Evaluating Interactive Archives

David Wilkinson, Luís Oliveira, Daniel Mossé, and Bruce Childers

Computer Science Department
University of Pittsburgh

***Abstract*—The concept of reproducibility has been the keystone of both ancient and modern scientific methods. In spite of this, digital science has recently been put to task to improve its failing record of repeatable experimentation. A plethora of digital archives have appeared in response, yet the community has not defined the end goal. There exists no means of comparing or evaluating digital archives nor the quality of preserved software, and thus no means of knowing if the tools are valid toward that goal. A metric for evaluating software sustainability is provided and used to define a metric for evaluating and comparing interactive software archives.**

I. The Crisis of Software Preservation

In the past several years, it has become evident that software—which is a particular form of digital media often overlooked with respect to preservation—is driving scientific research. A recent survey of UK scientists suggests that modern research is in fact impossible without software [1]. Yet, in a reproducibility study done within the field of Computer Science, which is obviously dominated by software-backed research, shows that only 32% of recent CS research is reproducible [2]. That is, the digital results of typical CS studies can not be replicated. Other fields have similarly raised this alarm: 90% of scientists surveyed by Nature indicated there is a slight or significant reproducibility crisis [3].

This crisis should be alarming to all of us. After all, the scientist Ibn al-Haytham in the early 11th century suggested science must be a process that encourages repetition [4]; this idea influenced Galileo and Newton hundreds of years later. Without the step of repeatability, coincidence could be mistaken for cause. Therefore, an experiment has to be devised in such a way that it can be independently verified by a second experiment. Yet, much of our digital science today can not be reproduced due to lost or unpublished software code. The CS reproducibility study showed that only 35% of code was publicly available. Even when code is available, this study found that 15% of code did not work at all when used by an independent reviewer [2]. This situation highlights the importance of software preservation in science.

Consequently, software preservation has become a new focus of the scientific world; commercial and academic projects have sprouted to fill the void for software preservation. There are emerging standards for packaging content in projects such as Popper [5], Research Objects [6], and DataMill [7]. Existing online code and data repository systems geared toward the tech industry such as GitHub [8], GitLab [9], and BitBucket [10]. Similar websites such as RunMyCode [11], MyExperiment [12], defunct Research Compendia, Zenodo [13], Open Science Framework [14], and two independent projects both called Datahub [15][16] have been created to host scientific code and data. There are also software frameworks that a researcher can leverage to build their own software, such as torch.ch [17] and GenePattern [18]. There are tools to capture and replay the dynamic execution of software such as ReproZip [19], CDE [20], and Sumatra [21]. Another strategy are curated collections of software such as Madagascar [22]. Tools to design and evaluate workflows, including Taverna [23], Galaxy [24], VisTrails [25], and Kepler [26]. A plethora of literate programming solutions to combine code and prose such as Jupyter [27], Collage [28], and Beaker Notebook [29] have appeared. Services have sprung up to run science code in the cloud, like Chameleon Cloud [30], NanoHub [31], and two commercial services: defunct Wind River Helix Lab Cloud and newcomer Code Ocean [32]. Finally, there are tools that describe a software environment and generate virtual machines on the fly such as Umbrella [33], Simulocean [34], and Occam [35]. In fact, the problem has shifted somewhat from a reproducibility crisis to *an issue of there being too many solutions and knowing which ones to use!*

II. Toward Evaluating Software Preservation

It is obvious that there is every motivation to create software archives that can keep digital artifacts for scientific research running. Yet, we have no means to evaluate these archival systems. How can we compare one artifact to another and determine which is packaged better? Turned around, how do we know which archive system meets our goals when there are so many? The broader scientific software community needs to put effort into defining what quality means for both artifacts and archives. In this short paper, we briefly define such a criteria, apply it by looking at existing archives and their drawbacks, and how it shows through experience that this metric is appropriate.

