Reproducibility, workflow, collaboration, reuse

ACM Reference Format:
Luis Oliveira, David Wilkinson, Daniel Mossé, and Bruce R. Childers. 2020. Stimulating Reproducible Software Artifacts. In *3rd International Workshop on Practical Reproducible Evaluation of Computer Systems (P-RECS 20), June 23, 2020, Stockholm, Sweden.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3391800.3398177

1 INTRODUCTION

Computational reproducibility has been elevated to a first-class concern, and researchers are now trying to make their software, data, and workflow artifacts more widely available through web pages or repositories, like GitHub [18], Bitbucket [6], GitLab [19], myExperiment [15], etc. However, the software, data, and metadata are often incomplete (e.g., authors might fail to provide scripts, parameters, configuration files, etc.) to fully repeat experiments, due to undocumented dependencies: either they don't compile or run (readily or at all), or they produce different results (e.g., a change in how `glibc` handles floating point numbers can lead to significantly different results [20]).

As a solution to these problems, we implemented an open-source prototype system called OCCAM [26] to define, conduct, and share experiments as executable content. The platform combines and expands the traits of many other frameworks to support the full research lifecycle, from developing and exploring initial ideas to associating executable content with scholarly publication [8]. OCCAM has a federated organization to marshal contributed data storage and computational resources to support many users that produce large amounts of data and require large amounts of processing power. OCCAM includes a way to identify and index content, a data store for inputs and results of experiments, and a mechanism to record provenance. OCCAM also permits the creation and sharing of computational experiments in the form of interactive, dynamic workflows that can be changed and re-executed to generate new results, all of which are saved and can then be shared.

2 RELATED WORK

The body of work in software and experiment curation mirrors the concern of many researchers in recent years, resulting in many different frameworks to share and rerun computational experiment but suffer from high fragmentation and limited scope. There are emerging standards for packaging content in projects such as Popper [23], Research Objects [5], and DataMill [13]. The most basic tools, and currently the ones with the most use, are online code and data repositories such as GitHub [18], GitLab [19], and BitBucket [6] commonly used by industry, academics, and individuals for collaborative development and code sharing. Similarly, but oriented towards scientific research, websites such as RunMyCode [30], MyExperiment [15], Zenodo [34], Open Science Framework [28], and two independent projects both called Datahub [12][11] have been created to share code, datasets, and experimental workflows. Like OCCAM, these platforms allow for search, find, and download software, but they do not support executable experiments.

There are tools that focus on rerunning software experiments while avoiding "dependency hell", where a user typically struggles to find undocumented dependencies and their correct versions. For that purpose, tools such as ReproZip [29], CDE [7], and Sumatra [14] capture the information about the running software, dependencies and their versions to ensure that nothing is different between executions. ReproZip and CDE go one step further and package all the necessary files such that the experiment can be reproduced in another computer through virtualization. However, this technique does not easily allow for software reutilization, for example, reading a different input file may require an unpackaged dependency. Umbrella [24] describes a software environment and generates VMs on the fly before running experiments, containing all the software required to run an experiment, making changes to the environment possible. Software packed in Umbrella is highly modularized and can be reused in different experiments; however, Umbrella expects all software to be in binary form, not source code; thus not allowing researchers to build on previous work and limiting scientific progress. In addition, these tools do not expose configuration options that allow users to understand or change the behavior of artifacts, also making the re-purposing of those tools difficult. In contrast, in OCCAM, developers have code together with build scripts, allowing other users to examine, clone, modify, build, and run their code (if permissions allow), thus minimizing the effort of improving an artifact and running new experiments with it.

Some services run code in the cloud, such as Umbrella, NanoHub [25], and Code Ocean [10]. However, NanoHub focuses on nanotechnology research, and Code Ocean neither tracks the history of the code nor allows the creation of experiments that combine several steps into a workflow. These aspects are vital reproducibility and curation of software systems; OCCAM provides natively for the creation and execution of experimental workflows that can be easily reconfigured and modified in order to test new hypothesis.

