An Update on Implementing Software Citation: Challenges in 2018

(aka Analysis & Planning or A&P document)
(Issues in GitHub)

Authors: Daniel S. Katz, Martin Fenner, Neil P. Chue Hong, Melissa Harrison, Lorraine Hwang, Tom Gillespie, Carly B. Robinson, Morane Gruenpeter, Matthew B. Jones, Robert Haines, Ilian Todorov, Daina Bouquin, Mingfang Wu, Stephan Druskat, Alejandra Gonzalez-Beltran, Ted Habermann, Qian Zhang, Alastair A. Kelly, Katrin Leinweber, Catherine Jones, Jessica Hausman

Date: 27 August 2018 - trying to complete by end of Jan 2019.

Table of Contents:

1. Introduction

The main output of the FORCE11 Software Citation working group was a paper on software citation principles published in September 2016. This paper laid out a set of six high-level principles for software citation (importance, credit and attribution, unique identification, persistence, accessibility, and specificity) and discussed how they could be used to implement software citation in the scholarly community. In a series of talks and other activities, we have promoted software citation using these increasingly accepted principles. At the time the initial paper was published, we also provided the following (old) guidance and examples on how to make software citable, though we now realize there are unresolved problems with that guidance. The purpose of this document is to provide an explanation of current issues impacting scholarly attribution of research software, organize updated implementation guidance, and identify where best practices and solutions are still needed.

1.1 Document Purpose and Intended Audience

This document is not intended to support typical researchers, editors, etc., for whom we will develop and point to other slimmer, more specific primers. It is not "user documentation", for

those who want to use software citation (e.g., cite the software they use, make their own software citable by others) in their writing, but "developer documentation", for those who are developing software citation practices and tools.

This document is aimed at various types of expert stakeholders (as follows) who have a level of expertise or advocacy that is above that of the average person who wishes to understand how to enact a specific software citation use case.

- **Communities that represent developers and other researchers, including software developers seeking to champion software citation** will use this document to provide more detailed and specific guidance to members of their community around using software citation
- **Publishers** will use this document to create policies and tools for software citation, including developing guidance and examples for authors, editors, and reviewers
- **Editors of journals or peer-reviewers** will use this document to develop workflows to ensure software used in publications is being cited properly
- **Librarians and archivists** will use this document to develop training and other resources for article authors and software developers to improve software citation practices; and advocate for or make changes to internal archival systems to enable software citation.
- **Indexing and abstracting services** will use this document to develop tools and policies to expose and index software citations and associated metadata
- **Funders** will use this document to develop policies that encourage practices that enable proper citation of software developed using grant funds, as well as proper citation of software in grant proposals

These expert stakeholders might be those at communities, institutions or journals who support researchers, or software citation champions.

It presents a view of the current state of software citation practice and organizes updated implementation guidance to enables these stakeholders to get up to speed with the current practice of software citation, the language and terminology that is being used, and the challenges that are still to be addressed.

1.2 Previous guidance

When the software citation principles were published, guidance on their implementation was generally given as follows. We quickly realized that this guidance was incomplete and insufficient, however,we will use the old guidance as a starting point for understanding its limits and what new work is needed.

Old guidance to making software citable:

- Publish the software:

    - If the software is on GitHub, follow steps in https://guides.github.com/activities/citable-code/
    - If the software is not on GitHub, submit it to Zenodo directly, or to figshare, or a suitable domain repository, with appropriate metadata (including authors, title, citations, and dependencies)
- Get a DOI
- Create a CITATION file and update your README to tell people how to cite
- If needed, also write a paper about the software that discusses performance, algorithms, and other features.

Old guidance to citing software:

- Check for a CITATION file or README that contains citation information; if such a file says how to cite the software itself, do that
- If there is no CITATION file or specifications in the README, do your best to follow the principles
    - Regarding authors of the software, if the software developers declare who the authors are, list them; otherwise, just name the project (Project X for open source software, Company Y for commercial software) as the authors
    - Try to include a method for identification that is machine actionable, globally unique, interoperable – perhaps a URL to a release, a company product number
    - If there's a landing page that includes metadata, point to that, not directly to the software (i.e., if software is on GitHub and in Zenodo, point to the Zenodo version and specifically to the landing page)
    - Include specific version/release information
    - If there is a software paper that includes information important to your use of the software, possible cite that *in addition to* the citation of the software itself

1.3 Moving forward

Since 2016, there have been many changes in the world relevant to software citation, including the introduction of software metadata specifications such as CodeMeta and the Citation File Format, and the universal software archive Software Heritage. Groups that want to implement software citation have also asked for specific guidance on what they should do with different types of software.

Today, a series of technical and community challenges exist that make moving beyond general acceptance to implementation and common use difficult. This document attempts to list them and some thoughts on how they might be addressed.

