

Chapter 1

The Software Heritage Open Science Ecosystem

Roberto Di Cosmo and Stefano Zacchiroli

Abstract

Software Heritage [5] is the largest public archive of software source code and associated development history, as captured by modern version control systems. As of February 2023 it has archived more than 12 billion unique source code files and 2 billion commits, coming from more than 180 million collaborative development projects. In this chapter we describe the Software Heritage ecosystem, focusing on research and open science use cases.

On the one hand Software Heritage supports empirical research on software by materialising in a single Merkle direct acyclic graph the development history of public code [19]. This giant graph of source code artifacts (files, directories, and commits) can be used—and has been used—to study repository forks [39], open source contributors [51, 52, 43], vulnerability propagation, software provenance tracking [44], source code indexing, and more.

On the other hand Software Heritage ensures availability and guarantees integrity of the source code of software artifacts used in any field that relies on software to conduct experiments, contributing to making research reproducible. The source code used in scientific experiments can be archived—e.g., via integration with open access repositories [15]—referenced using persistent identifiers [16] that allow downstream integrity checks, and linked to/from other scholarly digital artifacts [14].

Roberto Di Cosmo
Inria and Université Paris Cité, France, e-mail: roberto@dicosmo.org

Stefano Zacchiroli
LTCI, Télécom Paris, Institut Polytechnique de Paris, France e-mail: stefano.zacchiroli@telecom-paris.fr

1.1 The Software Heritage Archive

Software Heritage [19, 5] is a non-profit initiative started by Inria in partnership with UNESCO to build a long-term universal archive specifically designed for software source code, capable of storing source code files and directories, together with their full development histories.

Software Heritage’s mission is to collect, preserve, and make easily accessible the source code of *all publicly available software*, addressing the needs of a plurality of stakeholders, ranging from cultural heritage to public administrations and from research to industry.

The key principles that underpin this initiative are described in detail in two articles written for a broader audience in the early years of the project [19, 5]. One of these principles was to avoid any *a priori* selection of the contents of the archive, to avoid the risk of missing relevant source code, whose value will only become apparent later on. Hence one of the strategies enacted for collecting source code to archive is the large-scale automated crawling of major software development forges and distributions, as shown in Figure 1.1.

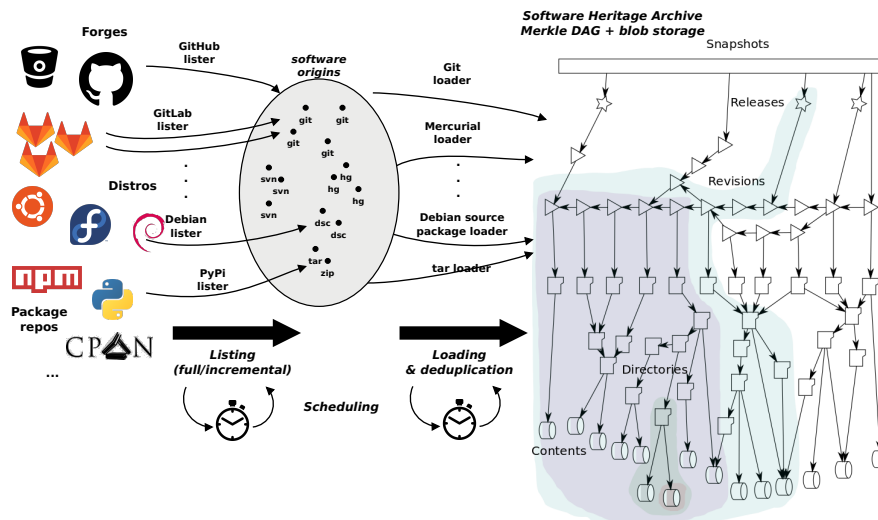


Fig. 1.1: Software Heritage data flow: crawling (on the left) and archival (right)

As a consequence of this automated harvesting, there is no guarantee that the content of the archive only contains quality source code, or only code that builds properly: curation of the contents will need to happen at a later stage, via human or automated processes that build a view of the archive for specific needs. It may also happen that the archive ends up containing content that needs to be removed, and

this required the creation of a process to handle take down requests following current legal regulations.¹

The sustainability plan is based on several pillars. The first one is the support of Inria, a national research institution that is involved for the long term. A second one is the fact that Software Heritage provides a common infrastructure catering to the needs of a variety of stakeholders, ranging from industry to academia, from cultural heritage to public administrations. As a consequence, funding comes from a diverse group of sponsors, ranging from IT companies to public institutions. Finally, an extra layer of archival security is provided by a network of independent international mirrors that maintain each a full copy of the archive.²

We recall here a few key properties that set Software Heritage apart from other scholarly infrastructures:

- Software Heritage *proactively* archives *all software*, making it possible to store and reference any piece of publicly available software relevant to a research result, independently from any specific field of endeavour, and even when the author(s) did not take any step to have it archived [19, 5];
- Software Heritage stores source code with its development history in a uniform data structure, a Merkle Directed Acyclic Graph (DAG) [33], that allows to provide uniform, *intrinsic* identifiers for tens of billions archived software artifacts, independently of the version control system (VCS) or package distribution technology used by software developers [17].

Relevance for software ecosystems. Software Heritage relates to software ecosystems, according to the seminal definition of Messerschmitt *et al.* [34] in two main ways. On the one hand, software products are associated to source code artifacts that are versioned and stored in VCSs. For Free/Open Source Software (FOSS), and more generally public code, those artifacts are distributed publicly and can be mined to pursue various goals. Software Heritage collects and preserves observable artifacts that originates from open source ecosystems, enabling others to access and exploit them in the foreseeable future.

On the other hand, Software Heritage provides the means to foster the sharing of even more of those artifacts in the specific case of open scientific practices—what we refer to as the “open science ecosystem” in this chapter. Contrary to software-only ecosystems, the open science ecosystem encompasses a variety of software and non-software artifacts (e.g., data, publications); Software Heritage has contributed to this ecosystem the missing piece of long-term archival and referencing of scientifically-relevant software source code artifacts.

¹ See <https://www.softwareheritage.org/legal/content-policy/> for details.

² More details can be found at <https://www.softwareheritage.org/support/sponsors> and <https://www.softwareheritage.org/mirrors>.

1.1.1 Data Model

Modern software development produces multiple kinds of source code artifacts (e.g., source code files, directories, commits), which are usually stored and tracked in version control systems, distributed as packages in various formats, or otherwise.

When designing a software source code archive that stores source code with its version control history coming from a disparate set of platforms, there are different design options available. One option is to keep a verbatim copy of all the harvested content, which makes it easy to immediately reuse the package or version control tool. However, this approach can result in storage explosion: as a consequence of both social coding practices on collaborative development platforms and the liberal licensing terms of open source software, those source code artifacts end up being massively duplicated across code hosting and distribution platforms.

Choosing a data structure that minimizes duplication is better for long-term preservation and the ability to identify easily code reuse and duplication.

This is the choice made by Software Heritage. Its data model is a Direct Acyclic Graph (DAG) that leverages classical ideas from content addressable storage and Merkle trees [33], that we recall briefly here.

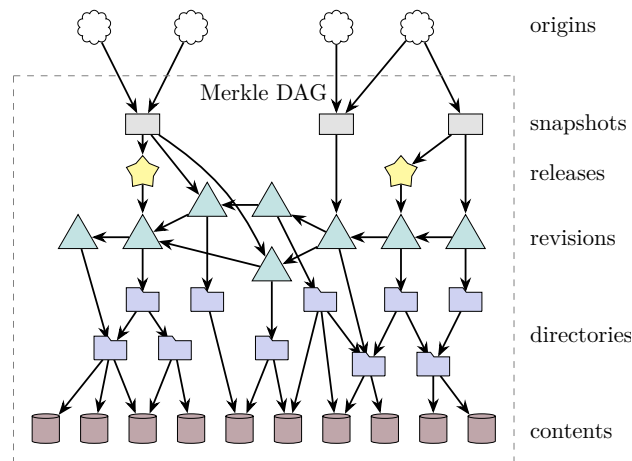


Fig. 1.2: Data model of the Software Heritage archive: a directed acyclic graph (DAG) linking together deduplicated software artifacts shared across the entire body of (archived) public code

As shown in Figure 1.2, the Software Heritage DAG is organized in five logical layers, which we describe below from bottom to top.