Several tools exist to implement workflows such as Taverna [33], Galaxy [16], and Kepler [4]. Taverna focuses on workflows alone and relies on external entities to provide the artifact services, while Galaxy tools cannot define their own isolated running environment, and Kepler only supports Java applications. In addition, similarly to some of previous tools, OCCAM tracks the lineage and provenance of all of its contents, allowing the inspection of the history of every piece of code and experiment; and OCCAM provides easy to use front-end services, that lower the technical barrier for new users. A fun example: we can play Super Mario Bros. in a NES emulator [3] and Doom [2] in a DOS emulator on OCCAM.

3 REPRODUCIBLE EXPERIMENTS

Different approaches have been taken towards creating better and more reproducible software, from building on-line code repositories for software (e.g., RunMyCode [30], MyExperiment [15]) to systems that capture the execution of software and allow to execute it again (e.g., Reprozip [29]). More recently, publishers and research funders have joined these efforts by implementing new procedures in the publishing process, such as Artifact Evaluation, and by requiring data management plans as part of approved projects.

Such commendable efforts have resulted in multiple tools that try to address specific concerns, but do not fully satisfy them [31]. A platform that provides all the services required to perform reproducible experiments is still missing. Below we describe the requirements for such software tool.

3.1 Levels of reproducibility

A set of requirements for reproducible research was recently proposed to evaluate software preservation tools according to their ability towards preserving verifiable software experiments [31]. The levels of reproducibility are organized as a pyramid, with degrees of preservation, see Fig. 1. In this paper, we divide the pyramid in three main categories, **Accessibility** (Shareable, retrievable, and discoverable artifacts), **Executability** (Deployable, and Repeatable artifacts), and **Interactive** (Configurable and derivable artifacts), each containing services that must be provided to preserve computational experiments. Note that each level of the pyramid provides more value than the previous, but previous levels are not strictly required by the latter. For example, an experiment can be repeatable without being shareable, however, it is not very useful for scientists to be able to repeat an experiment unless they have access to it, to inspect, study, and understand.

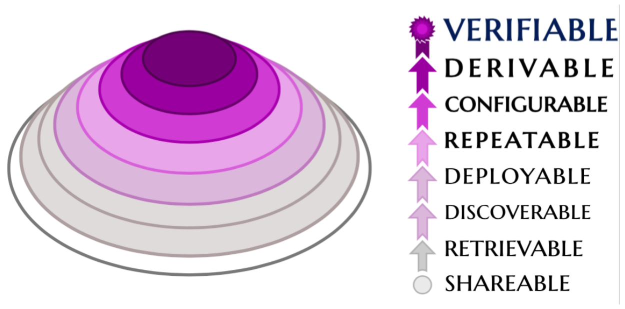

Figure 1: The reproducibility pyramid [31].

3.2 Accessibility of software artifacts

At the bottom of pyramid, the property that should be provided to everyone is **Accessibility** that includes sharing, retrieving, and discovering software artifacts. For artifacts to be useful, they must exist and be shareable. In order to be shared, artifacts need to be packaged/encapsulated. And the packaging needs to be flexible, because software is heterogeneous and may have different packaging requirements. Yet, some basic elements need to exist in any packaged artifact; an *ID*, which uniquely identifies the object, and a version to prevent unexpected results from untracked changes (e.g., a "temporary" change to a configuration for debug causing all experiments to stop working).

Assuming artifacts are shareable, they must also be findable (e.g., through a URL or search) and retrievable (i.e., downloaded). Some retrievable-only artifacts are restricted access artifacts; therefore, an artifact that can be discovered, that is, a public artifact, provides stronger reproducible results. Clearly, most on-line software repositories already fit this description.

A much more interesting prospect is to be able to discover artifacts using the results they produce, and vice-versa (rather than the usual search by keywords). This feature can be invaluable, allowing researchers to quickly discover relevant artifacts and experiments. For example, a researcher finds a many-variable plot that displays the information in a way that can be used for his work; with this feature, he could easily access the software that plotted that information and possibly use it in his own work.

3.3 Preserving Execution

At the center of the pyramid is the **Executability** level. To satisfy this level, a system must provide the means to deploy (i.e., run) the software artifacts, and when the executions is repeated (i.e., in the same experiment) it must produce the same conclusions.

The first aspect an experiment preservation system needs to determine is whether it preserves the binaries only or if it preserves the source code. The different trade-offs between the two approaches are summarized in Table 1.

