Software Deposit and Preservation Policy and Planning Workshop Report

Edited by Christopher Brown, Jisc; Neil Chue Hong, The Software Sustainability Institute; Mike Jackson, The Software Sustainability Institute.

1.0

doi:10.5281/zenodo.1303826

3rd July 2018

Contents

1. Introduction

On 7th March 2018, Software Sustainability Institute[1] and Jisc[2] organised a Software Deposit and Preservation Policy and Planning Workshop at Jisc's London offices. This workshop brought together members of pilot institutions from Jisc's Research Data Shared Service[3] (RDSS) and other institutions interested in research software preservation.

The workshop presented work done by the Software Sustainability Institute to examine the current workflows used to preserve software as a research object as part of research data management procedures. The attendees discussed these workflows, the nature of software deposits, how to collect software-related metadata and what metadata should be collected to satisfy the requirements of researchers, research data managers (RDMs), institutions and funders. The workshop concluded with a discussion of the best practices required for software preservation and ways to ensure that these can be adopted in research communities.

The outputs from this workshop are intended to guide the development of a reference implementation for a software deposit tool that supports common workflows, complemented with guidance on metadata collection and software deposit content. These will be presented at a second workshop, planned for summer 2018.

This report summarises the key findings and outcomes from the workshop.

1.1. Terminology

In this report, the following terminology is used.

Research software or **software** is any collection of scripts or code written for, or used within, a research context. For example, 50 lines of bash shell commands, 250 lines of R, or 10,000 lines of Fortran using MPI are all, for the purposes of this report, research software.

Source code repository is a version control tool, revision control system, source code management tool or source code repository i.e. any tool that manages versions of source code and related files. For example, Git, Mercurial, Subversion or Microsoft Team Foundation Version Control[4].

Repository hosting service is a service that hosts version control repositories within an institution, within a community, or for myriad individuals and communities. For example, GitHub[5], BitBucket[6], GitLab[7], CCPForge[8], or Microsoft Visual Studio Team Services[9].

Digital repository is a repository, archive or service that hosts digital artefacts, including research outputs such as papers, presentations, data sets and software. For example, Zenodo[10], Figshare[11], DSpace[12], or Samvera[13].

---

[1] The Software Sustainbility Institute, https://www.software.ac.uk
[2] Jisc, https://www.jisc.ac.uk
[3] Research data shared service, https://www.jisc.ac.uk/rd/projects/research-data-shared-service
[4] Use Team Foundation Version Control, https://docs.microsoft.com/en-us/vsts/tfvc/overview
[5] GitHub, https://github.com/
[6] BitBucket, https://bitbucket.com/
[7] GitLab, https://gitlab.com/
[8] CCPForge, https://ccpforge.cse.rl.ac.uk/gf/
[9] Microsoft Visual Studio Team Services, https://www.visualstudio.com/team-services/
[10] Zenodo, https://zenodo.org
[11] Figshare, https://figshare.com

Digital preservation system is a repository, archive or service that also hosts digital artefacts, including research outputs, but also implements strategies to afford long term access to the digital artefacts, in the face of technological changes. For example, Archivematica[14], Preservica[15] or Arkivum Perpetua[16].

1.2. Acknowledgements

Many thanks are due to those who contributed their time and expertise to this report. In particular to all those who contributed to the workshop: Steffan Adams, Cardiff University; Matthew Addis, Arkivum; Alessia Bardi, ISTI-CNR; David Clipsham, National Archives; Maria Guerreiro, eLife; Matthew Herring, University of York; Catherine Jones, STFC; Somaya Langley, University of Cambridge; Joanna Leng, University of Leeds; James Long, University of Plymouth; Mary McDerby, University of Manchester; Rachel MacGregor, University of Lancaster; Martin O'Reilly, Turing Institute; Edward Ransley, University of Plymouth; Ben Samuels, University of Lincoln; Matthew Siekier, University of Huddersfield; Justin Simpson, Artefactual Systems; Mark Woodbridge, Imperial College; Wei Xing, Crick Institute.

Jonathan Cooper, UCL, provided information about the Research Software Dashboard.

Naomi Penfold, eLife, provided feedback on a draft copy.

The work described in this report was funded by Jisc.

[12] DSpace, http://www.dspace.org
[13] Samvera, http://samvera.org
[14] Archivematica, https://www.archivematica.org/
[15] Preservica, https://preservica.com/
[16] Arkivum Perpetua, https://arkivum.com/perpetua/

2. A review of software deposit tools

Prior to the workshop, tools for creating a software deposit, collecting related metadata and submitting these into digital repositories were reviewed. These deposit tools, and target digital repositories, are shown in Table 1: Software deposit tools.

Workflow	Tool	Zenodo	Figshare	DSpace	Nature	Language
Manual		Y	Y	Y		
Event-driven	Zenodo-GitHub integration[17]	Y			Service	Python
Event-driven	Figshare-GitHub integration[18]		Y		Service	Unknown
Event-driven	fidgit[19]		Y		Service	Ruby
Interval-driven						
Manually-triggered (hosted)	Mozilla Science Code as a Research Object[2021]		Y		Service	JavaScript
Manually-triggered (hosted)	EasyDeposit[2223]			Y	Service Library for service development	PHP
Manually-triggered (local)	PyRDM[24]	Y	Y	Y	Library for service or application development	Python
Manually-triggered (directory)						

Table 1: Software deposit tools

From these tools, a set of common software deposit workflows, that are currently supported, or could be supported in future, were identified. These are discussed in section 3 Software deposit workflows.

2.1. Deposit tool limitations

The deposit tools have several limitations. These include:

[17] Making Your Code Citable, https://guides.github.com/activities/citable-code/, 10/2016. Service used 02/2018.
[18] How to connect figshare with your GitHub account, https://knowledge.figshare.com/articles/item/how-to-connect-figshare-with-your-github-account-1. Service used 02/2018.
[19] Fidgit, https://github.com/arfon/fidgit, release 0.0.3, aadf295e1102c75f16aec9f4aac1f94bf7a4cc40, 20/10/13; 7bb64db3c3991c69abe885601a215d5dd1da3e50, 07/11/17 (updates to documentation only).
[20] Code as a Research Object, http://mozillascience.github.io/code-research-object/. Service used 02/2018.
[21] Code as a research object, https://github.com/mozillascience/code-research-object/, e44d3d7e3733e38982b77fc3720b960ce343fe37, 28/12/14.
[22] EasyDeposit Client, http://easydeposit.swordapp.org/example/github/. Service used 02/2018 to submit to DSpace 6.2 Demo Server, https://demo.dspace.org/.
[23] EasyDeposit Client, https://github.com/stuartlewis/EasyDeposit, f887cdd7f41661a60494991debaed31a5dd42e9d, 11/04/16.
[24] PyRDM, https://github.com/pyrdm/pyrdm/, v0.3, 2fd0d507b84d76c8539fcaae28fac1ae261700fd, 16/06/16; e5a0c5516e3f82197c1d7b5f437ac31c9f169374, 24/10/17 (update to documentation only).

- **Only Git is supported**: Zenodo-GitHub integration, Figshare-GitHub integration, fidgit, Mozilla Science Code as a Research Object, EasyDeposit and PyRDM can only be used with Git source code repositories. Researchers might instead be using Mercurial, Subversion, Microsoft Team Foundation Version Control, for example, or they might not be using a source code repository at all.
- **Only GitHub is supported**: Zenodo-GitHub integration, Figshare-GitHub integration, fidgit, Mozilla Science Code as a Research Object, and EasyDeposit can only be used with GitHub. Researchers might instead be using BitBucket, GitLab, CCPForge or Microsoft Visual Studio Team Services, for example.
- **Only GitHub "release" events are handled**: Zenodo-GitHub integration, Figshare-GitHub integration, and fidgit only trigger deposits on receipt of GitHub "release" events. Other repository hosting services for Git repositories, for example BitBucket and GitLab, do not support the concept of "releases" or "release" events.
- **Deposits latest commit on default branch only**: Figshare-GitHub integration can initially only deposit the latest commit in the default branch of a Git repository. A researcher cannot choose another branch or a specific commit for their initial deposit.
- **Assumes Git "master" branch exists**: Mozilla Science Code as a Research Object and EasyDeposit can only deposit the latest commit on the "master" branch of a Git repository. It cannot be guaranteed that a Git repository has a "master" branch.
- **Assumes Git repository was cloned**: PyRDM assumes that the local Git repository is a clone of another repository. In practice, the Git repository might only exist locally.
- **Assumes Git clone has "origin" remote**: PyRDM assumes that the local Git repository clone has a remote called "origin". A clone cannot be guaranteed to have a remote called "origin".
- **Scrapes GitHub web pages**: EasyDeposit scrapes GitHub web pages for the message associated with the latest commit, rather than accessing this via the GitHub API.

The workflows of the next section have been defined to remove these limitations.

2.2. Metadata limitations

The review of the deposit tools revealed some of the issues that can arise when trying to automatically collect metadata for software deposits. These include:

- **Metadata unavailable via repository hosting service APIs:** When using the Figshare-GitHub integration, a researcher must enter values for the Figshare fields "Categories" and "Keyword(s)" before their initial deposit. The Figshare-GitHub integration, the GitHub API and its webhooks cannot provide this information.
- **Inconsistent versioning:** When using the Figshare-GitHub integration, a researcher may end up with a version tagged "Release 0.0.4" on GitHub having a version number of 2 in Figshare with no record of the association between the two version numbers. Zenodo and Zenodo-GitHub integration do preserve this association.
- **Contributors may not be synonymous with authors:** Zenodo-GitHub integration derives an author list based on the contributors. However, as Markus Ankerbrand notes[25], the software might have evolved sufficiently that it can be considered a new research output, for which the original contributors would warrant being cited, rather than listed as co-authors.

