Software Reuse, Repurposing and Reproducibility

Catherine Jones
STFC
Harwell Oxford
Didcot
44 (0)1235 445402
Catherine.jones@stfc.ac.uk

Brian Matthews
STFC
Harwell Oxford
Didcot
44 (0)1235 446648
Brian.matthews@stfc.ac.uk

Ian Gent
St Andrews University
North Haugh
St Andrews
+44 (0)1334 46 3247
Ian.gent@st-andrews.ac.uk

ABSTRACT
Software underpins the academic research process, regardless of discipline. With the increased focus on the long term value of data and other research outputs, then more attention needs to be paid to how software used in these processes is both identified and preserved for the long term as much data is meaningless without the related software. In this poster we describe the aims, objectives and current results of the Jisc funded project Software Reuse, Repurposing and Reproducibility (Software RRR). This poster discusses the issues around persistently identifying software, makes some recommendations for good practice, and discusses the relationship between identifying source code and a playable version of this software.

General Terms
Preservation strategies and workflows

Keywords
Software preservation

1. INTRODUCTION
Software underpins the academic research process, regardless of discipline. Software is written to be run, and while programmers might strive for elegance or beauty in the code, the overwhelming point of software is to execute it. To be able to understand and use/reuse and preserve data then the software code which generated, analysed or presented the data will need to be retained and executed. A starting point is the persistent identification of software to maintain the integrity of software as an item over time. This is an emerging area and services such as Zenodo (https://zenodo.org/) are enabling developers to persistently identify code.

Software is a composite artefact and may have different components bundled together. This can be seen by the following definition:

> *"Computer software includes computer programs, libraries and their associated documentation. The word software is also sometimes used in a more narrow sense, meaning application software only."(* https://en.wikipedia.org/wiki/Software Retrieved 12/6/2015)

Consequently, software cannot be treated in a similar manner to other digital artefacts (for example documents, media content or data) and needs separate consideration for preservation. Further, if the software is to remain reproducible and reusable, additional consideration needs to be taken to maintain its correct execution behaviour.

1.1 Aims of the software RRR project
The Software RRR project is investigating the persistent identification of software and how links can be made to runnable versions of software enabling preservation of functionality. The project builds on the Recomputation project (http://recomputation.org) [1] and earlier work on a framework for software preservation [2],[3].

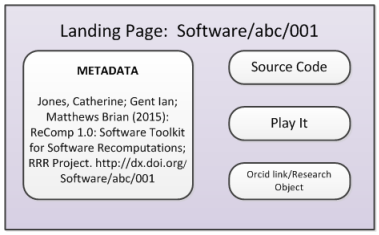

Figure 1 Landing Page

The figure above represents the vision of the project which is encapsulated in a landing page for a persistently identified software object with effective metadata, links to the downloads, including source code and a runnable version, together with hooks to other entities in the wider context such as Orcid, data and publications. Thus a user can: uniquely identify software released in a particular context (via software citation); access landing pages which give additional metadata to describe the software; access a runnable version of the software replicating its original behavior; and download packages with sufficient information to allow its reconstruction locally.

To realise this vision, we need to provide consistent guidelines for software identification together with local metadata and a virtualized platform for replay and recomputation. In the rest of this short paper, we concentrate on issues of persistently identifying software.

2. ISSUES IN PERSISTENT IDENTIFICATION
2.1 What is being identified?
A key issue is what exactly is being identified, as described in the previous section software is a complex object and may include one or more of: source code, executable version, packaged version, additional items such as included libraries and documentation. Further, software typically is an evolving artefact, with different expressions being made available through a software release cycle, reflecting the changes in functionality and computing environment which software undergoes over time.

We use a four level model of software to describe the different expressions a software system has over its release cycle. This model enables better understanding of what might need to be persistently identified.