Contents (or “blobs”) form the graph’s leaves, and contain the raw content of source code files, not including their filenames (which are context-dependent and stored only as part of directory entries).

Directories are associative lists mapping names to directory entries and associated metadata (e.g., permissions). Each entry can point to content objects (“file entries”), revisions (“revision entries”, e.g., to represent git submodules or subversion externals), or other directories (“directory entries”).

Revisions (or “commits”) are point-in-time representations of the entire source tree of a development project. Each revision points to the root directory of the project source tree, and a list of its parent revisions (if any).

Releases (or “tags”) are revisions that have been marked by developers as noteworthy with a specific, usually mnemonic, name (e.g., a version number like “4.2”). Each release points to a revision and might include additional metadata such as a changelog message, digital signature, etc.

Snapshots are point-in-time captures of the full state of a project development repository. While revisions capture the state of a single development line (or “branch”), snapshots capture the state of *all* branches in a repository and allow to reconstruct the full state of a repository that has been deleted or modified destructively (e.g., rewriting its history with tools like “git rebase”).

Origins represent the places where artifacts have been encountered in the wild (e.g., a public Git repository) and link those places to snapshot nodes and associated metadata (e.g., the timestamp at which crawling happened), allowing to start archive traversals pointing into the Merkle DAG.

The Software Heritage archive is hence a giant graph containing nodes corresponding to all these artifacts and links between them as graph edges.

What makes this DAG capable of deduplicating identical content is the fact that each node is identified by a cryptographic hash that concisely represent its contents, and that is used in the SWHID identifier detailed in the next section. For the blobs that are the leafs of the graph, this identifier is just a hash of the blob itself, so even if the same file content can be present in multiple projects, its identifier will be the same, and it will be stored in the archive only once, like in classical content addressable storage [41]. For internal nodes, the identifier is computed from the aggregation of the identifiers of its children, following the construction originally introduced by Ralph Merkle [33]: as a consequence, if a same directory, possibly containing thousands of files, is duplicated across multiple project, its identifier will stay the same, and it will be stored only once in the archive. The same goes for revision, releases and snapshots.

In terms of size the archive grows steadily over time as new source code artifacts get added to it, as shown in Figure 1.3. As of February 2023, the Software Heritage archive contained over 14 billions unique source code files, harvested from more than 210 million software origins.³

³ See <https://archive.softwareheritage.org> for these and other up-to-date statistics.



Fig. 1.3: Evolution of the Software Heritage Archive over time (February 2023)

1.1.2 Software Heritage Persistent Identifiers (SWHIDs)

As part of the archival process, a *Software Heritage Persistent Identifier (SWHID)*, is computed for each source code artifact added to the archive and can be used later to reference, lookup, and retrieve it from the archive. The general syntax of SWHIDs is shown in Figure 1.4.⁴

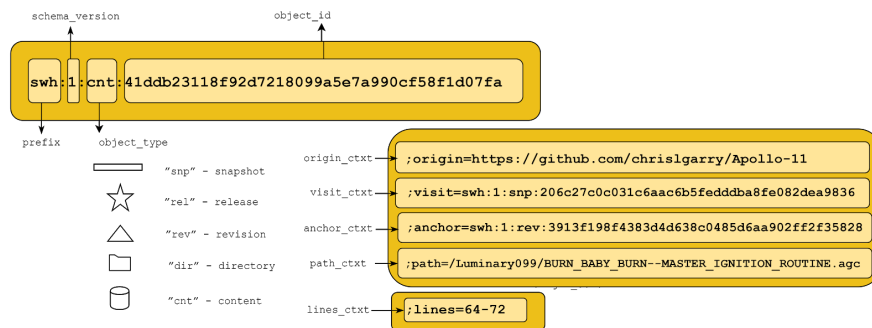


Fig. 1.4: Schema of the Software Heritage identifiers (SWHID)

SWHIDs are URIs [48] with a simple syntax. *Core* SWHIDs start with the "swh" URI scheme; the colon (:) is used as separator between the logical parts of identifiers; the schema version (currently 1) is the current version of this identifier schema; then follows the type of source code artifacts identified; and finally comes a hex-encoded

⁴ See <https://docs.softwareheritage.org/devel/swh-model/persistent-identifiers.html> for the full specification of SWHIDs.

(using lowercase ASCII characters) cryptographic signature of this object, computed in a standard way, as detailed in [16, 17].

Core SWHIDs can then be complemented by *qualifiers* that carry contextual *extrinsic* information about the referenced source code artifact:

origin: the *software origin* where an object has been found or observed in the wild, as an URI;

visit: persistent identifier of a *snapshot* corresponding to a specific *visit* of a repository containing the designated object;

anchor: a *designated node* in the Merkle DAG relative to which a *path to the object* is specified;

path: the *absolute file path*, from the *root directory* associated to the *anchor node*, to the object;

lines: *line number(s)* of interest, usually pointing within a source code file.

The combination of core SWHIDs and qualifiers provides a powerful means of referring in a research article all source code artefacts of interest.

By keeping all the development history in a single global Merkle DAG. Software Heritage offers unique opportunities for *massive analysis of the software development landscape*. By archiving and referencing all the publicly available source code, the archive also constitutes the ideal place to *preserve research software artifacts* and offers powerful mechanisms to *enhance research articles* with precise references to relevant fragments of source code, and contributes an essential building block to the software pillar of Open Science.

1.2 Large Open Datasets for Empirical Software Engineering

The availability of large amounts of source code that came with the growing adoption of open source and collaborative development has attracted the interest of software engineering researchers since the beginning of the 2000's, and opened the way to large-scale empirical software engineering studies and a dedicated conference, Mining Software Repositories.

Several shared concerns emerged over time in this area, and we recall here some of the ones that are relevant for the discussion in this chapter.

One issue is the significant overhead involved in the systematic extraction of relevant data from the publicly available repositories and their analysis for testing research hypotheses. Building a very large scale dataset containing massive amounts of source code with its version control history is a complex undertaking and requires significant resources, as shown in seminal work by Mockus in 2009 [35]. The lack of a common infrastructure spawned a proliferation of ad hoc pipelines for collecting and organising source code with its version control history, a duplication of efforts that were subtracted to the time available to perform the intended research and hindered their reusability. A few initiatives were born with the intention of improving this

unsatisfactory state of affairs: Boa [20] provides selected datasets (the largest and most recent one at the time of writing consists of about 8 million GitHub repositories sampled in October 2019) and a dedicated domain specific language to perform efficient queries on them, while World of Code [32] collects git repositories on a large scale and maintains dedicated data structures that ease their analysis.

The complexity of addressing the variety of existing code hosting platforms and version control systems resulted in focusing only on subsets of the most popular ones, in particular the GitHub forge and the git version control system, which raises another issue: the risk of introducing bias in the results. In empirical sciences, *selection bias* [25] is the bias that originates from performing an experiment on a non-representative subset of the entire population under study. It is a methodological issue that can lead to threats to the *external validity* of experiments, i.e., incorrectly concluding that the obtained results are valid for the entire population, whereas they might only apply to the selected subset. In empirical software engineering a common pattern that could result in selection bias is performing experiments on software artifacts coming from a relatively small set of development projects. It can be mitigated by ensuring that the project set is representative of the larger set of projects of interest, but doing so could be challenging.

