2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)

The Software Heritage Graph Dataset: Public Software Development Under One Roof

Antoine Pietri
Inria
France
antoine.pietri@inria.fr

Diomidis Spinellis†
Athens University of Economics and Business
Greece
dds@aueb.gr

Stefano Zacchiroli
University Paris Diderot and Inria
France
zack@irif.fr

***Abstract*—Software Heritage is the largest existing public archive of software source code and accompanying development history: it currently spans more than five billion unique source code files and one billion unique commits, coming from more than 80 million software projects.**

This paper introduces the *Software Heritage graph dataset*: a fully-deduplicated Merkle DAG representation of the Software Heritage archive. The dataset links together file content identifiers, source code directories, Version Control System (VCS) commits tracking evolution over time, up to the full states of VCS repositories as observed by Software Heritage during periodic crawls. The dataset's contents come from major development forges (including GitHub and GitLab), FOSS distributions (e.g., Debian), and language-specific package managers (e.g., PyPI). Crawling information is also included, providing timestamps about when and where all archived source code artifacts have been observed in the wild.

The Software Heritage graph dataset is available in multiple formats, including downloadable CSV dumps and Apache Parquet files for local use, as well as a public instance on Amazon Athena interactive query service for ready-to-use powerful analytical processing.

Source code file contents are cross-referenced at the graph leaves, and can be retrieved through individual requests using the Software Heritage archive API.

***Index Terms*—mining software repositories, source code, dataset, open source software, free software, digital preservation, development history graph**

I. Introduction

Public software development, i.e., the practice of collaboratively developing software using collaborative development platforms (also known as "forges" [1]) such as GitHub or GitLab is now well established. Further source code distribution mechanisms, by the means of GNU/Linux distributions and language-specific package managers [2], [3] are also becoming more and more popular, realizing in practice the half-century-old vision of off-the-shelf software components [4], [5].

The extensive public availability of software source code artifacts (source code files, commits, release information, etc.) is a significant asset for mining software repositories and, more generally, empirical software engineering research. The topics of corresponding studies range from the study of code clones [6]–[8] to automated vulnerability detection and repair [9]–[11]; and from code recommenders [12], [13] to software licence analysis and license compliance [14], [15]. However, many research studies are still being conducted on significantly smaller *subsets* of the *entire corpus of publicly available source code artifacts*.

The Software Heritage project [16], [17] aims at fixing this gap, by collecting, preserving, and sharing the entire body of publicly available software source code, together with the associated development history, as it is captured by modern version control systems (VCS) [18].

In this paper we introduce the *Software Heritage Graph Dataset*, a graph representation of all the source code artifacts archived by Software Heritage. The graph is a fully deduplicated Merkle DAG [19] that links together: source code file contents, source trees (directories) containing them, VCS commits and releases, up to crawling information indicating when and where (URLs) a given VCS repository state (which in turn binds all other artifacts together) has been observed.

The dataset captures the state of the Software Heritage archive on September 25th 2018, spanning a full mirror of Github and GitLab.com, the Debian distribution, Gitorious, Google Code, and the PyPI repository. Quantitatively it corresponds to 5 billion unique file contents and 1.1 billion unique commits, harvested from more than 85 million software origins (see Section III for more detailed figures).

Dataset availability: The Software Heritage graph is encoded in the dataset as a set of relational tables—roughly, one for each node type; see Section III for details—that can be queried using SQL. The dataset is available in three alternative formats, catering for different use cases:

- a Postgres [20] **database dump** in CSV format (for the data) and DDL queries (for recreating the DB schema): this format is best for local processing on a single server or high-end workstation;
- a set of **Apache Parquet files** [21] suitable for loading into columnar storage and scale-out processing solutions, e.g., Apache Spark [22];
- a public dataset on **Amazon Athena** [23], where the sample queries of Section IV can be tried live.

The first two formats are available for download (≈1 TB each) from Zenodo at https://zenodo.org/record/2583978, (doi:10.5281/zenodo.2583978). The Athena dataset [1] can also

† This work has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825328.

[1] https://registry.opendata.aws/software-heritage/

2574-3864/19/$31.00 ©2019 IEEE
DOI 10.1109/MSR.2019.00030

be queried from the AWS console by following the instructions in the `README.md` file.

