V1.0 for community review

Software Source Code Identification

Use cases and identifier schemes for persistent software source code identification

Editor: Morane Gruenpeter

Authors (in alphabetical order):

- Alice Allen, Astronomy Source Code Library & U. Maryland, USA
- Anita Bandrowski - University of California San Diego, USA
- Peter Chan - Stanford University Libraries, California, USA
- Roberto Di Cosmo - Software Heritage, Inria and University of Paris, France
- Martin Fenner - DataCite, Germany
- Leyla Garcia - ZB MED Information Centre for Life Sciences
- Morane Gruenpeter - Inria, Software Heritage, France
- Catherine M Jones - UKRI STFC, UK
- Daniel S. Katz - University of Illinois at Urbana-Champaign, USA
- John Kunze - California Digital Library, University of California, USA
- Moritz Schubotz - swMATH, FIZ Karlsruhe, Germany
- Ilian T. Todorov - UKRI STFC Daresbury Laboratory, UK
- And the participants of the SCID WG (listed in Appendix B)

Contents

Introduction

Software, and in particular source code, plays an important role in both industrial and academic research: it is used in all research fields to produce, transform and analyse research data, to simulate and understand natural phenomena, and is sometimes itself an object of research and/or an output of research (Clément-Fontaine, 2019).

Unlike research data and scientific articles, though, software source code has only very recently been recognised as important subject matter in a few initiatives (e.g CoSO[1], EOSC[2], FAIRsFAIR[3], FORCE11[4], Freya[5], Software Heritage[6], SSI[7], WSSSPE[8], Society of RSE[9], ReSA[10], URSSI[11] and more) related to scholarly publication and archiving (Abramatic et al., 2018). These initiatives are now working on a variety of plans for handling the identification of software artifacts.

At the same time, unlike research data and scientific articles, the overwhelming majority of software source code is developed and used outside the academic world, in industry and in developer communities where software is routinely either not formally identified or referenced at all, or is identified and referenced, in practice, through methods that are totally different from the ones typically used in scholarly publications.

The objective of the Software Source Code Identification Working Group (SCID WG) is to bring together a broad panel of stakeholders directly involved in software identification.

In this document, with inputs from a broad panel of stakeholders, we document the current ***state-of-the-art practice in software identification***, including use cases and identifier schemes from different academic domains and industry, clarifying and harmonizing the usage of different identifiers. We hope that this will provide solid ground on which to build recommendations for the academic community, and help academic and industrial stakeholders to adopt solutions compatible with each other and especially with the software development practice of tens of millions of developers worldwide.

---

[1] https://www.ouvrirlascience.fr/the-committee-for-open-science/
[2] https://www.eosc-portal.eu/
[3] https://www.fairsfair.eu/
[4] https://www.force11.org/
[5] https://www.project-freya.eu/
[6] https://www.softwareheritage.org/
[7] https://www.software.ac.uk/
[8] http://wssspe.researchcomputing.org.uk/
[9] https://society-rse.org/ - formerly an association - https://rse.ac.uk/about/ - leading to a larger network - https://researchsoftware.org/
[10] http://www.researchsoft.org/
[11] http://urssi.us/

About the Software Source Code Identification WG

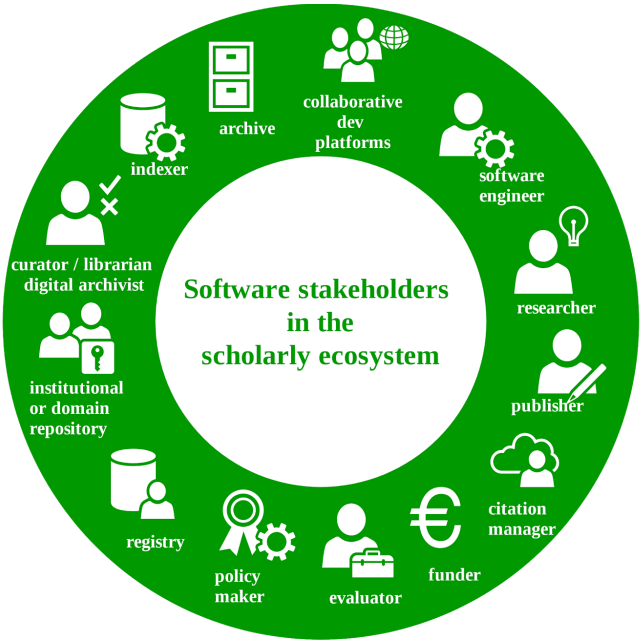
The SCID WG was spawned at RDA P11 in Berlin from the RDA Software Source Code IG and the FORCE11 Software Citation Implementation WG, as both had identified the challenge of software source code identification in the scholarly ecosystem.
These groups decided to create a joint working group under RDA and FORCE11 to involve a larger audience and to have a broder panel of stakeholders discussing the challenges and solutions for identification use cases and the different identifiers that are used for software.
The group's co-chairs are Roberto Di Cosmo, Martin Fenner and Daniel S. Katz.
It was endorsed by RDA's TAB in October 2018 and kicked-off its activity at RDA P13 in Philadelphia in April 2019. At first, a survey capturing the state of the art in software source code identification was sent to the WG.
In October 2019, during the FORCE2019 Full day hackathon on research software in Edinburgh, one of the parallel activities was on hacking the identifiers granularity levels.
In March 2020 at RDA VP15, the group held a session on the use cases and identifiers schemes.
For more information, see the group's web pages at https://www.rd-alliance.org/groups/software-source-code-identification-wg and the group's repository at https://github.com/force11/force11-rda-scidwg
Here are the links to the WG activity by chronological order:

- P13 slides:
  - https://www.rd-alliance.org/system/files/documents/2019-04-03_RDA-WG.handout.pdf
  - ASCL.net making codes discoverable for 20 years https://www.rd-alliance.org/system/files/documents/RDAPlenary13_SW_IGWGv2_1.pdf
- State of the art survey: https://docs.google.com/forms/d/e/1FAIpQLScRoCoJK1E3GDqRSjeirZiBbsL6T-xDi4A9NJuvTAgeqEwmAg/viewform?usp=pp_url
- FORCE2019 hackathon notes:
  -
- VP15
  - slides: https://docs.google.com/presentation/d/1Z7tbpnWn_HyES2pxAyVIeNfCDmPBF_zbxcJ71oYnxD0/edit?usp=sharing
  - notes: https://docs.google.com/document/d/1C3Q-O0FHg9pbVDH35olfhFTEGaZ1nnebBoFg_hX1xC8/edit?usp=sharing

Definitions

Actors in the scholarly ecosystem

In this section, we provide a full list of the actors regarding the software artifact, specifying the actor's role, and including examples. Some of the actors were specified in the software citation principles (Smith et al., 2016). This list is in alphabetical order.



Archive

Refers to organizations or initiatives aiming to preserve human knowledge and particularly, in our case, the preservation of software source code. They need not be limited to a specific institution or domain. e.g. SWH, Zenodo.

Citation manager

Refers to organizations or people creating services or tools for citation management
e.g. Zotero, Mendeley, EndNote.

Collaborative dev. platforms

Refers to platforms where collaborative software development is possible on public or private repositories. There is no distinction in the version control technology used (git, mercurial, svn, etc.) e.g. GitHub, Gitlab, Bitbucket.

Curator / librarian / digital archivist

Refers to the people moderating and curating research artifacts or software artifacts in archives, institutional repositories, or libraries.
e.g. the HAL moderators described in (Di Cosmo et al., 2019)

Funder

Refers to organizations or people funding research using software or directly funding software. Academic software projects tend to involve support from funding agencies. Funding Agencies also evaluate the projects and research they are funding.
E.g NSF, NIH, or Wellcome Trust

Indexer

Refers to a research engine, a service or a person building a catalog and providing access to the aggregated collection of data regarding links between scholarly research outputs, including papers, data, and software. One main part of the indexer is adding subject classification and disambiguation improving findability of software items.
e.g Scopus, Web of Science, Google Scholar, and Microsoft Academic Search.

Institution, research center or university

Refers to organizations employing researchers. Sometimes these organizations are the copyright holders of the research outputs. These organizations evaluate the researchers and research artifacts under their supervision.
e.g MIT, ENS, Inria

Institutional, national or domain repository

Refers to a digital archive collecting and preserving the copies of the intellectual outputs of a specific institution or domain[12]
e.g institutional repository, HAL,

Library

Refers to an organization that holds a curated collection of resources. Libraries can provide emulation services, enabling access and reuse of legacy software. For this report, we will refer only to libraries collecting software source code.
e.g Stanford Library, Bibliothèque National de France (BNF)

Package manager

Refers to a collection of software tools that are publicly available on an accessible online platform that facilitates software installation, configuration, upgrade or removal[13].
e.g PyPi, NPM

---

[12] Institutional repository, https://en.wikipedia.org/w/index.php?title=Institutional_repository&oldid=960627744 (last visited June 10, 2020).
[13] Package manager, https://en.wikipedia.org/w/index.php?title=Package_manager&oldid=953745584 (last visited Apr. 30, 2020).

Policy maker

Refers to people and organizations in charge of institutional, national or international policies. e.g institutional committees, the European Commission, National research commissions or organizations.