1.4 Future changes to this document

Once this document is completed, it will still not answer all the questions about software citation, particularly as the scholarly world and the scholarly communications community continue to

change. In addition, as discussed in Section 1.1, the guidance here is not aimed at the average person who needs to enact a software citation use case. Therefore:

- Communities should build further documents based on this, more specific for their communities and use cases.
- Later versions of this document itself will need be written, perhaps with small changes initially, but eventually with larger changes.

2. Software types

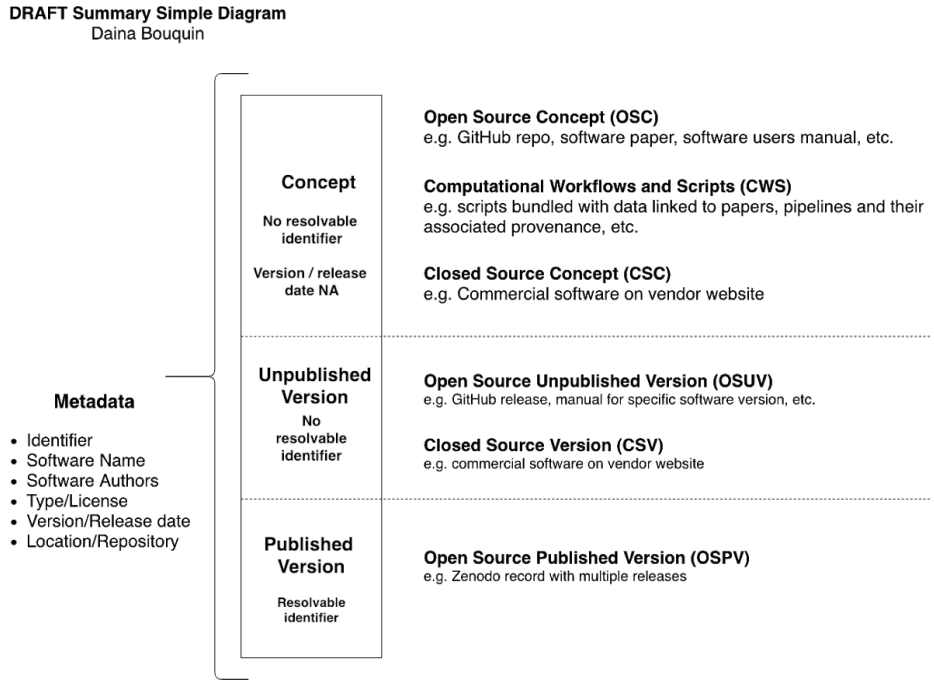
In order to cite software, it must be distinctly identified, and appropriate metadata about it must be gathered. There are many approaches to categorizing software, each with different metadata facets. For instance, software may be categorized by function, by layer (e.g., infrastructure, library, tool), or by the software delivery type (e.g., source code, binary executable, package, container, service). In this document, we will primarily consider two aspects of software categorization that are important for software citation and can be applied to the wide spectrum of software that we might choose to cite: the copyright status and the publishing status. For each type, the method for identification and the location (or absence) of metadata may differ.

Software can be open source or closed source (aka proprietary software). It is also both a concept (all versions of the software; see Section 6.6 for discussion about this choice of wording) and a set of individual versions. Likewise, software can be "published" by permanently archiving it and creating a resolvable identifier. Software is "archived" once it has been deposited in a repository (along with appropriate metadata), where the curators of that repository steward the software with long-term preservation as their goal (e.g., Zenodo, figshare, institutional archival repositories). Without archiving the software and obtaining a resolvable identifier it is "unpublished." Unpublished software is often made available by a hosting organization that does not commit to long term preservation (e.g., GitHub). Thus, in the terminology used here, we consider code publicly available online as open source but unpublished, unless a specific version of the software (and associated metadata) has been deposited and made available via an organization that archives and provides identifiers for software deposits[1].

The software citation principles and discussion essentially say that versions of open source software should be published, and that those versions (and their corresponding DOIs) should then be cited. Today, with Software Heritage, we also have the possibility of citing unpublished versions (though Software Heritage doesn't store any of the metadata that we would like, unless it has been stored in the repository by the software authors) of open source software, by

[1] This distinction is similar to that of posting a PDF document on a web site (making it available) vs submitting it to a publisher or repository that will archive it and create a landing page and an identifier. In both cases, the document has been made available, but only in the latter case has it been published. The key difference is the intent of the document host: only short term availability or also long term archiving.