[25] Ankenbrand, A. (2016) "How to properly use zenodo DOIs for derived software?", 08/09/2016. https://openscience.uni-bielefeld.de/985/how-to-properly-use-zenodo-dois-for-derived-software

- **Limitations in metadata conversion:** EasyDeposit can deposit software into DSpace via the SWORD[26] API with METS[27] and SWAP[28]-encoded metadata. However, DSpace's SWAP-to-DSpace Internal Model crosswalk[29] supports only a handful of SWAP values. For example, the values "publisher", "subject", "rights", and any type that is not "JournalArticle" are not supported.

More generally, each digital repository exposes different APIs and differ in what they view to be mandatory or optional metadata.

[26] Jones, R. and Lewis, S. (eds). SWORD 2.0 Profile, 2011, http://swordapp.org/sword-v2/
[27] Metadata Encoding and Transmission Standard (METS), http://www.loc.gov/standards/mets/
[28] SWAP Scholarly Works Application Profile (SWAP), http://www.ukoln.ac.uk/repositories/digirep/index/Eprints_Application_Profile
[29] DSpace SWORD SWAP ingest cross-walk, https://github.com/DSpace/DSpace/blob/master/dspace/config/crosswalks/sword-swap-ingest.xsl e0bd496e644d170789f3b15811a481f5c263e9cb, 24/11/16 (marked as "work in progress").

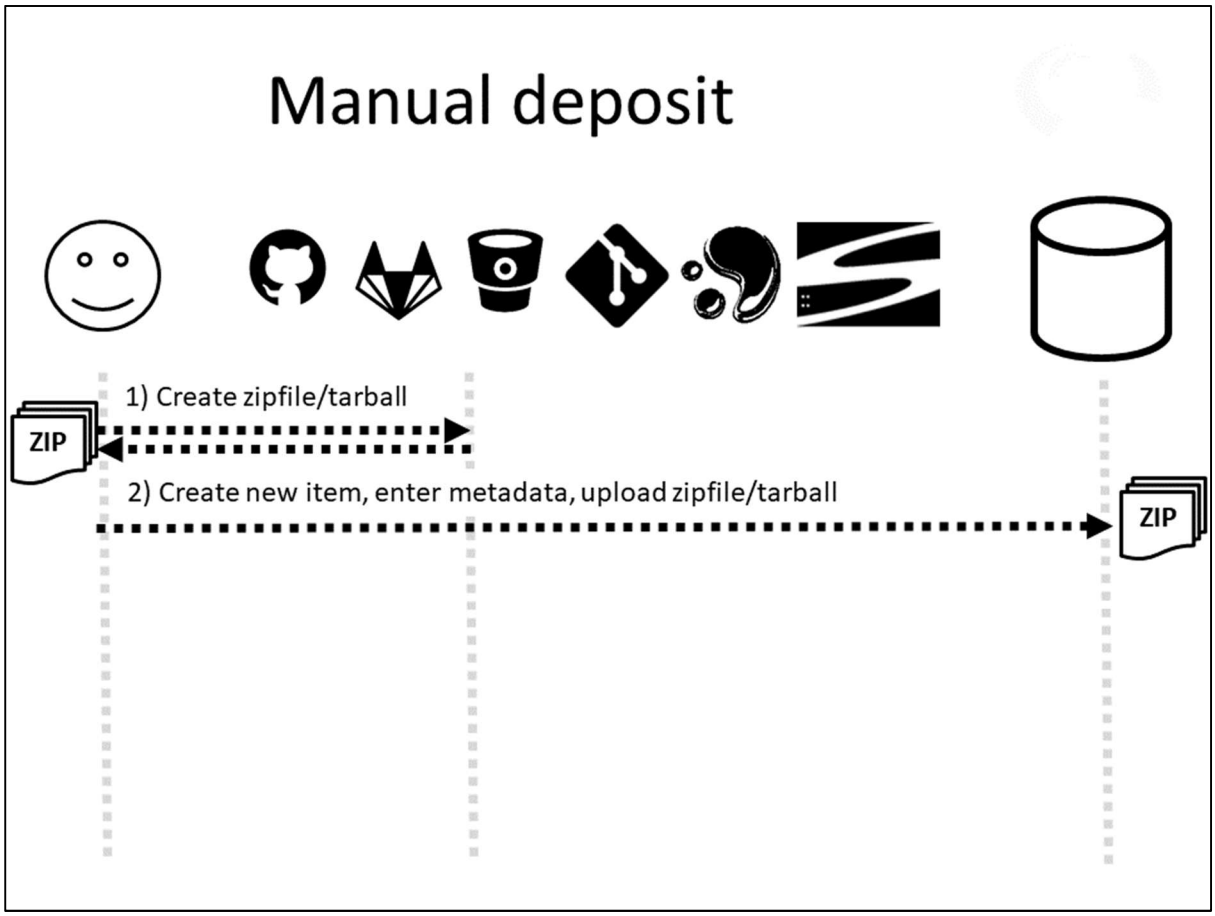
3. Software deposit workflows

From the review of deposit tools, a set of common software deposit workflows, which are currently supported or could be supported in future, were identified.

The following aspects were considered out-of-scope when identifying these workflows (however, this should not be interpreted to mean that these aspects are in any way unimportant):

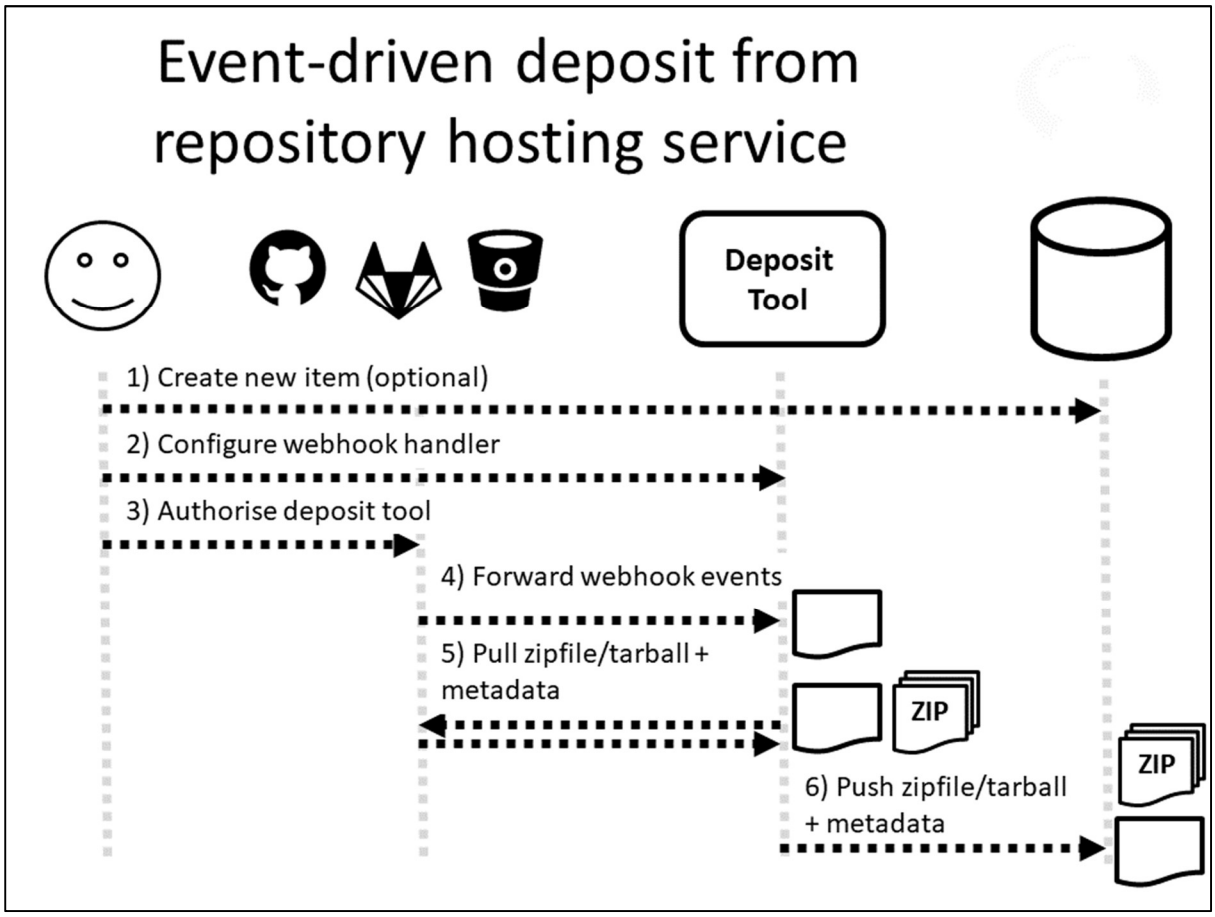
- Incremental software deposits i.e. uploading software in groups of files at separate intervals over time. A deposit is considered as a single zipfile/tarball containing the software, submitted along with complementary metadata.
- Post-deposit activity, including any review of a deposit by an RDM, archivist or other stakeholder, or subsequent updates to related metadata, by researchers, their representatives or other stakeholders, after the initial deposit.
- Authentication and authorisation, whether source code repository-, repository hosting service- or digital repository-specific. Delegation of rights, for example a researcher who makes a deposit, but wants to allow colleagues to update its related metadata, can be done out-of-band.
- Binary deposits, for example deposits that consist primarily of executables, binary libraries (e.g. JAR files, LIB files, DLLs), Docker or Singularity containers, or virtual machine images.

3.1. Manual deposit



Within this workflow there is no deposit tool. A researcher is responsible for gathering both their software and metadata and making their deposit into the digital repository manually e.g. via a web form offered by the digital repository.