Publisher or publication venue

Refers to scholarly publishers disseminating research outputs (articles, data, software or any other digital object) after peer review. Including journals, conferences, or other named "collections" created by defined groups under the guidance and rules of a publisher.
It includes journals (e.g JOSS), conferences with artifact evaluation committees (e.g POPL)

Registry

Refers to an organization providing an online catalog of items usually stored elsewhere by others. Each catalog item describes a software project with a set of metadata.
e.g ASCL, SwMath, SciCrunch, Wikidata

Researcher as a software user (RSU)

We have separated researchers into two categories. This one is for researchers who use software without taking part in its creation. A researcher can be in both categories in different situations. Both refer to researchers at all career stages, including students, postdocs, staff researchers, tenure-track faculty members, professors and non-academic researchers.

Researcher as a software author (RSA)

Refers to researchers in all stages of the researcher's career as stated above, participating in the creation of software. A software author can be a research software engineer (RSE), but this category of actors isn't exclusively for RSEs. A software author may have contributed in one or more roles identified in (Alliez et al., 2020):

- Design
- Architecture
- Debugging
- Maintenance
- Coding
- Documentation
- Testing
- Support
- Management

Software engineer

Refers to people that take part in the software creation endeavor and can take one of the roles in the RSA category, without being researchers.

What do we want to identify or the granularity of software?

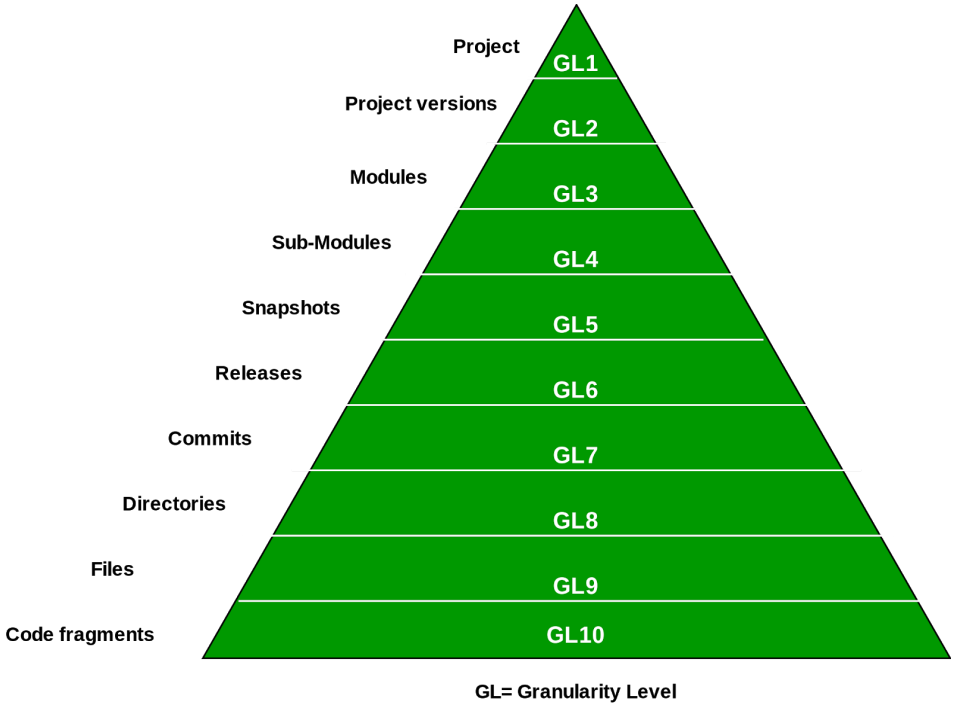
The first question that comes up when revisiting granularity is, what is software?
From the Cambridge dictionary :
Software is the [set of] instructions that control what a computer does; computer programs: the programs that you put into a computer to make it do particular jobs:

Yet this definition focuses on the software executable, understandable by the machine. In research, the emphasis should be on the source code, which is readable by humans and captures the human knowledge (Abramatic et al., 2018).

This is why the scope of this WG is source code and we will not address use cases intended for the usage of the executables or use cases that make use of proprietary software, because we do not have access to the source code.

Identification target

Before reviewing different technologies in academia or in industry that are used to identify software artifacts, it is necessary to break down the different meanings behind "software" and specify the exact target of identification.
In (Jones et al. 2016), clarifying which item is being identified is important since software is complex and evolving.
Furthermore the landscape of software projects is vast with different structures, lifetime evolution, communities and more (Aliiez et al., 2019). When it comes to structure, some projects are monolithic and some can be a composition of modules. A proposition to decompose the source code of a software project into different levels of granularity, is the basis to agree upon the identifier which could be used with that item. FIrst we need to agree on the terminology and on the granularity level each target represents. Note, since different software structures exist, not all have all levels of granularity.



To do so we will use a granularity level scale from 1 to 10, where 1 is the most global element and where 10 is the smallest element. Granularity level 1, abbreviated as GL1, is coarse grained, while granularity level 10, abbreviated as GL10, is fine-grained.

Software project

Granularity level: **coarse-grained** (GL1)

A software project can be found on a dedicated landing page or registry on which the project, as a whole is described. This page gives access to all software modules, versions and download links to source code or executables. A software project can be sometimes referred to as a software collection or a software concept (Katz et al., 2019).
In some cases, identifying the project is needed without any specificity of a sub-module or a version, for example, when an institution identifies the software developed by its researchers. At this granularity level it is complicated to identify the source code without identifying a very long list of artifacts, this is why an extrinsic identifier is recommended.
The software project can be represented as a metadata record in a registry.
A software project can also have the notion of version at a coarse-grained granularity level, e.g Python 2 and Python 3 (GL2).

Software module

Granularity level: **medium-grained** (GL3)

A specific module of a larger software project or collection. Modules were introduced in the late 1960s, also called assembly or package, referring to a software architecture separating functionalities into smaller interchangeable pieces[14]. A software project can be very complex with many modules (GL3) and even sub-modules(GL4) that can be written and used separately.

Software version

Granularity level: **fine-grained** (GL5)

The software artifact is always versioned, this is why the target is a software version. It can have many instantiations for different environments, but for this document we will only distinguish two forms:

1. Executable (e.g download link): we will not address this target in the rest of the document
2. Software source code

[14] Wikipedia contributors. (2020, April 6). Modular programming. In *Wikipedia, The Free Encyclopedia*. Retrieved 15:33, May 19, 2020, from https://en.wikipedia.org/w/index.php?title=Modular_programming&oldid=949451352

- *Dynamic source code or current version* - The source code can have a dynamic representation on a collaborative development platform (a.k.a the Github/ Gitlab/ Bitbucket repository). On which the development history is also presented.
- *Snapshot*- a capture of the complete situation in a software repository, including branches, releases and all the development history. This archived artifact is specific to Software Heritage and provides access to the complete archived copy of the development history of a project. (GL5)
- *Release* - a specific version can be shared as a release on a package manager or as a tar file on a website (GL6)
- *Commit / a specific point in development history* - in a version control system, this is the mechanism that captures the modifications in each iteration during the software development. Each commit is signed by the author. (GL7)
- *Directory* - a static version of the source code without the evolution of the development history. Usually what institutional repositories, libraries and Zenodo collects.(GL8)
- *File* - a static file in a directory. (GL9)
- *Code fragment* - an implemented algorithm or function represented in a few lines of code in a static file. (GL10)

Software context

- Complementary artifacts
  - Software artifacts that are external to the source code (build scripts, run scripts, test cases, etc.)
  - Documentation (which is external to the source code)
  - the software environment (can be a Docker image or other emulation solutions)
  - *Data* (input/output data)
  - tutorial (Jupyter notebook)
  - software images (screenshots)

- *Reference publication* - The article describing the software and source code
  Examples:
  - the IPOL Journal · Image Processing On Line [15] publishes articles that contain a peer reviewed software artifact and demo.
  - SwMath has the **standard article** property identifying the publication describing the software.
- Documentation (reference manual, build instructions, API calls, README file, etc.)

[15] https://www.ipol.im/

What is at stake

It is important to clearly identify the different concerns that come into play when addressing software, and in particular its source code, as a research output, that can be classified as follows:

Archival

- ensure (research) software artifacts *are not lost;* they must be properly archived, to ensure we can retrieve them at a later time

Reference

- ensure (research) software artifacts *can be precisely identified;* software artifacts must be properly referenced to ensure we can identify the exact code, among many potentially archived copies, used for reproducing a specific experiment

Description

- make it easy to *discover* (research) software artifacts; they must be equipped with proper *metadata* to make it easy to find them in a catalog or through a search engine

Credit

- ensure *proper credit* is given *to authors and contributors* ; research software must be properly cited in research articles in order to give credit to all that contributed to it

As already pointed out in the literature, these are different and separate concerns. Establishing proper credit for contributors via citations or providing proper metadata to describe the artifacts requires a curation process (Allen & Schmidt, 2015; Alliez et al., 2020; Bönisch et al., 2012) and is way more complex than simply providing stable, intrinsic identifiers to reference a precise version of a software source code for reproducibility purposes (Alliez et al., 2020; Di Cosmo et al., 2020; Howison & Bullard, 2016). Also, as remarked in (Alliez et al., 2020; Hinsen, 2013), research software is often a thin layer on top of a large number of software dependencies that are developed and maintained outside of academia, so the usual approach based on institutional archives is not sufficient to cover all the software that is relevant for reproducibility of research.

