Strategies for Biomedical Software Management, Sharing, and Citation

Daniel S. Katz[1], Kyle E. Niemeyer[2], Arfon M. Smith[3]

[1]University of Illinois Urbana-Champaign, d.katz@ieee.org
[2]Oregon State University, kyle.niemeyer@oregonstate.edu
[3]Space Telescope Science Institute, arfon@stsci.edu

Abstract: This document is an open response to the NIH Request for Information (RFI): Strategies for NIH Data Management, Sharing, and Citation, Notice Number: NOT-OD-17-015, written by the leaders of the FORCE11 Software Citation Working Group from its inception in mid-2015 through today. This group produced a set of Software Citation Principles and related discussion, which are the basis for this document. Here, we describe research software, summarize the software citation principles, discuss open issues related to software citation, and make recommendations to the NIH.

Introduction

As we wrote in [1], we believe (and think NIH also believes) that "software is a critical part of modern research and yet there is little support across the scholarly ecosystem for its acknowledgement and citation." Of course, this is also true about data. And while software can be considered data, it is not merely data. As we discuss in [2]:

> In the context of research (e.g., in science), the term "data" usually refers to electronic records of observations made in the course of a research study ("raw data") or to information derived from such observations by some form of processing ("processed data"), as well as the output of simulation or modeling software ("simulated data"). In the following, we use the term "data" in this specific sense.
>
> The confusion about the distinction between software and data comes in part from the much wider sense that the term "data" has in computing and information science, where it refers to anything that can be processed by a computer. In that sense, software is just a special kind of data.

Even after discriminating between software and data, software is still a very general term, and we believe that different types of software should be treated differently for the purposes of management, sharing, and citation. For example, general-purpose software such as that used for email and text processing does not need to be considered here; we should focus on research software, meaning software used where the choice or version of software impacts the research results and citation (and associated credit) is of value to the software authors.

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.2640v1 | CC BY 4.0 Open Access | rec: 17 Dec 2016, publ: 17 Dec 2016

Research Software

We can further break research software into two categories: (1) software intended just for use by the developer (e.g., student, postdoc, faculty, staff possibly not on an academic career path) to solve one or more specific research problems; and (2) software intended for wide use beyond the developers, to be used in a wide variety of research problems. In both cases, the software should be managed, shared, and cited, but for somewhat different purposes, and with potentially different requirements.

For software created by a developer for their own use in research, we can ask what could be done for that developer, and what could be done for others. A minimum goal for the developer is that the software is correct, and that the developer can return to it at some future time, and still understand it, and still be able to use it—at least to repeat what that developer previously did. This implies that the developer should be at least minimally trained as a software developer (perhaps through a Software Carpentry[1] course), and that the software be archived in some way. If we assume that the software has been used for some research purpose, then the developer should also be able to document this usage in the record of the research, which in the case of a paper should be in the form of a citation. This implies that the software, in addition to being archived, should also be published to make it citable. This also enables others to possibly use the software for another purpose, but this may depend on the quality of the software and its documentation.

Conventionally, the primary research output of the developer is the research paper, not the software; the developer gets credit for the research paper if the software is not reused, though the developer may also get credit for the software if it is reused. The developer also likely wants to spend the least effort possible to create the software; the software is “complete” once it has been used for the research purpose.

If the software was created by the developer (or a community of developers) with the intention of it being shared, quality measures also probably include how friendly the software is to users (e.g., in documentation) and how responsive it is to the users (e.g., in tracking and responding to bugs and feature requests). Much like the software created for the developer’s use, this software-as-infrastructure should be archived and published by its developers, with a goal that the contributors are recognized for their work, since the software itself is the research output of their work.

This software is likely never “complete”, though versions are released. There is generally always more work that could be done for research software, and what actually gets done is limited by the available resources (i.e., time and funding).

In both cases, archiving, publication, and then citation enable—but are not sufficient for—reproducibility, since reproducibility depends a number of factors beyond identification of

[1] https://software-carpentry.org

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.2640v1 | CC BY 4.0 Open Access | rec: 17 Dec 2016, publ: 17 Dec 2016

the software. These include compiler flags, versions of software dependencies (e.g., libraries, operating system), usage options, and the environment in which the software is run.

In addition to citations providing credit (if properly indexed), they can also help understand use of software. This serves a number of needs, including those of users who can choose software to use based on if it already is widely used, and funders who can choose what software to support, e.g., in the NSF SI2 program[2].

Software Citation Principles

Our working group included 55–60 people from a mix of backgrounds, including researchers, librarians, publishers, sociologists, and indexers. Starting with the FORCE11 Data Citation Principles [3], we used a set of previous work and meetings, use cases, and multiple rounds of feedback and discussion (both inside and outside the working group) to decide on a set of six Software Citation Principles, as published in [1] and described briefly here:

- Importance: Software is a legitimate and citable product of research, accorded the same importance as other research products, such as publications and data.
- Credit and attribution: Citations should facilitate credit attribution for all contributors.
- Unique identification: Citations should be machine actionable, globally unique, and interoperable.
- Persistence: Unique identifiers and metadata describing the software and its disposition should persist, even beyond the lifespan of the software.
- Accessibility: Software citations should facilitate access to the software itself and to its associated materials.
- Specificity: Software citations should facilitate identification of, and access to, the specific version of software that was used.