	Binaries	Code
Pros	• Small storage footprint • Only non-statically linked runtime libraries must be preserved	• Can be rebuilt for any compatible system • Code can be inspected and modified to correct bugs
Cons	• Highly dependent on running environment (needs compatible SW libraries) • Limited portability to other systems (needs the same HW)	• Larger storage footprint (code needs to be stored) • Needs to preserve the source of all dependencies for building and running

Table 1: Pros and cons of approaches for software preservation

Binary preservation (i.e., storing binary files instead of source code) is popular for software distribution as evidenced by many OS package management systems. It has also been used in tools that replicate previous results (e.g., ReproZip [29], Umbrella [24], CDE [7]). The advantage to this choice is the smaller storage footprint and reduced installation times versus distributing code and compiling it. However, this approach only preserves the ability of running the same experiment multiple times with no possibility of modifying the software or inspecting the code. There is also often limited information about how the software was compiled.

Conversely, preserving all the source code enables reproducible research. Most software contains errors [22], and a failed reproduction study only shows there is an error, not where the error is (e.g., is it a software bug, is it a dataset formatting error, etc). As such, preserving the code, the ability to inspect it, and to use it at a later time, is vital to verify claims and root out those errors. Preserving the code also allows other researchers to build on it, by re-using and modifying software artifacts, and deploying the artifact with possibly different (but compatible) library versions, which permits leveraging the software in new ways for new results. To be thorough, even visualize tools used should be preserved.

Preserving software is more than storing source code. The ability to execute that source code, both for replicating previous results and to produce new results, needs to be preserved. We advocate full **source code preservation**, which implies that both build and run instructions must be preserved. Building and running software poses its own challenges that must be solved. Namely, to avoid undocumented dependencies and their versions, the commonly called *dependency hell*, all dependencies must be documented (as metadata) and preserved as artifacts. Then, all elements in an experiment can be modified and re-compiled, allowing users to build on them or quickly reuse them in experiments, even if their original artifact source code is no longer available [21].

Even re-executing the same software with reliable deployment and repeatable results is not easy: It requires preserving not only the artifacts themselves, but also their dependencies and the commands to run the software [27]. For example, updating a library may lead to accidentally breaking the execution of some artifact. Therefore, experiments should be sandboxed in a *virtual environment* (e.g., Docker containers or VirtualBox virtual machines) that contain the correct software. Only then can experiments reliably produce the same results, instead of unexpectedly different results. In fact, there are two main approaches to sandboxing experiments: (1) create the environment in a VM and preserve the VM; or, (2) preserve the environment files (i.e., all software dependencies with enough detail to reconstruct the environment and to execute the software) and create the environment on-demand. The pros and cons of these approaches are summarized in Table 2.

For the first approach, several VM technologies can be used to preserve an experiment's execution environment. Some technologies, such as VirtualBox and Xen, implement full system virtualization, while others, such as Docker and LXC, implement OS-level virtualization where some of the components are shared with the host system, notably the kernel. Preserving and distributing a VM [1, 9] is a widely used approach to facilitate the evaluation of experiments and artifacts as part of peer review. However, in the long term, preserving VMs is not the best solution [32]. First, VM are dependent on the software that runs them, and therefore, when that software is deprecated, the VM may become useless. Second, reusing VMs is both inefficient and complex; for example, (a) changing the input of a software artifact preserving the original experiment requires the clone of the complete VM and (b) extracting a library from a VM to use in another VM can be daunting.

In contrast, storing all artifacts and their dependencies separately, and assembling them on-demand is a method that is not dependent on any specific technology. In fact, it can be used with any virtualization system, or natively deployed (sometimes it may even be required for a hardware-dependent application). Due to the flexibility of this approach, any execution back-end can be replaced, and possibly emulated, if the original is no longer available.

3.4 Interactivity

At the top level of the pyramid, an **Interactive** system that allows experiments and software artifacts to be (re-)configured and derived (i.e., also allowing for modification of existing artifacts), although providing a replay button is important (e.g., to deploy experiments on different hardware to verify software behavior).

From our experience as programmers, all code has bugs, it's just a matter of time until they surface. As such, unless we are able to modify the code to fix those bugs and correct previous results, preserving the source code does not provide much improvement over binaries. Similarly, static experiments that cannot be modified are of limited usefulness, for example, for reviewers and other scientists that want to review and reuse the experiment to explore different configurations or datasets, explore the configuration space to produce corroborating results, or simply to compare results with similar artifacts.