Use cases

During the lifespan of the SCID WG we collected and analysed a number of use cases, with the actors and identification targets defined in this document. For each use case we have noted the facet (Archive, Reference, Describe or Cite). In this section we will list the complete collection of use cases with a very short summary. In Appendix A, we provide a set of analyzed use cases that emerged at the RDA VP15 session.

The use cases collection

Actor	**Use case description**	**Action**	**Identification target**
Archive	Identify all the software artifacts I hold	Archiving, referencing	Release and smaller artifacts
Citation manager	Curate the software citation entries	Credit	Project, release
Collaborative dev. platforms	Provide access to the most recent state of the software artifact in an online repository (e.g Gitlab, GitHub) and to its development history	Accessing	Dynamic VCS online copy
Curator / librarian / digital archivist	Catalog and browse the development history of legacy software source code for preservation purposes (The Apollo mission source code is a good scenario on how making code available on GitHub isn't enough for persistence purposes[16])	Archiving	Project, release and smaller artifacts depending on the reference
Data center	Identify the software tools we produce to support the use (e.g., reading, visualizing) of our data products.	Archiving, referencing	Archived copy, specifically release that represents the researcher's use of the tool/package.
Evaluator	Measure the importance of department X's contribution to software package Y, relative to other contributors.	Credit	Project, module and release, specifically

[16] https://www.softwareheritage.org/2019/07/20/archiving-and-referencing-the-apollo-source-code/

	Also classify the overall size (scope and complexity) of that package.		identifying the authors and contributors at
Funder	Track the software I funded and see if it was “published” and how it was used.	Referencing	Any item (all granularity levels)
Indexer	Classify software with metadata	Describing	Project
Institution, research center or university	Measure the impact of the software developed by us. Who is using this software?	Referencing	Any item (all granularity levels)
Institution, research center or university	Count the citations of the software. target: software release, particular version, organization	Referencing	Any item in the software project
Institutional or domain repository	Preserve software that is deposited with metadata	Archiving, describing	Release
Library	Collect,catalog, preserve software. Provide necessary environments (in virtual machines) to run the software.	Archiving, referencing, describing, providing environments to run the software	From project to release in the software project.
Package manager	Finding, installing, maintaining or uninstalling software packages, using a command to do so	Referencing	Release
Policy maker	Publish policies for research products including software	Referencing	Software project
Publisher	Create/retrieve identifiers quickly for use in the paper for all software including commercial packages.	Referencing, describing	Any item (all granularity levels)
Publisher	Add software source code or access to software source code that needs to be published along with a publication. For	Archiving, referencing, describing,	Any item (all granularity levels)

	example the journal IPOL[17] (Image processing online) publishes the software artifact and demo alongside the article. Recently IPOL started archiving the software artifacts in Software Heritage.	Credit	
Registry	Identify and curate the software entries I hold	Archiving, referencing, describing, credit	Project
Researcher as a software user (RSU)	Access and use SSC no longer available on a collaborative platform	Archiving	Snapshot, release, revision, directory
RSU	Reference SSC used in an article (McCaffrey algorithm in SageMath[18] detailed in this blog post[19])	Referencing	Any item (all granularity levels)
RSU	Search and find appropriate SSC using rich metadata	Describing	Project
RSU	Attribute to others their software contributions to publications (and have the skills/knowledge to do so).	Credit	Any item (all granularity levels)
Researcher as a software author (RSA)	Reproduce an experiment detailed in an article (replication studies)	Referencing	Release, revision, directory, file, fragment of code
RSA	Get (and give) credit for research SSC via correct citations to articles and data	Credit	Project
RSA	Find the publications that have used (and referenced) the software packages I wrote, so that I track the reuse of my work.	Referencing	Release

[17] https://www.ipol.im/
[18] swh:1:cnt:c60366bc03936eede6509b23307321faf1035e23;origin=https://github.com/sagemath/sage;lines=473-537
[19] https://msdn.microsoft.com/magazine/ee310028

RSA	Track how my software might relate to other software (as in versions or dependencies).	Referencing, Describing	Project, module, sub-module and release
RSA (team leader)	Ensure that my team members can get credit for their software development and that the group's output can be cited, re-used and associated with the group.	Credit	Project, module, sub-module and release
RSA	Contribute and improve existing SSC. As a software developer contributing to a large scale Open Source project I would like to have credit for the parts I contributed. Understand *how authorship will be managed* at the level of the overall project and how can I publish (e.g. in a software journal) my contributions to the overall project.	Credit	Project
Software engineer	Know if others are using my code, and whether they are giving me credit or they are just "copying" it (plagiarism). **Challenges**: here you need a reference corpus of source code, and sophisticated tools to track software contributors; there are tools for this in industry, and we are working on open source versions at Software Heritage.	Credit	Any item (all granularity levels)
Software engineer	Track the provenance of the tools I am re-using. In this way I can give credit to others, I know who contact to in case of doubts on the code I am re-using, and I know how that code is currently supported **Challenges**: - rapid evolution of software packages (incompatibility) - unique identification of software origin	Referencing, Credit	Any item (all granularity levels)

Identifiers schemes

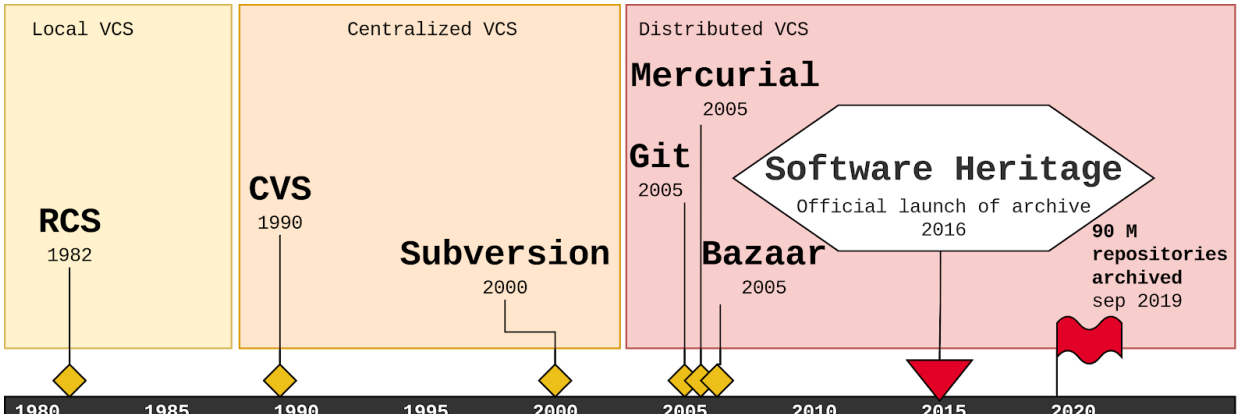
This section presents several identifier schemes that are used in different settings to designate software artifacts. Some of them are specifically designed for software artifacts, others are digital identifiers systems that can be used for any kind of object (not necessarily digital ones). Some of them rely on *intrinsic identifiers*, computed from the object itself, others rely on *extrinsic identifiers* which are not computed from the object itself, and a registry (centralised or distributed) that maintains the relationship between identifiers and objects.

We refer the interested reader to (Di Cosmo et al., 2018, 2020) for an extensive analysis of the properties of these two classes of schemes.

Intrinsic identifier schemes

Cryptographic Hashes in Distributed Version Control Systems

Version control system (VCS)[20] are essential tools in software development. They are used to control the evolution of a software project, by recording changes made to the source code files, usually called versions, and providing mechanisms to compare different versions, restore a previous version, and merge changes from multiple versions.



As shown in the figure above, version control systems have evolved a lot over the last decades, moving from simple tools that could only operate on a local machine, like RCS, to systems that relied on a central server to allow concurrent modifications to a large software project, like CVS or Subversion, and finally to Distributed Version Control Systems (DVCS), that enabled fully

[20] See https://en.wikipedia.org/wiki/Version_control for more information.

distributed software development, without relying on any central server[21]. Since the beginning of the 2000's, DVCS have grown extremely popular, in particular because of the broad adoption of Git.

In order to build DVCS, it was necessary to find a mechanism that allowed any peer in a distributed development network to identify in the exact same way the same state of the software project, *without depending on any registry*, For a single file, the solution was well known: a cryptographically strong hash can compute from any file a short "*signature*" that provides an *intrinsic* identifier for the file. The extra step needed was a way of getting intrinsic identifiers not only for a single file, but for a full project, with its completed directory structure. The key insight to do this comes from the seminal work of Ralph Merkle (Merkle, 1987), that showed how one could compute a single, strong cryptographic signature over a tree structure, by building what is now broadly known as a Merkle tree. This technology is now largely used not only in DVCS, but also in blockchains and distributed file systems.

The key point to retain from all this for the purpose of this report is the fact that today *tens of millions of software developers* use daily tools that rely on ***intrinsic identifiers*** for software projects computed along the principles of Merkle trees. These identifiers are often referred to as "*commit hashes*", but the notion is more generic than that, as not only commits have identifiers.

Here is an example:

- The Git identifier of the source code of the release 5.6 of the Linux kernel: **7111951b8d4973bda27ff663f2cf18b663d15b48** (this identifier can be used for example to access the copy of this source code on GitHub at https://github.com/torvalds/linux/tree/7111951b8d4973bda27ff663f2cf18b663d15b48)

These intrinsic identifiers are quite powerful, as they allow not only to identify an object, but also to verify that the designated object has not been modified: it suffices to recompute the intrinsic identifier from the object itself to spot any alteration, due to the strong cryptographic properties of the hash algorithm used (see for example (Di Cosmo et al., 2018) for more details).