Finally, there is the issue of enabling *reproducibility* of large-scale experiments—i.e., the ability to replicate the findings of a previous scientific experiment, by the same or a different team of scientists, reusing varying amounts of the artifacts used in the original experiment [30].⁵ Large-scale empirical experiments in software engineering might easily require shipping *hundreds* of GiB up to a few TiB of source code artifacts as part of replication packages, whereas current scientific platform for data self archival usually cap at tens of GiB.⁶

The comprehensiveness of the Software Heritage archive, that makes available the largest public corpus of source code artifacts in a single logical place, helps with all these issues:

- reduces the *opportunity cost* of conducting large-scale experiments by offering at regular intervals as *open datasets* full dumps of the archive content
- contributes to *mitigate* selection bias and the associated external validity threats by providing a corpus that strives to be *comprehensive* for researchers conducting empirical software engineering experiments targeting large project populations.
- the persistence offered by an independent digital archive, run by a non profit open organisation, eases the process of ensuring the *reproducibility* of large-scale experiments, avoiding the need to re-archive the same open source code artifacts in multiple papers, a wasteful practice that should be avoided if possible. Using Software Heritage it is enough to thoroughly document in replication packages

⁵ For the sake of conciseness we do not differentiate here between repeatability, reproducibility, and replicability; we refer instead the interested reader to the ACM terminology available at <https://www.acm.org/publications/policies/artifact-review-and-badging-current>. To varying degrees Software Heritage helps with all of them, specifically when it comes to mitigating the risk of losing availability to source code artifacts.

⁶ For comparison: the total size of source code archived at Software Heritage is ≈ 1 PiB at the time of writing.

Table 1.1: Comparison of infrastructures for performing empirical software engineering research

Criteria	Infrastructure		Boa	World of Code
	SWH on S3	SWH graph (on premise)		
host organisation	non profit foundation		research project	research project
purpose	archival & research			
scope	all platforms		GitHub, SourceForce	Git hosting
dataset	open		closed	closed
access	free		on demand	on demand
query language	SQL Athena	graph API	custom DSL	custom API
cost	5\$/TB	10K\$ setup	free	free
dataset update frequency	6 months		≈ yearly	≈ yearly
reproducibility	named dataset	SWHID list	named dataset	named dataset

the SWHIDs (see Section 1.1.2) of all source code artifacts⁷ used in an empirical experiment to enable other scientists to reproduce the experiments later on [14].

Table 1.1 summarises the above points, comparing with a few other infrastructures designed specifically for software engineering studies.

In the rest of this section we briefly describe the datasets that Software Heritage curates and maintains to the benefit of other researchers in the field of empirical software engineering.

Before detailing the available datasets, we recall that building and maintaining the Software Heritage infrastructure that is instrumental to build them is a multi-million dollars undertaking. We are making significant efforts to reduce the burden on the prospective users, by providing dumps at regular intervals that help with reproducibility and making them directly available on public clouds like AWS. Researchers can then either run their queries directly on the cloud, paying only the compute time, or download them for exploiting them on their own infrastructure.

To give an idea of the associated costs for researchers, SQL queries on the graph datasets described in 1.2.1.1 can be performed using Amazon Athena for approximately 5\$ per Terabyte scanned at the time of writing. For example, an SQL query to get the 4 topmost commit verb stems from over 2 billion revisions scans

⁷ As it will become clear in Section 1.1.2, in most cases it will be sufficient to list the SWHIDs of the releases or repository snapshots.

approximately 100 Gigabytes of data, and provides the user with the answer in less than a minute, for a total cost of approximately 50 cents, a minimal fraction of the cost one would incur to set up an on premise solution.

When SQL queries are not enough (typically when a graph traversal is needed), the cost of a cloud solution may quickly become significant, and it may become more interesting to set up an on premise solution. The full compressed graph dataset can be exploited using medium range server grade machines that are accessible for less than 10 thousand dollars.

1.2.1 The Software Heritage Datasets

The entire content of the Software Heritage archive is publicly available to researchers interested in conducting empirical experiments on it. At the simplest level, the content of the archive can be browsed interactively using the Web user interface at <https://archive.softwareheritage.org/> and accessed programmatically using the Web API documented at <https://archive.softwareheritage.org/api/>. These access paths, however, are not really suitable for large-scale experiments due to protocol overheads and rate limitations enforced to avoid depleting archive resources. To address this, several curated datasets are regularly extracted from the archive and made available to researchers in ways suitable for mass analysis.

1.2.1.1 The Software Heritage *Graph* Dataset

Consider the data model discussed in Section 1.1.1. The entire archive graph is exported periodically as the *Software Heritage Graph Dataset* [40]. Note the word “graph” in there, which characterises this particular dataset and denotes that *only* the graph is included in the dataset, up to the content of its leaf nodes, excluded (for size reasons). This dataset is suitable for analysing source code *metadata*, including commit information, file names, software provenance, code reuse, etc.; but not for textual analyses of archived source code, as that is stored in graph leaves (see the blob dataset below for how to analyse actual code).

The data model of the graph dataset is a relational representation of the archive Merkle DAG, with one “table” for each type of node: blobs, directories, commits, releases, and snapshots. Each table entry is associated with several attributes, such as multiple checksums for blobs, file names and attributes for directories, commit messages and timestamps for commits, etc. The full schema is documented at <https://docs.softwareheritage.org/devel/swh-dataset/graph/schema.html>.

In practical terms, the dataset is distributed as a set of Apache ORC files for each table, suitable for loading into scale-out columnar-oriented data processing frameworks such as Spark and Hadoop. The ORC files can be downloaded from the public Amazon S3 bucket <s3://softwareheritage/graph/>. At the time of

writing the most recent dataset export has timestamp 2022-12-07 so, for example, the first ORC files of the commit table are:

```

1 $ aws s3 ls --no-sign-request
   ↪ s3://softwareheritage/graph/2022-12-07/orc/revision/
2 2022-12-13 17:41:44 3099338621 revision-[..]-f9492019c788.orc
3 2022-12-13 17:32:42 4714929458 revision-[..]-42da526d2964.orc
4 2022-12-13 17:57:00 3095895911 revision-[..]-9c46b558269d.orc
5 [..]
```

The current version of the dataset contains metadata for 13 billion source code files, 10 billion directories, 2.7 billion commits, 35 million releases and 200 million VCS snapshots, coming from 189 M software origins. The total size of the dataset is 11 TiB, which makes it unpractical for use on personal machines, as opposed to research clusters. For that reason hosted versions of the dataset are also available on Amazon Athena and Azure Databricks. The former can be queried using the Presto distributed SQL engine without having to download the dataset locally. For example, the following query will return the most common first word stems used in commit messages across more than 2.7 billion commits in just a few seconds:

Listing 1.1: Simple SQL query to get the 4 topmost commit verb stems

```

1 SELECT count(*) as c,word FROM (
2   SELECT word_stem(lower(split_part(trim(from_utf8(message)), ' ', 1)))
   ↪ as word
3   from revision WHERE length(message) < 1000000)
4 WHERE word != ''
5 GROUP BY word ORDER BY c DESC LIMIT 4
```

For the curious reader the (unsurprising) results of the query look like this:

Count	Word
294 369 196	updat
178 738 450	merg
152 441 261	add
113 924 516	fix

More complex queries and examples can be found in previous work [40]. For more details about using the graph dataset we refer the reader to its technical documentation at <https://docs.softwareheritage.org/devel/swh-dataset/graph/>.

In addition to the research highlights presented later in this chapter, the Software Heritage graph dataset has been used as subject of study for the 2020 edition of the MSR (Mining Software Repositories) mining challenge, where students and young researchers in software repository mining have used it to solve the most interesting mining problems they could think of. To facilitate their task “teaser” datasets —data samples with exactly the same shape of the full dataset, but much smaller— have also been produced and can be used by researchers to understand how the dataset works before attacking its full scale. For example, the `popular-3k-python` teaser contains

a subset of 2.197 popular repositories tagged as implemented in Python and being popular according to various metrics (e.g., GitHub stars, PyPI download statistics, etc.). The `gitlab-all` teaser corresponds to all public repositories on `gitlab.com` (as of December 2020), an often neglected ecosystem of Git repositories, which is interest to study to avoid (or compare against) GitHub-specific biases.

1.2.1.2 Accessing Source Code Files

All source code files archived by Software Heritage are spread across multiple copies and also mirrored to the public Amazon S3 bucket `s3://softwareheritage/content/`. From there, individual files can be retrieved, possibly massively and in parallel, based on their SHA1 checksums. Starting from SWHIDs one can obtain SHA1 checksums using the `content` table of the graph dataset and then access the associated content as follows:

```

1 $ aws s3 cp s3://softwareheritage/content/\
2   8624bcdae55baeef00cd11d5dfcfa60f68710a02 .
3 download: s3://softwareheritage/content/8624b[...] to ./8624b[...]
4
5 $ zcat 8624bcdae55baeef00cd11d5dfcfa60f68710a02 | sha1sum
6 8624bcdae55baeef00cd11d5dfcfa60f68710a02 -
7
8 $ zcat 8624bcdae55baeef00cd11d5dfcfa60f68710a02 | head
9     GNU GENERAL PUBLIC LICENSE
10    Version 3, 29 June 2007
11 [...]