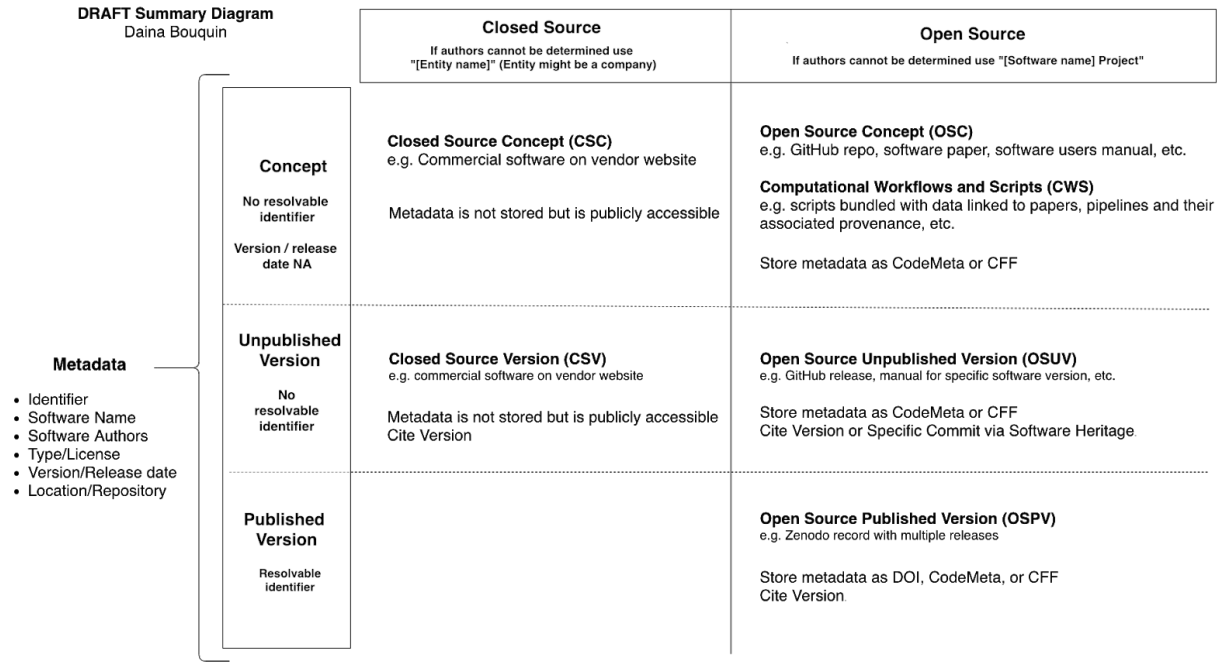
pointing to an archived copy of a specific commit or release in the code repository. We do not currently have an acceptable solution for closed source software. In practice, a piece of software does not always fit neatly into a single type; there may be hybrids such as notebooks or combinations of open and closed source.



Has versions

1. **CSV** (closed source version) - A version of closed source, possibly commercial, software. An example is version 14.3 of SAS/STAT.
2. **CSC** (closed source concept) - The software concept for a closed source, possibly commercial software package, such as SAS/STAT.
3. **OSPV** (open source published version) - A published version of an open source software package. An example is version 1.2 of the Application Skeleton package (https://github.com/applicationskeleton/Skeleton), found on GitHub at https://github.com/applicationskeleton/Skeleton/releases/tag/v1.2, published by Zenodo as https://doi.org/10.5281/zenodo.13750.
4. **OSUV** (open source unpublished version) - An unpublished version of an open source software package, for example, https://github.com/applicationskeleton/Skeleton as of commit 81c66c0db5c381dacc0841a4c16e0b3876b15b89.
5. **OSC** (open source concept) - The software concept for an open source software comprhensive package, for example, the Application Skeleton package (https://github.com/applicationskeleton/Skeleton).
6. **CWS** (computational workflows and scripts) - a computational workflow or script that is published within mixed-type Research Objects that represent computational pipelines with their associated provenance. For example, the Chlorophyll R Script contained in

the Gentry et al. research data package (https://doi.org/10.5063/F1CF9N69) that is cited in an associated paper (https://doi.org/10.1038/s41559-017-0257-9).



3. Technical challenges

The primary technical challenges to software citation implementation are how to identify software, how to cite it once it has been identified, and how citations are counted.

3.1 How to identify software

How to identify software to be cited is an open question that has been discussed in the FORCE11 software citation working groups, and will be discussed in the RDA Software Source Code Identifaction Working Group.