Taken alone, though, such hashes do not allow knowing whether the designated artefact is a file, a directory, a commit or a release, nor what exact hashing algorithm has been used to compute them: we depend on external knowledge for that, for example on the fact of knowing that the identifier is used in the Git version control system.

A slightly more general and structured approach has been adopted for defining SWHIDs, that are described below.

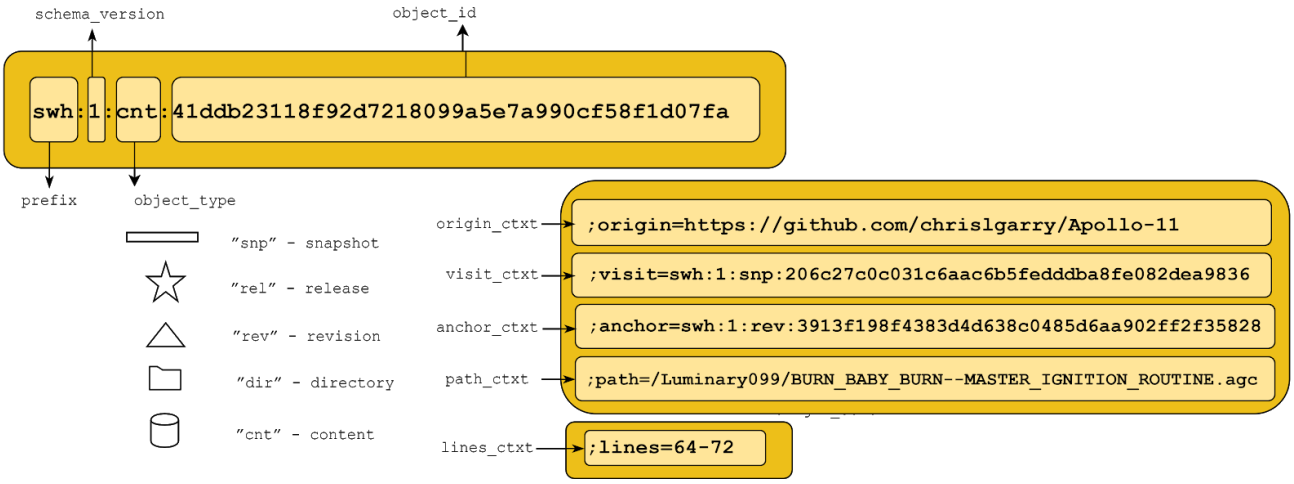
[21] One should not be misled by the popularity of platforms like GitHub, GitLab.com or Bitbucket: these platforms offer convenient facilities for developer interaction, but for the version control system point of view, they are just peers in a network of distributed nodes.

Software Heritage Identifiers (SWHID)

Software Heritage is a non profit multi-stakeholder initiative to build a universal archive of software source code, started in 2015 under Inria's impulsion, in partnership with UNESCO. The main goal is to ensure long term access to the source code of all software ever written that is publicly available. For that reason, the choice of the identifiers for the software artefacts contained in the archive was of paramount importance. It turned out that the key requirements were very similar to those identified by Distributed Version Control Systems, and naturally led to choosing *intrinsic identifiers* based on the same principles of Merkle tree signature. A full discussion of the motivations behind this choice can be found in (Di Cosmo et al., 2018, 2020).

One key difference between usual hashes used in DVCS and Software Heritage identifiers (SWHIDs) is the fact that they do not depend on the version control system, if any, used for maintaining a software artefact: a SWHID can be computed for any software artefact, even if it is distributed as a package, a zip file, or in any other form.

The full specification is available online. The structure of a SWHID is shown in the figure below.



SWHIDs are URIs, with a clearly defined prefix **swh** followed by the version of the hashing algorithm employed, and a **tag** that allows to identify the **type** of the object identified, and only then one finds the intrinsic software fingerprint. In version 1 of the schema this fingerprint is *fully compatible with **git** intrinsic identifiers*, a property which is extremely convenient for users of this popular version control system. Additional *qualifiers* may be used to provide rich contextual information about the object (or fragment of object) that is denoted by the identifier. See the official documentation for more details.

SWHIDs are supported by the following resolvers:

- archive.softwareheritage.org
- n2t.net
- Identifiers.org

SWHIDs are currently used in the following services:

- The HAL french national open access repository
- The swMATH.org registry of mathematical software
- Wikidata

Full guidelines are available to trigger archival of any publicly available software artefact, with a particular focus on use in the scholarly world to enrich research articles with SWHIDs that enhance the reproducibility of the published results: see Save and Reference guide[22].

Tools to verify and independently compute SWHIDs are also readily available, specifically the swh-identify module[23].

Here are a few examples of SWHIDs (they are clickable, and will be resolved directly), to simplify the SWHIDs visibility, only the identifier itself is shown, without the complementary context elements:

- **Snapshot**: a capture of the entire Darktable repository (4 May 2017, GitHub): including all branches, releases and development history up to this point in time

  swh:1:snp:c7c108084bc0bf3d81436bf980b46e98bd338453

- **Release**: version 2.3.0 of Darktable, dated 24 December 2016

  swh:1:rel:22ece559cc7cc2364edc5e5593d63ae8bd229f9f

- **Revision**: a commit in the development history of Darktable

  swh:1:rev:50d91bdfc94cb9d3aa01634ac0b003d76e799bf1

- **Directory**: a directory from the computer game Quake III Arena

  swh:1:dir:c6f07c2173a458d098de45d4c459a8f1916d900f

[22] https://www.softwareheritage.org/save-and-reference-research-software/
[23] swh:1:rev:b4fbdeb30f02ba3d428b372aef5b904cf2125221;origin=https://pypi.org/project/swh.model/;visit=swh:1:snp:fd30d9054acb716addee49506465bc5f8043c194

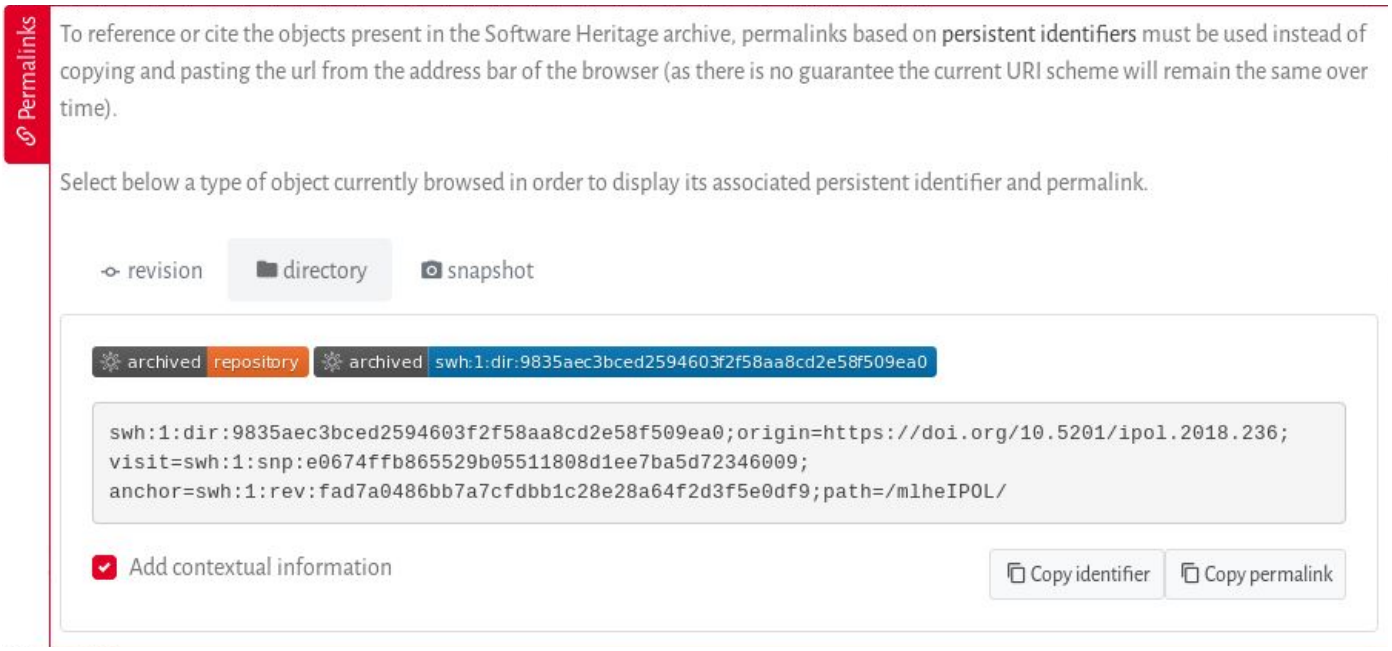
- **Content**: full text of the GPL3 license (which appears in many projects):

  swh:1:cnt:94a9ed024d3859793618152ea559a168bbcbb5e2

- **Code fragment**: Apollo 11 source code excerpt “Please crank the silly thing around” (here the additional *lines* parameter is visible, since it defines the start and end of the code fragment)

  swh:1:cnt:64582b78792cd6c2d67d35da5a11bb80886a6409;lines=245-261/

When you are browsing the Software Heritage archive, you can find on the right a permanent red tab called ‘Permalinks’ with the possibility to identify all the artifacts you are viewing with or without context qualifiers. The image below is a screenshot of the opened tab with the chosen directory identifier with context of an Ipol deposit[24]:



[24]https://archive.softwareheritage.org/swh:1:dir:9835aec3bced2594603f2f58aa8cd2e58f509ea0;origin=https://doi.org/10.5201/ipol.2018.236;visit=swh:1:snp:e0674ffb865529b05511808d1ee7ba5d72346009;anchor=swh:1:rev:fad7a0486bb7a7cfdbb1c28e28a64f2d3f5e0df9;path=/mlheIPOL//

Extrinsic identifier schemes

ARK (Archival Resource Key)

With no fees, 3.2 billion ARKs have been assigned by 615 institutions to things digital, physical, and abstract. Resolution is decentralized or, if the provider prefers, centralized via n2t.net.