```

Note that individual files are `gzip`-compressed to further reduce storage size.

The general empirical analysis workflow involves three simple steps: identify the source code files of interest using the metadata available in the graph dataset, obtain their checksum identifiers, and then retrieve them in batch and in parallel from public cloud providers. This process scales well up to many million files to be analysed. For even larger-scale experiments, e.g., analysing *all* source code files archived at Software Heritage, research institutions may consider setting up a local mirror of the archive.⁸

1.2.1.3 License Dataset

In addition to datasets that correspond to the actual content of the archive, i.e., source code artifacts as encountered among public code, it is also possible to curate *derived* datasets extracted from Software Heritage for the specific use cases or fields of endeavours.

⁸ See <https://www.softwareheritage.org/mirrors/> for details, including storage requirements. At the time of writing a full mirror of the archive requires about 1 PiB of raw storage.

As of today one notable example of such a derived dataset is the *license blob dataset*, available at <https://annex.softwareheritage.org/public/dataset/license-blobs/> and described in [52]. It consists of the largest known dataset of the complete texts of free/open source software (FOSS) license variants. To assemble it the authors collected from the Software Heritage archive all versions of files whose names are commonly used to convey licensing terms to software users and developers, e.g., COPYRIGHT, LICENSE, etc. (the exact pattern is documented as part of the dataset replication package).

The dataset consists of 6.5 million unique license files that can be used to conduct empirical studies on open source licensing, training of automated license classifiers, natural language processing (NLP) analyses of legal texts, as well as historical and phylogenetic studies on FOSS licensing. Additional metadata about shipped license files are also provided, making the dataset ready to use in various empirical software engineering contexts. Metadata include: file length measures, detected MIME type, detected SPDX [46] license (using ScanCode [36], a state-of-the-art tool for license detection), example origin (e.g., GitHub repository), oldest public commit in which the license appeared. The dataset is released as open data as an archive file containing all deduplicated license files, plus several portable CSV files for metadata, referencing files via cryptographic checksums.

1.3 Research Highlights

The datasets discussed in the previous section have been used to tackle research problems in empirical software engineering and neighbouring fields. In this section we provide brief highlights on the most interesting of them.

1.3.1 Enabling Artifact Access and (Large-Scale) Analysis

Applied research in various fields has been conducted to ease access to such a huge amount of data as the Software Heritage archive for empirical researchers. This kind of research is not, strictly speaking, research *enabled by* the availability of the archive to solve software engineering problems, but rather research *motivated by* the practical need of empowering fellow scholars to do so empirically.

As a first example *SwhFS* (the “*Software Heritage File System*”) [6] is a virtual filesystem developed using the Linux FUSE (Filesystem in User Space) framework that can “mount”, in the UNIX tradition, selected parts of the archive as if they were available locally as part of your filesystem. For example, starting from a known SWHID, one can for instance:

```
1 $ mkdir swarfs
2 $ swh fs mount swarfs/ # mount the archive
3 $ cd swarfs/
```

```

4
5 $ cat archive/swh:1:cnt:c839dea9e8e6f0528b468214348fee8669b305b2
6 #include <stdio.h>
7
8 int main(void) {
9     printf("Hello, World!\n");
10 }
11
12 $ cd archive/swh:1:dir:1fee702c7e6d14395bbf\
13 5ac3598e73bcbf97b030
14 $ ls | wc -l
15 127
16 $ grep -i antenna THE_LUNAR_LANDING.s | cut -f 5
17 # IS THE LR ANTENNA IN POSITION 1 YET
18 # BRANCH IF ANTENNA ALREADY IN POSITION 1

```

In the second example we are grepping through the code of Apollo 11 guidance computer code, searching for reference to antennas.

SwhFS allows to bridge the gap between classic UNIX-like mining tools, which are often relied upon in the fields of empirical software engineering and software repository mining, as well as by the Software Heritage APIs. However, it is not suitable for very large scale mining, due to the fact that seemingly local archive access pass through the public Internet (with caching, but still not suitable for large experiments).

swh-graph [10] is a way to enable such large-scale experiments. The main idea behind its approach is to adapt and apply *graph compression* techniques, commonly used for graphs such as the Web or social network, to the Merkle DAG graph that underpins the Software Heritage archive. The main research question addressed by *swh-graph* is:

Is it possible to efficiently perform software development history analyses at ultra-large scale, on a single, relatively cheap machine?

The answer is affirmative. As of today the entire structure of the Software Heritage graph (≈ 25 billion nodes + 350 billion edges) can be loaded in memory on a single machine equipped with ≈ 200 GiB of RAM (roughly: 100 GiB for the direct graph + 100 GiB for its transposed version, which is useful in many research use cases such as source code provenance analysis). While significant and not suitable for personal machines, such requirements are perfectly fine for server-grade hardware on the market, with an investment of a few thousand US dollars in RAM. Once loaded the entire graph can be visited in full in just a few hours and a single path visit from end-to-end can be performed in tens of nanoseconds per edge, close to the cost of a single memory access per edge.

In practical terms, this allows to answer queries such as “where does this file/directory/commit come from” or “list the entire content of this repositories” in fractions of seconds (depending just on the size of the answer, in most cases) fully in memory, without having to rely on a DBMS or even just disk accesses. The price to pay for this is that: (1) the compressed graph representation loaded in memory is derived from the main archive and not incremental (it should periodically be recreated) and

(2) only the graph structure and selected metadata fit in RAM, others reside on disk (although using compressed representations as well [38]) and need to be memory mapped for efficient access to frequently accessed information.

Finally, the archive also provides interesting use cases for database research. Recently, Wellenzohn *et al.* [49] has used it to develop a novel type of *content-and-structure (CAS) index*, capable of indexing over time the evolution of properties associated to specific graph nodes, e.g., a file content residing at a given place in a repository changing over time together with its metadata (last modified timestamp, author, etc.). While these indexes existed before, their deployment and efficient pre-population were still unexplored at this scale.

1.3.2 Software Provenance and Evolution

The peculiar structure—a fully deduplicated Merkle DAG—and comprehensiveness of the Software Heritage archive provides a powerful observation point and tool on the evolution and provenance of public source code artifacts. In particular it is possible, on the one hand, to navigate the Merkle DAG *backwards*, starting from any artifact of interest (source code file, directory, commit, etc.), to obtain the full list of all places (e.g., different repositories) where it has ever been distributed from. This area is referred to as *software provenance* and, in its simplest form, deals with determining the *original* (i.e., earliest) distribution place of a given artifact. More generally, being able to identify *all* places that have ever distributed it provides a way to measure software impact, track out of date copies or clones, and more.

Rousseau *et al.* [44] used the Software Heritage archive in a study that made two relevant contributions in this area. First, exploiting the fact that commits are deduplicated and timestamped, they verified that *the growth of public code* as a whole, at least as it is observable from the lenses of Software Heritage *is exponential*: the amount of original commits (i.e., commits never observed before throughout the archive, no matter the origin repository) in public source code doubles every ≈ 30 months and has been doing so for the past 20 years. If, on the other hand, we look at original source code blobs (i.e., files whose content has never been observed before throughout the archive, up to that point in time), the overall trends remains the same, only the speed changes: the amount of original public source code blobs doubles every ≈ 22 months. These are remarkable findings for software evolution, which had never been verified before at this macro-level.

Second, the authors showed how to model software provenance compactly, so that it can be represented (space-)efficiently at the scale of Software Heritage, and can be used to address software audit use cases which are commonplace in open source compliance scenarios, merger and acquisition audits, etc.