3.2. Event-driven deposit from repository hosting service



GitHub, GitLab and BitBucket all support APIs to access zipfiles/tarballs and metadata for specific versions of code held within Git repositories and, for BitBucket, Mercurial repositories also. GitHub, GitLab and BitBucket also all support webhooks that are raised when repository-specific events occur. GitHub supports “release” events that are triggered when a new release is created. Though GitLab and BitBucket have no concept of releases, they do, as does GitHub, support events related to tags:

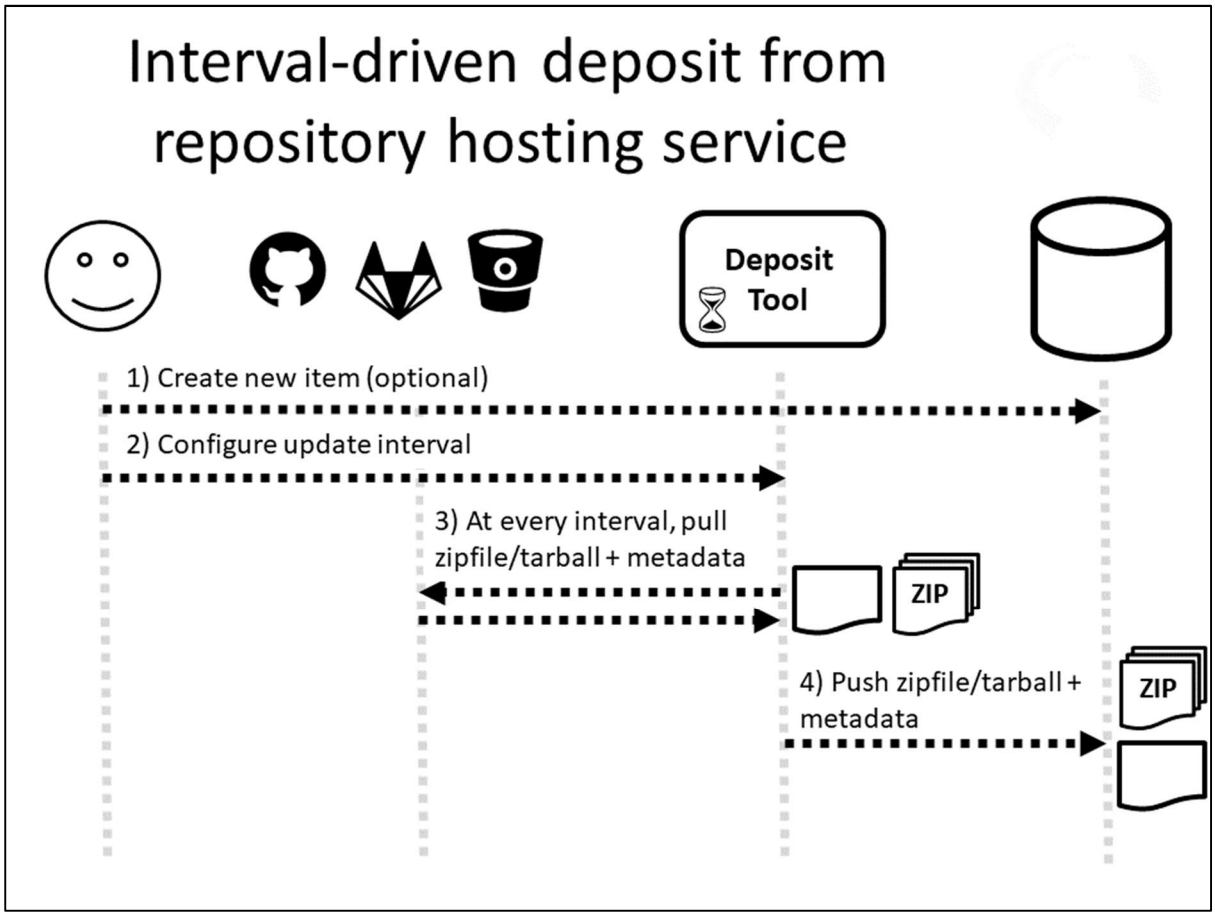
- GitHub “create” (new branch/tag) event.
- BitBucket “push” (new branch/tag) event.
- GitLab (new) “tag” event.

Within this workflow, the deposit tool subscribes to the repository hosting service. When an event is created e.g. in response to a release being created on GitHub, the deposit tool gets the version of the software from the repository hosting service, compiles associated metadata and makes the deposit into the digital repository.

The use of a tag pattern allows a researcher to define a naming convention, so that only versions tagged in a specific way, defined by the researcher, will be deposited by the deposit tool.

This workflow was inspired by Zenodo-GitHub integration, Figshare-GitHub integration, and fidgit.

3.3. Interval-driven deposit from repository hosting service

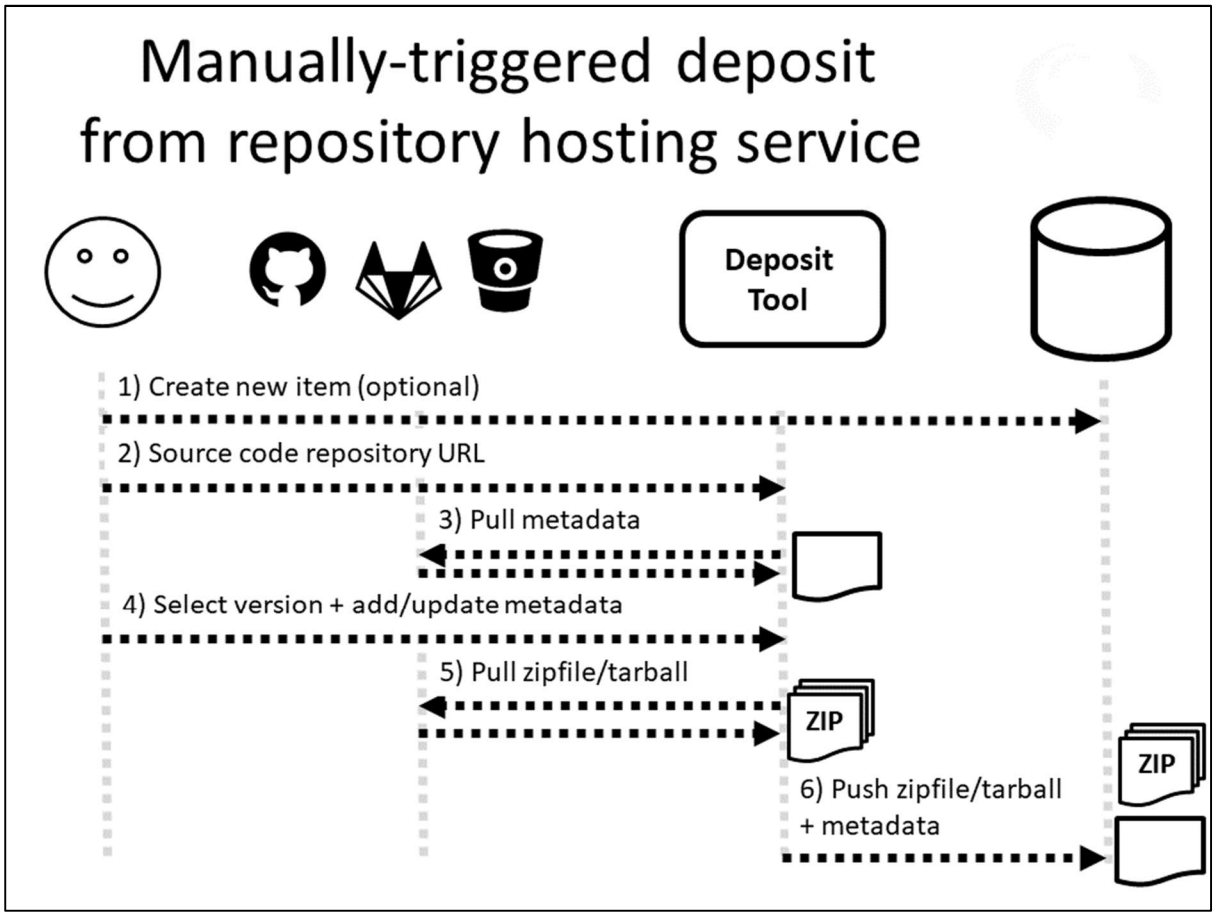


This workflow is like the event-driven deposit from repository hosting service workflow. However, instead of deposits being triggered by events from the repository hosting service, the deposit tool is configured with a query interval. At each interval, the deposit tool queries the repository hosting service for new releases, for GitHub, or tags matching a specific pattern, for Github, BitBucket or GitLab, compiles associated metadata and makes the deposit into the digital repository.

None of the reviewed deposit tools supports this workflow. Instead, this workflow was inspired by University College London's Research Software Dashboard[30] which provides up-to-date status information on research software under development at UCL. One of UCL's motivations for using interval-driven deposit was to avoid the need for researchers to configure webhooks in GitHub, BitBucket or GitLab, thereby removing a barrier to use by researchers.