1. CSV - Perhaps this is a URL to a release, or a company product number, or simply a product name and version number. For example, SAS/STAT ver. 14.3
2. CSC - Perhaps this is a URL to the software, or a company product number. This could simply be a product name, such as SAS/STAT.
3. OSPV - The software is identified by the publisher, likely with a DOI. For example, version 1.2 of the Application Skeleton package published by Zenodo is https://doi.org/10.5281/zenodo.13750.
4. OSUV - Let's assume the software has been archived by Software Heritage, since they are already archiving a large amount of open source software and aim to do this universally. Use their search and browse capability to find the version of the software you want to cite. If the repository you are looking for is not archived yet by Software

Heritage, you can use the Save Code Now feature and request that a repository be archived, using its URL[2]. Use the permalinks button on the right to retrieve the persistent identifier (or the URL). For example, if https://github.com/applicationskeleton/Skeleton as of commit 81c66c0db5c381dacc0841a4c16e0b3876b15b89 is https://archive.softwareheritage.org/swh:1:rev:81c66c0db5c381dacc0841a4c16e0b3876b15b89;origin=https://github.com/applicationskeleton/Skeleton/ in Software Heritage's archive.

5. OSC - The software can be identified by the software repository, and is sometimes represented by a software paper or the software users manual.

3.2 Metadata for software citation

The generally required metadata for software citation from the Software Citation Principles, are listed as follows:

- Identifier - Applicable to all; persistent identifiers preferred. Versionless DOIs can be used for CSC and OSC to identify software as a concept rather than a specific version/release, though there are not generally available tools to generate/issue such DOIs at this time.
- Software name - Applicable to all
- Authors - Applicable to all; if not able to be determined, one can fall back to "[Software name] Project" for open source projects, or "[Entity name]" or “[Company name]”commercial projects.
- Type - applicable to all software
- Version/Release date - not applicable to CSC and OSC
- Location/Repository - Applicable to all

3.3 Recording metadata

Metadata for software citation can be stored and made machine actionable in one of three places, or may not stored at all. These include the metadata associated with an identifier (e.g., a DOI[3]), in CodeMeta.json and in CITATION.cff files in a software repository alongside the code to be cited.

---

[2] The timing of archiving may vary based on:
1. The URL provided. If it is from a known domain (github for example) it will be accepted directly, while an unknown URL domain will have to pass a manual review, and the time needed for this depends on human resources at the time of submission.
2. The size of the repository. The number of commits can make a big difference in the ingestion time.
3. The maintenance state of the Software Heritage infrastructure
Archiving a normal size repository can take an hour, but this is not guaranteed. You can follow the request status on the "browse save requests" tab.

[3] Version 4.1 of the DataCite schema includes the metadata needed for software citation according to Smith et al.

CodeMeta is a machine-actionable general-purpose exchange format for software metadata expressed in JSON-LD format (i.e. JavaScript Object Notation for Linking Data). The use of JSON-LD in the CodeMeta specification means that the software metadata includes some structure (in JSON format) and can be extended with semantics via context files, which provide mappings of the elements to vocabularies. The provided context in CodeMeta relies mostly on the schema.org[4] vocabulary.

The Citation File Format (CFF) is a machine-actionable, human-readable and -writable metadata format based on the Software Citation Principles. CFF relies on the YAML format for representing the citation metadata, which is a superset of JSON (i.e., JavaScript Object Notation).

Both formats can be used to provide citation-relevant metadata for a software and may both be present. The primary difference between CFF and CodeMeta is that CFF is a "front-end" format, and CodeMeta is a "back-end" format. See section 3.3.1 for a comparison.

1. CSV - metadata is not stored, or is stored but is not publicly accessible
2. CSC - metadata is not stored, or is stored but is not publicly accessible
3. OSPV - metadata is stored with the DOI, and can also be stored via CodeMeta or the Citation File Format (CFF) in the repository.
4. OSUV - metadata can be stored via CodeMeta or CFF in the repository
5. OSC - metadata can be stored with the DOI (if published), and can also be stored via CodeMeta or CFF in the repository

Unresolved Issues:

- When there is no stored metadata, what should people wanting to cite the software do to meet the software citation principles?
- When there is more than one set of metadata stored, which is primary? It is important for discovery that the metadata about the object describes the type of object. Object typing for software is a developing field and may be addressed in the RDA Software Source Code Identification Working Group.
- Note that R has a set of guidance already: https://cran.r-project.org/doc/manuals/R-exts.html#CITATION-files And this is the recommended way by bioconductor (https://www.bioconductor.org/developers/package-guidelines/#citation) that now assigns DOIs to code

3.3.1 CodeMeta and the Citation File Format (CFF) comparison

The differences between the two are found in the following table.

[4] Schema.org is a collaborative project to create & maintain vocabularies for structured data on the internet. It enables common understanding of shared terms.