1.3.3 Software Forks

The same characteristics that enable studying the evolution and provenance of public code artifacts can be leveraged to study the global ecosystem of software forks. In particular, the fact that commits are fully deduplicated allows to detect forks—both collaborative ones, such as those created on social coding platforms to submit pull requests, and hostile ones used to bring the project in a different direction—even when they are not created on the same platform. It is possible to detect the fork of a project originally created on GitHub and living on GitLab.com, or vice-versa, based on the fact that the respective repositories share a common commit history.

This is important as a methodological point for empirical researchers, because by relying only on platform metadata (e.g., the fact that a repository *has been created* by clicking on a “fork” button on the GitHub user interface) researchers risk overlooking other relevant forks. In previous work Zacchiroli [52] provided a classification of the type of forks based on whether they are explicitly tracked as being forks of one another on a coding platform (Type 1 forks), they share at least one commit (Type 2), or they share a common root directory at some point in their histories (Type 3). He empirically verified that between 3.8% and 16% forks could be overlooked by considering only type 1 forks, possibly inducing a significant threat to validity for empirical analyses of forks that strive to be comprehensive.

Along the same lines, Bhattacharjee *et al.* [9] (participants in the MSR 2020 mining challenge) focus their analyses on “cross-platform” forks between GitHub and GitLab.com, identifying several cases in which interesting development activity can be found on GitLab even for projects initially mirrored from GitHub.

1.3.4 Diversity, Equity, and Inclusion

Diversity, equity, and inclusion studies (DE&I) are hot research topics in the area of human aspects of software engineering. Free/open source software artifacts, as archived by Software Heritage, provides a wealth of data for analysing evolutionary DE&I trends, in particular in the very long term and at the largest scale attempted thus far.

A recent study by Zacchiroli [51] has used Software Heritage to explore the trend of *gender diversity* over a time period of 50 years. He conducted a longitudinal study of the population of contributors to publicly available software source code, analysing 1.6 billion commits corresponding to the development history of 120 million projects, contributed by 33 million distinct authors over a period of 50 years. At this scale authors cannot be interviewed to ask their gender, nor cross-checking with large-enough complementary dataset was possible. Instead, automated detection based on census data from around the world and the gender-guesser tool (benchmarked for accuracy, and popular in the field) was used. Results show that, while the amount of commits by female authors remains very low overall (male authors have contributed more than 92% of public code commits over the 50 years leading to 2019), there

is evidence of a stable long-term increase in their proportion over all contributions (with the ratio of commits by female authors growing steadily over 15 years, reaching in 2019 for the first time 10% of all contributions to public code).

Follow up studies have added the spatial dimension investigating the *geographic gap* in addition to the gender one. Rossi *et al.* [42] have developed techniques to detect the geographic origin of authors of Software Heritage commit, using as signals the timezone offset and the author names (compared against census data from around the world). Results over 50 years of development history show evidence of the early dominance of North America in open source software, later joined by Europe. After that period, the geographic diversity in public code has been constantly increasing, with more and more contributions coming from Central and South Asia (comprising India), Russia, Africa, Central and South America.

Finally, Rossi *et al.* [43] put together the temporal and spatial dimension using the Software Heritage archive to investigate whether the ratio of women participation over time shows notable differences around the world, at the granularity of 20 macro regions. The main result is that the increased trend of women participation is indeed a world-wide phenomenon, with the exception of specific regions of Asia where the increase is either slowed or completely flat. An incidental finding is also worth noting: the positive trend of increased women participation observed up to 2019 has been reversed by the COVID-19 pandemic, with the *ratio* of both contributions by and active female authors decreasing sharply starting at about that time.

These studies show how social aspects of software engineering can benefit from large-scale empirical studies and how they can be enabled by comprehensive, public archives of public code artifacts.

1.4 Building the Software Pillar of Open Science

Software plays a key role in scientific research, and it can be a tool, a result, and a research object. [...] France will support the development and preservation of source code – inseparable from the support of humanity’s technical and scientific knowledge – and it will, from this position, continue its support for the Software Heritage universal archive. So as to create an ecosystem that connects code, data and publications, the collaboration between the national open archive HAL, the national research data platform Recherche Data Gouv, the scientific publishing sector and Software Heritage will be strengthened.

Second french national plan for open science, July 2021 [23]

Software is *an essential research output*, and its source code implements and describes data generation and collection, data visualisation, data analysis, data transformation, and data processing with a level of precision that is not met by scholarly articles alone. Publicly accessible software source code allows a better understanding of the process that leads to research results, and open source software allows researchers to build upon the results obtained by others, provided proper mechanisms are put in place to make sure that software source code is preserved and that it is referenced in a persistent way.

There is a growing general awareness of its importance for supporting the research process [12, 47, 26]. Many research communities focus on the issue of *scientific reproducibility* and strongly encourage making the source code of the artefact available by archiving it in publicly accessible long-term archives; some have even put in place mechanisms to assess research software, like the *Artefact Evaluation* process introduced in the ESEC-FSE 2011 conference and now widely adopted by many computer science conferences [13], and the ACM *Artifact Review and Badging* program.⁹ Other raise the complementary issues of making it easier to discover existing research software, and giving academic credit to its authors [45, 27, 31].

These important issues are similar in spirit to those that led to the now popular FAIR data movement [50], and as a first step 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. They can be classified as follows:

Archival: software artifacts must be properly **archived**, to ensure we can *retrieve* them at a later time;

Reference: 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: software artifacts must be equipped with proper **metadata** to make it easy to *find* them in a catalog or through a search engine;

Citation: research software must be properly **cited** in research articles in order to give *credit* to the people that contributed to it.

These are not only different concerns, but also *separate* ones. Establishing proper *credit* for contributors via *citations* or providing proper metadata to *describe* the artifacts requires a *curation* process [11, 7, 15] and is way more complex than simply providing stable, intrinsic identifiers to *reference* a precise version of a software source code for reproducibility purposes [27, 8, 17]. Also, as remarked in [26, 8], 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.

In this section, we focus on the first two concerns, *archival* and *reference*, that can be addressed fully by leveraging the Software Heritage archive, but we also describe how Software Heritage contributes through its ecosystem to the two other concerns.

1.4.1 Software in the Scholarly Ecosystem

Presenting results in journal or conference articles has always been part of the research activity. The growing trend, however, is to include software to support or demonstrate such results. This activity can be a significant part of academic work and must be properly taken into account when researchers are evaluated [8, 45].

⁹ <https://www.acm.org/publications/policies/artifact-review-badging>

Software source code developed by researchers is only *a thin layer* on top of the complex web of software components, most of them developed outside of academia, that are necessary to produce scientific results: as an example, Figure 1.5 shows the many components that are needed by the popular `matplotlib` library [28].

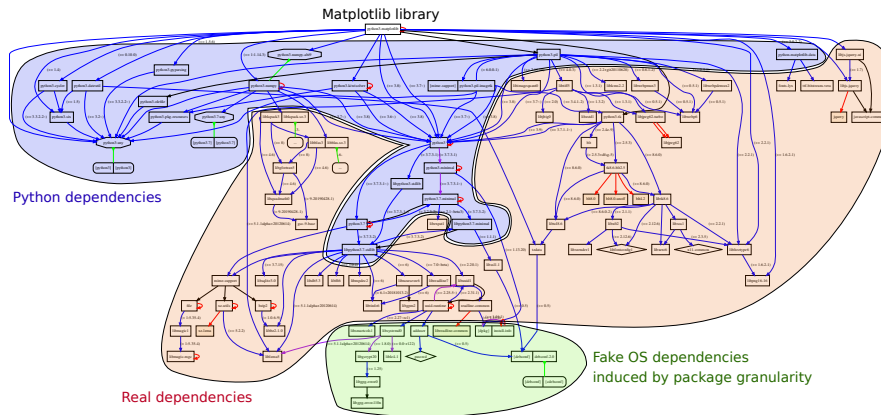


Fig. 1.5: Direct and indirect dependencies for a specific python package (`matplotlib`). In blue the Python dependencies, in red the “true” system dependencies incurred by python (e.g., the `libc` or `libjpeg62`), in green some dependencies triggered by the package management system but which are very likely not used by python (e.g., `adduser` or `dpkg`).

