Software Heritage

Source Code Analysis at the Scale of the World

Stefano Zacchiroli

Univ. Paris Diderot & Inria — zack@upsilon.cc, @zacchiro

24 April 2019

Gran Sasso Science Institute — L'Aquila, Italy

Software Heritage

THE GREAT LIBRARY OF SOURCE CODE

Outline

1. Source code in science
2. Software Heritage
3. Research challenges & roadmap
4. Conclusion

(Free) Software is everywhere

Software source code is *special*

Harold Abelson, Structure and Interpretation of Computer Programs

"Programs must be written for people to read, and only incidentally for machines to execute."

Quake III source code (excerpt)

```
float Q_rsqrt( float number )
{
    long i;
    float x2, y;
    const float threehalfs = 1.5F;

    x2 = number * 0.5F;
    y = number;
    i = * ( long * ) &y; // evil floating point bit level hacking
    i = 0x5f3759df - ( i >> 1 ); // what the fuck?
    y = * ( float * ) &i;
    y = y * ( threehalfs - ( x2 * y * y ) ); // 1st iteration
// y = y * ( threehalfs - ( x2 * y * y ) ); // 2nd iteration, this
can be removed

    return y;
}
```

Net. queue in Linux (excerpt)

```
/*
 * SFB uses two B[l][n] : L x N arrays of bins (L levels, N bins per level)
 * This implementation uses L = 8 and N = 16
 * This permits us to split one 32bit hash (provided per packet by rxhash or
 * external classifier) into 8 subhashes of 4 bits.
 */
#define SFB_BUCKET_SHIFT 4
#define SFB_NUMBUCKETS  (1 << SFB_BUCKET_SHIFT) /* N bins per Level */
#define SFB_BUCKET_MASK (SFB_NUMBUCKETS - 1)
#define SFB_LEVELS      (32 / SFB_BUCKET_SHIFT) /* L */

/* SFB algo uses a virtual queue, named "bin" */
struct sfb_bucket {
        u16             qlen; /* length of virtual queue */
        u16             p_mark; /* marking probability */
};
```

Len Shustek, Computer History Museum

"Source code provides a view into the mind of the designer."

Reproducibility in the digital age

For an experiment involving software, we need

- open access to the scientific article describing it
- open data sets used in the experiment
- source code of all the components
- environment of execution
- stable references between all this

Reproducibility in the digital age

For an experiment involving software, we need

- open access to the scientific article describing it
- open data sets used in the experiment
- source code of all the components
- environment of execution
- stable references between all this

Remark

The first two items are already widely discussed!

… what about *software*?

Software is knowledge

Software is *an essential component* of modern scientific research

Top 100 papers (Nature, October 2014)

> *[…] the vast majority describe experimental methods or software that have become essential in their fields.*

`http://www.nature.com/news/the-top-100-papers-1.16224`

Software and reproducibility

A fundamental question

How are we doing, regarding reproducibility, in *Software*?

The case of Computer Systems Research

A field with Computer experts … we have high expectations!
Christian Collberg set out to check them.

Measuring Reproducibility in Computer Systems Research

Long and detailed technical report, March 2014
`http://reproducibility.cs.arizona.edu/v1/tr.pdf`

Collberg's report from the trenches

Analysis of 613 papers

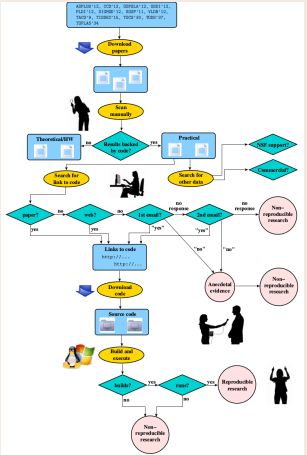
- 8 ACM conferences: ASPLOS'12, CCS'12, OOPSLA'12, OSDI'12, PLDI'12, SIGMOD'12, SOSP'11, VLDB'12
- 5 journals: TACO'9, TISSEC'15, TOCS'30, TODS'37, TOPLAS'34