Feature	CodeMeta	CFF (Citation File Format)
Metadata type	*general-purpose metadata*	*citation metadata*
Targeted at	*machine exchange*	*human and machine actors*
Linked data	*yes*	*no*
Self-documenting	*no (perhaps via file name)*	*yes*
Enforces Principles	*N/A*	*yes*
Secondary references	*yes (general)*	*yes (scoped)*
Implementation	*JSON-LD*	*YAML*

While CFF is a suitable format for the initial provision of software citation metadata by the creators of a software, the metadata it provides should also be transferred to CodeMeta downstream in the software citation workflow. Transferring the CFF metadata lets software authors leverage CodeMeta's linked-data features and can enable more comprehensive documentation of the software metadata beyond bibliographic elements

A good example for how the two formats can be used together is the Netherlands eScience Center's Research Software Directory (e.g., https://research-software.nl/software/xenon), where CFF is used to provide the actual citation metadata input (converted to BibTeX etc.), and is also used to generate CodeMeta JSON-LD using cffconvert, which is embedded in a directory entry's landing page where it can be located by search engines.

3.4 Presenting and converting citation metadata

Once citation metadata is known and stored, it can be presented in various formats, such as a text citation, or in a format such as bibtex, RIS, etc. This metadata can also be stored in a reference manager, which can then do this conversion.

For example, DataCite's Content Resolver can return a representation of DOI in different formats. An example of this is returning the metadata for version 1.2 of the Application Skeleton package (with doi https://doi.org/10.5281/zenodo.13750) in bibtex:

```
> curl https://data.datacite.org/application/x-bibtex/10.5281/zenodo.13750

@misc{https://doi.org/10.5281/zenodo.13750,
  doi = {10.5281/zenodo.13750},
  url = {https://zenodo.org/record/13750},
  author = {Katz, Daniel S. and Merzky, Andre and Turilli, Matteo and Wilde,
Michael and Zhang, Zhao},
```

```
  keywords = {computer science, application skeleton, co-desiour proposalgn,
distributed computing, many-task computing, parallel computing},
  title = {Application Skeleton V1.2},
  publisher = {Zenodo},
  year = {2015}
}
```

Unresolved Issues

- Note that this type is "misc", not "software", since there is no software type in bibtex.
- Reference management tools need to be able to parse CFF and CodeMeta files
- Indexers need to be able to ingest CFF and CodeMeta files

3.5 How to cite software in text

How to cite identified software is also a challenge.

1. CSV - Authors need to do their best at gathering the metadata for a citation and formatting it appropriately.
2. CSC - Authors need to do their best at gathering the metadata for a citation and formatting it appropriately.
3. OSPV - the metadata we need for citation should be available as part of its publication, and we can cite the software similarly to how we would cite any other published digital object, with the exceptions of identifying it as software, by adding "[Software source code]" or similar in the citation, and identifying the version (much as you would identify volume, issue, or page numbers for a journal publication.) In the specific example (https://doi.org/10.5281/zenodo.13750) we have a landing page associated with the DOI that provides a suggested citation:
   Daniel S. Katz, Andre Merzky, Matteo Turilli, Michael Wilde, & Zhao Zhang. (2015, January 5). Application Skeleton v1.2. Zenodo. http://doi.org/10.5281/zenodo.13750
   This is not perfect, since it is not marked as software, and the version is not listed, except in the title, but it's fairly easy to start with this and fix it. We could also use https://citation.crosscite.org/ to get a citation in various styles, such as IEEE, which gives:
   D. S. Katz, A. Merzky, M. Turilli, M. Wilde, and Z. Zhang, “Application Skeleton V1.2.” Zenodo, 05-Jan-2015.
4. OSUV - For example, for https://github.com/applicationskeleton/Skeleton as of commit 81c66c0db5c381dacc0841a4c16e0b3876b15b89, I first look for the metadata in the archive of the repository. In this case, there is an AUTHORS file that tells me the authors of the software. If this wasn't here, I would name the authors as "Application Skeleton Project." To cite this version, I might use (in a more or less IEEE style):
   Daniel S. Katz and Andre Merzky and Matteo Turilli and Michael Wilde and Zhao Zhang, "Skeleton," [Software source code], commit 81c66c0db5c381dacc0841a4c16e0b3876b15b89, 2016.

https://archive.softwareheritage.org/swh:1:rev:81c66c0db5c381dacc0841a4c16e0b3876b15b89;origin=https://github.com/applicationskeleton/Skeleton/
(Or we could use the compact identifiers version: https://identifiers.org/swh:1:rev:81c66c0db5c381dacc0841a4c16e0b3876b15b89
If the software has been previously versioned, use the previous version's citation information as a starting point, adding to it any updates made to the AUTHORS, CITATION, README, ACKNOWLEDGEMENT etc. files as appropriate, along with the short commit hash.