4 OVERVIEW OF OCCAM

In this section we describe our system to create reproducible research softwere, and refer back to Section 3.1 pointing at the desired

	Preserve system-level VM	Build on demand
Pros	• Mostly independent of the running host • Ready to use	• Software can be deployed in different virtualization technologies • Allow composable software modules
Cons	• Large storage footprint • Dependent on the virtualization software • Small changes require large data duplication	• Highly dependent on documentation (dependencies must be documented or it will fail) • Possibly high overhead on first execution (pay one time only!)

Table 2: Pros and cons of different approaches for sandboxing executions

properties. OCCAM is a platform to create, run and share software-based experiments. Users contribute their own artifacts, which can be combined with other (own or others') artifacts and used in *workflows* to define and conduct experiments. An experiment is the execution of a workflow, which is, in turn, a directed acyclic graph of artifacts that includes configurations, input parameters, input data, and code. Workflows, software artifacts, datasets, and results from experiments, like all content curated in OCCAM, can also be shared with (**shareable**), modified and reused (**configurable and derivable**) by specific groups of users. OCCAM supports a range of software and data types; it is designed to be agnostic to the underlying contents (i.e., the artifacts can interoperate).

OCCAM was designed specifically for *executable experiments*, where all execution is automated (**deployable**). OCCAM orchestrates the execution of the workflow, including (1) container provisioning, job scheduling on distributed resources, and data transfers; (2) recording and curation of results from execution of experiments; and, (3) capturing provenance and lineage information about an executed experiment (**repeatable**). With this information, OCCAM provides access to any experiment in a way that it can be inspected, modified and rerun in a subsequent experiment.

OCCAM has a multi-instance organization to marshal contributed computational and storage resources into a federation. As depicted in Fig. 2, a user can instantiate their own local instance of OCCAM, which can use private resources to execute experiments and store content. The user's local instance can be connected to other instances to form a federation, which may be private (single machine, research group's, or a university's computational cluster or data repository) or public such as cloud hardware resources.We provide access control for each user and each resource.

Using OCCAM, users can import their own artifacts into a federation, create and conduct experiments in any instance they have access to, and share experiments and results with other users (e.g., collaborators or reviewers). After a user marks a workflow as shareable, other users can access/retrieve/use (**retrievable**) that workflow either by knowing its identifier or by using the search utility, which allows for searching by name, type, or other tags present in the metadata of each object (**discoverable**). Experiments in OCCAM are modifiable and re-runnable to allow users—regardless of their role, i.e., researcher, reviewer, and paper reader—to interact with the experiments. For example, experiments can be shared for Artifact Evaluation [9], where peer reviewers check that the artifacts and experiments support the conclusions of a scholarly article under review. Lastly, if a paper contains an OCCAM link, a user can re-run that experiment or run it with different configuration parameters that were not considered in the original article. In fact,

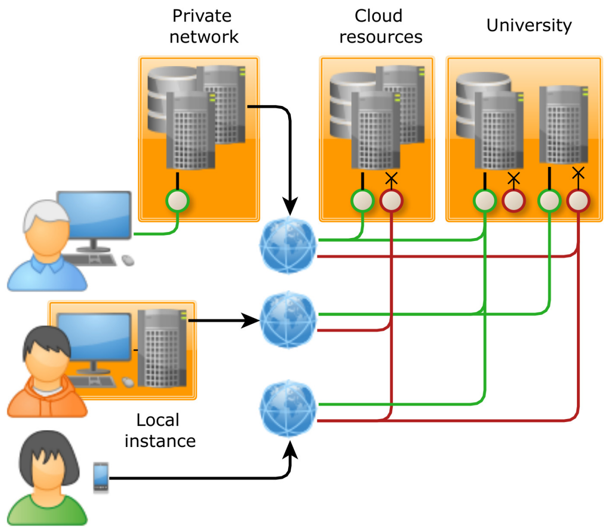

Figure 2: OCCAM's federation: Multiple resources linked

with appropriate support in a digital library, we have implemented a mechanism in which clicking on a plot in a published article takes the user to the experiment setup in OCCAM [17].

5 CONCLUSIONS

We presented our philosophy of execution environments for reproducible research and introduced several desired properties of such systems, namely the desire for artifacts to be shareable, retrievable, discoverable, deployable, repeatable, configurable, and derivable. We also presented an overview of OCCAM, our framework for the interactive curation of artifacts and experiments, our design of how OCCAM provides the desired properties.