[30] Research Software Dashboard, UCL, http://www.ucl.ac.uk/isd/services/research-it/research-software/dashboard

3.4. Manually triggered deposit from repository hosting service



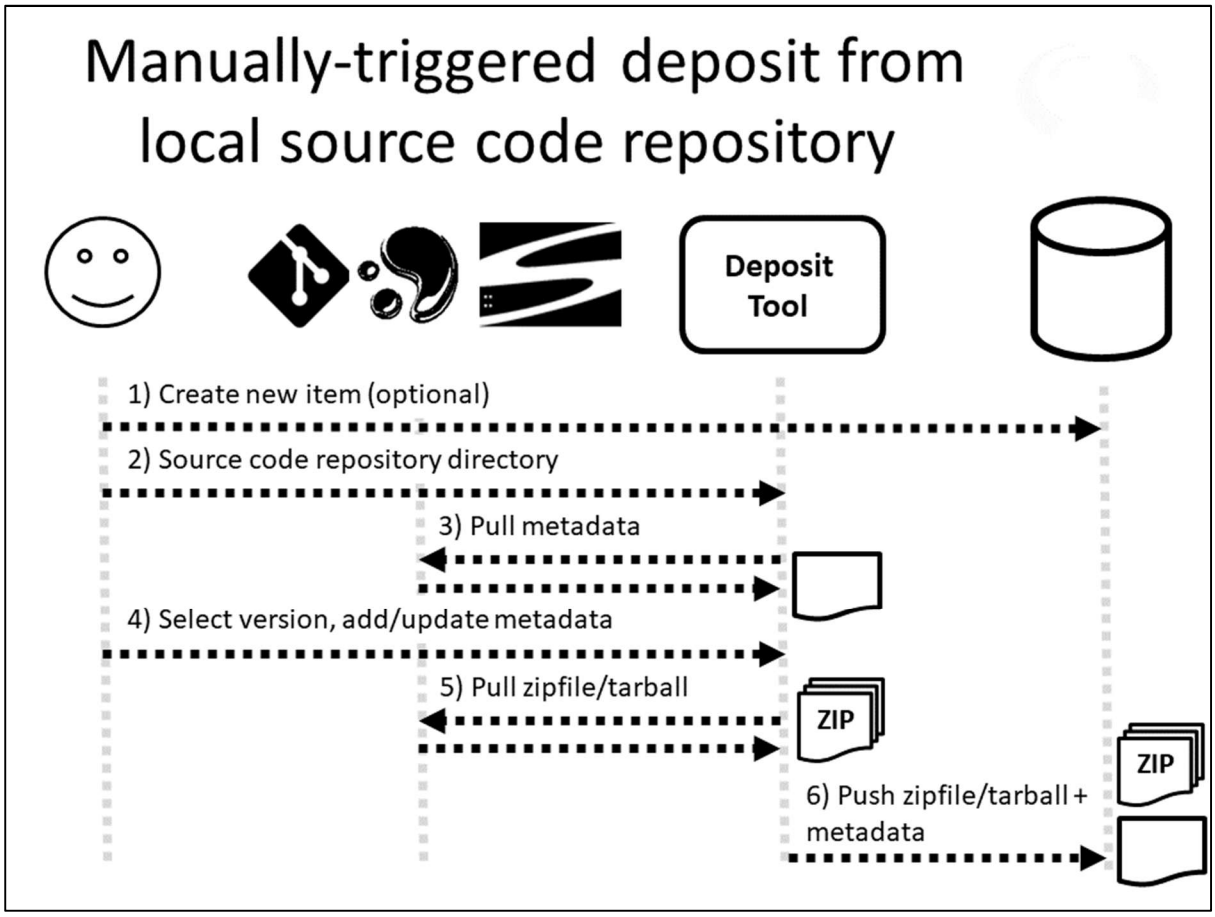
GitHub, GitLab and BitBucket all support APIs to access zipfiles/tarballs and metadata for specific versions of code held within Git repositories and, for BitBucket, Mercurial repositories also. A version can be the latest version on the current/default branch, the latest version on a named branch, a tag, a specific commit or, for GitHub a release.

Within this workflow, a researcher requests that a deposit of their software be made. This might be done by running a command at the command-line, running an application, entering a URL into a web form, or dragging and dropping a URL onto a bookmarklet.

Depending upon the nature of the deposit tool, the researcher might specify the version to deposit or the available versions might be presented to them by the deposit tool, after it has queried the repository hosting service. The deposit tool gets the version of the software from the repository hosting service, compiles associated metadata and makes the deposit into the digital repository.

This workflow was inspired by Figshare-GitHub integration, Mozilla Science Code as a Research Object, and EasyDeposit.

3.5. Manually triggered deposit from local source code repository



zipfiles/tarballs of specific versions of Git, Mercurial and Subversion source code repositories can be created using Git's "archive" command, Mercurial's "archive" command and Subversion's "export" command (for local or remote source code repositories). A version can be the latest version on the current/default branch, the latest version on a named branch, a tag, or a specific commit.

Within this workflow, a researcher requests that a deposit of their software be made. This might be done by running a command at the command-line, running an application, entering a directory into a web form, or dragging and dropping a directory onto a bookmarklet or application.

If using Git, the deposit tool could be exposed via an "archive" remote and, whenever a researcher wants to start a deposit, they push to this remote[31]. The "archive" remote is exposed by the deposit tool and handles the deposit from thereon. A similar approach could be done within Subversion with a deposit triggered by a post-commit hook triggered via a copy to a "tags" directory with a tag name conforming to some naming scheme.

Depending upon the nature of the deposit tool, the researcher might specify the version to deposit or the available versions might be presented to them by the deposit tool, after it has queried the source code repository.

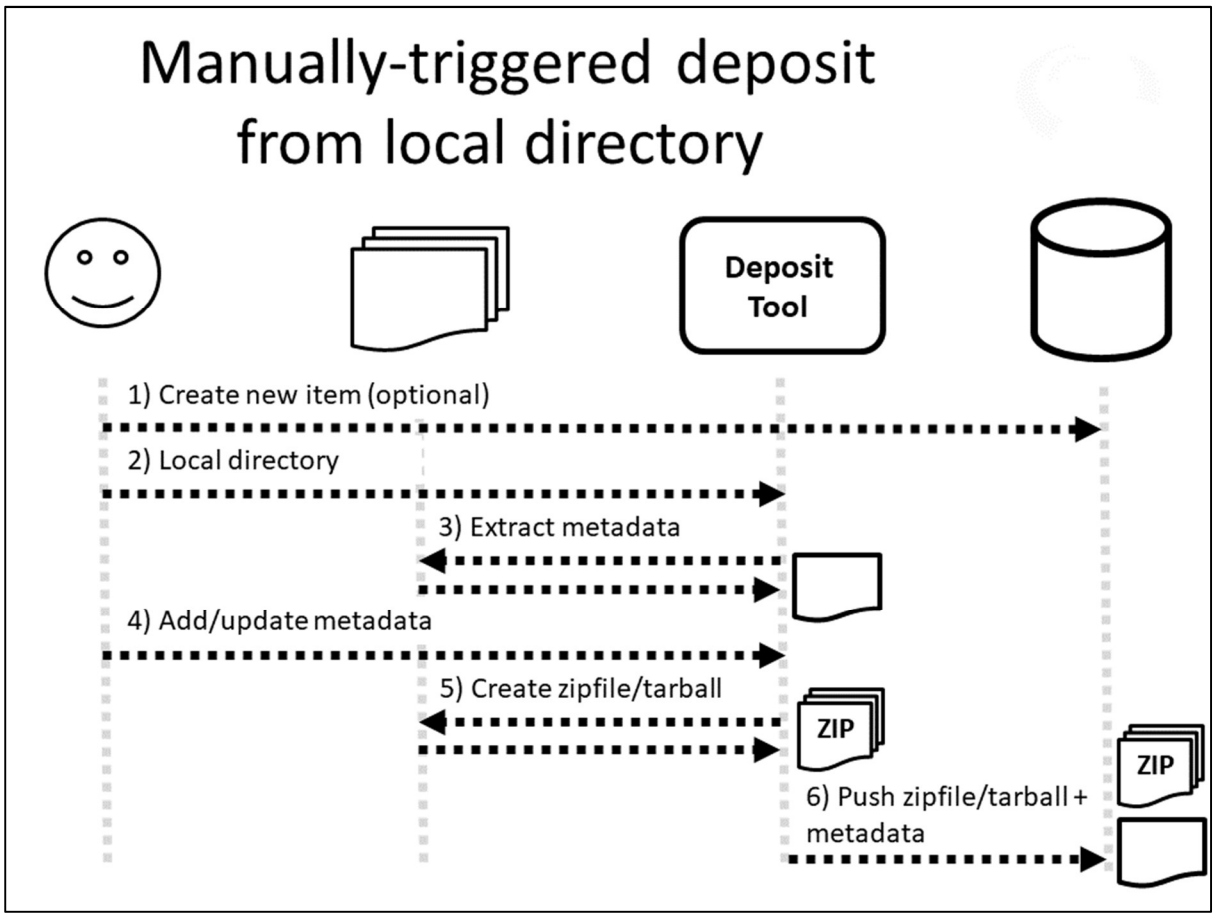
The deposit tool gets the version of the software from the repository hosting service, compiles associated metadata and makes the deposit into the digital repository.

[31] Thanks to Mark Woodbridge for this idea. Mark comments that "it's the Heroku model i.e. push somewhere other than origin in order to trigger an action."

This workflow was inspired by PyRDM.

3.6. Manually triggered deposit from local directory

This workflow was added post-workshop for completeness.