5. OSC - While the idea of an identifier (such as a DOI) has always been possible to associate with software concept, there is no public way to directly obtain such an identifier. If there was such a way to do so, as of DataCite schema 4.1, the relation types `hasVersion` and `isVersionOf` would allow us to link `concept DOIs` and `version DOIs`. Some indirect options include
    a. If a version of the software is published to zenodo, a concept identifier is created, and multiple versions that are then published all link to the concept, using the relation types.
    b. Some in life sciences might use an RRID for a software concept, but this type of identifier is unknown outside life sciences, and the relation types cannot be used to link versions to the concept.
    c. Organizations who register DataCite DOIs can take advantage of the built-in support for codemeta. An example of this is https://doi.org/10.5438/qeg0-3gm3, which points to https://github.com/datacite/maremma. This DOI used the GitHub URL as metadata. When the DOI was created, the DataCite DOI registration service fetched the codemeta.json from the repository, converted the metadata into DataCite XML, and stored them.

All software citations should be included in the references section of papers rather than in acknowledgments or footnotes to ensure citations can be properly indexed, as stated in the software citation principles paper.

4. Community challenges

In order to implement software citation, in addition to the technical challenges in the previous section, we also have to bring this into the scholarly culture. We see at least four ways of looking at and interacting with the scholarly community. These can generally be viewed as points of leverage, at which we can make changes that will affect the overall community.

4.1 Disciplinary communities

One way of implementing software citation is by working with disciplinary communities (e.g., astronomy), which might involve professional societies (e.g., AAS), publishers (e.g., AAS again, and also see the next subsection), archives (e.g. Zenodo), indices (e.g., ADS), and other organizations.

To impact these communities, we likely need representatives for each, and those representatives should form a group to share their experiences, successes, lessons, and challenges with each other.

Actions for disciplinary communities include:

- Providing guidance for how software is cited in publications and events organized by the community, oriented towards editors, organizers, authors, and reviewers.
- Providing guidance for how software is recognized and counted within discipline
- Providing guidance for what approaches to provide software citation metadata are recommended on submission to publications and events, oriented towards authors and reviewers
- Guidance and resources to help disciplinary communities advocate for other stakeholders to adopt best practices

4.2 Publishers

Similar to disciplinary communities, each publisher has their own systems, practices, and culture, but also share a larger culture.

To impact these publishers, we likely need a representative for each, and those representatives should form a group to share their experiences, successes, lessons, and challenges with each other.

Actions for publishers include:

- Promote the need for authors to provide links to published copies of their code and determine how publishers can identify these
- Determine how software is internally identified in publisher systems now and how this could be improved
- Decide whether software associated with the publication (software developed by the authors of the publication) and software that is reused (software not developed by the authors of the publication, or not developed for work described in the publication) should be differentiated from each other
- Determine what policies should be adopted to support software citation
- Provide guidance and training for how software should be cited in publications to authors, reviewers, and editors; enforce these guidelines
- Ensure that publication workflows do not degrade metadata required for software citation
- Provide discipline-specific examples of software-related publisher policies, with annotations to explain why they are well-made and the important elements they include

4.3 Repositories

Repositories archive both data and software from computational workflows, often in integrated packages that document processes used in scholarly and other work. These integrated

packages often contain software, but that software is typically not specifically designed for reuse. Rather, it implements the particular computations needed for an analysis or project, and it in turn depends on all of the types of software described in Section 2. Challenges surrounding these mixed data and software packages include:

- Embedding of software (e.g. scripts, Jupyter notebooks) with data inputs, outputs, and other computation artifacts in mixed packages that are assigned a single DOI in which the software may not be individually referenceable
- Metadata for packages derives primarily from data repository communities but includes sections on software used, and these repositories do not yet embrace CodeMeta and similar efforts
- Need for documenting the provenance relationships between software, data, and products that describes the lineage of computational workflows
- Software in repository packages that import other software packages (e.g., via implicit language-specific library imports) but does not document these dependencies in metadata

4.4 Indexers

Indexers such as Google, Elsevier, Smithsonian Astrophysical Observatory (SAO) and NASA, etc. have created and maintain indices such as Google Scholar, Web of Science, the Astrophysics Data System (ADS).

These indices need to:

- Store a software type
- Capture software mentions in publications, etc.
- Provide metrics across software versions
- Develop and pilot tactics for determining metrics across different representations of software

4.4.1 How to count citations

- How to add across versions
    - Tooling
    - Indexing services
- There is a possible concern with identifying repeated citations to software in different locations within a publication (e.g. citation within a figure caption and a citation in a reference list)
    - We need to ensure any particular software is only counted once.
    - Given that we are promoting (and counting) formal citations for software (in the references section), this should not be an issue.
- How to identify software citations in reference lists
    - Dereferencing of DOI metadata?
- Should we attempt to count citations to software which do not have PIDs with adequate associated metadata?

Should we follow software dependency trees to count further citations?

4.5 Funders

As the organizations that support much of the work that leads to research software, funders have a large part to play in the collection of information relating to software production, particularly through research impact assessment platforms (e.g., ResearchFish). They may also mandate policies (e.g. data management plans) that may steer communities towards best practice in software citation, for instance to make it clear how software citations should be assessed in peer review.