To our knowledge, there is no other evaluation metric for digital archive systems. However, there has been prior discussion about what quality means for the artifacts themselves. C. Titus Brown briefly illustrated in 2013 what they called "The Ladder of Academic Software Reusability + Sustainability" [36] which defined a series of goal posts software can achieve toward sustainability, which is the ability for that software to be reused in the future. At the first level, software had to be downloadable. At another level, it had to be installable and runnable. At a higher level, the software can repeat its prior results, and at the highest rung the code could be modified

Presented at Gateways 2017, University of Michigan, Ann Arbor, MI, October 23-25, 2017. https://gateways2017.figshare.com/

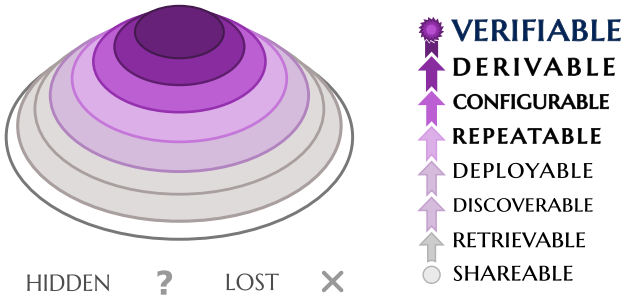

Fig. 1: The Degrees of Software Sustainability

and redistributed. In this design, software at a higher position on the ladder is presumed to have the attributes at the lower rungs. The higher the software, the higher the quality of its preservation.

This type of ladder inspired Neil Chue Hong's "Five Stars of Research Software," [37] which is a short essay that similarly defines a set of important attributes with respect to sustainable research software. In a follow-up paper in 2014, he goes on to define levels of quality for software reusability. At level 0, software can merely be retrieved. Jumping to level 3, software can be legally modified and has adequate usage documentation and attribution. At level 4, the software has good automated test coverage and has documented dependencies. [38] The final level is an idealistic goal: 100% test coverage, correctness, and so on. It is this high bar that developers are meant to adhere for the most reusable software.

The open data community has offered its own version of sustainability guidelines called FAIR [39]. This breaks down to Findable, defined as searchable and globally identifiable; Accessible, retrievable via an open and public protocol; Interoperable, formatted in a common and known fashion; and Reusable, appropriately licensed and well-described. It should be noted that FAIR is meant to promote the accountable preservation of data and not necessarily software. However, software archival is clearly important, if not essential, toward their goal of interoperability and reusability. This is yet another example of a community pushing the importance of software repeatability and derivation.

III. The Ladder of Software Sustainability

At their core, these types of ladders and guidelines describe software quality in broad strokes. They do not define specifically what types of features or best practices achieve the sustainability they covet. This vagueness is a weakness since it is difficult to understand how to apply such qualifications to a particular artifact. For instance, Hong specifically suggests documentation and automated testing help lower the technical barrier of derivation which improves the reuse of software. That does seem intuitive; reuse obviously implies use, and knowing how to interact with software certainly plays a part. However, does that relate directly to its quality? For one, it is not clear good documentation correlates with software reusability. Nor is it clear how one would measure what good means, or how to compare the quality of multiple artifacts.

However, these works do illustrate common and accepted notions of software quality that bridge communities. First, they strongly suggest that availability is the absolute minimum requirement. Second, both ladders show derivation, and making that process easier, as the goal toward the higher end of software preservation. These ladders represent the goals of the digital research community. In order to weigh the effectiveness of any archival system, the community does need a definition of the quality of a software artifact. To that end, The Ladder of Software Sustainability shown in Figure 1 is introduced as a refinement of the aforementioned work.

Much like the work of Brown and Hong, the first rung of the ladder represents the shareability of the artifact. Simply put, the software is packaged. If the software cannot be packaged, it must be lost, where it no longer exists, or hidden, where it exists but is privately held. This is the minimum requirement for software to exist. The next rung captures the idea of software being retrievable, which means the artifact can be downloaded if you know where it is. Sometimes this means you are given a link or permission to download the software where it may be hidden for most people. Therefore, a stronger artifact is one which is "discoverable" by some search engine or through a public service. In short, a public entity has a stronger quality than an obscured or private entity.

The next two rungs capture the execution of software. A "deployable" artifact can be executed, but may not be correct or may generate different results in different environments. A program that can avoid those pitfalls would be considered "repeatable". This difference may not seem significant at first. However, isolating the environmental effects of differing hardware or system software has inspired an entire field of software in the form of virtualization tools. Hardware virtualization, within tools like VMWare [40] or VirtualBox [41], emulate hardware interfaces to fool software into seeing the same machine each execution. Other tools, such as Docker [42], fool software into seeing the same operating system regardless of the system software the individual is personally using. Each technique may be the difference between software being deployable or repeatable.