OCCAM brings together features that exist in other frameworks, thus improving access to computational artifacts, and providing the infrastructure to replicate the results created by those artifacts. OCCAM preserves software and artifacts, as well as provenance of software and results, allowing further examination of and experimentation with the software, configuration, input data and parameters that generated results, plots, and data of an experiment. OCCAM overheads are small when compared with the actual execution of common simulators.

REFERENCES

[1] [n.d.]. Guidelines for Packaging AEC Submissions. www.artifact-eval.org/guidelines.html. [Online; accessed 18-Jun-2018].

[2] 2018. DOOM 1.9 Shareware Installer. https://occam.cs.pitt.edu/QmTmfRq3HUvV2itA8dfqariyXNvxSUMnuocTSpMoZBEdjC/5dqtbxG91jBgHn2STp3x5dd6vDUmoU

[3] 2018. Super Mario Bros. (JU). https://occam.cs.pitt.edu/QmaYMdsbVoBX31Tej64PuCRYoeMJ2WmnR1FnsSdZPvoxQV/5du5rM2XWup4coXDvHeqf7fzPntU6w

[4] Ilkay Altintas, Chad Berkley, Efrat Jaeger, Matthew Jones, Bertram Ludascher, and Steve Mock. 2004. Kepler: an extensible system for design and execution of scientific workflows. In *Scientific and Statistical Database Management, 2004. Proceedings. 16th International Conference on.* IEEE, 423–424.

[5] Sean Bechhofer, John Ainsworth, Jitenkumar Bhagat, Iain Buchan, Philip Couch, Don Cruickshank, Mark Delderfield, Ian Dunlop, Matthew Gamble, Carole Goble, Danius Michaelides, Paolo Missier, Stuart Owen, David Newman, David De Roure, and Shoaib Sufi. 2013. Why Linked Data is Not Enough for Scientists. 29 (02 2013).

[6] Bitbucket [n.d.]. Bitbucket: Unlimited private code repositories. bitbucket.org. [Online; accessed 17-Nov-2014].

[7] cde [n.d.]. CDE: Lightweight application virtualization for Linux. http://www.pgbovine.net/cde.html. [Online; accessed 29-Sep-2017].

[8] Bruce Childers, Jack Davidson, Wayne Graves, Bernard Rous, and David Wilkinson. 2016. Active curation of artifacts is changing the way digital libraries will operate. In *Proceedings of the Fourth Workshop on Sustainable Software for Science: Practice and Experiences (WSSSPE4)*, Vol. 1686.

[9] Bruce R. Childers, Grigori Fursin, Shriram Krishnamurthi, and Andreas Zeller. 2016. Artifact Evaluation for Publications (Dagstuhl Perspectives Workshop 15452). *Dagstuhl Reports* 5, 11 (2016), 29–35. https://doi.org/10.4230/DagRep.5.11.29

[10] codeocean [n.d.]. Code Ocean - Discover and Run Scientific Code. https://codeocean.com. [Online; accessed 29-Sep-2017].

[11] datahub-io [n.d.]. DataHub. http://datahub.io. [Online; accessed 18-Sep-2017].

[12] datahub-mit [n.d.]. DataHub. https://datahub.csail.mit.edu/www/. [Online; accessed 18-Sep-2017].

[13] DataMill [n.d.]. DataMill - Open Benchmarking Platform. https://datamill.uwaterloo.ca/. [Online; accessed 29-Aug-2016].

[14] Andrew P Davison, Michele Mattioni, Dmitry Samarkanov, and B Teleńczuk. 2014. Sumatra: a toolkit for reproducible research. *Implementing reproducible research* 57 (2014).

[15] David De Roure, Carole Goble, and Robert Stevens. 2009. The design and realisation of the Virtual Research Environment for social sharing of workflows. *Future Generation Computer Systems* 25, 5 (2009), 561–567.

[16] Galaxy [n.d.]. Galaxy Project – Online bioinformatics analysis for everyone. https://galaxyproject.org/. [Online; accessed 29-Aug-2016].