Actions for funders include:

- Determine how best to reference software in proposals, research outcomes submissions and research output management plans
- Provide additional guidance about submitting software as research outputs to funders, especially focused on how software should be submitted in the case where a DOI is assigned by the funder
- Provide guidance about expectations of submitting software versions to funders

4.6 Institutions

Institutions, as the organization that generally employ researchers, have a large amount of influence on the community culture. They have a large part to play in the collection of information relating to software production in particular through current research information systems (CRIS) such as PURE, Converis or Symplectic Elements.

Actions for institutions include:

- Determine how to best measure and evaluate researchers' output, including their software contributions (multi-dimensional as software is not just code)
- Provide guidance on how to record software as a research outcome or impact in their CRIS
- Incorporate software citation principles and metadata standards into their institutional repository workflows and metadata schemas

5. Previous thoughts on challenges

Note that some of us also were thinking about this in 2016, when we published "The Challenge and Promise of Software Citation for Credit, Identification, Discovery, and Reuse". In that paper, we listed the following challenges — and possible solutions/methods.

- Identify necessary metadata associated with software for citation — We suggest metadata here, and the CodeMeta project is working to determine minimal metadata.

- Standardize proper formats for citing software in publications — The FORCE11 Software Citation Working Group is defining, and gaining community acceptance for, software citation principles, after which a follow-on group will begin implementation efforts.
- Establish mechanisms for software to cite other software (i.e., dependencies) — Software publication as software papers allows this. For software that is directly published, this is an open challenge.
- Develop infrastructure to support indexing of software citations within the existing publication citation ecosystem — The FORCE11 Software Citation Implementation Working Group will also work on this in collaboration with publishers and indexers.
- Determine standard practices for peer review of software — Professional societies and science communities need to determine how this will happen.
- Increase cultural acceptance of the concept of software as a digital product) — Acceptance will happen over time; unclear how to accelerate this process.

6. Discussion topics (based on items and comments above)

6.1. Goal of this document/process

Based on discussion in Section 1.

- What's the goal of this document? And how does this related to the software citation principles document? Are we creating an updated set of principles? Determining the work the FORCE11 group now needs to do? Adding detail beyond what's in the software citation principles?
- Perhaps think about this as the next layer of the discussion?
- List of challenges? Guidance for the group on how to address challenges?
- A way to organize guidance that we provide to others? (and to explain where we still have questions and uncertainties)
- Ultimately it should prove a number of case studies where examples from different communities, software projects in different languages manage to get the software citation principles as closely implemented as possible. (Ilian)

6.2. Repositories

Based on discussion in Section 1.

- Should we provide any guidance on which repositories should be used for software?
- What functionality should repositories provide to support software citation?

6.3. Citation files

Based on discussion in Section 1.

- We have multiple types of citation files now: codemeta.json, citation.cff, CITATION files as proposed/used by CRAN. What guidance can we provide users, if any? Can we provide tools or methods for translation between these files?

6.4. Types of software

Based on discussion in Section 2.

- We need to determine what scheme to use for the various software types. Dan originally proposed a simple listing. Alejandra proposed looking at various dimensions.
- We need to add other types that are not in the document now, such as services, executables, containers, packages or modules (that only work within a framework).
- Should this be a table? Showing dimensions and values? A drawing?

6.5. "Published" language

Based on discussion in Section 2.

- In the software citation principles paper, we used "published" to mean than an author had submitted the software to an archival repository along with metadata, and that the repository had issued an identifier. There is some confusion about this term. Is there a better term?
- We also need to make clear that made public is not the same as "published"

6.6. "Concept" language

We have chosen to use the word "concept" in this document to refer to the generic idea of a piece of software. This is distinct from a version of the concept. For example, Microsoft Word is a concept, and Microsoft Word 13 and Microsoft Word 16.0.1 are both versions. We are aware that there is other language used in other communities, such as "work" (or "creative work"), "product", "software family", and "project", but given that there is no consensus for the appropriate work in the context of software, we have chosen to use "concept".

Here, we use this to distinguish between the abstract concept vs the specific representations (in this case in the form of software versions). This is similar to the dichotomy in datasets, where there is the conceptual dataset and the specific representation (in W3C DCAT - and schema.org

as is based in DCAT, as well as in DATS) this is Dataset vs Distribution). In REST, there is the distinction between resource and representation.