all very practical oriented

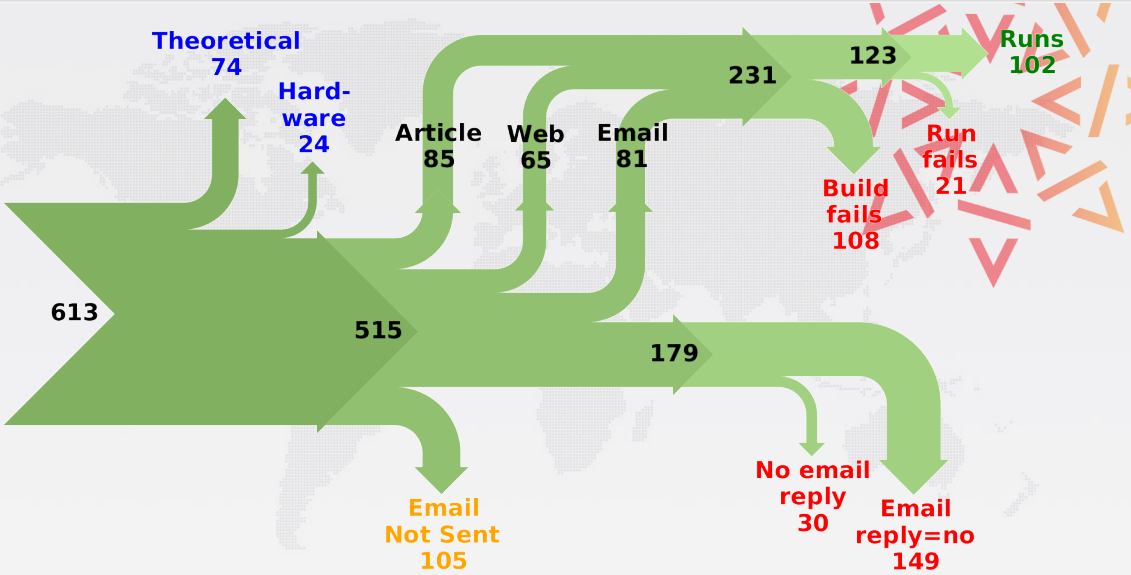
The basic question

can we get the code to build and run?

The workflow



The result


Theoretical
74
Hard-
ware
24
Article
85
Web
65
Email
81
231
123
Runs
102
Run
fails
21
Build
fails
108
613
515
179
Email
Not Sent
105
No email
reply
30
Email
reply=no
149

The reasons (or, “the dog ate my program”)

Why so much software fails to pass the test?

Many issues, nice anecdotes, and it finally boils down to

- *Availability*
- *Traceability*
- Environment
- Automation (do *you* use continuous integration?)
- Documentation
- Understanding (including free/open source software)

The reasons (or, “the dog ate my program”)

Why so much software fails to pass the test?

Many issues, nice anecdotes, and it finally boils down to

- *Availability*
- *Traceability*
- Environment
- Automation (do *you* use continuous integration?)
- Documentation
- Understanding (including free/open source software)

The first two are important *software preservation issues*

Yes, code is fragile:

it can be destroyed, and we can lose trace of it

Software is fragile

Like all digital information, FOSS is fragile

- inconsiderate and/or malicious code loss (e.g., Code Spaces)
- business-driven code loss (e.g., Gitorious, Google Code)
- for obsolete code: physical media decay (data rot)

Software is fragile

Like all digital information, FOSS is fragile

- inconsiderate and/or malicious code loss (e.g., Code Spaces)
- business-driven code loss (e.g., Gitorious, Google Code)
- for obsolete code: physical media decay (data rot)

Where is the archive…

where we go if (a repository on) GitHub or GitLab.com goes away?

Software is spread all around

Fashion victims

- many disparate development platforms
- a myriad places where distribution may happen
- projects tend to migrate from one place to another over time

Software is spread all around

Fashion victims

- many disparate development platforms
- a myriad places where distribution may happen
- projects tend to migrate from one place to another over time

Where is the place …

where we can find, track and search *all* source code?

Software lacks its own research infrastructure

A wealth of software research on crucial issues…

- safety, security, test, verification, proof
- software engineering, software evolution
- big data, machine learning, empirical studies

Software lacks its own research infrastructure

A wealth of software research on crucial issues…

- safety, security, test, verification, proof
- software engineering, software evolution
- big data, machine learning, empirical studies

If you study the stars, you go to Atacama…

… where is the *very large telescope* of source code?