Assigners choose the form of the identifier (for example, to match up with legacy SVN commit ids) or they generate new unique opaque strings. For the latter, they can opt for strings that are long but convenient (eg, generating UUIDs) or compact with check digits (eg, minting Noids).

Each ARK string becomes resolvable when registered with a redirection target URL or when an ancestor of the ARK is registered to point to a remote resolver. If the remote resolver can deal with the descendants, it permits one ARK to resolve to millions of descendant ARKs. This is called the “suffix passthrough” mechanism and is similar to PURL’s “partial redirect” mechanism. In this way ARKs may be registered either individually or with one ARK registration that can stand in for millions of ARKs.

An ARK Example

```
ark:/12345/b67c89d/sys/io/socket.py
```

where `12345` is the institution, `b67c89d` the overall thing, `/sys`, `/sys/io`, and `/sys/io/socket.py` are optional subthings, and `.py` is an optional variant qualifier.

An ARK such as `ark:/12345/f98g76` is best cited in actionable form, eg,

```
https://n2t.net/ark:/12345/f98g76
```

ARKs appear in the Data Citation Index, Wikipedia, Wikidata, ORCID profiles, and the Internet Archive.

ASCL-ID

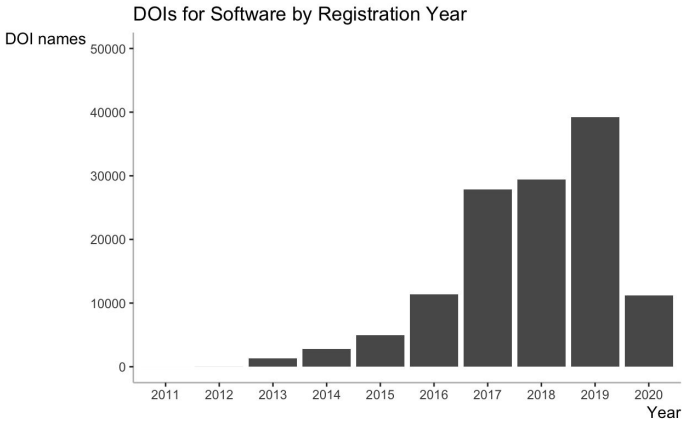
Astrophysics Source Code Library: Registry and repository for source code in astrophysics started in 1999.
Items registered by authors (or sometimes journal editors or users) or added by ASCL editors based on their appearing in the astrophysics literature and is assigned a unique ID.

Identifiers are **ascl:yymm.xxx**, where yy & mm are year & month of addition to ASCL, and xxx indicates that software was the xxx'th ASCL entry in the month

ASCL is indexed by the SAO/NASA Astrophysics Data System (ADS) and Web of Science; entries can be cited using their unique ASCL identifier
ASCL aims to improve the transparency, reproducibility, and falsifiability of research by making software source code discoverable and citable.
ASCL initially required code deposit but most software authors were reluctant to deposit code, because the repository didn't grow, ASCL dropped the requirement to deposit code, though it still accepts code deposits. Pointing to software download location is easily done and removes barriers to growth.
Metadata is regularly curated by an editor who actively performs curation through daily "random code" posting activity. The curation triggered by a link checker, "suggest a change," and the editor corresponds with the authors.
Site link curation
Links are checked with two link checkers, one twice weekly, the other continuously When links are consistently down for period of time, editor seeks new link Result: Links are consistently healthy; link health is reported twice weekly on public dashboard

DOI (Digital Object Identifier)

The DOI is a persistent identifier supporting standard citation metadata (title, authors, publication year, publication venue, etc.), and linking to other PIDs.
DOI registration is provided by currently eight DOI registration agencies, who coordinate their work via the DOI Foundation[25]. The DOI registration agency DataCite is the primary DOI registration agency for registering DOIs for software, DataCite has registered 128,276 DOIs for software as of March 26,2020. DOI registration agencies in turn work with publishers and repositories, in the case of software, the majority of DOIs (84%) have been registered via the Zenodo repository, which is offering a GitHub integration workflow for archiving and metadata registration since 2014[26].



Formal software citations using DataCite DOIs and metadata are still not common in the scholarly literature, but their number is increasing, and DataCite is collaborating with Crossref to exchange this information using the Crossref/DataCite Event Data service. As of July 2, 2020 this service has captured 5,219 software citations in the scholarly literature using DOIs.[27] One random example would be :

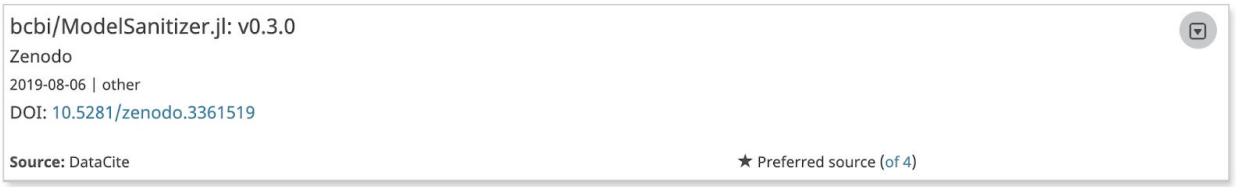
- Urai, A. E., de Gee, J. W., Tsetsos, K., & Donner, T. H. (2019). Choice history biases subsequent evidence accumulation. eLife, 8. https://doi.org/10.7554/elife.46331

citing the software :

- Urai, A. (2016). Motionenergy: Motion Energy V1.0. Zenodo. https://doi.org/10.5281/ZENODO.594505

DOIs not only align well with software citation workflows, but also support linking with other identifiers, for example the ORCID ID identifier for researchers. If the ORCID ID is included in the DataCite metadata of the software, and the researcher has given DataCite permission to do so, DataCite will automatically update the ORCID record of the researcher with the software record.

[25] https://www.doi.org/registration_agencies.html
[26] Fenner, M., Katz, D. S., Nielsen, L. H., & Smith, A. (2018, May 17). DOI Registrations for Software. https://doi.org/10.5438/1NMY-9902
[27] https://api.datacite.org/events?citation-type=ScholarlyArticle-SoftwareSourceCode



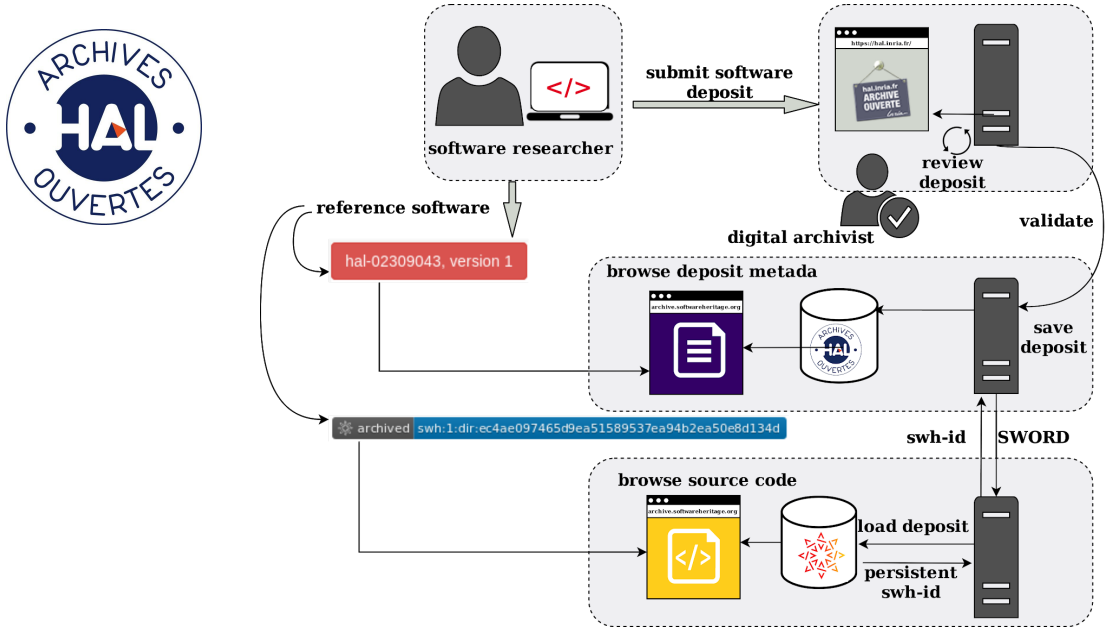
Software in ORCID record for https://orcid.org/0000-0002-9247-0530

The software can also be linked to other persistent identifiers, e.g. Crossref Funder IDs for funding, Research Organisation Registry IDs (ROR) for author affiliations, DataCite DOIs for datasets, and of course DataCite DOIs, SWHIDs, Arks or ASCL-IDs for other software.