Applying the Software Citation Principles

While the principles themselves are quite brief and clear, how they are applied is much less clear. Our work [1] includes a large amount of discussion and recommendations, some of which is very briefly summarized in this section. Note that these points **are not** principles or requirements; they are discussion, and in some cases, recommendations. For full context, please see [1].

- Software should be cited on the same basis as any other research product, such as a paper or book.
- Some software which is captured as part of data provenance may not be cited: citation is partly a record of software important to a research outcome, where provenance is a

[2] https://www.nsf.gov/publications/pub_summ.jsp?ods_key=nsf16532

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.2640v1 | CC BY 4.0 Open Access | rec: 17 Dec 2016, publ: 17 Dec 2016

record of all steps (including software) used to generated particular data within the research process.

- Software papers are a workaround for the current lack of software citation practices. Software itself should be cited as a first priority, and if a software paper exists and contains results (performance, validation, etc.) that are important to the work, then the software paper should also be cited.
- Software peer-review is an important concern going forward, but it is mostly outside the scope of a discussion about software citation.
- Strict journal citation limits and page limits that include references are problematic for software citation.
- While there are a number of different types of identifiers that can be used, we recommend the use of DOIs for a specific version of a software that is going to be cited.
- We believe the principles apply to all of these forms of software (not just source code, but also executables, containers, virtual machine images, and software available as a service) though the implementation of them will certainly differ based on software type.
- Cited software does not need to be freely available, but the citation metadata should provide enough information that the software can be accessed, even if closed, commercial, etc.
- Software identifiers should resolve to a persistent landing page that contains metadata and a link to the software itself, rather than directly to the source code files, repository, or executable.

Open issues

The following two issues cannot be resolved within the software citation principles, which might imply a gap in the principles that will need to be solved at a later time.

1. Push vs. pull for citations. To make software citation work as described by the principles, software developers need to register versions of the software, and software users need to cite registered software. This implies that there are two citation roles that need to be performed: creation (push) and usage (pull), with creation required before usage. However, this breaks down in number of cases, such as in commercial software where there may be no person with an incentive to create a citation, or when a version of open-source software is used that doesn't match a version that was published. Examples of this latter case may be the desired citation of a version 4.1.1 of software, where the developers only publish major (4.0) or even minor (4.1) releases, or a version of software without even a version number (e.g., the latest, unreleased software downloaded from a GitHub repository).

   As we stated in the discussion section of [1], when the software that someone wants to cite software is not published, the best option today seems to be to cite the software by name or URL, and version number or release date. Of course, the fact that these may not be unique will lead to difficulties in understanding the impact of such software (see point 2 below as well.) Another option is that a third-party service is used, such as

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.2640v1 | CC BY 4.0 Open Access | rec: 17 Dec 2016, publ: 17 Dec 2016

SciCrunch[3] in biosciences or ASCL[4] in astronomy, which would allow people to register software that they hadn't authored, though most often the registration is to a package, not to a specific version (again, see point 2 below.)

2. Grouping citations across versions or by project. While citations at the level of software releases enable each release to have a distinct set of contributors who are credited by the citations for that release, this alone makes it impractical to track citations for a project (i.e., a software package or a repository), which is important to the project team and to the funders of that project, as well as to those who study the interplay between projects. The ability to do this does not naturally appear from the software citation principles being applied to the current general citation system.

   There are two types of possible solutions. The first is to ask users to cite both the specific software version and the overall software package, which would require yet another cultural change. Also related to this discussion is the use of Research Resource Identifiers (RRIDs), a set of identifiers that have been introduced for "finding or generating stable unique identifiers" for key biological resources: Antibodies, Model Organisms, and Tools (software, databases, services)[5]. As of December 2, 2016, the SciCrunch.org Research Identification Portal lists almost 2500 software resources. Each RRID entry contains an RRID itself, a URL for the resource, a short text description of the resource, and a set of additional metadata, including metrics. These "software" resources are a mix of services, tar files, software project home pages, software project repositories, and non-software links, including research group home pages and text documents. ASCL curates a similar catalog[6] for astronomy software.

   The second type of solution is to use metadata to describe these relationships, which might mean automatically creating a "container" or "package" DOI when the first version of a software package is cited, then using "relationship type" field in the DataCite metadata for that and future versions. This does not require any culture change, but does require infrastructure changes, particularly by the services that create DOIs for software, as well as some consensus and agreement about how to implement those changes. This solution is discussed in some detail in one of the open issues in the software citation working space on GitHub[7].