Outline

1. Source code in science
2. Software Heritage
3. Research challenges & roadmap
4. Conclusion

Software Heritage

THE GREAT LIBRARY OF SOURCE CODE

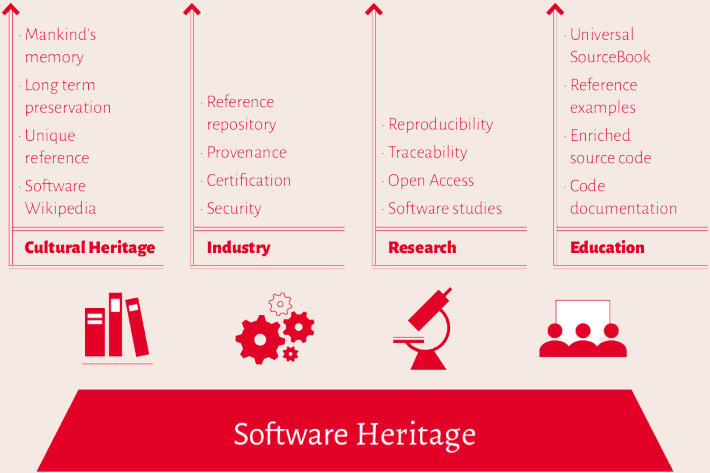
Our mission

Collect, preserve and share the *source code* of *all the software* that is publicly available.

Past, present and future

Preserving the past, *enhancing* the present, *preparing* the future.

We are working on the foundations



Archiving goals

Targets: VCS repositories & source code releases (e.g., tarballs)

We DO archive

- file content (= blobs)
- revisions (= commits), with full metadata
- releases (= tags), ditto
- where (origin) & when (visit) we found any of the above

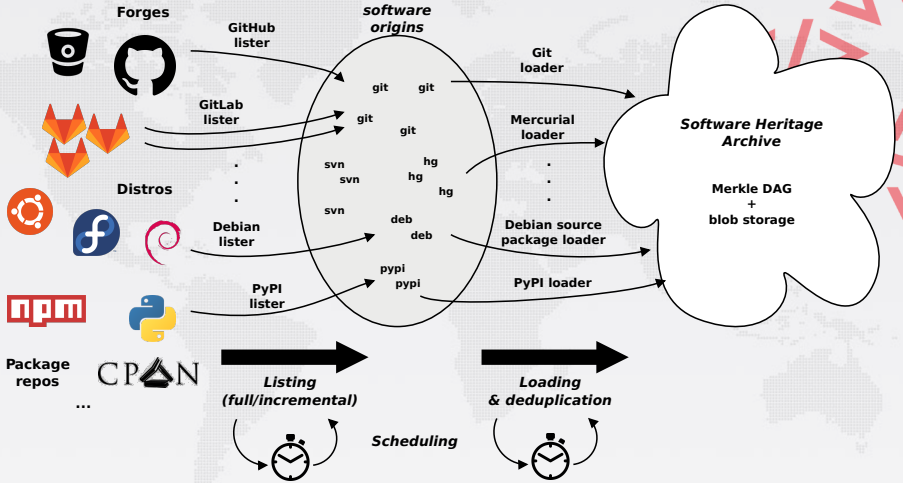
… in a VCS-/archive-agnostic canonical data model

We DON'T archive

- homepages, wikis
- BTS/issues/code reviews/etc.
- mailing lists

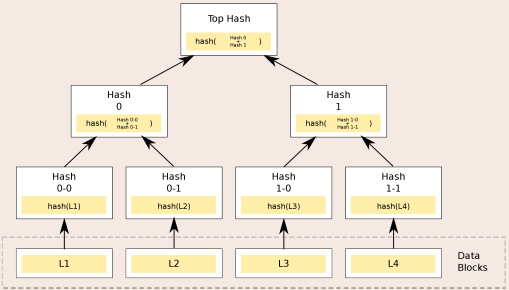
Long term vision: play our part in a *"semantic wikipedia of software"*

Data flow


Forges
GitHub lister
GitLab lister
Distros
Debian lister
PyPI lister
Package repos
CPAN
...
software origins
git
git
git
git
svn
svn
svn
hg
hg
hg
deb
deb
pypi
pypi
Git loader
Mercurial loader
Debian source package loader
PyPI loader
Software Heritage Archive
Merkle DAG
+
blob storage
Listing (full/incremental)
Scheduling
Loading & deduplication

Merkle trees

Merkle tree (R. C. Merkle, Crypto 1979)