II. Data Gathering

The Software Heritage Graph Dataset has been assembled as part of the regular crawling activities of Software Heritage, dating back to 2015. Due to the long-term archival nature of Software Heritage, the dataset is not entirely reproducible, because the archived source code artifacts might have disappeared from the original hosting place: a VCS might have been deleted, a commit or branch rewritten (e.g., using `git rebase`), a package removed from a distribution.

What we list below are hence the steps needed to recreate the dataset from scratch, archiving what is available *today* from the included software distribution places.

As a preliminary step one should setup local storage. As the official getting started guide for Software Heritage developers covers this in detail, we refer readers to it.[2]

Software Heritage tracks a curated list of software distribution places. For each different type of distribution place there exists a "lister" component, capable of enumerating all software origins (VCS repository, source package, etc.) hosted there. The next step to recreate the dataset is hence to run the lister for each tracked software distribution place, that is: GitHub, GitLab.com, Debian, and PyPI. (Gitorious and Google Code having disappeared, they cannot be re-archived today). All listers are available from the Software Heritage forge.[3][4]

Then, all software origins enumerated by running the listers should be crawled, retrieving all the source code artifacts found there (file contents, commits, directories, releases, etc.). For each origin having a different type (Git repository, Debian package, PyPI package, etc.), this step boils down to running the matching "loader" component on the origin URL. Be warned that as of this writing Software Heritage tracks more than 80 million origins, and therefore crawling all of them might require months, if not years, and significant computing and bandwidth resources. By default loaders also store file *contents*, which are not included in the graph dataset and would require ≈200 TB of storage space as of this writing; the storage layer can be configured to not store contents, reducing the storage fingerprint (the database storing the graph will still need ≈5 TB of fast storage, e.g., SSDs).

At the end of the process, database dumps can be obtained by running `pg_dump`; Parquet files by using PyArrow. A public dataset on Athena can be obtained by importing the Parquet files on Amazon S3.

III. Data Model

A particular property of software development is that source code artifacts are massively duplicated across hosts and projects [17]. In order to enable tracking of software artifacts across projects, and reduce the storage size of the graph, the Software Heritage Graph Dataset is stored as a Merkle directed acyclic graph (DAG) [19]. By using persistent, cryptographically-strong hashes as node identifiers [24], the graph can be deduplicated by sharing all identical nodes.

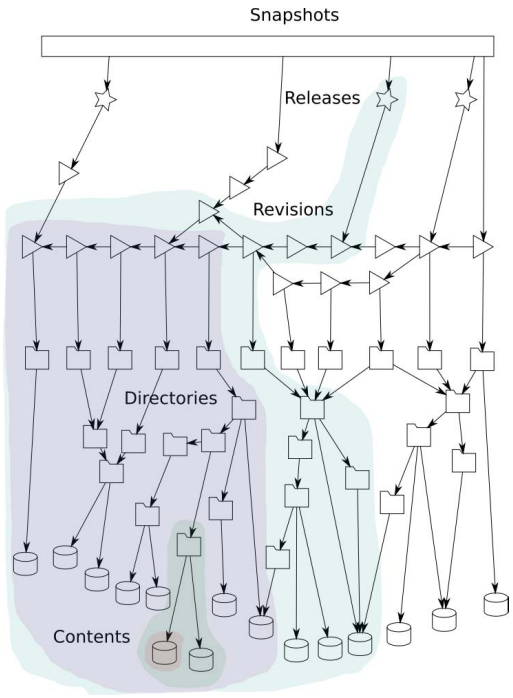

Fig. 1: Data model: a uniform Merkle DAG containing source code artifacts and their development history

As shown in Fig. 1, the Software Heritage DAG is organized in five logical layers, which we describe 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). The dataset contains cryptographic checksums for all contents though, that can be used to retrieve the actual files from any Software Heritage mirror using a Web API[5] and cross-reference files encountered in the wild, including other datasets.

Directories are lists of named directory entries. Each entry can point to content objects ("file entries"), revisions ("revision entries"), or other directories ("directory entries").

Revisions (or "commits") are point-in-time captures 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.

Releases (or "tags") are revisions that have been marked as noteworthy with a specific, usually mnemonic, name (e.g., a version number). Each release points to a revision and might include additional descriptive metadata.

Snapshots are point-in-time captures of the full state of a project development repository. As revisions capture the

[2] https://docs.softwareheritage.org/devel/getting-started.html

[3] https://forge.softwareheritage.org/source/swh-lister/

[4] Specific versions of these software components are available as part of the dataset, in `swh-environment.tar.gz`.