The following two rungs speak toward the artifact's ability to change. The prior work of Brown and Hong feature derivation as a goal post, but do not go into detail beyond the application of open source licensing. Yet, there are other forms of derivation and change that could be enabled within archives. For one, it seems reasonable to have the ability to change artistic artifacts or manipulate scientific visualizations through software modification. There is also the creation of new artifacts via either cloning existing ones or using software to produce data which itself needs to be archived. The very nature of preserving such data suggests the requirement of preserving the software that is used to read it. This complicates the scope of derivation. In the end, we want to lose this read-only, "do not touch" mentality (borrowed from existing museums,) since it does not apply for digital curation, particularly with respect

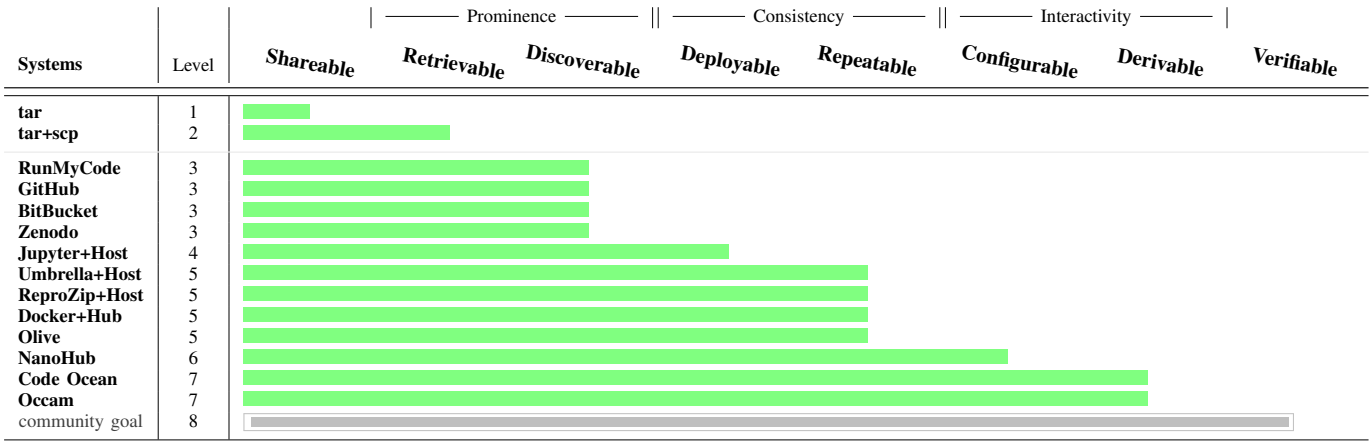

TABLE I: Comparing different archival systems in their support for software sustainability. Some presume to have some "Host" serving the artifacts, such as GitHub. tar, an application that simply packages files together, and scp, an application that simply copies files across machines, serve as a baseline. This is determined by examining their feature sets in the other tables.

to interactive software and data, and to the general goal of scientific exploration.

To this end, the "configurable" rung speaks to the specific act of being able to modify how an artifact runs. That is, software may be able to take different input files than ones used in a prior experiment. Another case is when software has defined configuration parameters which can be manipulated without changing its code. The modification of code is then pushed to the "derivable" rung. Closed source software may be configurable, but it is not easily derivable. However, being derivable and configurable increases the flexibility of reusing the software.

Yet, by itself, this ladder falls prey to a similar flaw. It is still difficult to understand the value of applying it to a particular artifact when they can be produced or packaged in a variety of ways using different tools. Simply, how do you compare one against another? If one tool provides a mechanism for repeatability, such as Docker, and another tool uses a different strategy, say VMWare, which is better? It would reduce to comparing those specific technologies, implying this problem is alleviated when comparing artifacts that were produced with the same tools. Therefore, applying this evaluation metric to the artifact is less informative than applying it to the product of particular tools. That is, the archival system itself can be evaluated by looking at the degrees of sustainability that are promoted by that system. The more features in particular areas defined by the ladder a tool has, the stronger its support for that aspect of preservation.