[17] William C. Garrison, Adam J. Lee, and Timothy L. Hinrichs. 2014. An Actor-based, Application-aware Access Control Evaluation Framework. In *Proceedings of the 19th ACM Symposium on Access Control Models and Technologies* (London, Ontario, Canada) *(SACMAT '14)*. ACM, New York, NY, USA, 199–210. https://doi.org/10.1145/2613087.2613099

[18] GitHub [n.d.]. GitHub: Build software better, together. github.com. [Online; accessed 12-Dec-2014].

[19] GitLab [n.d.]. The leading product for integrated software development - GitLab. gitlab.com. [Online; accessed 9-Oct-2017].

[20] Tristan Glatard, Lindsay B. Lewis, Rafael Ferreira da Silva, Reza Adalat, Natacha Beck, Claude Lepage, Pierre Rioux, Marc-Etienne Rousseau, Tarek Sherif, Ewa Deelman, Najmeh Khalili-Mahani, and Alan C. Evans. 2015. Reproducibility of neuroimaging analyses across operating systems. *Frontiers in Neuroinformatics* 9 (2015), 12. https://doi.org/10.3389/fninf.2015.00012

[21] Software Heritage. [n.d.]. Software is fragile. ([n. d.]). https://www.softwareheritage.org/mission/software-is-fragile/ [Online; accessed 30-Oct-2018].

[22] Darrel C Ince, Leslie Hatton, and John Graham-Cumming. 2012. The case for open computer programs. *Nature* 482, 7386 (2012), 485.

[23] I. Jimenez, A. Arpaci-Dusseau, R. Arpaci-Dusseau, J. Lofstead, C. Maltzahn, K. Mohror, and R. Ricci. 2017. PopperCI: Automated reproducibility validation. In *2017 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. 450–455. https://doi.org/10.1109/INFCOMW.2017.8116418

[24] H. Meng, D. Thain, A. Vyushkov, M. Wolf, and A. Woodard. 2016. Conducting reproducible research with Umbrella: Tracking, creating, and preserving execution environments. In *2016 IEEE 12th International Conference on e-Science (e-Science)*. 91–100. https://doi.org/10.1109/eScience.2016.7870889

[25] nanoHUB [n.d.]. nanoHUB.org — Online Simulation and More for Nanotechnology. nanohub.org. [Online; accessed 20-Sep-2014].

[26] Occam [n.d.]. Distribution at. occam.cs.pitt.edu.

[27] Luís Oliveira, David Wilkinson, Daniel Mossé, and Bruce Childers. 2018. Supporting Long-term Reproducible Software Execution. In *Proceedings of the First International Workshop on Practical Reproducible Evaluation of Computer Systems* (Tempe, AZ, USA) *(P-RECS'18)*. ACM, New York, NY, USA, Article 6, 6 pages. https://doi.org/10.1145/3214239.3214245

[28] OpenScienceFramework [n.d.]. OSF | Home. https://osf.io/. [Online; accessed 29-Aug-2016].

[29] ReproZip [n.d.]. ReproZip - About. https://vida-nyu.github.io/reprozip/. [Online; accessed 29-Aug-2016].

[30] RunMyCode [n.d.]. Run My Code. https://runmycode.org/. [Online; accessed 29-Aug-2016].

[31] David Wilkinson, Luis Oliveira, Bruce Childers, and Daniel Mosse. 2017. Evaluating Interactive Archives. https://doi.org/10.6084/m9.figshare.5483836.v1

[32] David Wilkinson, Luís Oliveira, Daniel Mossé, and Bruce Childers. 2018. Software Provenance: Track the Reality Not the Virtual Machine. In *Proceedings of the First International Workshop on Practical Reproducible Evaluation of Computer Systems* (Tempe, AZ, USA) *(P-RECS'18)*. ACM, New York, NY, USA, Article 5, 6 pages. https://doi.org/10.1145/3214239.3214244

[33] Katherine Wolstencroft, Robert Haines, Donal Fellows, Alan Williams, David Withers, Stuart Owen, Stian Soiland-Reyes, Ian Dunlop, Aleksandra Nenadic, Paul Fisher, et al. 2013. The Taverna workflow suite: designing and executing workflows of Web Services on the desktop, web or in the cloud. *Nucleic acids research* (2013), gkt328.

[34] zenodo [n.d.]. Zenodo. https://zenodo.org/. [Online; accessed 29-Aug-2016].