[5] https://archive.softwareheritage.org/api/

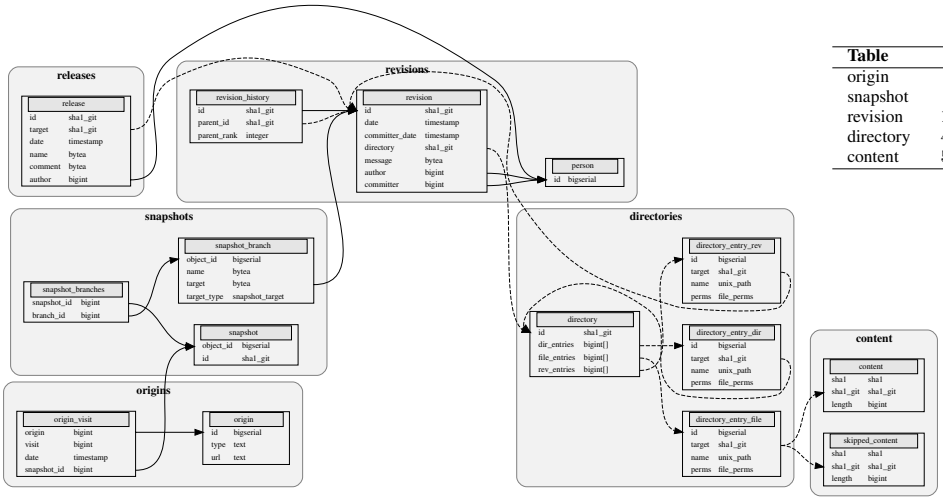

Table	# of objects
origin	85 143 957
snapshot	57 144 153
revision	1 125 083 793
directory	4 422 303 776
content	5 082 263 206

Fig. 2: Simplified schema of the Software Heritage Graph Dataset and the number of artifacts in it

Listing 1: Most frequent file name

```
SELECT FROM_UTF8(name, '?') AS name,
  COUNT(DISTINCT target) AS cnt
FROM directory_entry_file
GROUP BY name
ORDER BY cnt DESC
LIMIT 1;
```

state of a single development line (or "branch"), snapshots capture the state of *all* branches in a repository and allow to deduplicate unmodified forks across the archive.

Deduplication happens implicitly, automatically tracking byte-identical clones. If a file or a directory is copied to another project, both projects will point to the same node in the graph. Similarly for revisions, if a project is forked on a different hosting platform, the past development history will be deduplicated as the same nodes in the graph. Likewise for snapshots, each "fork" that creates an identical copy of a repository on a code host, will point to the same snapshot node. By walking the graph bottom-up, it is hence possible to find all *occurrences* of a source code artifact in the archive (e.g., all projects that have ever shipped a specific file content).

The Merkle DAG is encoded in the dataset as a set of relational tables. In addition to the nodes and edges of the graph, the dataset also contains crawling information, as a set of triples capturing where (an origin URL) and when (a timestamp) a given snapshot has been encountered. A simplified view of the corresponding database schema is shown in Fig. 2; the full schema is available as part of the dataset distribution. The sizes of the dataset's most notable tables are shown in the legend appearing in the Figure's top right.

IV. Sample Queries and Research Questions

To further illustrate the dataset's affordances and as motivating examples regarding the research possibilities it opens, below are some sample SQL queries that can be executed with the dataset on AWS Athena.

Listing 2: Most common commit operations

```
SELECT COUNT(*) AS c, word
FROM
  (SELECT LOWER(REGEXP_EXTRACT(FROM_UTF8(
        message), '^\w+')) AS word
  FROM revision )
WHERE word != ''
GROUP BY  word
ORDER BY  COUNT(*) DESC LIMIT 20;
```

Listing 1 shows a simple query that finds the most frequent file name across all the revisions. The result, obtained by scanning 151GB in 3′40″, is `index.html`, which occurs in the dataset 182 million times.

As an example of a query useful in software evolution research, consider the Listing 2. It is based on the convention dictating that commit messages should start with a summary expressed in the imperative mood [25, 3.3.2.1]. Based on that idea, the query uses a regular expression to extract the first word of each commit message and then tallies words by frequency. By scanning 37 GB in 30″ it gives us that commits concern the following common actions ordered by descending order of frequency: *add, fix, update, remove, merge, initial, create*. (We had to manually stem some verbs, because the most recent version of the Presto query engine, which provides a built-in stemming function, is not yet available on Athena.)