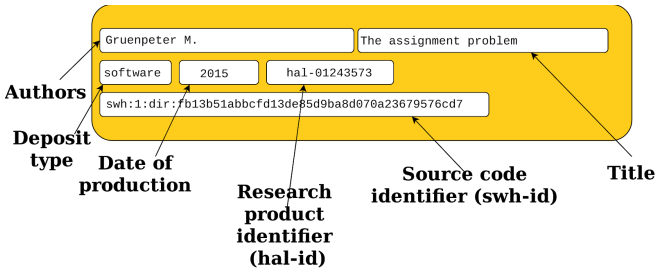
Another key use case for DataCite DOIs for software besides citing and referencing described above is discovery. The registration of standard metadata in a central registry simplifies discovery, and DataCite metadata includes a number of metadata fields that help with discovery, e.g. description, keywords or subject area. This information is not only available via DataCite APIs and the DataCite Search web interface, but also via the many aggregators who harvest DataCite metadata.

The HAL-ID

The French national open archive (HAL) is an open access repository, designed for the deposit of different types of research outputs to which a HAL-ID is assigned. The software source code deposit is possible from September 2018 on all HAL instances with transfer of the source code into Software Heritage (after the contributors validation).
The user can deposit easily a source code archive (in .zip or .tar.gz formats) alongside required metadata. For more information about the deposit process here is the deposit guide.
The HAL-ID is a persistent identifier to which each version gets a postfix v[x].
Software deposits also get a SWHID generated by Software Heritage after the source code is transferred. As stated in (Di Cosmo et al. 2019), the HAL-ID is a direct access to the metadata and answers the attribution use case of the research product, while the SWHID references the specific source code and answers the reproducibility use case.
In the diagram below, you can see how the deposit mechanism works for a software researcher.



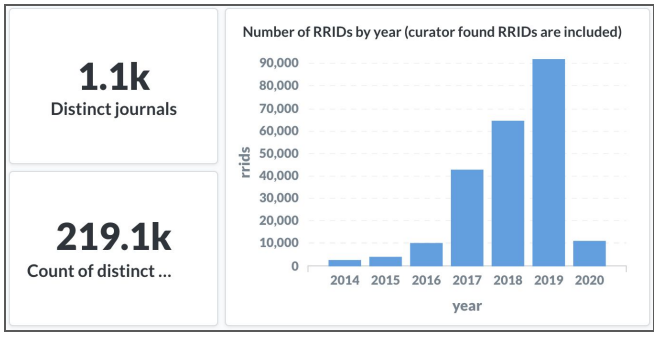
The main added value in the deposit process with HAL is including human moderation and curation of the content and specifically of the metadata associated with that content. We need to keep in mind that quality metadata is hard to come by with automated processes, as mentioned in (Alliez et al. 2019). Insuring quality metadata behind a registered identifier is key when it comes to giving credit to authors.
On the metadata record, a citation is suggested with both identifiers, one for the landing page on HAL and the other one for the actual content (the directory on Software Heritage).

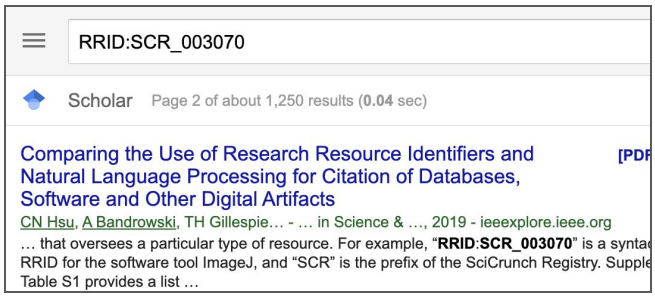


RRID

Research Resource Identifiers are used mostly in biomedicine, registered via SciCrunch: a system that aggregates ~25 RRID registries or repositories, such as the antibody registry, or Addgene repository.
The SciCrunch registry is a listing of low granularity (~GL1) software projects (e.g., SPSS, ImageJ), services (e.g., core facilities), and data projects (e.g., NeuroMorpoho.org) that may need to be cited as aggregate entities in the scientific literature.

RRID format: RRID:SCR_001622 (SCR = repository code, 001622 = local identifier)
Why register? Journals ask authors to do so, and RRIDs are part of MDAR & JATS (standards used by journals)
Where are they? Mainly in the methods section (most found within a table of research materials)
Usage: Started in 2014 with 25 journals, and continues to grow (current ~1000 journals (Most visible proponents: Cell, Nature, eLife), >20K papers, >200K RRIDs used by authors)





swMATH-ID

swMATH provides information on software referenced in mathematical publications. This *publication based approach* uses heuristics to detect software references in the zbMATH database. In a second step the heuristics results are checked by the human editor and complemented with false negatives. If a software was detected in the publication the relation between the software and the article is classified into one of the following two categories. Either it is an article that mentions the software, e.g., since it was used to derive results, or the article is a so-called standard article that describes the software or a significant modification of the software. If the article describes a new software (that did not exist in swMATH before) it will be added to the swMATH database. To do this the software is identified with a numeric Identifier, e.g., 825=SageMath

In addition the following metadata will be inserted by the editor:

- Authors
- Description of the software
- Links to the code
- Link to the homepage
- Keywords related to the software

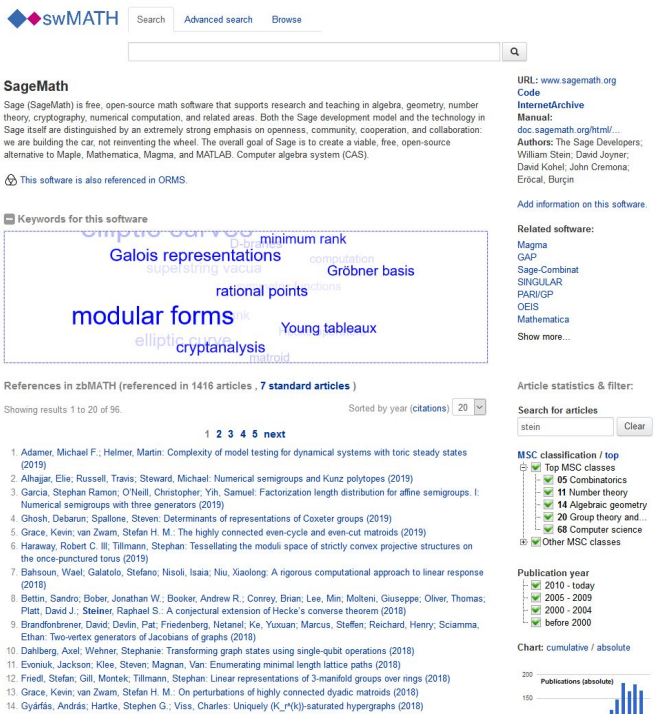
In addition the following information is derived from the publications related to the software:

- Classification of the software
- Information on citations in mathematical Publications

The dataset is manually curated and carefully checked using a test system that is released in a weekly schedule as a static snapshot. The release date is shown in the footer of every page on the swMATH.org homepage.

Currently the swMATH team performs an effort to establish back and forth linking with:

- Wikidata
- Software Heritage.



While swMATH provides automatically generated links to related software, the heuristics suffer from the typical drawbacks with machine learning approaches. To establish high quality well-defined links between software implementing the same algorithm, we are investigating options building an algorithm database that links between software, publication and algorithm[28]. Please refer to the scientific publications on swMATH for more technical details see (Bönisch et al. 2013, Chrapary et al. 2017 and Holzmann et al. 2016).

[28] https://opus4.kobv.de/opus4-zib/frontdoor/index/index/docId/7579

Wikidata entities

The wikidata entities are numeric identifiers prefixed with Q, e.g, Q1165184=SageMath. The Software class is identified by the entity Q7397 and each software entity is an *instant of (P31)* the software class or one of its sub-classes (like free and open-source software).

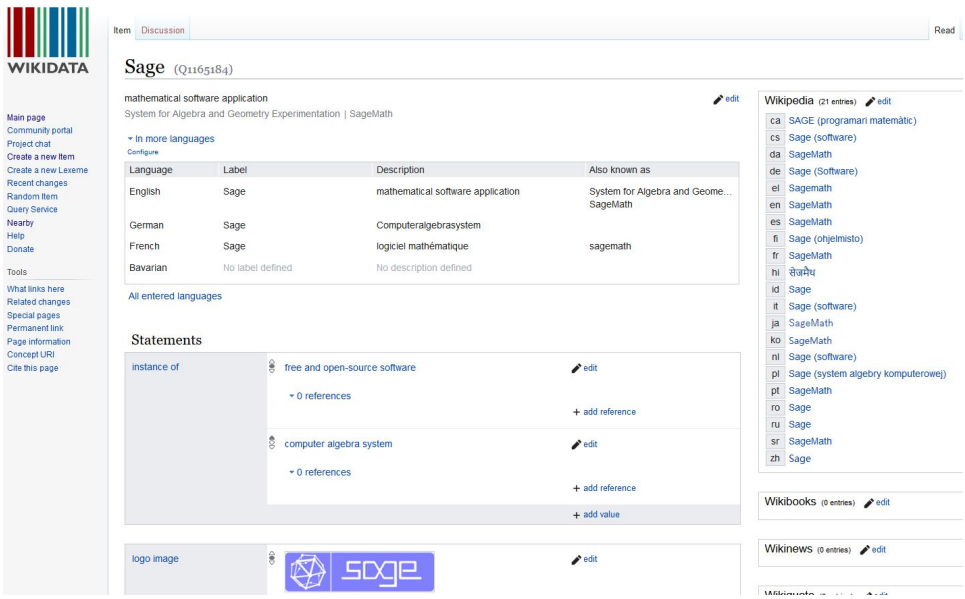
The information on the version of the software entity is maintained with the property *software version identifier* (P348).