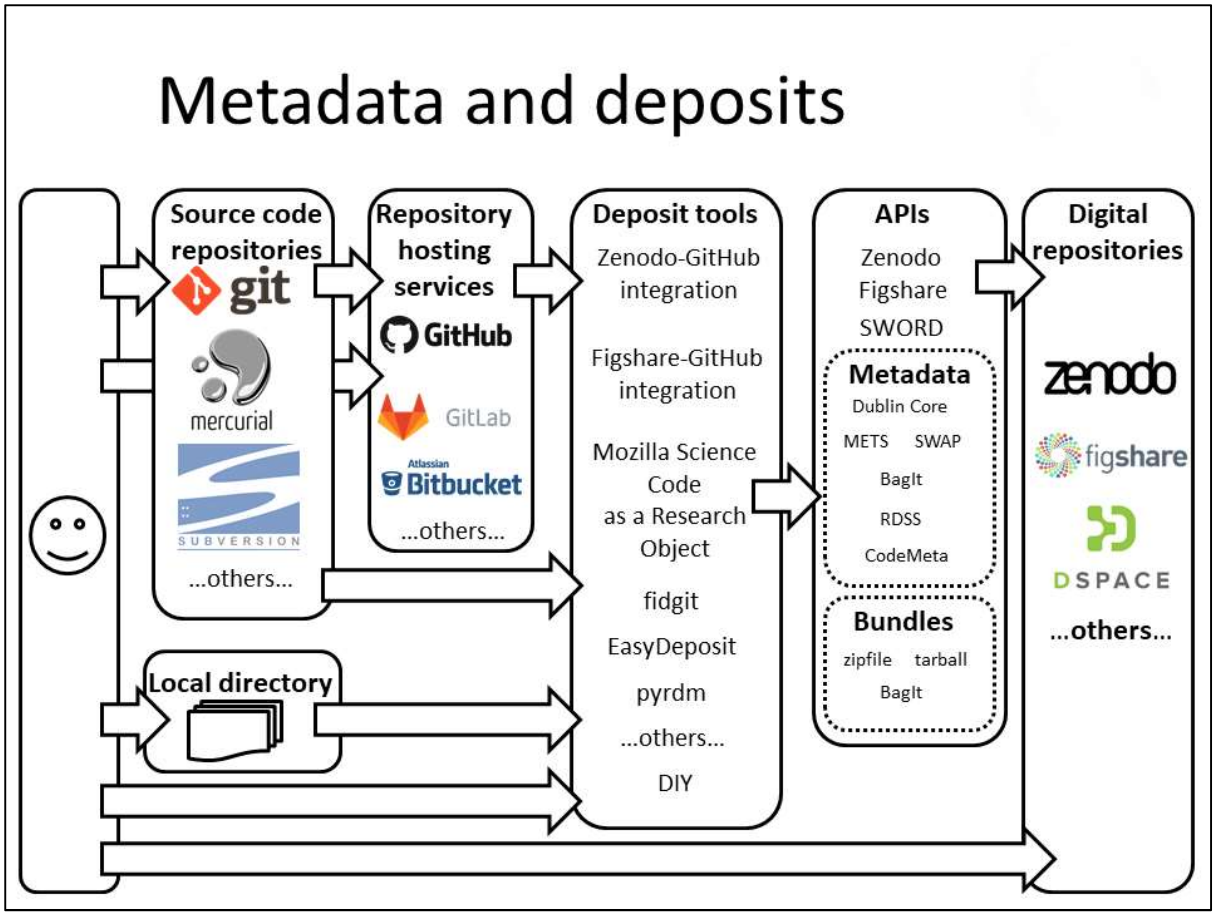


Within this workflow, a researcher requests that a deposit of their software be made. The deposit tool creates a zipfile/tarball of a folder containing the software, compiles associated metadata and makes the deposit into the digital repository.

This workflow was inspired by the previous workflow, but without the assumption that a researcher is using a source code repository.

3.7. Metadata

Metadata for software deposits can originate from myriad sources. These include: researchers who deposit software, directories with software, source code repositories, repository hosting services (including their APIs and events) and existing items in digital repositories (e.g. for deposits corresponding to a previous version of the software).



3.8. Deposit tool configuration

Configuration required by deposit tools to support each workflow is summarised in Table 2: Workflow configuration.

	Manual	Event-driven	Interval-driven	Manually-triggered (hosted)	Manually-triggered (local)	Manually-triggered (directory)
Repository hosting service connection		Y	Y	Y		
Source code repository URL		Y	Y	Y		
Source code repository directory					Y	
Source code directory						Y
Digital repository connection		Y	Y	Y	Y	Y
ID of digital repository item to update		Y	Y	Y	Y	Y
Tag pattern e.g. `[0-9]+(\.[0-9]+)*` (optional)		Y	Y			
Default metadata (optional, digital repository-specific, researcher and/or deposit		Y	Y	Y	Y	Y

tool could provide defaults)						
Query interval		Y				

Table 2: Workflow configuration

3.9. Pros and cons

The pros and cons of each workflow can be summarised as in Table 3: Workflows pros and cons.

	Manual	Event-driven	Interval-driven	Manually-triggered (hosted)	Manually-triggered (local)	Manually-triggered (directory)
Deposit made exactly when researcher wants	Y			Y	Y	Y
Deposit fully automated once configured		Y	Y			
Researcher does not have to enable webhooks	Y		Y	Y	Y	Y
Researcher does not need to use repository hosting service	Y				Y	Y
Many deposits could be triggered, consuming archive resources and possibly violating "acceptable use" or other terms of service. Researcher needs to be aware of these.		Y	Y			
Researcher must add/edit metadata	Y					
Researcher may need to add/edit metadata		Y	Y	Y	Y	Y
Researcher may have to create item before initial deposit		Y	Y	Y	Y	Y
Researcher must upload zipfile/tarball	Y					
Data files or temporary files might be deposited	Y					Y

Table 3: Workflows pros and cons

4. A review of the software deposit workflows

The following observations relating to software deposit workflows arose during the workshop.

4.1. Terminology

The term “repository” can mean different things to researchers, research software engineers (RSEs), software developers, research data managers or librarians. This motivated the inclusion of section 1.1 Terminology in this report.

4.2. Encouraging software deposit

Encouraging researchers to deposit their software into a digital repository is a fundamental challenge.

There is not always a requirement, from institutions, publishers or funders, to deposit software or software descriptions, as supplementary material for research publications such as journal papers.

Software is often not part of any peer review process.

Principal investigators can be very protective of their ideas and may be reluctant to deposit research software for others to access.

Software should be deposited not only for preservation, but also for reuse.

It can be challenging to encouraging deposits for reuse, if it is unclear that the software will be reused.

Some communities will be more willing to deposit software, and use a specific workflow, than others. It depends upon both context and project.

Digital preservation needs to be integrated into software development processes from the outset. It is very hard to apply as an after-thought.

4.3. Comparing workflows

Researchers can publish their research outputs in many different digital repositories.

Even within a single institution, a whole range of processes and tools (e.g. source code repositories or repository hosting services) will be used, often siloed from other parts of same institution.

Different workflows are required for different institutions or communities and for different projects, within the same institution or community.

There will always be a need, and desire, for the manual workflow, as it represents the “lowest common denominator”. There are a lot of researchers who work within this manual world. For example, a case study for Cambridge University is software that can arrive on a USB flash drive, as part of the associated materials for a PhD thesis.

A disadvantage of the manual workflow is that researchers might not deposit their software, because this workflow requires the most action from them.

Deposits from a local repository or directory could be considered the option with least overhead.

A deposit from a local directory may incur the risk of (possibly large or large numbers of) data files, temporary files, or scratch files, being inadvertently included as part of a deposit.

Interval- or event-driven workflows can be termed “automatic harvesting”.

Event-driven workflows can help researchers move away from manual deposit.

For bigger projects, interval- or event-driven workflows may be preferable.

An institutional digital repository might impose limits on deposits (e.g. number or size) and either block deposits that exceed these, or charge researchers if they exceed their limit. If using an interval- or event-driven workflow, a researcher might get a bill they did not expect.

Digital repositories could be modified to prompt a researcher to publish, or provide a reference to, software, if the researcher deposits a data set with analysis results.

4.4. When is software ready for deposit

Researchers may not understand the concepts of software releases and release management. They may lack the skills or knowledge to assess whether their software is in a suitable state for release, or deposit. This is complicated by the constantly evolving nature of software.

4.5. Combining content from different sources

A software deposit that forms a single usable bundle, or intellectual entity, might be a combination of files from different locations e.g. code on GitHub plus some local scripts in a folder. A more advanced version of the manually triggered workflows would be to allow content from multiple sources to be pulled together into a single deposit. An alternative is to create multiple, related, deposits.

4.6. Source code repositories

Many workflows assume the use of a source code repository. However, getting researchers to use a source code repository is a significant challenge. Many researchers do not use them or do not see their value.

Using a source code repository could help to encourage researchers to deposit their software once the initial configuration of a deposit tool is done. However, researchers still need to make the effort to do this initial set up.

4.7. Deposit dates versus publication dates

The choice of both workflow, and when a deposit is published, can be affected by when a researcher wants to publish their software. For example, a researcher may want deposits to be done routinely during their project. Alternatively, they may want to delay publishing their software until they have published results, derived from the software, in a related paper.

There is a distinction between the time when software is deposited into a digital repository and the time when it is published.

Researchers should be made aware of the embargo features of digital repositories.

Deposit tools need to allow researchers to configure any embargo-related features of the digital repositories with which they interact.

4.8. Copyright, IP and licensing

Not all software can be assumed to have an open source licence, or any licence at all.

Publishing source code in a digital repository does not imply that the software is open source.

Requiring researchers to select a licence within a deposit tool could ensure that all deposited software has a licence. However, mandating this selection places an additional burden upon the researcher.

Some digital repositories adopt a default licence (e.g. CC-BY-4.0 for Zenodo) unless explicitly overridden by a researcher.

It is unclear whether a researcher can be considered to have given their informed consent for a choice of licence if a licence is automatically selected by default.

A default licence may be more permissive or open than a researcher might prefer.

If the software deposit itself has a licence, then selection of a default licence by a digital repository can result in two, very possibly inconsistent, licences, the one within the software deposit and the one recorded within the metadata held by the digital repository.

A principal investigator, or another stakeholder, may need to decide upon a licence, not the developer of the research software (for example, if a research software engineer is temporarily working with a research group) or the researcher who is responsible for depositing the software. It is essential that all stakeholders agree on how the software is to be published, and under what licence(s), before a deposit workflow is adopted.

4.9. Additional source code repositories, repository hosting services and digital repositories

In addition to Git, Mercurial, Subversion, GitHub, GitLab and BitBucket other source code management systems are also used, or mandated, by universities. For example, researchers using Microsoft Visual Studio Team Services can use either Git or Team Foundation Version Control.