Note: In library science, the difference between a conceptual project and its specific instantiations has been addressed in a concept model called FRBR (Functional Requirements for Bibliographic Records). While the specific language of FRBR might be confusing to non-librarians, it is mentioned here for librarians. The model includes work, expression, manifestation, and item. The first two probably address the issue here: A "work" is the conceptual level of a creation. For example, the novel "Gone With the Wind". This would correlate to "concept" as used here. Microsoft Word 2016 would be a software work. An "expression" is what we might call an edition of that work. For example, the original text and a translated text of "Gone With the Wind" would both be considered expressions. Updated editions that do things like correct typos and modernize spelling would also be considered expressions. This would probably correspond to versions or releases of software. Microsoft Word 2016 for Windows 32 bit version 16.16 would be an expression.

"A Framework for Software Preservation" follows the FRBR model, but uses the terms product, version, variant, and instance in place of work, expression, manifestation, and item.

6.7. Layers of software (via Hinsen)

Based on discussion in Section 2.

- Matt Jones suggested adding some explanation about software layers. Does this make sense to add? Or is it out of scope for a citation discussion?

6.8. Non-developer identification of software

Based on discussion in Section 3.1 and 3.2 and 3.3 and 3.5

- For items like commercial software packages, how should they be identified? This is a concern in the cases where the developer does not create an identifier.
- One option is to use RRIDs
    - https://www.force11.org/group/resource-identification-initiative
    - https://scicrunch.org/resources
- This is also somewhat what ASCL offers.
- In 3.3, we need to know where to find metadata for such software.
- If there is a primary identifier (e.g. a DOI), we want to avoid creating a new one.

Daina to write suggested text to:

6.9. Software Heritage and software identification

Based on discussion in Section 3.1 and 3.5

- In 3.1.4, there is an example of identification of software in software heritage: https://archive.softwareheritage.org/swh:1:rev:81c66c0db5c381dacc0841a4c16e0b3876b15b89;origin=https://github.com/applicationskeleton/Skeleton/
- In a comment, another way to identify this same software is https://identifiers.org/swh:1:rev:81c66c0db5c381dacc0841a4c16e0b3876b15b89
- Which, if either, should we recommend?

6.10. Bibtex

Based on discussion in Section 3.4

- There is a bibtex software type, but a lot more is needed around it to make it truly useful.
- For example, the DataCite Content Resolver in Section 3.4 returns a bibtex item of type @misc for software, and this seems likely to be the case for other tools as well.
- Also, there are needs around how a bibtex entry of type @software is processed. Currently, there is no specific driver set up in standard.bbx, it uses the one defined for @misc. See https://github.com/force11/force11-sciwg/issues/48#issuecomment-371871401 for more info.
- Journal and conference-specific class files might also need to be updated to be able to properly process @software entries

6.11. Text citation styles for software

Based on discussion in Section 3.5

- How can software be identified as software in text citations when the style guides do not include it?

6.12. Generating DOIs for "unversioned" software "packages"/"works"

Based on discussion in Section 3.5

- There is no public way to directly obtain a DOI for "unversioned" software "packages"/"works".

- This can be done as a byproduct of depositing software with Zenodo, as depositing the first version will create a DOI for that specific version and a DOI for the set of versions that are created with Zenodo, but this is not the same as being able to do this generically.
- Organizations who register DataCite DOIs can do this via Fabrica, but this is a small number of organizations.

6.13. Counting citations across versions

Based on discussion in Section 3.6

- This text needs to be cleaned up and made easier to read and understand.
- This has now been moved to 4.4.1

6.16. Publisher items

Based on discussion in Section 4.2

- Need to address issues here.

6.17. Funder items

Based on discussion in Section 4.5

- Need to address issues here.

6.18. Institution items

Based on discussion in Section 4.6

- Need to address issues here.

6.19. Discussion of specific tools

Based on discussion in many sections. These tools include Software Heritage, CFF, CodeMeta, cffconvert

- Do we want to talk about specific tools? Doing so makes the document more useful, but also makes it likely to go out of data quickly.

Our decision is to try to talk at a high level, but also include specific tools as examples, and to emphasize particular tools/standards that we think should be adopted.

6.20. Out of scope items

- Transitive credit. Citing software directly in a paper or product is analogous to traditional citation of scholarly works, but software is much more frequently 'cited' via dependency chaining and provenance tracking. Katz & Smith discuss transitive citation and the potential impact that would have on our understanding of the impact of software on scholarly and other endeavours. Both dependency chains and provenance graphs provide mechanisms for assigning credit for an impactful outcome such as a manuscript publication to the data and software on which the findings were based. Citations to a downstream package can then be properly attributed to upstream software and data that were fundamental to the cited outcomes, even though only the proximate package was directly cited in the reference list of the paper. While this concept is a valuable idea in discussion of credit, it is out of scope in a discussion of citation.
- Review of software is out of scope in a discussion of citation, as citation indicates that software has been used, whether or not it has been reviewed. Similarly, review of publications and review of proposals/grants are out of scope, except that the processes should have policies that ensure software citation is implemented and used within them.