   In that issue, Martin Fenner points out that the same idea is also in the JISC recommendations for software citation [4]. They talk about a model of software entities and the "product" level vs. the "version" level, and they describe the usefulness of identifiers for the "product" level: "Using an identifier at this level may be appropriate to reference the general concept of a particular software artefact regardless of the specific

[3] https://scicrunch.org
[4] http://ascl.net/
[5] https://scicrunch.org/resources
[6] http://ascl.net/code/all
[7] https://github.com/force11/force11-scwg/issues/157

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.2640v1 | CC BY 4.0 Open Access | rec: 17 Dec 2016, publ: 17 Dec 2016

version, or the continued use of this software over a long period. It [is] of use if different versions are going to be referenced as it can stand as a unifying record."

Conclusions

The FORCE11 Software Citation Working Group has almost completed its set of activities. We are in the process of creating an infographic, and have started a brief paper that works through some of the software citation use cases in detail, describing what the roles of each group of stakeholders are. After this, we will end our work, and report to FORCE11 that we have finished. We expect that FORCE11 will then start a Software Citation Implementation Group, possibly with new leadership, and that this group will work with all the software citation stakeholders to gather endorsements for the principles, and to implement them. The principles document also includes a path for updates, which also might be suggested and made by the Implementation Group.

We specifically recommend the following to NIH (these are point-by-point responses to a selected set of the "invited areas for comment"):

The NIH seeks comment on any or all of the following topics to help formulate strategic approaches to prioritizing its data management and sharing activities:

- *The highest-priority types of data to be shared and value in sharing such data;*

  We recommend that NIH should recognize the critical importance of research software, both as it supports other research products (e.g., manuscripts, data) and as a primary research product, by endorsing and implementing the Software Citation Principles [1].

- *The length of time these data should be made available for secondary research purposes, the appropriate means for maintaining and sustaining such data, and the long-term resource implications;*

  We recommend that NIH adopt (current) best practises for handling research software:
  1. Use a community platform for hosting/versioning software
  2. Archive said software in, e.g., Zenodo, figshare
  3. Use metadata that are compatible with and translateable to CodeMeta (https://codemeta.github.io/) metadata files

- *Barriers (and burdens or costs) to data stewardship and sharing, and mechanisms to overcome these barriers; and*

  We recommend that NIH consider allocating funds for long-term and sustained effort on these topics. The Software Citation Principles [1] makes specific recommendations which places requirements on sharing and stewardship. Quoting directly from [1]:

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.2640v1 | CC BY 4.0 Open Access | rec: 17 Dec 2016, publ: 17 Dec 2016

4. Persistence: Unique identifiers and metadata describing the software and its disposition should persist—even beyond the lifespan of the software they describe.

5. Accessibility: Software citations should facilitate access to the software itself and to its associated metadata, documentation, data, and other materials necessary for both humans and machines to make informed use of the referenced software.

We note that the NIH's own workshop focused on Software Discovery in the biosciences [5] discussed many related topics and made recommendations in the form of a proposed roadmap.

Disclosures

Katz's opinions are influenced by the four years he spent as an NSF Program Director leading the Office of Cyberinfrastructure's & Division of Advanced Cyberinfrastructure's software-as-infrastructure programs. Niemeyer's opinions are informed by his research activities developing and using, and then citing, research software. Smith's perspectives are influenced and informed by his three years working for GitHub Inc. as their liaison to academic and research communities. Any opinion, finding, and conclusion, or recommendation expressed in this document are that of the authors and does not necessarily reflect the views of their former or current employers, FORCE11, or other members of the FORCE11 Software Citation Working Group.

References

[1] Smith AM, Katz DS, Niemeyer KE, FORCE11 Software Citation Working Group. (2016) Software citation principles. *PeerJ Computer Science* 2:e86 DOI: 10.7717/peerj-cs.86

[2] Katz DS, Niemeyer KE, Smith AM, Anderson WL, Boettiger C, Hinsen K, Hooft R, Hucka M, Lee A, Löffler F, Pollard T, Rios F. (2016) Software vs. data in the context of citation. *PeerJ Preprints* 4:e2630v1 https://doi.org/10.7287/peerj.preprints.2630v1

[3] Data Citation Synthesis Group. (2014) Joint Declaration of Data Citation Principles. Martone M (ed.) San Diego CA. FORCE11. URL: https://www.force11.org/group/joint-declaration-data-citation-principles-final [Accessed 2016-11-23]

[4] Gent I, Jones C, Matthews B. (2015) Guidelines for Persistently Identifying Software Using DataCite: A JISC Research Data Spring Project. Version 1.0, URL: http://rrr.cs.st-andrews.ac.uk/wp-content/uploads/2015/10/guidelines-software-identification.pdf [Accessed 2016-12-02]

[5] NIH Software Discovery Workshop (2014), URL: http://www.softwarediscoveryindex.org [Accessed 2016-12-05]

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.2640v1 | CC BY 4.0 Open Access | rec: 17 Dec 2016, publ: 17 Dec 2016