CCPForge is a repository hosting service offered by Jisc, STFC[32] and EPSRC[33]. CCPForge can host Git, Subversion and CVS source code repositories.

Institutions can have their own services, for example the University of Edinburgh's DataShare[34], DataVault[35], currently under development by the Universities of Edinburgh and Manchester, or institution-specific Microsoft SharePoint[36] services.

4.10. Reviewing deposits

Depending upon the workflow, there are three points at which a software deposit could be reviewed:

- Prior to deposit e.g. by a researcher, their peers, or a research software engineer within their project or institution.
- Post-deposit but pre-publication e.g. by a digital repository manager or a research software engineer.
- Post-publication e.g. by the research community.

[32] STFC, http://www.stfc.ac.uk/
[33] EPSRC, https://www.epsrc.ac.uk
[34] Edinburgh DataShare, https://datashare.is.ed.ac.uk
[35] DataVault, https://github.com/DataVault/datavault
[36] Microsoft SharePoint, https://products.office.com/en-gb/sharepoint/collaboration

A review could range from a simple visual inspection, following a checklist, through to an attempt to build and run the software.

Automated testing and continuous integration could afford some form of automated checking of deposits.

The nature and degree to which a software deposit can be reviewed depends upon the expertise, time and effort available.

4.11. Post-deposit activities

Ease of maintenance depends on the quality of the software and its related documentation.

Both researchers and digital repositories may want to remove deposits that are deemed to be no longer worth keeping. For example, if a software deposit has a major bug it might be worth removing that deposit, replacing it with a “tombstone” page explaining why it has been removed.

It may be desirable to keep only specific versions of software that enable reuse and are considered to have long-term value.

To promote reuse, digital repositories could support a Docker-type preview of the research software they host, so a researcher can see if some research software may be useful to them.

5. Software deposits and documentation

The following observations relating to the nature and content of a software deposit arose during the workshop.

5.1. How a deposit will be used

Software should be deposited not only for preservation, but also for reuse.

Carole Goble, in "Reproducibility, Research Objects and Reality"[37], slide 23, provides definitions for 5 "Rs" of research: rerun, repeat, replicate, reproduce, reuse.

Three common scenarios for the use of software deposits are:

- Look at the source code to see how data is generated or to spot bugs. This is a minimum requirement.
- Rerun the software to validate research findings i.e. check whether using the same software on the same data produces the same results.
- Rerun the software with different data i.e. reuse the same software on different data.

Software that is executable today may not be executable in the future. Language features are deprecated and removed, libraries become unavailable, HPC services reach their end date and are shut down etc. Consequently, a software deposit that can be used for reproducibility and reuse today, may only serve as a historical record in future.

5.2. Deposit content

A deposit could consist of any of:

- Source code.
- Build scripts e.g. Makefiles.
- Binary executables.
- Container/image files e.g. Docker[38], Singularity[39], with a runtime environment and dependencies.
- Virtual machine images, with a runtime environment and dependencies.
- Ansible[40] playbooks.
- Sample input and output data.
- etc.

A software deposit should have, or be complemented with:

- Licence, which might not necessarily be an open source licence.
- README.
- Documentation.
- etc.

[37] Goble, C. (2016), Reproducibility, Research Objects and Reality, Leiden Bioscience Lecture, 24 November 2016, https://www.slideshare.net/carolegoble/reproducibility-research-objects-and-reality-leiden-2016
[38] Docker, https://www.docker.com/
[39] Singularity, http://singularity.lbl.gov/
[40] Ansible, https://www.ansible.com/

A software deposit should not contain IP addresses, usernames, passwords or personal or sensitive information or data that the researcher does not want to be exposed.

5.3. Documentation

Documentation is important for reproducibility.

It is not enough to publish software, there is a need to complement with information on how to build and deploy a running instance of the software.

A significant challenge is that many researchers struggle with the basics of all aspects of software development e.g. using source code repositories, documenting code, and managing releases. Researchers may not have the time, effort, skills or expertise to perform these activities. Consequently, documentation is often incomplete or inaccurate. The problem is not how to describe software, it's how to make it easy for researchers who develop, and use, research software to provide these descriptions.

Researchers should document, for both reproducibility and reuse:

- Runtime environment e.g. operating system, hardware.
- Configuration e.g. parameters, settings, command-line options.
- Input and output formats.
- Sample inputs and expected outputs/errors.
- Third-party dependencies e.g. libraries, software, infrastructure, services.
- How to build/create a running instance of the software.
- Skills needed to install, use, inspect, develop the software.

See, for example, the documentation for ffmprovisr[41]

Common Workflow Language (CWL)[42] and GA4GH Task Execution Schema (TES)[43] both offer ways to more formally represent software, processes, inputs, outputs and the relationships between these. CWL can represent inputs, outputs, parameters, processes and command-line tools. TES can represent inputs, outputs, processes and resources.

Documentation should include a point-of-contact, in case a researcher experiences problems in trying to use the software.

Research software can depend upon other software, services and infrastructure to which access by other researchers may not be readily available. For example, proprietary software (e.g. Matlab or Autodesk), embedded software (e.g. within instruments), online services (Software-as-a-Service), university, community and national HPC services, virtual digital environments and science gateways. Researchers should record these dependencies too. However, it is not always clear how this could be done. For example, a service can evolve over time, unknown to the researchers using it. In such circumstances, the notion of a "version" becomes problematic.

The FORCE 11[44] Software citation principles[45] are of value when describing dependencies.

---

[41] ffmprovisr, https://amiaopensource.github.io/ffmprovisr/
[42] Common Workflow Language (CWL), http://www.commonwl.org/
[43] GA4GH Task Execution Schema (TES), https://github.com/ga4gh/task-execution-schemas
[44] FORCE 11, https://www.force11.org/
[45] 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. https://peerj.com/articles/cs-86/

Templates and lists of important information may help to deliver improved documentation.

Documentation that is machine-readable could contribute to automated metadata extraction, but development of templates for such documentation would take more time.

6. Software metadata

The following observations relating to software metadata arose during the workshop.

6.1. General

As for documentation (see section 5.3 Documentation), the problem is not how to describe software, it's how to make it easy for researchers who develop, and use, research software to provide these descriptions.

Different researchers working in different fields may have different views as to what metadata is important.

To mint a digital object identifier (DOI)[46], a digital repository (or human agent such as a librarian) usually requires a certain minimum set of metadata to be completed.

What should be mandatory and what can be optional can vary, depending upon what the metadata is required for and the context in which it is to be used. For example, metadata for software discovery may differ from metadata for software credit. As another example, a researcher making or checking a citation, a researcher trying to replicate research results, and a researcher looking for research software to use for a specific purpose all have different metadata requirements.

The semantics of metadata is important, especially when it can be defined by individual researchers or projects. For example, topics attached to Git source code repositories on GitHub might, for one researcher, be used to describe the software and what it does, for another they might describe the readiness of the code e.g. alpha, beta, or production. Similarly, for tag names used within Git, Mercurial or Subversion repositories.

Certain metadata is immutable or static. For example: depositor, date, version, operating system, programming language, authors) at the point of deposit.

Other metadata is mutable or evolves over time. For example: descriptions, keywords, related references, citations, applicable domains, tested platforms.

Some metadata fields may be more sensitive than others and might need to be access-restricted or controlled. It could be possible to infer commercially sensitive information from certain metadata.

6.2. CodeMeta

CodeMeta[47] is a schema specifically designed to "assist the transfer of software and software metadata between the entities that author, archive, index and distribute and use the software". It is based on schema.org[48] terms, and adds terms for software that have no equivalents in schema.org. CodeMeta is complemented with cross-walks to other metadata schemas. CodeMeta could be used as a means to preserve metadata that would otherwise be lost when a deposit passes through APIs (as mentioned in section 2.2 Metadata limitations).

[46] Digital Object Identifier System, https://www.doi.org/
[47] The CodeMeta Project, https://codemeta.github.io/
[48] schema.org, https://schema.org

6.3. Collecting metadata

6.3.1. Observational metadata

Making a wrong assertion about the value of a piece of metadata, is worse than not making any assertion at all.

Observational metadata views metadata not as assertions, but as observations, statements made by some agent at a specific point in time.

PREMIS[49] and PROV[50] are metadata schemas which enable observational metadata to be captured and represented.

Metadata should be captured as observational metadata. For example, if an email address is taken from GitHub's API by a deposit tool, the metadata can record not just the email address, but that it originated from GitHub's API, was captured by the deposit tool, and the date when this occurred.

Like several digital repositories or digital preservation systems, Archivematica can derive some metadata from deposits themselves. For example, for an audio file, how long it is. However, Archivematica also captures the source of such metadata, Archivematica itself in this example.

6.3.2. Approaches to collecting metadata

Calcyte[5152] is a work-in-progress tool that supports the capture and management of metadata for data files. Researchers record their metadata in an Excel spreadsheet. Calcyte converts this into an RDF-A[53] metadata schema, supported by schema.org, which can be embedded into HTML web pages so it can be crawled by search engines. It bundles static HTML repositories for distribution via BagIt[54] in a format called a Data Crate[55]. One possible strength of this approach is that spreadsheets are very accessible and familiar to all.

Technical metadata (e.g. programming language) can be derived automatically, but not always correctly.

Components that harvest metadata could use algorithms to extract metadata by analysing files (e.g. README and LICENCE files) or using Natural Language Processing (NLP), for example to extract keywords from descriptions. The Digital Public Library of America[56] uses these techniques.

There is no one best way to collect metadata. Metadata can come from different, possibly multiple, sources.