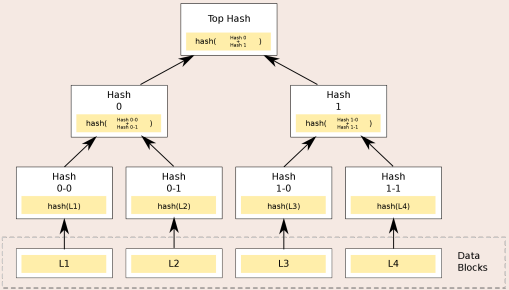


Combination of

- tree
- hash function

Merkle trees

Merkle tree (R. C. Merkle, Crypto 1979)



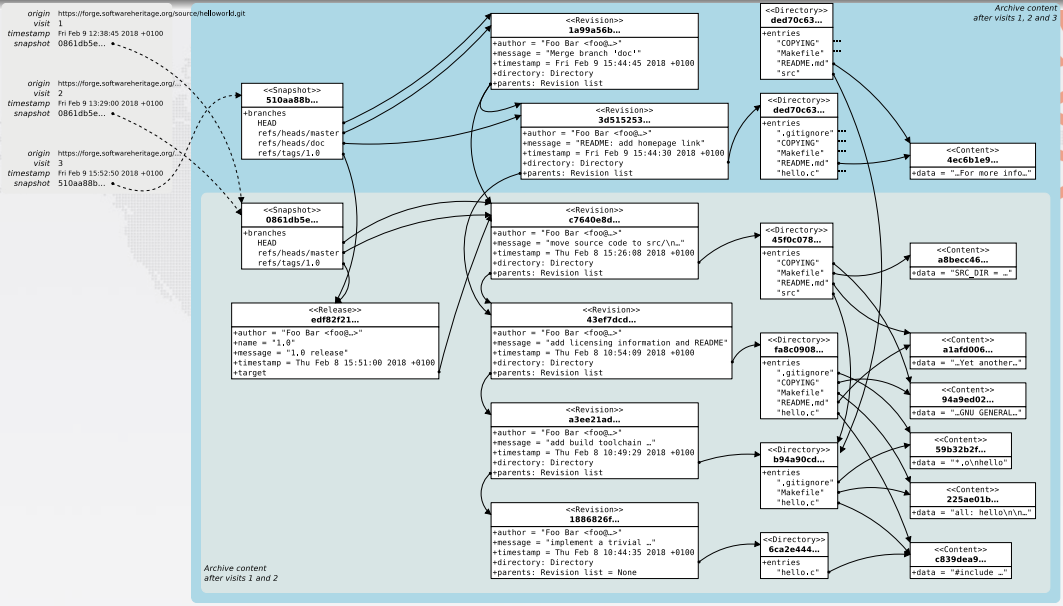
Combination of

- tree
- hash function

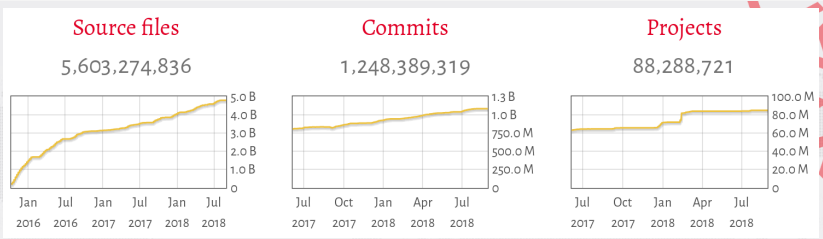
Classical cryptographic construction

- fast, parallel signature of large data structures
- widely used (e.g., Git, blockchains, IPFS, …)
- built-in deduplication

The archive: a (giant) Merkle DAG

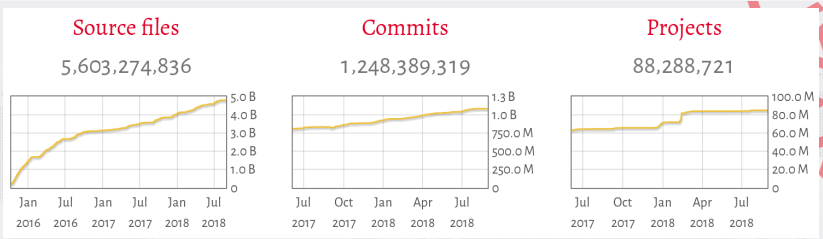


Archive coverage — archive.softwareheritage.org



GitHub debian GitLab Google code {P} python™ Package Index GITORIOUS GNU HAL archives-ouvertes.fr Inria inventeurs du monde numérique

Archive coverage — archive.softwareheritage.org



GitHub · debian · GitLab · Google code · {P} · python Package Index · GITORIOUS · GNU · HAL archives-ouvertes.fr · Inria inventeurs du monde numérique

- 200 TB (compressed) blobs, 6 TB database (as a graph: 10 B nodes + 100 B edges)
- The *richest* public source code archive, … and growing daily!