As a consequence, scholarly infrastructures that support software source code written in academia must go the extra mile to ensure they adopt standards and provide mechanisms that are compatible with the ones used by tens of millions of non-academic software developers worldwide. They also need to ensure that the large amount of software components that are developed outside academia, but are relevant for research activities, are properly taken into account.

Over the recent years, there have been a number of initiatives to add support for software artifacts in the scholarly world, that fall short of satisfying these requirements. They can be roughly classified in two categories:

overlays on public forges provide links from articles to the source code repository of the associated software artifact as found on a public code hosting platform (forge); typical examples are websites like <https://paperswithcode.com/>, <http://www.replicabilitystamp.org/> and the *Code and data* links recently introduced in *ArXiv.org*.

deposits in academic repositories take snapshots of a given state of the source code, usually in the form of a `.zip` or `.tar` file, and store it in the repository exactly like an article or a dataset, with an associated publisher identifier; typical examples in Computer Science is the ACM Digital Library, but there are a

number of general academic repositories where software artefacts have been deposited, like FigShare and Zenodo.

The approaches in the first category rely on code hosting platforms that do not guarantee *persistence* of the software artifact: the author of a project may alter, rename, or remove it, and we have seen that code hosting platforms can be discontinued, or decide to remove large amount of projects.¹⁰

The approaches in the second category do take into account persistence, as they archive software snapshots, but they loose the *version control history* and do not provide the *granularity* needed to reference the internal components of a software artifact (directories, files, snippets).

And none of the initiatives in these categories provides a means to properly archive and reference the numerous external dependencies of software artefacts.

This is where Software Heritage comes into play for Open Science, by providing an *archive designed for software* that provides persistence, preserves the version control history, supports granularity in the identification of software artefacts and their components, and harvests all publicly available source code.

The differences described above are summarised in the following table, where we only consider infrastructures in the second category described above, as they are the only one assuming the mission to archive their contents. We also take into account additional features found in academic repositories, like the possibility of depositing content with an *embargo* period, which is not possible on Software Heritage, and the existence of a curation process to obtain qualified metadata, which is currently out of scope of Software Heritage.

1.4.2 Extending the Scholarly Ecosystem Architecture to Software

In the framework of the European Open Science Cloud initiative (EOSC), a working group has been tasked in 2019 to bring together representatives from a broad spectrum of scholarly infrastructures to study these issues and propose concrete ways to address theme. The result, known as the EOSC Scholarly Infrastructures for Research Software (SIRS) report [18] was published in 2020 and provides a detailed analysis of the existing infrastructures, their relationships, and the workflows that are needed to properly support software as a research result on par with publications and data.

Figure 1.6 presents the main categories of identified actors:

Scholarly repositories: services that have as one of their primary goals the long-term preservation of the digital content that they collect.

Academic publishers: organisations that prepare submitted research texts, possibly with associated source code and data, to produce a publication and manage the dissemination, promotion, and archival process. Software and data can be part

¹⁰ Google Code and Gitorious.org were shut down in 2015, Bitbucket removed support for the Mercurial VCS in 2020, and in 2022 Gitlab.com considered removing all projects inactive for more than a year.

Table 1.2: Comparison of infrastructures for archiving research software. The various granularities of identifiers are abbreviated with the same convention used in SWHIDs (*snp* for snapshot, etc.), plus the abbreviation *frg* that stands for the ability to identify a *code fragment*.

Infrastructure \ Criteria	Software Heritage	ACM DL	HAL	Figshare	Zenodo
identifier	intrinsic	extrinsic	extrinsic + <i>intrinsic</i> (via SWH)	extrinsic	extrinsic
granularity	snp, rel, rev dir, cnt, frg	dir	dir	dir	rel, dir
archival	harvest deposit save code now	deposit	deposit	deposit	deposit
history	full VCS	no	no	no	releases
browse code	yes	no	no	no	no
scope	universal	discipline	academic	academic	academic
embargo	no	no	yes	yes	yes
curation	no	yes	yes	no	no
integration	BitBucket, SourceForge, GitHub, Gitea, Gitlab, HAL, etc.		SWH		GitHub

of the main publication, or assets given as supplementary materials depending on the policy of the journal.

Aggregators: services that collect information about digital content from a variety of sources with the primary goal of increasing its discoverability, and possibly adding value to this information via processes like curation, abstraction, classification, and linking.

These actors have a long history of collaboration around research articles, with well defined workflows and collaborations. The novelty here is the fact that to handle research software, it is no longer possible to work in isolation inside the academic world, for the reasons explained previously: one needs a means to share information and work with other ecosystems where software is present, like industry and public administration.

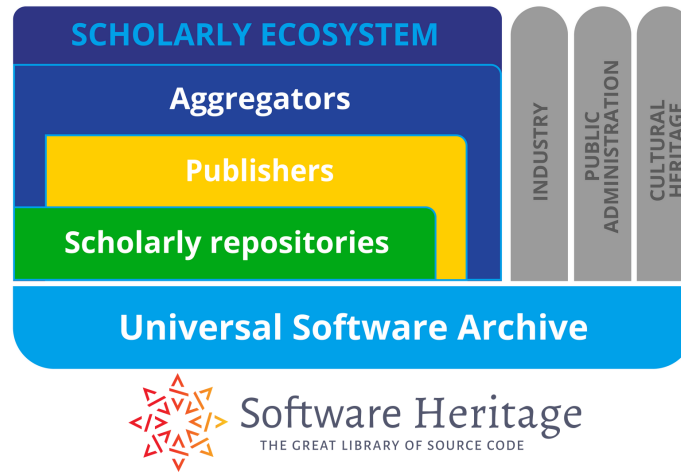


Fig. 1.6: Overview of the high level architecture of scholarly infrastructures for research software, as described in the EOSC SIRS report

One key finding of the EOSC SIRS Report is that Software Heritage provides the shared basic architectural layer that allows to interconnect all these ecosystems, because of its unified approach to archiving and referencing all software artefacts, independently of the tools or platforms used to develop or distribute the software involved.

1.4.3 Growing Technical and Policy Support

In order to take advantage of the services provide by Software Heritage in this setting, a broad spectrum of actions have been started, and are ongoing. We briefly survey here the ones that are most relevant at the time of writing.

At the national level, France has developed a multi-annual plan on Open Science that includes research software [22, 23], and consistently implemented this plan through a series of steps that range from technical development to policy measures.

On the technical side, the French national open access repository HAL [15] (analogous to the popular arXiv service¹¹) has been integrated with the Software Heritage archive. The integration allows researchers to have their software projects archived and referenced in Software Heritage, while curated rich metadata and citation information is made available on HAL [15], with a streamlined process depicted in Figure 1.7.

¹¹ <https://arxiv.org>

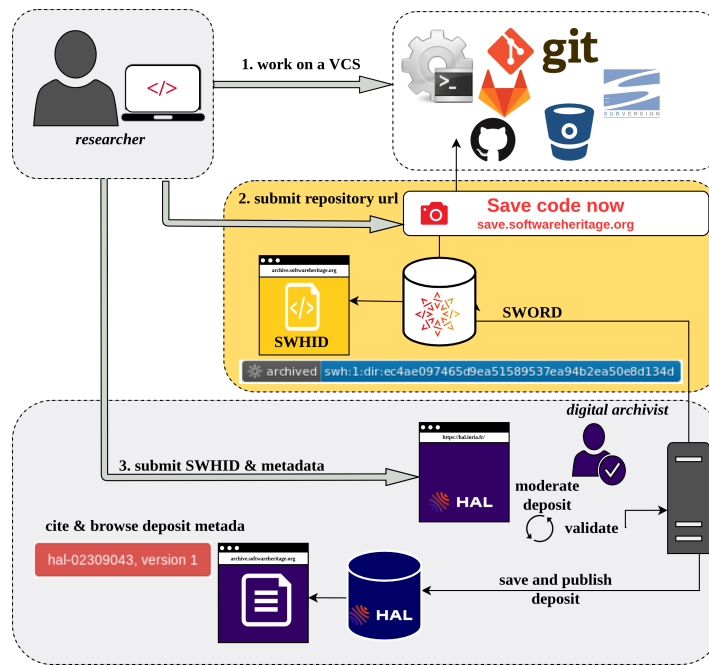


Fig. 1.7: Overview of the interplay between HAL and Software Heritage for research software

On the policy side, the second french national plan for open science [23], published in July 2021, prescribes the use of Software Heritage and HAL for all the research software produced in France, and Software Heritage is now listed in the official national roadmap of research infrastructures published in February 2022 [24].