6.3.3. Metadata sources

Metadata that could be provided by researchers who write software includes:

- Author

---

[49] PREMIS, Library of Congress, http://www.loc.gov/standards/premis/
[50] PROV, W3C, https://www.w3.org/TR/prov-overview/
[51] Sefton, P. (2017) Calcyte: A simple tool for describing, packaging and publishing data collections. eResearch Australasia, October 2017. https://conference.eresearch.edu.au/2017/08/calcyte-a-simple-tool-for-describing-packaging-and-publishing-data-collections/
[52] Calcyte, https://codeine.research.uts.edu.au/eresearch/calcyte/
[53] RDF-A, W3C, https://www.w3.org/TR/rdfa-primer/
[54] The BagIt File Packaging Format (V0.97), https://tools.ietf.org/html/draft-kunze-bagit-14
[55] Research Data Crate, https://github.com/UTS-eResearch/datacrate
[56] Digital Public Library of America (DPLA), https://dp.la/

- Date
- Version
- Licence
- Description
- Documentation, or location thereof.
- Link to source code repository.
- Platform.
- Dependencies and requirements.
- Subject or area of research.
- Optional descriptive metadata, possibly specified using CodeMeta.

Metadata that could be automatically derived includes:

- Author
- Date
- Version
- Contributors
- Description
- Language
- Size
- Test coverage
- Quality-assurance
- Benchmarks
- README file location.
- Licence
- Virus checking results.

Metadata that can be provided by users of the software:

- Does the software work.
- Their opinion as to the quality, utility, and usability of the software.

6.3.4. Validating collected metadata

Automatically derived metadata might not be correct.

Researchers need to check metadata, especially that which has been automatically derived.

For an institutional digital repository, it is important that a human agent checks the metadata, for example, a librarian or other repository officer.

Metadata guidance for researchers can also serve as guidance for librarians or repository officers.

6.4. Metadata for software preservation

There is significant overlap between documentation, discussed earlier, and metadata for software preservation. A lot of information that should be within documentation e.g. languages, operating systems, dependencies, input and output formats etc. can also serve as valuable metadata.

PREMIS 3[57], p33, states that "The only mandatory semantic units that apply to all categories of Object (Intellectual Entity, Representation, File, and Bitstream) are "objectIdentifier" and "objectCategory"". PREMIS can serve as a starting point for metadata but it is inadequate for enabling software reuse.

At a minimum, the following metadata needs to be collected for preservation:

- What the software is.
- What it is for.
- Creator
- Rights and licensing.

It is not unreasonable to expect researchers to provide such descriptive metadata. However, the quality may vary.

Rights and licensing are important to allow other researchers to know whether they have the right to retain and edit the software to keep it usable.

There is a need for a software equivalent to DataCite[58].

Many schemas and ontologies for software metadata, and resources from which such an equivalent schema and ontology could be derived or expanded, already exist. These include:

- Common Workflow Language (CWL).
- GA4GH Task Execution Schema (TES).
- CodeMeta.
- The Software Sustainability Institute's Software Evaluation: Criteria-based Assessment[59].
- The Software Sustainability Institute's Checklist for a Software Management Plan[60].
- STFC's The Significant Properties of Software: A Study[61].
- researchobject.org[62]'s Research Object ontology[63], for describing workflow-centric research objects.

6.5. Metadata for linkage with other research objects

Software does not necessarily sit alone from other digital content e.g. software might be a script that works alongside processing large collection of images etc.

Research objects related to software can include:

---

[57] PREMIS Editorial Committee (2015) PREMIS Data Dictionary for Preservation Metadata version 3.0, June 2015. http://www.loc.gov/standards/premis/v3/premis-3-0-final.pdf

[58] DataCite, https://www.datacite.org/

[59] Jackson, M., Crouch, S. and Baxter, R. (2011) Software Evaluation: Criteria-based Assessment. The Software Sustainability Institute, November 2011. https://www.software.ac.uk/software-evaluation-guide

[60] The Software Sustainability Institute. (2016). Checklist for a Software Management Plan. v0.1. Available online: https://www.software.ac.uk/software-management-plans.

[61] Matthews, B., McIlwrath, B., Giaretta, D., and Conway, E. (2008) The Significant Properties of Software: A Study, STFC, March 2008. https://www.webarchive.org.uk/wayback/archive/20100624233431/http://www.jisc.ac.uk/media/documents/programmes/preservation/spsoftware_report_redacted.pdf

[62] researchobject.org, http://www.researchobject.org

[63] Soiland-Reyes, S. and Bechhofer, S. (eds.) (2016) Research Object ontology, 1.0.0-SNAPSHOT, 28 January 2016, http://wf4ever.github.io/ro/2016-01-28/

- Publications
- Data, both input and output.
- Data management plans, for the above data.
- Software management plans, for the software.
- Other research software.
- Experimental observation equipment.
- Publicly funded facilities and services.

To link research objects to software requires the following metadata:

- For research software:
    - Authors
    - Location
    - Date
    - Version e.g. unique identifier and/or timestamp.
- For relationships to other research object:
    - Nature/semantics of relationship to other research object.
    - Type of other research object.
    - Version (unique identifier/timestamp).

Taken together, research software, data, facilities, equipment and an overarching research question can be viewed as a research activity or experiment, worthy to be published.

Conversely, a publication can be considered as a narrative that describes how the research objects are used together to reply to the research question.

Establishment of a research object relationship semantics, neither too broad nor too narrow, is essential.

The RDSS canonical data model[64] defines a “RelationType” with relations between research objects including, but not limited to, “cites”, “isCitedBy”, “references”, “isReferencedBy”, “continues”, “isContinuedBy”, “documents”, “isDocumentedBy”, “isDerivedFrom” etc.

6.6. Metadata for software discovery

Software should be catalogued for discovery and reuse.

The following metadata can contribute to assisting the discovery of research software:

- Deposit date.
- Version
- Contributors, which may not just be those who wrote the actual code.
- Subject-specific keywords.
- Relevant identifiers e.g. organisation identifiers.
- Algorithms implemented within the software, to allow for the discovery of more generic functionality which could be used elsewhere. For example, another researcher may be interested in the implementation of a specific algorithm within the software, rather than the software itself.

[64] JiscRDSS Data model documentation store, https://github.com/JiscRDSS/rdss-canonical-data-model/, v3.0.0, 88658d7b6aa32a4696b6d0d34ea905daa186c37b, 14/03/18.

- References/citations to the use of the software elsewhere e.g. in conjunction with or within other software.
- Number of downloads. This can be used as an indicator of quality.
- Source code e.g. statements which import or include modules, packages or libraries.

6.7. Metadata for research credit

Citation is still *the* currency in academia.

Mandatory metadata for research credit, with an indication of the ease or otherwise of capturing this metadata, could include:

- Identities of the contributors (Easy).
- Funders, represented as unambiguous identifiers (preferably ORCiD[65]) (Easy).
- DOI, that both:
    - Resolve in such a way that the software can be found (Easy).
    - Unambiguously identify a version of the software, and thereby the contributors (Easy).
- Licence (Easy).

Optional metadata could include:

- A distinctive, memorable name (Easy).
- Contributors' affiliations at time of publication (Easy). This should be derivable from an appropriately populated ORCiD record.
- Nature of contributors' roles e.g. Project manager, Developer, Maintainer, User support etc. (Easy)
- Degree of contribution (Difficult). This could be determined by the number of commits to the source code repository, the number of lines of code changed, influence on design or quality etc.
- Component(s) contributed to (Difficult).
- Associations with relevant journal articles i.e. papers that have used the software. This is a transitive link. (Easy).

There is a risk of DOI proliferation.

Overarching DOIs which exist for the lifetime of a piece of software, with all contributors, but which are associated with individual DOIs for specific versions, can help to preserve the relationship between different versions of software that has been deposited.

A CONTRIBUTOR file could include contributor names with the weights of their contributions.

However, automated credit assignment is unreliable. This is true for a binary "is a contributor" assignment and, especially, for contribution weights.

Alternatively, contributors could be assigned to one or more contribution areas from a defined project credit taxonomy.

A citation can be viewed as an endorsement, or as a criticism, depending upon the rationale for the citation.

[65] ORCiD, https://orcid.org/

The ability to alter or revoke both contributors or associations with other research objects may be desirable for practical, ethical or other reasons.

The original authors of long-lived, highly influential software could find their contribution “diluted” within an ever-growing list of contributors.

It may be appropriate for an institutional RSE service to be acknowledged instead of, or in addition to, individuals. It may also be appropriate for an institutional RSE service to request such acknowledgement.

If research software has been developed by an institution’s central funds, rather than via research grants, recognition on the institution’s research management system may even be more important than authorship of or acknowledgment within a published paper.

One can debate whether there is a role for “software” at all or whether the primary entities are research conclusions (presented in papers) and the people who delivered these (lab scientists, software developers etc).

Many of these issues are not necessarily unique to research software.

7. Best practice development and adoption

From the foregoing, many challenges were identified. These include:

- How to encourage researchers to deposit their software in the first place.
- There is not always a requirement, from institutions, publishers or funders, to deposit software or software descriptions, as supplementary material for research publications such as journal papers.
- Software is often not part of any peer review process.
- Researchers can be very protective of their ideas and may not want to share their research software out of a concern for being "scooped" or losing opportunities for commercial exploitation.
- Many researchers struggle with the basics of all aspects of software development e.g. using source code repositories, documenting software, and managing releases.
- Researchers may not have the time, effort, skills or expertise to perform these activities and may lack the skills or knowledge to assess whether their software is in a suitable state for release or deposit.
- Digital preservation needs to be integrated into software development processes from the outset. It is very hard to apply as an after-thought.