In the end, the idea represented by this ladder is that software artifacts gain, for each quality they capture, an increasingly stronger base for which to independently verify the artifact's behavior and output. Software verification is the notion of the preservation of correctness. It is then the final rung and the ultimate step in the wider goal of digital preservation and scientific digital reproducibility. That is, software is preserved so that it can be improved over time. And once software is correct, by whatever measure, it then doesn't disappear and it doesn't become obsolete. This is why the evaluation of archival systems is so strongly related to the definition of software sustainability. When a system provides the means to promote a particular level of support, it is then a better system than one that does not. A system that produces repeatable artifacts is better than a hosting provider that only maintains retrievable objects, because it gets us closer to verifiable and accountable software. Overall, good software archival systems go beyond preservation or scientific purposes and are vehicles for building more robust computing platforms.

IV. The Evaluation of Software Archives

The sustainability ladder has been applied with respect to several tools. Table I gives a representable list of projects and how they compare against the kinds of software preservation they enable. The following tables relate to each pair of rungs in our ladder. For each table, properties are marked for a set of existing projects related to reproducibility. Shareability is well-understood and common among all of the projects and is omitted. Table II shows properties related to retrieval, and discovery which we term Prominence. Table III shows deployability and repeatability, or, when taken together, we call Consistency. Table IV shows configurability and derivability, which imbue the concept of Interactivity. Projects tend to focus on one of these three categories as they emphasize features and attributes common to them.

Based on your needs, you can use this breakdown to find the appropriate tool for your purpose. For many people, the simple hosting of data is all they need. For instance, the publishing of graphs or datasets. Here, you may only need tools which provide strong retrieval and discovery (Table II). However, for those running software, you should focus more on projects (or combinations of tools) that fulfill the deployable and repeatable aspects of the sustainability ladder. The fulfillment of the entire research life-cycle, which entails the proposal, hypotheses, experimentation, and conclusion

Systems Compared Through Prominence	Artifact Downloads	Versioning	Peer-to-Peer	Access Control	Unique Identifiers	Search Capability	Keyword/Tag	Linked Relationships	Embeddable Artifacts	Metadata Standards	Provenance
RunMyCode	✓	×	×	×	×	✓	×	×	×	×	×
GitHub, BitBucket, GitLab	✓	✓	×	✓	×	✓	×	×	×	×	×
Zenodo	✓	✓	×	×	✓	✓	✓	×	×	✓	×
Jupyter, Umbrella, ReproZip†	✓	×	×	×	×	×	×	×	×	×	×
Olive	✓	✓	×	✓	✓	✓	×	✓	✓	✓	×
NanoHub	✓	✓	×	✓	✓	✓	✓	✓	✓	×	×
Code Ocean	✓	✓	×	×	✓	✓	✓	×	✓	×	×
Occam	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
		Retrievable					Discoverable				

TABLE II: Comparing different archival systems based on Prominence: how well artifacts are retrieved or replicated between archives or from the archive to an interested party. †: Jupyter, Umbrella, and ReproZip are not hosting providers and do not have discoverability features, but they can be combined with GitHub or similar services that do.

Systems Compared Through Consistency	Non-Local Execution	Commodity/Cloud Execution	Local/Native Execution	Isolates Software Environment	Hardware Virtualization	Runs Graphical / GUI Software	Self-Archival
RunMyCode	×	×	×	×	×	×	×
GitHub, BitBucket, GitLab	×	×	×	×	×	×	×
Zenodo	×	×	×	×	×	×	×
Jupyter	✓	×	✓	×	×	×	×
Umbrella	✓	✓	✓	✓	✓	×	×
ReproZip	✓	✓	✓	✓	✓	×	×
Docker Cloud	✓	✓	✓	✓	×	×	×
Olive	✓	✓	×	✓	✓	✓	×
NanoHub	✓	✓	✓	✓	✓	✓	×
Code Ocean	✓	✓	✓	✓	×	×	×
Occam	✓	×	✓	✓	×	✓	✓
		Deployable			Repeatable		

TABLE III: Comparing different archival systems based on Consistency: how well software artifacts deploy and run in their intended manner. Self-Archival describes the ability to archive data as it is produced.