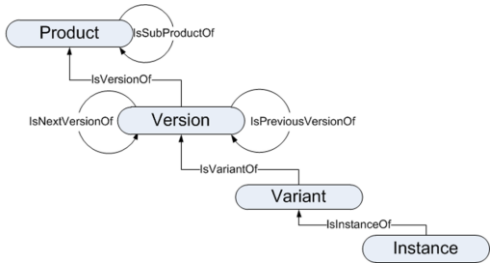

Figure 2 Levels of Software

- **Product:** The whole top-level conceptual entity encompassing the whole lifecycle of the software, and is how the system may be commonly or informally referred to
- **Version:** is an expression of the product which provides a single coherent presentation of the product with a well-defined functionality and behaviour and usually in how it interacts with the computing environment.
- **Variant** is a version adapted to a specific operating environment
- **Instance** is an actual deployment of a software product which is to be found on a particular environment or machine.

A particular software citation will typically refer to a particular expression of the software which is used in a particular context, thus the same expression should be used to validate the results.

2.2 Environment

The environment that the software was built and operates in is a vital part of ensuring software is not just preserved but remains runnable. Metadata supplied with the software expression should describe its target environment. This is a complex area and has not been addressed in this phase of the project.

2.3 Metadata

Metadata plays an important role in the discovery, access, management & preservation of software, and thus we need to consider the appropriate metadata to provide. We address the use of DataCite discovery metadata to describe software in the next section.

3. DATACITE METADATA

DataCite (www.datacite.org) issues Digital Object Identifiers for data and other research artefacts. While it is not the only persistent identification system available, its wide adoption means it is an important source for identification of software, and consequently we concentrate on how to adapt Datacite DOIs for the citation of software expressions.

Datacite provides set of metadata elements to characterise digital objects for search and discovery [4]. The DataCite elements have been analysed to propose an appropriate profile for describing software. The approach taken is not to prescribe the content of any specific element but to describe the importance and enable the potential user to establish the correct answer for theirown situation. This poster will discuss how some key elements are used in the context of identifying software.

3.1 Creator

This element identifies the people responsible for the software. However this may not be straightforward to ascertain as software has a long life-span and may be worked on by many people. The point during the development cycle that the first DOI is given may also affect those identified as creators.

3.2 Title

The title of the resource is a mandatory field and can contain significant information. In a software context, there are some specific issues. If it a piece of software written by a single person for a specific project does it actually have a name? Is the official name different from the common name? What effect is versioning or branching of code going to have on the name? Will the name used be unique enough for it to be found and distinguished from other search results?

3.3 ResourceType

There is a resource type of Software, but this is a rather wide category and at present there aren't many formal suggestions for how this might be broken down further. This is an area with potential for further work.

3.4 Description

This field is designed to enable the addition of further information to assist in the understanding of the object being identified. Currently the two subtypes being used for software are Abstract and Other. These do not encourage the use of this field for technical information that may be needed to understand the object and a new subtype with a more descriptive label may be of assistance

4. FURTHER WORK

The first phase of this project has been concerned with persistent identification. The next phase is concerned with how software may be captured in such a way to ensure it remains runnable, thus preserving the performance. Being able to link the different software artefacts together in a fixed complex object will enable the long term preservation of software

ACKNOWLEDGMENTS

This work has been funded by Jisc in the Research@Risk scheme. We are grateful to our colleagues Tom Griffin, Simon Dobson and John McDermott for their comments and advice.

REFERENCES

[1] Arabas, S. et. al. 2014, Case Studies and Challenges in Reproducibility in the Computational Sciences. 1st Summer School on Experimental Methodology in Computational Science Research, St Andrews, August 4-8, 2014. arXiv:1408.2123

[2] Matthews, B., Shaon, A., Bicarregui, J., Jones, C., Woodcock, J., and Conway, E. 2009. Towards a Methodology for Software Preservation. In 6th International Conference on Preservation of Digital Objects (iPres 2009), San Francisco, USA, 5-6 Oct 2009.

[3] Matthews, B., Shaon, A., Bicarregui, J., and Jones, C. 2010. A Framework for Software Preservation. International Journal of Digital Curation 5, no. 1.

[4] Datacite Metadata Working Group, 2015. DataCite Metadata Schema for the Publication and Citation of Research Data. Version 3.1, June 2015, doi:10.5438/001