The workshop attendees discussed the development of best practices to address these challenges, and how to achieve adoption in communities. The most important observations were that:

- Researchers need to be encouraged to use source code repositories to more effectively document their software, and to deposit their software.
- There is a need to understand what would make researcher's life easier in terms of preparing software for deposit, documenting and describing their software, and depositing their software.
- Researchers need to want to adopt best practices and, for this to happen, there need to be clear benefits for them to do so e.g. time saved, greater impact, additional funding, and enhanced reputation.
- Best practices need to be community-driven.
- Guidance for stakeholders e.g. researchers, research data managers, institutions and publishers, needs to speak their respective languages, not require significant time or effort to apply, and be free of developer jargon.
- Research software development needs to be cost-effective for all stakeholders. Demonstrating that best practices can reduce costs for all stakeholders may help stakeholders to either adopt those best practices and incentivise or mandate them.

7.1. Delivering best practice and guidance

RSEs, RDMs, repository managers, research IT teams, research administration and support services and infrastructure managers within institutions, and funders and publishers can all play a part in promoting and delivering best practice and providing guidance.

Institutions should be a researcher's first port of call. However, relationships with an institution are often temporary, so best practices that a researcher can readily take with them and apply at other institutions are highly desirable.

RSEs and RDMs not always aligned. A stronger alignment between RSEs and RDMs, between research software development and archiving and preservation, could help to deliver a more unified

and consistent view of research software best practices to researchers. For example, well documented software with a licence, can provide all the information needed to complete metadata fields when the software comes to be deposited.

RSEs may be best placed to mediate between researchers and RDMs.

Researchers could be encouraged to produce a software management plan early on. This could help RSEs to identify what guidance a specific researcher will need and at what time.

Best practices and guidance needs to be promoted and delivered by trusted, independent authorities for information dissemination and who understand local factors affecting researchers.

OpenAIRE and Zenodo, for example, could support RDMs.

7.2. Incentives

Promoting best practices in themselves may not be sufficient. If adopting a best practice does not affect impact, there is less, if any, incentive for researchers to implement it.

If the benefits of adopting best practices can be easily identified and assessed by researchers, then this can drive their adoption. For example, researchers may be more open to depositing research software in digital repositories if they know that if such deposited software, citable and with a DOI, can be submitted as a REF-able research output.

Institutions, funders, publishers, and government should all be encouraged to recognise the value of research software as a first-class citizen of research.

Digital repositories should be encouraged to treat research software as distinct class of research output (not just a type of data), complemented with software-specific metadata.

Evidence of impact for software, that should be considered by institutions, funders and government, for example as part of REF, could include:

- Number of citations for a DOI for software, and not just a related paper that describes or mentions that software.
- Number of downloads.
- Exploitation and reuse within the institution (for both teaching and research), research community, within other research communities, and within industry.
- Royalties received or licences sold.
- Commercialisation e.g. spin-outs.

Other financial or reputational incentives for recognising the adoption of best practices by researchers can include:

- Digital badges.
- Financial incentives or prizes.
- Free passes to future conferences.
- Discount in article processing charges (APCs) for a future submission/publication.
- Altmetric[66] statistics.
- Social media or press coverage etc.

[66] Altmetric, https://www.altmetric.com/

The issuer of incentives such as digital badges needs an agent both with the authority to assess whether a badge should be issued and who is widely recognised. For example, the Software Sustainability Institute could issue badges, but there is a question as to whether the Institute is widely recognised.

Badges could be issued by digital repositories automatically. For example, pages for software deposits onto Zenodo using the Zenodo-GitHub plugin are marked with “Available in GitHub”, which implies that the researcher is both using a source code repository and a repository hosting service.

There is a question as to who receives recognition, an individual researcher, their institution, or both. For example, researchers can take papers with them to other organisations.

7.3. Mandates

Institutions, funders and publishers often mandate the deposit of, and long-term access to, research data.

The RDSS project is driven by funders having requirements for publication in open access digital repositories and to make underlying data publicly available.

Institutional policies often reflect mandates from funding bodies. However, there is a need for consistent policies across institutions, as researchers can move between institutions.

Institutions, funders and publishers could mandate the deposit of, and long-term access to, research software.

Publishers can mandate that software be provided as supplementary information, if the software underpins the published research. Publishers need to be convinced of the merits of this.

If deposits are to be mandated, then there need to be checks for compliance to these policies. Appropriate penalties for non-compliance are also required.

7.4. Recommendations

Recommendations for future activities to develop and encourage best practices are as follows.

7.4.1. Best practice and guidance

Provide guidance on software deposit and deposit workflows, supported by currently maintained deposit tools. This should include the pros and cons of deposit workflows, which digital repositories support embargoes, and how these can be used.

Provide guidance on what should be included in a source code software deposit. This should include how check source code, documentation and other files to ensure no personal or secure information is inadvertently exposed.

Provide guidance how to document a source code software deposit, focusing on what needs to be provided to promote both reproducibility and reuse.

Provide guidance on software licence types and which licences are suitable for which purposes e.g. reproducibility, reuse, protecting intellectual property etc. This should not be restricted to solely open source licences.

Provide guidance on how to describe dependencies including third-party software, libraries, embedded software, online services, HPC platforms, virtual digital environments and science gateways.

Develop short guides on software management plans, data management plans and how they complement each other. Consider contributing these as Software/Data/Library Carpentry lesson.

Provide guidance on methods for researchers to gather information on the use of their software, especially in a non-academic context, to be able to demonstrate its impact. This should include licences that require attribution to be given; CITATION files; restricting access to and use of research software via accounts or keys; number of downloads, forks or clones; community surveys; web searches etc.

Provide guidance on what should be included in other types of software deposit e.g. containers or virtual images. This should include how check source code, documentation and other files to ensure no personal or secure information is inadvertently exposed.

Provide guidance how to document other types of software deposit e.g. containers or virtual images, focusing on what needs to be provided to promote both reproducibility and reuse.

Develop a guide on "4 easy ways to boost your impact via software deposition", drawing upon the foregoing, which can be completed by researchers within a few hours. Consider contributing this as a Software/Data/Library Carpentry lesson.

Provide guidance on the circumstances in which a deposit corresponding to a specific version of research software could be "demoted" or removed.

Provide guidance on what types of deposit (source code, binaries, containers, images) effectively serve what purposes (rerun, repeat, replicate, reproduce, reuse) and the pros and cons of each type of deposit.

Provide guidance on how software deposits can best be reviewed and quality assured, pre-publication, pre-deposit and post-deposit. This should include when the software is reviewed, who performs the review, how it is reviewed, what review criteria can and should be used, to what extent it can be reviewed, and the pros and cons of various approaches. A review could range from a simple visual inspection, following a checklist, through to an attempt to build and run the software.

Provide guidance on how software deposits can be maintained in the short-, medium- and long-term. This should include whether maintenance should be done at all, who performs the maintenance, the nature of the maintenance, what resources within or related to the software are needed to maintain it, how this maintenance is funded, and the pros and cons of various approaches.

Develop a software-equivalent of the FAIR[67] data principles.

Provide support to deliver Software/Data/Library Carpentry lessons, mentioned above, to anyone who currently delivers these workshops e.g. RSEs, librarians, and RDMs.

7.4.2. Schemas, ontologies, specifications, templates

Develop mappings from GitHub and other repository hosting service APIs to the RDSS data model. This could possibly be done using via CodeMeta crosswalks between, for example, GitHub's API and CodeMeta and CodeMeta and the RDSS data model.

[67] Wilkinson, M. D., M. Dumontier, I. J. Aalbersberg, G. Appleton, M. Axton, A. Baak, N. Blomberg, et al. 2016. "The FAIR Guiding Principles for scientific data management and stewardship." Scientific Data 3 (1): 160018. doi:10.1038/sdata.2016.18. http://dx.doi.org/10.1038/sdata.2016.18.

Develop standard formats for README files and related files (e.g. LICENCE, CONTRIBUTORS) that allow for metadata to be automatically extracted.

Develop shareable, machine-readable, descriptions of software, infrastructures and services.

Develop, or customise an existing, research object relationship semantics for expressing linkages between research software to other research objects.

Develop a project credit taxonomy to express contributions to research software.

7.4.3. Policy

Lobby funders and government to recognise software as a first-class research output and to adopt ways of assessing the impact of research software that are best suited to software.

7.4.4. Research

Survey what researchers needs are and evaluate who the stakeholders are. A Sustainable Software Sustainability workshop run by Digital Archiving and Networked Services (DANS) and the Software Sustainability Institute[68], identified the following stakeholders: researchers, software developers, institutes, funders, industry and innovation centres, publishers, and the cultural (heritage) sectors.

Investigate how users of research software could be involved in validating its metadata.

Investigate whether, and how, source code itself should, and could, be made searchable, via a digital repository API, to aid discovery, or whether searches should be restricted to over metadata only.

Investigate whether if software is submitted to an institutional, subject area or general digital repository, whether information required for preservation be collected and the deposit automatically submitted to a central digital preservation system. Similarly, if an associated software paper exists, investigate whether information be gathered on submission to enable automatic submission to the central digital preservation system on acceptance or publication. This would help to reduce duplicated effort in gathering information related to deposits.

From a publishers' perspective, there is a desire for solutions for the permanent archival of software associated with published papers. Investigate Software Heritage[69] and other digital preservation systems as possible options.

7.4.5. Deposit tools

Develop deposit tool(s) that support multiple workflows.

Prototype a deposit tool that supports licence selection.

Prototype a deposit tool that supports embargo specification.

Prototype a deposit tool that captures observational metadata from researchers, directories, source code repositories and repository hosting services using PROV or PERMIS.

Test deposit tool(s) with Jisc-RDSS services.

---

[68] Aerts, Dr. (PhD) P.J.C. (DANS) (2017): Sustainable Software Sustainability – Workshop report. DANS. https://doi.org/10.17026/dans-xfe-rn2w

[69] Software Heritage, https://www.softwareheritage.org/