SQL queries can also be used to express more complex tasks. Consider the research hypothesis that weekend work on open source projects is decreasing over the years as evermore development work is done by companies rather than volunteers. The corresponding data can be obtained by finding the ratio between revisions committed on the weekends of each year and the total number of that year's revisions (see Listing 3). The results, obtained by scanning 14 GB in 7″ are inconclusive, and point to the need for further analysis, which is left as an exercise to the reader.

The provided dataset forms a graph, which can be dif-

Listing 3: Ratio of commits performed during each year's weekends

```
WITH revision_date AS
  (SELECT FROM_UNIXTIME(date / 1000000) AS date
  FROM revision)
SELECT yearly_rev.year AS year,
  CAST(yearly_weekend_rev.number AS DOUBLE)
  / yearly_rev.number * 100.0 AS weekend_pc
FROM
  (SELECT YEAR(date) AS year, COUNT(*) AS number
  FROM revision_date
  WHERE YEAR(date) BETWEEN 1971 AND 2018
  GROUP BY  YEAR(date) ) AS yearly_rev
JOIN
  (SELECT YEAR(date) AS year, COUNT(*) AS number
  FROM revision_date
  WHERE DAY_OF_WEEK(date) >= 6
      AND YEAR(date) BETWEEN 1971 AND 2018
  GROUP BY  YEAR(date) ) AS yearly_weekend_rev
  ON yearly_rev.year = yearly_weekend_rev.year
ORDER BY  year DESC;
```

Listing 4: Average number of a revision's parents

```
SELECT AVG(fork_size)
FROM (SELECT COUNT(*) AS fork_size
        FROM revision_history
        GROUP BY  parent_id);
```

ficult query with SQL. Therefore, questions associated with the graph's characteristics, such as closeness, distance, and centrality, will require the use of other query mechanisms. Yet, interesting metrics can be readily obtained by limiting scans to specific cases, such as merge commits. As an example, Listing 4 calculates the average number of parents of each revision (1.088, after scanning 23 GB in 22″) by grouping revisions by their parent identifier. Such queries can be used to examine in depth the characteristics of merge operations.

Although the performance of Athena can be impressive, there are cases where the available memory resources will be exhausted causing an expensive query to fail. This typically happens when joining two equally large tables consisting of hundreds of millions of records. This restriction can be overcome by sampling the corresponding tables. Listing 5 demonstrates such a case. The objective here is to determine the modularity at the level of files among diverse programming languages, by examining the size of popular file types. The query joins two 5 billion row tables: the file names and the content metadata. To reduce the number of joined rows a 1% sample of the rows is processed, thus scanning 317 GB in 1′20″. The order of the resulting language files (JavaScript>C>C++>Python>PHP>C#> Ruby) indicates that, with the exception of JavaScript, languages offering more abstraction facilities are associated with smaller source code files.

V. Limitations and Extensions

As discussed in Section II, the dataset is not fully reproducible. This is a limitation, but one that is very hard to avoid for datasets originating from long-term archival.

The dataset cannot claim full coverage of the entire software commons, and several forges and package repositories are still missing from it. Still, it is the largest dataset of its kind (by several order of magnitude). Also, to mitigate this limitation

Listing 5: Average size of the most popular file types

```
SELECT suffix,
  ROUND(COUNT(*) * 100 / 1e6) AS Million_files,
  ROUND(AVG(length) / 1024) AS Average_k_length
FROM
  (SELECT length, suffix
  FROM
    -- File length in joinable form
    (SELECT TO_BASE64(sha1_git) AS sha1_git64, length
    FROM content ) AS content_length
  JOIN
  -- Sample of files with popular suffixes
  (SELECT target64, file_suffix_sample.suffix AS suffix
  FROM
    -- Popular suffixes
    (SELECT suffix FROM (
      SELECT REGEXP_EXTRACT(FROM_UTF8(name),
       '\.[^.]+$') AS suffix
    FROM directory_entry_file) AS file_suffix
    GROUP BY  suffix
    ORDER BY  COUNT(*) DESC LIMIT 20 ) AS pop_suffix
  JOIN
    -- Sample of files and suffixes
    (SELECT TO_BASE64(target) AS target64,
      REGEXP_EXTRACT(FROM_UTF8(name),
  '\.[^.]+$') AS suffix
    FROM directory_entry_file TABLESAMPLE BERNOULLI(1))
    AS file_suffix_sample
  ON file_suffix_sample.suffix = pop_suffix.suffix)
  AS pop_suffix_sample
  ON pop_suffix_sample.target64 = content_length.sha1_git64)
GROUP BY  suffix
ORDER BY  AVG(length) DESC;
```

we plan to periodically publish new versions of it, as the Software Heritage archive extends its coverage.