This approach is now being pushed forward at the European level, through funding for consortia that will build the needed connectors between Software Heritage and several infrastructures and technologies used in academia, using the French experience as a reference. Most notably, the FAIRCORE4EOSC [2] European project include plans to build connectors with scholarly repository systems like Dataverse [4] and InvenioRDM [29] (the white-label variant of Zenodo), publishers like Dagstuhl [3] and Episcience [1], and aggregators like swMath [21] and OpenAire [37].

1.4.4 Supporting Researchers

The growing awareness about the importance of software as a research output will inevitably bring new recommendations for research activity, that will eventually become obligations for researchers, as we have seen with publications and data.

Through the collaboration with academic infrastructures, Software Heritage is striving to develop mechanisms that minimise the extra burden for researchers, and we mention here a few examples.

A newly released extension, codename `updateswh`, for the popular web browsers Firefox and Google Chrome allows to trigger archival in just one click for any public repository hosted on BitBucket, GitLab (.com, and any instance), GitHub and any instance of Gitea. It also allows to access in one click the archived version of the repository and obtain the associated SWHID identifier.

Integration with web hooks is available for a variety of code hosting platforms, including BitBucket, GitHub, GitLab.com and Source forge, as well as for instances of GitLab and Gitea, which enable owners of projects hosted on those platforms to trigger archival automatically on any new release, reducing the burden on researchers even more.

Software Heritage will try to detect and parse intrinsic metadata present in software projects independently of the format chosen, but we see the value of standardising on a common format. This is why, with all academic platforms we are working with, we are advocating the use of `codemeta.json`, a machine readable file based on the CodeMeta extension of `schema.org`, to retrieve automatically metadata associated to software artifact, in order to avoid the need for researchers to fill forms when declaring software artifacts in academic catalogs, following the schema put in place with the HAL national open access portal.

Finally, we have released the `biblatex-software` bibliographic style extension to make it easy to cite software artefacts in publications written using the popular \LaTeX framework.

1.5 Conclusions and Perspectives

In conclusion, the Software Heritage ecosystem is a useful resource for both software engineering studies and for Open Science. As an infrastructure for research on software engineering, the archive provides numerous benefits. The SWHID intrinsic identifiers make it easier for researchers to identify and track software artifacts across different repositories and systems. The uniform data structure used by the archive abstracts away all the details of software forges and package managers, providing a standardised representation of software code that is easy to use and analyse. The availability of the open datasets makes it possible to tailor experiments to one's needs and improves their reproducibility. An obvious direction at the time of writing is to leverage Software Heritage's extensive source code corpus for pre-training large language models. Future collaborations may lead to integrate functionalities like the

domain-specific language from the Boa project or the efficient data structures of the World of Code project, enabling researchers to run more specialised queries and achieve more detailed insights.

Regarding the Open Science aspect, Software Heritage already offers the reference archive for all publicly available research software. The next step is to interconnect it with a growing number of scholarly infrastructures, which will increase reproducibility of research in all fields, and support software citation directly from the archive, contributing to increasing visibility of research software.

Going forward, we believe that Software Heritage will provide a unique observatory for the whole software development ecosystem, both in academia and outside of it. We hope that with growing adoption it will play an increasingly valuable role in advancing the state of software engineering research and in supporting the software pillar of open science.

References

1. Episciences. <https://www.episciences.org>. Accessed 2023-04-15
2. FAIRCORE4EOSC project. <https://faircore4eosc.eu>. Accessed 2023-04-15
3. Schloss Dagstuhl. <https://www.dagstuhl.de>. Accessed 2023-04-15
4. The Dataverse Project. <https://dataverse.org>. Accessed 2023-04-15
5. Abramatic, J.F., Di Cosmo, R., Zacchiroli, S.: Building the universal archive of source code. *Communications of the ACM* **61**(10), 29–31 (2018). DOI 10.1145/3183558
6. Allançon, T., Pietri, A., Zacchiroli, S.: The software heritage filesystem (SwhFS): Integrating source code archival with development. In: International Conference on Software Engineering (ICSE). IEEE (2021). DOI 10.1109/ICSE-Companion52605.2021.00032
7. Allen, A., Schmidt, J.: Looking before leaping: Creating a software registry. *Journal of Open Research Software* **3**(e15) (2015). DOI 10.5334/jors.bv
8. Alliez, P., Di Cosmo, R., Guedj, B., Girault, A., Hacid, M.S., Legrand, A., Rougier, N.: Attributing and referencing (research) software: Best practices and outlook from INRIA. *Computing in Science and Engineering* **22**(1), 39–52 (2020). DOI 10.1109/MCSE.2019.2949413. Available from <https://hal.archives-ouvertes.fr/hal-02135891>
9. Bhattacharjee, A., Nath, S.S., Zhou, S., Chakroborti, D., Roy, B., Roy, C.K., Schneider, K.A.: An exploratory study to find motives behind cross-platform forks from software heritage dataset. In: International Conference on Mining Software Repositories (MSR), pp. 11–15. ACM (2020). DOI 10.1145/3379597.3387512
10. Boldi, P., Pietri, A., Vigna, S., Zacchiroli, S.: Ultra-large-scale repository analysis via graph compression. In: International Conference on Software Analysis, Evolution and Reengineering (SANER), pp. 184–194. IEEE (2020). DOI 10.1109/SANER48275.2020.9054827
11. Bönisch, S., Brickenstein, M., Chrapary, H., Greuel, G., Sperber, W.: swmath - A new information service for mathematical software. In: MKM/Calculemus/DML, *Lecture Notes in Computer Science*, vol. 7961, pp. 369–373. Springer (2013)
12. Borgman, C.L., Wallis, J.C., Mayernik, M.S.: Who’s got the data? interdependencies in science and technology collaborations. In: Computer Supported Cooperative Work (CSCW), vol. 21, pp. 485–523 (2012). DOI 10.1007/s10606-012-9169-z
13. Childers, B.R., Fursin, G., Krishnamurthi, S., Zeller, A.: Artifact Evaluation for Publications (Dagstuhl Perspectives Workshop 15452). *Dagstuhl Reports* **5**(11), 29–35 (2016). DOI 10.4230/DagRep.5.11.29
14. Di Cosmo, R.: Archiving and referencing source code with software heritage. In: International Conference on Mathematical Software (ICMS), *Lecture Notes in Computer Science*, vol. 12097, pp. 362–373. Springer (2020). DOI 10.1007/978-3-030-52200-1_36