An Identifier can be merged to remove duplicates

One important drawback is that Wikidata is open to editing by the community without the curation and supervision of an authority. A possibility to circumvent this drawback would be maintaining a local Wikibase, which will provide a controlled access environment and flexible modeling.

There are 3401 “external” identifiers in Wikidata[29]. Amongst these identifiers, you can find the following for software:

- Arch package,
- Debian stable package,
- Fedora package,
- Free Software Directory entry
- Freebase,
- Gentoo package,
- Open Hub,
- Quora topic,
- Ubuntu Package,
- swMATH work ID,
- SWH release ID,
- and many more ….



[29] https://www.wikidata.org/wiki/Wikidata:Identifiers retrieved on July 2nd 2020

Summary of findings

It appears clearly from the discussion that there is not a single solution that fits all use cases. To clarify the challenge,

Granularity level (GL)	ID target	Extrinsic identifiers									Intrinsic identifiers	
		ASCL	***ARK***	***DOI***	***HAL***	***URL***	***RRID***	***SwMath***	***Wikidata***		***Hash***	***SWH***
									entity	*property*		
GL1	project	X	X		X	X	X	X	X			
GL2	project version		X						X			
GL3	module		X						X			
GL4	repository		X			X				X		
GL5	repository snapshot		X							X		X
GL6	release		X	X						X	X	X
GL7	commit		X							X	X	X
GL8	directory		X		X*					X		X
GL9	file		X									X
GL10	Code fragment		X									X

* The HAL-ID when combined with a SWHID can identify also the directory of the source code in its metadata

Conclusion

The SCID WG was launched to resolve a crucial matter in citation: which identifier to use? After the work of the FORCE11 Software Citation WG introducing the Software Citation Principles (Smith et al. 2016), it was clear that unique identification, persistence and specificity are important for citation, but there is still a gap between the principles and the reality of the current state of the art of software identification. During community discussions we agreed that we need a consensus on terminology and use cases before producing concrete recommendations on identifiers.

We have come to a consensus on naming the stakeholders and identification targets. Decomposing the software as a concept to smaller identifiable digital artifacts using a scale for the granularity level of the digital artifact, which helped in the specification and analysis of the use cases. We have shown a large panorama of identifiers schemes: both extrinsic and intrinsic identifiers and designated the identification targets they can cater.

Lastly we have summarized the findings in a complete table matching identifiers schemes to identification targets, which emphasizes the difficulty to use one identifier for all use cases. By doing so, we can conclude that a strategy of combining multiple identifiers to cover all the facets of software is needed to answer the software citation predicament, especially if we wish a citation to capture the fundamental use cases (discoverability, access, persistence, reproducibility and reuse).

The next step would be to produce a set of recommendations based on these findings.

Bibliography

Abramatic, J.-F., Di Cosmo, R., & Zacchiroli, S. (2018). Building the Universal Archive of Source Code. *Commun. ACM*, *61*(10), 29–31. https://doi.org/10.1145/3183558

Allen, A., & Schmidt, J. (2015). Looking Before Leaping : Creating a Software Registry. *Journal of Open Research Software*, *3*(e15). http://dx.doi.org/10.5334/jors.bv

Alliez, P., Di Cosmo, R., Guedj, B., Girault, A., Hacid, M.-S., Legrand, A., & Rougier, N. (2020). Attributing and Referencing (Research) Software : Best Practices and Outlook From Inria. *Computing in Science Engineering*, *22*(1), 39‑52. https://doi.org/10.1109/MCSE.2019.2949413

Bönisch, S., Brickenstein, M., Greuel, G.-M., & Sperber, W. (2012). SwMATH – citations for your mathematical software. *journalId:00006143*, *2012*. https://doi.org/10.1007/BF03345852

Bönisch, S., Brickenstein, M., Chrapary H., Greuel G.-M., & Sperber W.(2013). “SwMATH – A New Information Service for Mathematical Software.” In Intelligent Computer Mathematics, edited by Jacques Carette, David Aspinall, Christoph Lange, Petr Sojka, and Wolfgang Windsteiger, 7961:369–73. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-39320-4_31.

Chrapary, H., Dalitz, W., Neun, W., & Sperber W. (2017). Design, Concepts, and State of the Art of the swMATH Service. *Math.Comput.Sci.* 11, 469–481. https://doi.org/10.1007/s11786-017-0305-5

Clément-Fontaine, Mélanie, Roberto Di Cosmo, Bastien Guerry, Patrick MOREAU, and François Pellegrini. (2019). Encouraging a Wider Usage of Software Derived from Research. Research Report. Committee for Open Science’s Free Software and Open Source Project Group. https://hal.archives-ouvertes.fr/hal-02545142

Di Cosmo, R., Gruenpeter, M., & Zacchiroli, S. (2018, septembre). Identifiers for Digital Objects : The Case of Software Source Code Preservation. *Proceedings of the 15th International Conference on Digital Preservation, iPRES 2018, Boston, USA*. https://doi.org/10.17605/OSF.IO/KDE56

Di Cosmo R., Gruenpeter M., Marmol B., Monteil A., Romary L., Sadowska J. (2019, December) Curated Archiving of Research Software Artifacts : lessons learned from the French open archive (HAL). 2019.https://hal.archives-ouvertes.fr/hal-02475835

Di Cosmo, R., Gruenpeter, M., & Zacchiroli, S. (2020). Referencing Source Code Artifacts : A Separate Concern in Software Citation. *Computing in Science & Engineering*. https://doi.org/10.1109/MCSE.2019.2963148 https://hal.archives-ouvertes.fr/hal-02446202

Greuel, Gert-Martin, and Wolfram Sperber. (2014). "SwMATH – An Information Service for Mathematical Software." In Mathematical Software – ICMS 2014, edited by Hoon Hong and Chee Yap, 8592:691–701. Lecture Notes in Computer Science. Berlin, Heidelberg: *Springer Berlin Heidelberg*. https://doi.org/10.1007/978-3-662-44199-2_103.

Hinsen, K. (2013). Software Development for Reproducible Research. *Computing in Science and Engineering*, *15*(4), 60–63. https://doi.org/10.1109/MCSE.2013.91

Holzmann, H., Runnwerth, M., & Sperber W. (2016). "Linking Mathematical Software in Web Archives." In Mathematical Software – ICMS 2016, edited by Gert-Martin Greuel, Thorsten Koch, Peter Paule, and Andrew Sommese, 9725:419–22. *Lecture Notes in Computer Science. Cham: Springer International Publishing.* https://doi.org/10.1007/978-3-319-42432-3_52

Howison, J., & Bullard, J. (2016). Software in the scientific literature : Problems with seeing, finding, and using software mentioned in the biology literature. *Journal of the Association for Information Science and Technology*, *67*(9), 2137‑2155.

https://doi.org/10.1002/asi.23538

Jones, C. M., Matthews, B., Gent, I., Griffin, T., & Tedds, J. (2017). Persistent identification and citation of software. *International Journal of Digital Curation*. Vol 11 Iss 2 104-114. https://doi.org/10.2218/ijdc.v11i2.422

Katz, D. S., Bouquin, D., Hong, N. P. C., Hausman, J., et al. (2019). Software citation implementation challenges. *arXiv preprint arXiv:1905.08674*.

Merkle, R. C. (1987). A Digital Signature Based on a Conventional Encryption Function. *Advances in Cryptology - CRYPTO '87, A Conference on the Theory and Applications of Cryptographic Techniques, Santa Barbara, California, USA, August 16-20, 1987, Proceedings*, 369–378. https://doi.org/10.1007/3-540-48184-2_32

Rios, F., Almas, B., Contaxis, N., Jabloner, P., Kelly, H., Chassanoff, A., Potterbusch, M., Work, L. (2018, February 7). *Report 1: Exploring Curation-ready Software: Use Cases.* https://doi.org/10.17605/OSF.IO/8RZ9E

Smith, A. M., Katz, D. S., & Niemeyer, K. E. (2016). Software citation principles. *PeerJ Computer Science*, *2:e86*. https://doi.org/10.7717/peerj-cs.86

Appendix A - Use Cases Analysis

The use cases analysis were taken from the VP15 activity and elaborated.