Web API

RESTful API to programmatically access the Software Heritage archive
`https://archive.softwareheritage.org/api/`

Features

- pointwise browsing of the archive
  - … snapshots ⟶ revisions ⟶ directories ⟶ contents …
- full access to the metadata of archived objects
- crawling information
  - *when have you last visited this Git repository I care about?*
  - *where were its branches/tags pointing to at the time?*

Endpoint index

`https://archive.softwareheritage.org/api/1/`

Bulk download

Vault service

- source code is thoroughly deduplicated within the Software Heritage archive
- bulk download of large artefacts (e.g., a Linux kernel release) requires collecting millions of objects
- the Software Heritage Vault cooks and caches source code bundles for bulk download needs

Tech bits

- RESTful API to request downloads, notifications, and monitoring
- `docs.softwareheritage.org/devel/swh-vault`

Web user interface

Browser-based interface to browse the Software Heritage archive
https://archive.softwareheritage.org/browse/

Features

- all REST API features, but good looking :-)
  - browsing: snapshots → revisions → directories → contents …
  - access to metadata and crawling information
- origin search, as full text indexing of origin URLs
- bulk download, via integration with the Vault

Outline

1. Source code in science
2. Software Heritage
3. Research challenges & roadmap
4. Conclusion

Realizing the "large telescope of source code" in practice

Requirements

- Availability: Software Heritage mirror, relatively up-to-date
- Efficiency: massive computing resources with fast access to the mirror
- Sustainability: pay-per-use or bring-your-own-computing

Challenges

- mirroring
- compression
- efficient processing
- experiments description language
- big code analysis (i.e., ML on source code)

… at the scale of the world!

Challenge #1: Mirroring

Thomas Jefferson, February 18, 1791

Let us save what remains: not by vaults and locks which fence them from the public eye and use in consigning them to the waste of time, but by such a multiplication of copies, as shall place them beyond the reach of accident.

Challenge #1: Mirroring

Thomas Jefferson, February 18, 1791

Let us save what remains: not by vaults and locks which fence them from the public eye and use in consigning them to the waste of time, but by such a multiplication of copies, as shall place them beyond the reach of accident.

Mirroring: the good

- big, but not *that* big
- append-only archive (in theory), easy to journal and incrementally update

Challenge #1: Mirroring (cont.)

Mirroring: the bad — Merkle DAGs with "holes"

- the world sucks: corrupted repositories, takedown notices, data losses
  - nodes can go missing at archival time or disappear later on
- top-level hash(es) no longer capture the full state of the archive

Open questions

- how do you capture such full state then?
  - by extension: how do you *timestamp* the archive?
- how do you efficiently check if something is to be re-archived?
- ultimately, what's your notion of having "fully archived" something?

Challenge #2: Compression — file contents

Storage figures

- file contents: ~400 TB (raw), ~200 TB compressed (1-by-1 content compression)
- median compressed size: 3 KB, i.e., a lot (~5 B) of very small files

Practical problem: scale-out object storages are not designed for this workload.

Challenge #2: Compression — file contents

Storage figures

- file contents: ~400 TB (raw), ~200 TB compressed (1-by-1 content compression)
- median compressed size: 3 KB, i.e., a lot (~5 B) of very small files

Practical problem: scale-out object storages are not designed for this workload.

Content compression

- low compression ratio (2x) with 1-by-1 compression
- typical Git/VCS packing heuristics do not work here, because contents occur in many different contexts
- early experiences with Rabin-style compression & co. were unsatisfactory
  - increased deduplication granularity (e.g., SLOCs) will likely suffer of the same problem
- research lead: object packing, using heuristics that maximize the chances of compressability (e.g., having *a* file name in common)

Challenge #2: Compression — graph

Storage figures

- Merkle DAG: ~10 B nodes, ~100 B edges
  - breakdown by node type: ~40% contents, ~40% directories, ~10% commits

Still outside the limits of what (cheaply & trivially) fits in RAM.