Systems Compared Through Interactivity	Standardized Configuration	Extensible Input	Composition / Workflows	Preserves Building/Compiling	Multiple Methods of Interaction	Artifact Cloning / Forking
RunMyCode, GitHub, BitBucket, GitLab, Zenodo	×	×	×	×	×	×
Jupyter, Umbrella, ReproZip	×	×	×	×	×	×
Olive	×	×	×	×	×	×
NanoHub	✓	✓	×	×	×	✓
Code Ocean	✓	✓	×	✓	✓	✓
Occam	✓	✓	✓	✓	✓	✓
		Configurable			Derivable	

TABLE IV: Comparing different archival systems based on Interactivity: how well software artifacts can be changed or used in new ways to create new things or, more specifically, verify scientific results. Many of the projects are omitted because they have no focus on interactivity.

looping continuously, requires much stronger tools which can preserve software artifacts that can change over time. For this, we all look toward the final set of attributes.

Furthermore, we can use these guidelines to envision what people find important and what can still be done in the space of digital archives. Looking at the tables, we can see the idea of discovery and prominence is well understood. There are a variety of competitive solutions to essentially host software artifacts. We can also see that there is a high interest in the development of tools for the consistency of software execution. This is not a surprise as this has been driven by both scientific and industry communities to better handle increasingly prevalent cloud computing environments.

Table IV is important for understanding what comes next. When we accept software sustainability as being more than just preservation of an executable, we see that we still have some room for growth. The sparsity of the table on support for artifact interactivity represents the state-of-the-art. Combined with the ladder, which measures the strength of an artifact, this shows us that these projects are certainly doing something new and beneficial toward digital preservation and that there are still new types of archives worth building. As a community, we need to come together and determine what else comes between now and the overall goal of preserving verifiable software. It is this frontier of software archival that we now need to focus our attention.

REFERENCES

[1] S. Hettrick, M. Antonioletti, L. Carr, N. Chue Hong, S. Crouch, D. De Roure, I. Emsley, C. Goble, A. Hay, D. Inupakutika, M. Jackson, A. Nenadic, T. Parkinson, M. I. Parsons, A. Pawlik, G. Peru, A. Proeme, J. Robinson, and S. Sufi, "Uk research software survey 2014," Dec. 2014. [Online]. Available: https://doi.org/10.5281/zenodo.14809

[2] C. Collberg and T. A. Proebsting, "Repeatability in computer systems research," *Commun. ACM*, vol. 59, no. 3, pp. 62–69, Feb. 2016. [Online]. Available: http://doi.acm.org/10.1145/2812803

[3] M. Baker, "1,500 scientists lift the lid on reproducibility," *Nature*, vol. 533, pp. 452–454, May 2016.

[4] H. I. a.-H. Ibn-al Haitam and S. A. I., *The optics. on direct vision*. The Warburg Institute, Univ. of London, 1989.

[5] "Popper," http://falsifiable.us/, [Online; accessed 18-Sep-2017].

[6] S. Bechhofer, J. Ainsworth, J. Bhagat, I. Buchan, P. Couch, D. Cruickshank, M. Delderfield, I. Dunlop, M. Gamble, C. Goble, D. Michaelides, P. Missier, S. Owen, D. Newman, D. De Roure, and S. Sufi, "Why linked data is not enough for scientists," vol. 29, 02 2013.

[7] "DataMill - Open Benchmarking Platform," https://datamill.uwaterloo.ca/, [Online; accessed 29-Aug-2016].

[8] "GitHub: Build software better, together," github.com, [Online; accessed 12-Dec-2014].

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

[10] "Bitbucket: Unlimited private code repositories," bitbucket.org, [Online; accessed 17-Nov-2014].

[11] "Run My Code," https://runmycode.org/, [Online; accessed 29-Aug-2016].

[12] D. De Roure, C. Goble, and R. Stevens, "The design and realisation of the virtual research environment for social sharing of workflows," *Future Generation Computer Systems*, vol. 25, no. 5, pp. 561–567, 2009.

[13] "Zenodo," https://zenodo.org/, [Online; accessed 29-Aug-2016].

[14] "OSF — Home," https://osf.io/, [Online; accessed 29-Aug-2016].

[15] "DataHub," https://datahub.csail.mit.edu/www/, [Online; accessed 18-Sep-2017].

[16] "DataHub," http://datahub.io, [Online; accessed 18-Sep-2017].

[17] "Torch — Scientific computing for LuaJIT," https://torch.ch, [Online; accessed 29-Sep-2017].

[18] M. Reich, T. Liefeld, J. Gould, J. Lerner, P. Tamayo, and J. P. Mesirov, "Genepattern 2.0," *Nat Genet*, vol. 38, no. 5, pp. 500–501, May 2006. [Online]. Available: http://dx.doi.org/10.1038/ng0506-500

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

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

[21] A. P. Davison, M. Mattioni, D. Samarkanov, and B. Teleńczuk, "Sumatra: a toolkit for reproducible research," *Implementing reproducible research*, vol. 57, 2014.