A.1 Use case: Reproduce an experiment

<table>
<tr><td colspan="2">Use case summary
As a researcher I want to reproduce an experiment that I have read about in a paper, including testing the software parts. This paper can be my own paper, as portrayed in the `10 years reproducibility challenge`[30] where paper authors are asked to reproduce their own experiment and describe the process.</td></tr>
<tr><td>Actor/s
author of paper and code,
researcher who wants to reproduce the results</td><td rowspan="2">Step by step scenario
Author seeking to have reproducible code:
needs to specify the exact versions of software that were used, and how they were used, perhaps in a methods sections
needs to specify (at least) software environment in metadata
Can specify dependencies and documentation with build instructions
Researcher seeking to reproduce experiment:
needs identifier to access source code of specific version
needs documentation and metadata on software environment and dependencies
needs identifier and access to emulated environment (if the environment is deprecated)</td></tr>
<tr><td>Goal
Reach the same results as the published paper, using the description provided by the paper itself

Example
Ten Years Reproducibility Challenge
https://github.com/ReScience/ten-years</td></tr>
<tr><td>Target for identifier
Metadata record / software source code artifact / software executable (with/without container) /</td><td>Granularity level (bold selection)
project / collection / repository / branch / release / commit / directory / file / lines of code</td></tr>
<tr><td colspan="2">Identifiers schemes</td></tr>
<tr><td colspan="2"><table><tr><td>ASCL</td><td>ARK</td><td>DOI</td><td>HAL</td><td>Hash</td><td>RRID</td><td>SWH</td><td>SwMath</td><td>Wikidata</td><td>Other:</td></tr></table></td></tr>
</table>

[30] https://github.com/ReScience/ten-years/issues/1

Challenges with reproducibility use case:

- Identifying the specific software that was used (version, packages, dependencies)
    - **Key for this group - need to be able to identify the exact version of the software**
- Documenting how the software was used (inputs, options, environment/platform incl. compiler and compiler flags, operating system)
    - Can this be made machine-readable/automated?
- If not in a container environment, still executable?
    - Is the container itself going to be (re)usable in X years?
- Can the reproducer obtain the same environment that was used originally? No/yes it depends - (platform independency?)
    - Is there an emulation solution (and identifier)?
- Is the programming language still supported (like e.g. Python 2.7 and Python 3.0 now).

A.2 Use case: Access the software source code

Use case summary
As a researcher as a user (RSU) I want to access the software source code that is described in an article. Once I can access the source code, I want to know, for instance, whether the license allows me to build upon this code.

Notes: There is an important difference between repository vs archive; persistent identifiers are aligned with this use case.

Actor/s
Researcher as a user (RSU)

Goal
Ability to use an identifier to access the content (here the software source code) of a reference or a citation in an article.

Examples

An article from 2012 with the original pdf[31] referencing the Gitorious repository (which disappeared when Gitorious closed) and an updated pdf[32] with references to content in Software Heritage:

`swh:1:rev:0064fbd0ad69de205ea6ec6999f3d3895e9442c2;origin=https://gitorious.org/parmap/parmap.git;visit=swh:1:snp:78209702559384ee1b5586df13eca84a5123aa82`

Step by step scenario
Researcher as a user (RSU):

- Access the research article
- Determine the availability (i.e., location) of the corresponding software from the article itself
- Access the hosting location
- If the hosting location does not include the source code, get from this location a link to the source code
- Access the source code together with its metadata (e.g., authors, contact, license)
- Ideally, examine additional information such as version, branches, commit history as that will provide an idea of how this source code is supported and maintained

Target for identifier
Metadata record / **software source code artifact** / software executable (with/without container) /

Granularity level (bold selection)
project / collection / **repository** / branch / **release** / commit / **directory** / **file** / **lines of code**

Identifiers schemes – any PID scheme that supports persistence of content

ASCL	ARK	DOI	HAL	Hash	RRID	SWH	SwMath	Wikidata	Other:

[31] Danelutto, M., & Di Cosmo, R. (2012). A “minimal disruption” skeleton experiment: seamless map & reduce embedding in OCaml. *Procedia Computer Science*, *9*, 1837-1846. https://doi.org/10.1016/j.procs.2012.04.202
[32] https://www.dicosmo.org/share/parmap_swh.pdf

A.3 Use case: get credit for a software artifact

Use case summary

As a software author, I want to get credit for when my software is used, and to know when it is used (and for what purpose) as evidence when I apply for funding for future development and maintenance.

Proxies for Credit includes:

- Quality of software as measured by peer review, test coverage, documentation
- Used as dependency by other software packages, including stars and forks
- Number of downloads
- Citations in the literature

Main actor/s

- Software author
- Funding Agency
- Review Panel

Secondary actor/s[33]

- Software users
- Code Hosting Platform(s)
- Publications Index

Goal

Get credit and recognition for my work with the possibility to count all the citations for my software and where and how it was used

Example

For software identified with a PID and indexed, the index provides a means to find and count citations to that PID, an example on Google scholar[34] with 29 citations for version 0.8 of the software:

- Newville, M., Stensitzki, T., Allen, D. B., & Ingargiola, A. 2014, LMFIT: NonLinear Least-Square Minimization and Curve-Fitting for Python, zenodo, doi:10.5281/zenodo.11813

Step by step scenario

Researcher as software author (RSA)

- first define **who are the software authors**
- add this information to the artifact in an added metadata file (e.g codemeta.json, CITATION.cff, AUTHORS or on the README etc.)
- share the artifact with a software release with a metadata record

Note that assembling citations for software identified with multiple PIDs can be problematic.

Target for identifier

Metadata record / software source code artifact / software executable (with/without container) /

Granularity level (bold selection)

project / collection / repository / branch / **release** / **commit** / **directory** / **file** / **lines of code**

Identifiers schemes and examples: wherever the authors list is accurate and public

ASCL	ARK	DOI	HAL	Hash	RRID	SWH	SwMath	Wikidata	Other:

[33] secondary actors are entities who can provide credit to software authors

[34] https://scholar.google.com/scholar?cites=13647197374772619471&as_sdt=400005&sciodt=0,14&hl=en

A.4 Use case: Find software answering a problem

Use case summary

As a researcher I want to find the software to solve a problem or to advance in my research. Where do I start?

I can go to wikidata or on another search engine, with a query and get a list of software that matches my query.

Actor/s

- a researcher as a software user (RSU)

Goal

find the right tool for analysis using semantic search

Example

Query on Wikidata with a specific identifier:

```
SELECT ?item ?itemLabel ?value
{
        ?item wdt:P6138 ?value .
        SERVICE wikibase:label {
bd:serviceParam wikibase:language "en,en"  }
}
```

P6138 is a SWHID and the query retrieves all entities with a SWHID

Step by step scenario

- the researcher
  - goes to a search engine (e.g Wikidata, Wikipedia)
  - enters descriptive properties in the search box (e.g tags, description , programming language, data formats, etc.)
  - the request is transformed to a sparql query to the wikidata knowledge graph or other
  - a resulting list of matching software is returned
  - The researcher chooses an item from the list
  - lands on the software page software including the identifier that might allow other use cases (access, download, reuse, etc.)

Target for identifier

Metadata record / software source code artifact / software executable (with/without container) /

Granularity level (bold selection)

project / collection / **repository** / branch / **release** / commit / directory / file / lines of code

Identifiers schemes and examples

ASCL	ARK	DOI	HAL	Hash	RRID	SWH	SwMath	Wikidata	Other:

Appendix B - List of working group participants

The use cases analysis were taken from the VP15 activity and elaborated

Ajit Singh
Alejandra Gonzalez-Beltran
Alexandra Delipalta
Alice Allen
Andrea Mannocci
Andrea Dell'Amico
Andrew Treloar
Brian Matthews
Catherine Jones
Christopher Erdmann
Daina Bouquin
Daniel Scharon
Daniel S. Katz
Dawit Tegbaru
Dawn Wright
Doina Cristina Duma
Emmy Tsang
Eric Quinton
Fernando Niño
Fotis Psomopoulos
Geneviève Michaud
Heather Yager
Ian Bruno
Ilian Todorov
Ingemar Häggström
Jamie Lupo-Petta
Janos Mohacsi
Jean-Christophe Malapert
Jenny Thomas
Jeremy Cope
Jonathan Bisson
Jonathan Tedds
Jose Benito Gonzalez Lopez
Joshua Taillon
Julia Collins
Jürgen Knödlseder
larson reever
Leonardo Candela
Lesley Wyborn
Leslie Hsu
Limor Peer
Marieke Willems
Marios Chatziangelou
Mark Leggott
Martin Fenner
Martina Stockhause
Mateusz Kuzak
Mingfang Wu
Minho Lee
Mohammad Akhlaghi
Morane Gruenpeter
Naeem Muhammad
Neil Chue Hong
Pablo Orviz
Paolo Manghi
Paula Martinez Lavanchy
Paula Andrea Martinez
Peter Neish
Peter Doorn
Qian Zhang
Rahul Tomar
Réka Kósa
Rik Janssen
Rob Hooft
Roberto Di Cosmo
Rouven Schabinger
Sam Bradley
Sandor Brockhauser
Sarah Jones
Shahanshah Manzoor
Shelley Stall
Sophie Fortuno
Stefanie Kethers
Stéphane Laurière
Stéphanie Rennes
Stephanie van de Sandt
Susheel Varma
Syeda Tasnim Jannat
Thomas Morrell
Tim Dennis
Timea Biro
Tovo Rabemanantsoa
Violaine Louvet
Volodymyr Kushnarenko
Wendy Hagenmaier