Challenge #2: Compression — graph

Storage figures

- Merkle DAG: ~10 B nodes, ~100 B edges
  - breakdown by node type: ~40% contents, ~40% directories, ~10% commits

Still outside the limits of what (cheaply & trivially) fits in RAM.

Graph compression

- related: Web graph compression techniques in the style of, e.g.:

  Paolo Boldi, Sebastiano Vigna
  The WebGraph framework I: compression techniques
  WWW 2004

- challenges: no canonical identifiers with good locality properties, different node types/subgraphs, live updates
- research lead: graph topology characterization

Challenge #3: Efficient processing — graph

Big graphs calls for scale-out processing, e.g.:

Malewicz, Grzegorz, et al.
Pregel: a system for large-scale graph processing
ACM SIGMOD 2010

Roy, Amitabha, et al.
Chaos: Scale-out graph processing from secondary storage
ACM SOSP 2015

support very large graphs, exploiting topological characteristics (e.g., small world)

Software Heritage graph characteristics

- scale-free, not small world (?)
- connected components size distribution unclear yet

Approach: topological characterization

- if amenable to scale-out ⟶ distribution
- if not ⟶ scale-up using graph compression

Challenge #3: Efficient processing — file contents

Code search: a natural need

Find code snippets for reuse, find reverse dependencies for maintenance, impact analysis, etc.

Approaches vary with their level of code "understanding"

- full-text search: treat source code as text
- symbol extraction (e.g., ctags)
- AST search (language-specific)

Challenge #3: Efficient processing — file contents (cont.)

Russ Cox

Regular Expression Matching with a Trigram Index or How Google Code Search Worked

2012, https://swtch.com/ rsc/regexp/regexp4.html

Use regexs, Luke!

- build an inverted index of trigrams for all source code files
- compile regex to trigrams, find *potential* matches
- `grep` each potential match using map reduce

Application

- Google Code Search (now gone)
- Debsources (`https://sources.debian.org`)
  - scale up to 1 B SLOCs (Debian development branch)

Challenge

Scale it up to Software Heritage, several orders of magnitude later

Challenge #4: Experiment definition

Problem

How would researchers define their experiments on the dataset?

- corpus selection (code? history? sampling? etc.)
- how will they run their tools in a fully-deduplicated world?
- job scheduling (well-known HPC problem)

Related work (at a smaller scale)

Robert Dyer et al.
Boa: Ultra-large-scale software repository and source-code mining
ACM TOSEM 25.1 (2015): 7

Approach

formal languages, DSLs, tool adapters/porting

Challenge #5: Machine Learning on code

Big Code = Big Data + Source Code

Popular research trends: NLP on code / SBSE using ML, e.g.:

Miltiadis Allamanis et al.
A survey of machine learning for big code and naturalness
ACM Computing Surveys

Xiaodong Gu, Hongyu Zhang, Sunghun Kim
Deep Code Search
ICSE 2018

Common assumptions: AST availability, "small" datasets (e.g., random GitHub samples)

Example (Universal programming language detection)

- programming language detection at the scale of Software Heritage
- no AST, only bytes (not characters!)
- supervised or unsupervised (would be best for language evolution)

Outline

1. Source code in science
2. Software Heritage
3. Research challenges & roadmap
4. Conclusion

Wrapping up

- Software Heritage archives all software source code with its development history.
- It is a major endeavor that benefits society, science, and industry.
- For computer scientists, it is a gold mine of research opportunities. Wanna join?

References

Antoine Pietri, Diomidis Spinellis, Stefano Zacchiroli
The Software Heritage graph dataset: public software development under one roof
MSR 2019: Mining Software Repositories, IEEE

Jean-François Abramatic, Roberto Di Cosmo, Stefano Zacchiroli
Building the Universal Archive of Source Code
Communication of the ACM, October 2018

Roberto Di Cosmo, Morane Gruenpeter, Stefano Zacchiroli
Identifiers for Digital Objects: the Case of Software Source Code Preservation
iPRES 2018: Intl. Conf. on Digital Preservation

Roberto Di Cosmo, Stefano Zacchiroli
Software Heritage: Why and How to Preserve Software Source Code
iPRES 2017: Intl. Conf. on Digital Preservation