[22] "Madagascar," http://www.ahay.org, [Online; accessed 09-Jun-2017].

[23] K. Wolstencroft, R. Haines, D. Fellows, A. Williams, D. Withers, S. Owen, S. Soiland-Reyes, I. Dunlop, A. Nenadic, P. Fisher *et al.*, "The taverna workflow suite: designing and executing workflows of web services on the desktop, web or in the cloud," *Nucleic acids research*, p. gkt328, 2013.

[24] "Galaxy Project – Online bioinformatics analysis for everyone," https://galaxyproject.org/, [Online; accessed 29-Aug-2016].

[25] L. Bavoil, S. P. Callahan, P. J. Crossno, J. Freire, C. E. Scheidegger, C. T. Silva, and H. T. Vo, "Vistrails: Enabling interactive multiple-view visualizations," in *VIS 05. IEEE Visualization, 2005.* IEEE, 2005, pp. 135–142.

[26] 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 *Scientific and Statistical Database Management, 2004. Proceedings. 16th International Conference on*. IEEE, 2004, pp. 423–424.

[27] "Project Jupyter Home," http://jupyter.org/index.html, [Online; accessed 29-Aug-2016].

[28] M. Sato, S. Matsuoka, P. M. Sloot, G. D. van Albada, J. Dongarra, P. Nowakowski, E. Ciepiela, D. Harlak, J. Kocot, M. Kasztelnik, T. Bartyski, J. Meizner, G. Dyk, and M. Malawski, "Proceedings of the international conference on computational science, iccs 2011 the collage authoring environment," *Procedia Computer Science*, vol. 4, pp. 608 – 617, 2011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877050911001220

[29] "Beaker Notebook," http://beakernotebook.com/, [Online; accessed 29-Sep-2017].

[30] "Chameleon Cloud," https://www.chameleoncloud.org/, [Online; accessed 29-Sep-2017].

[31] "nanoHUB.org — Online Simulation and More for Nanotechnology," nanohub.org, [Online; accessed 20-Sep-2014].

[32] "Code Ocean - Discover and Run Scientific Code," https://codeocean.com, [Online; accessed 29-Sep-2017].

[33] H. Meng and D. Thain, "Umbrella: A portable environment creator for reproducible computing on clusters, clouds, and grids," in *Proceedings of the 8th International Workshop on Virtualization Technologies in Distributed Computing*, ser. VTDC '15. New York, NY, USA: ACM, 2015, pp. 23–30. [Online]. Available: http://doi.acm.org/10.1145/2755979.2755982

[34] "SIMULOCEAN," http://xsede.simulocean.org/, [Online; accessed 29-Aug-2016].

[35] "OCCAM: Open Curation for Computer Architecture Modeling," http://occam.cs.pitt.edu, 2016, [Online; accessed 12-Aug-2016].

[36] C. T. Brown, "The ladder of academic software notsuck," 03 2013. [Online]. Available: http://ivory.idyll.org/blog/ladder-of-academic-software-notsuck.html

[37] N. C. Hong, "The five stars of research software," 04 2013. [Online]. Available: https://www.software.ac.uk/blog/2016-10-07-five-stars-research-software

[38] N. Chue Hong, "Minimal information for reusable scientific software," WorkingPaper ¡importModel: WorkingPaperImportModel¿, 7 2014.

[39] M. D. Wilkinson, M. Dumontier, I. J. Aalbersberg, and et al., "The fair guiding principles for scientific data management and stewardship," vol. 3, pp. 160 018 EP –, Mar 2016, comment. [Online]. Available: http://dx.doi.org/10.1038/sdata.2016.18

[40] "VMware Player," http://www.vmware.com/products/player/, [Online; accessed 11-Sep-2012].

[41] Oracle, "VirtualBox," http://www.virtualbox.org, [Online; accessed 10-Jan-2015].

[42] D. Merkel, "Docker: Lightweight linux containers for consistent development and deployment," *Linux J.*, vol. 2014, no. 239, Mar. 2014. [Online]. Available: http://dl.acm.org/citation.cfm?id=2600239.2600241