15. Di Cosmo, R., Gruenpeter, M., Marmol, B.P., Monteil, A., Romary, L., Sadowska, J.: Curated Archiving of Research Software Artifacts : lessons learned from the French open archive (HAL) (2019). Presented at the International Digital Curation Conference, submitted to IJDC
16. Di Cosmo, R., Gruenpeter, M., Zacchiroli, S.: Identifiers for digital objects: the case of software source code preservation. In: International Conference on Digital Preservation (iPRES) (2018). DOI 10.17605/OSF.IO/KDE56
17. Di Cosmo, R., Gruenpeter, M., Zacchiroli, S.: Referencing source code artifacts: a separate concern in software citation. *Computing in Science and Engineering* **22**(2), 33–43 (2020). DOI 10.1109/MCSE.2019.2963148
18. Di Cosmo, R., Lopez, J.B.G., Abramatic, J.F., Graf, K., Colom, M., Manghi, P., Harrison, M., Barborini, Y., Tenhunen, V., Wagner, M., Dalitz, W., Maassen, J., Martinez-Ortiz, C., Ronchieri, E., Yates, S., Schubotz, M., Candela, L., Fenner, M., Jeangirard, E.: Scholarly Infrastructures for Research Software. European Commission. Directorate General for Research and Innovation (2020). DOI 10.2777/28598
19. Di Cosmo, R., Zacchiroli, S.: Software Heritage: Why and how to preserve software source code. In: International Conference on Digital Preservation (iPRES) (2017)
20. Dyer, R., Nguyen, H.A., Rajan, H., Nguyen, T.N.: Boa: A language and infrastructure for analyzing ultra-large-scale software repositories. In: International Conference on Software Engineering (ICSE), pp. 422–431 (2013)
21. FIZ Karlsruhe GmbH: swMATH mathematical software. <https://swmath.org> (2023). Accessed 2023-04-15
22. French Ministry of Research and Higher Education: French National Plan for Open Science. <https://www.enseignementsup-recherche.gouv.fr/fr/le-plan-national-pour-la-science-ouverte-les-resultats-de-la-recherche-scientifique-ouverts-tous-49241> (2018)
23. French Ministry of Research and Higher Education: French second national plan for open science: Support and opportunities for universities’ open infrastructures and practices. <https://www.enseignementsup-recherche.gouv.fr/fr/le-plan-national-pour-la-science-ouverte-2021-2024-vers-une-generalisation-de-la-science-ouverte-en-48> (2021)
24. French Ministry of Research and Higher Education: Feuille de route nationale des infrastructures de recherche. <https://www.enseignementsup-recherche.gouv.fr/fr/feuille-de-route-nationale-des-infrastructures-de-recherche> (2022)
25. Heckman, J.: Varieties of selection bias. *The American Economic Review* **80**(2), 313–318 (1990)
26. Hinsen, K.: Software development for reproducible research. *Computing in Science and Engineering* **15**(4), 60–63 (2013). DOI 10.1109/MCSE.2013.91
27. Howison, J., Bullard, J.: 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 (2016). DOI 10.1002/asi.23538
28. Hunter, J.D.: Matplotlib: A 2D graphics environment. *Computing in Science & Engineering* **9**(3), 90–95 (2007). DOI 10.1109/MCSE.2007.55
29. Invenio: InvenioRDM. <https://inveniosoftware.org/products/rdm/>. Accessed 2023-04-15
30. Ivie, P., Thain, D.: Reproducibility in scientific computing. *ACM Computing Surveys* **51**(3), 63:1–63:36 (2018). DOI 10.1145/3186266
31. Lamprecht, A.L., Garcia, L., Kuzak, M., Martinez, C., Arcila, R., Martin Del Pico, E., Dominguez Del Angel, V., van de Sandt, S., Ison, J., Martinez, P.A., McQuilton, P., Valencia, A., Harrow, J., Psomopoulos, F., Gelpi, J.L., Chue Hong, N., Goble, C., Capella-Gutierrez, S.: Towards FAIR principles for research software. *Data Science* **3**(1), 37–59 (2020). DOI 10.3233/DS-190026
32. Ma, Y., Bogart, C., Amreen, S., Zaretski, R., Mockus, A.: World of code: an infrastructure for mining the universe of open source VCS data. In: International Conference on Mining Software Repositories (MSR), pp. 143–154. IEEE (2019). DOI 10.1109/MSR.2019.00031

33. Merkle, R.C.: A digital signature based on a conventional encryption function. In: *Advances in Cryptology (CRYPTO)*, pp. 369–378 (1987). DOI 10.1007/3-540-48184-2%5C.32
34. Messerschmitt, D.G., Szyperski, C.: *Software ecosystem: understanding an indispensable technology and industry*. MIT press (2003)
35. Mockus, A.: Amassing and indexing a large sample of version control systems: Towards the census of public source code history. In: *International Working Conference on Mining Software Repositories (MSR)*, pp. 11–20. IEEE (2009). DOI 10.1109/MSR.2009.5069476
36. nexB: ScanCode. <https://www.aboutcode.org/projects/scancode.html>. Accessed 2023-04-15
37. Openaire. <https://www.openaire.eu>. Accessed 2023-04-15
38. Pietri, A.: *Organizing the graph of public software development for large-scale mining. (organisation du graphe de développement logiciel pour l’analyse à grande échelle)*. Ph.D. thesis, University of Paris, France (2021)
39. Pietri, A., Rousseau, G., Zacchiroli, S.: Forking without clicking: On how to identify software repository forks. In: *International Conference on Mining Software Repositories (MSR)*, pp. 277–287. ACM (2020). DOI 10.1145/3379597.3387450
40. Pietri, A., Spinellis, D., Zacchiroli, S.: The Software Heritage graph dataset: public software development under one roof. In: *International Conference on Mining Software Repositories (MSR)*, pp. 138–142 (2019). DOI 10.1109/MSR.2019.00030
41. Quinlan, S., Dorward, S.: Venti: A new approach to archival data storage. In: *Conference on File and Storage Technologies (FAST)*. USENIX Association (2002). URL <https://www.usenix.org/conference/fast-02/venti-new-approach-archival-data-storage>
42. Rossi, D., Zacchiroli, S.: Geographic diversity in public code contributions: An exploratory large-scale study over 50 years. In: *International Conference on Mining Software Repositories (MSR)*, pp. 80–85. ACM (2022). DOI 10.1145/3524842.3528471
43. Rossi, D., Zacchiroli, S.: Worldwide gender differences in public code contributions (and how they have been affected by the COVID-19 pandemic). In: *International Conference on Software Engineering - Software Engineering in Society Track (ICSE-SEIS)*, pp. 172–183. ACM (2022). DOI 10.1109/ICSE-SEIS55304.2022.9794118
44. Rousseau, G., Di Cosmo, R., Zacchiroli, S.: Software provenance tracking at the scale of public source code. *Empirical Software Engineering* **25**(4), 2930–2959 (2020). DOI 10.1007/s10664-020-09828-5
45. Smith, A.M., Katz, D.S., Niemeyer, K.E.: Software citation principles. *PeerJ Computer Science* **2**, e86 (2016). DOI 10.7717/peerj-cs.86
46. Stewart, K., Odence, P., Rockett, E.: Software package data exchange (SPDX) specification. *IFOSS L. Rev.* **2**, 191 (2010)
47. Stodden, V., LeVeque, R.J., Mitchell, I.: Reproducible research for scientific computing: Tools and strategies for changing the culture. *Computing in Science and Engineering* **14**(4), 13–17 (2012). DOI 10.1109/MCSE.2012.38
48. T. Berners-Lee R. Fielding, L.M.: Uniform resource identifier (URI): Generic syntax. RFC 3986, RFC Editor (2005)
49. Wellenzohn, K., Böhlen, M.H., Helmer, S., Pietri, A., Zacchiroli, S.: Robust and scalable content-and-structure indexing. *The VLDB Journal* (2022). DOI 10.1007/s00778-022-00764-y
50. Wilkinson, M.D., Dumontier, M., Aalbersberg, I.J., Appleton, G., Axton, M., Baak, A., Blomberg, N., Boiten, J.W., da Silva Santos, L.B., Bourne, P.E., Bouwman, J., Brookes, A.J., Clark, T., Crosas, M., Dillo, I., Dumon, O., Edmunds, S., Evelo, C.T., Finkers, R., Gonzalez-Beltran, A., Gray, A.J., Groth, P., Goble, C., Grethe, J.S., Heringa, J., ’t Hoen, P.A., Hooft, R., Kuhn, T., Kok, R., Kok, J., Lusher, S.J., Martone, M.E., Mons, A., Packer, A.L., Persson, B., Rocca-Serra, P., Roos, M., van Schaik, R., Sansone, S.A., Schultes, E., Sengstag, T., Slater, T., Strawn, G., Swertz, M.A., Thompson, M., van der Lei, J., van Mulligen, E., Velterop, J., Waagmeester, A., Wittenburg, P., Wolstencroft, K., Zhao, J., Mons, B.: The FAIR guiding principles for scientific data management and stewardship. *Scientific Data* **3**(1), 160018 (2016). DOI 10.1038/sdata.2016.18

51. Zacchiroli, S.: Gender differences in public code contributions: a 50-year perspective. *IEEE Software* **38**(2), 45–50 (2021). DOI 10.1109/MS.2020.3038765
52. Zacchiroli, S.: A large-scale dataset of (open source) license text variants. In: *International Conference on Mining Software Repositories (MSR)*, pp. 757–761. ACM (2022). DOI 10.1145/3524842.3528491