As a first future extension we plan to include actual file contents as part of the dataset. This is currently impractical due to the storage size (≈200 TB), but we are exploring cloud storage with bring-your-own-compute service model solutions.

As further extensions, it would be useful to combine the Software Heritage Graph Dataset with neighboring datasets that contain non-source-code development artifacts, such as issues, code review discussions, wiki pages, etc. We are exploring an integrated dataset with GHTorrent [26], even thought that would narrow the scope to GitHub "only".

VI. Related Work

To the best of the authors' knowledge no existing dataset contains a comparable amount of data about source code artifacts extracted from public software development.

GHTorrent [26] covers GitHub and includes non-source-code artifacts (e.g., issues, pull requests, etc.), but lacks other software distribution places (e.g., GitLab, Debian, PyPI). Also, no deduplication applies to GHTorrent, making cloning tracking more challenging. CodeFeedr [27] is a natural evolution of GHTorrent, but it is oriented toward expanding coverage for additional sources of non-source-code artifacts.

The Debsources dataset [28] contains a wealth of information about Debian releases over several decades, but stops at the release granularity (rather than commits) and is at least three orders of magnitude smaller than the present dataset.

Acknowledgements: We thank the Amazon AWS Public Dataset Program team for hosting the dataset. We thank Roberto Di Cosmo for insightful discussions on the dataset and for providing the key resources that made this work possible.

REFERENCES

[1] Megan Squire. The lives and deaths of open source code forges. In Lorraine Morgan, editor, *Proceedings of the 13th International Symposium on Open Collaboration, OpenSym 2017, Galway, Ireland, August 23-25, 2017*, pages 15:1–15:8. ACM, 2017.

[2] Riivo Kikas, Georgios Gousios, Marlon Dumas, and Dietmar Pfahl. Structure and evolution of package dependency networks. In Jesús M. González-Barahona, Abram Hindle, and Lin Tan, editors, *Proceedings of the 14th International Conference on Mining Software Repositories, MSR 2017, Buenos Aires, Argentina, May 20-28, 2017*, pages 102–112. IEEE Computer Society, 2017.

[3] Pietro Abate, Roberto Di Cosmo, Louis Gesbert, Fabrice Le Fessant, Ralf Treinen, and Stefano Zacchiroli. Mining component repositories for installability issues. In Massimiliano Di Penta, Martin Pinzger, and Romain Robbes, editors, *12th IEEE/ACM Working Conference on Mining Software Repositories, MSR 2015, Florence, Italy, May 16-17, 2015*, pages 24–33. IEEE Computer Society, 2015.

[4] M Douglas McIlroy, J Buxton, Peter Naur, and Brian Randell. Mass-produced software components. In *Proceedings of the 1st International Conference on Software Engineering, Garmisch Pattenkirchen, Germany*, pages 88–98, 1968.

[5] Diomidis Spinellis. Cracking software reuse. *IEEE Software*, 24(1):12–13, January/February 2007.

[6] Jeffrey Svajlenko and Chanchal Kumar Roy. Fast and flexible large-scale clone detection with cloneworks. In Sebastián Uchitel, Alessandro Orso, and Martin P. Robillard, editors, *Proceedings of the 39th International Conference on Software Engineering, ICSE 2017, Buenos Aires, Argentina, May 20-28, 2017 - Companion Volume*, pages 27–30. IEEE Computer Society, 2017.

[7] Yuichi Semura, Norihiro Yoshida, Eunjong Choi, and Katsuro Inoue. Ccfindersw: Clone detection tool with flexible multilingual tokenization. In Jian Lv, He Jason Zhang, Mike Hinchey, and Xiao Liu, editors, *24th Asia-Pacific Software Engineering Conference, APSEC 2017, Nanjing, China, December 4-8, 2017*, pages 654–659. IEEE Computer Society, 2017.

[8] Suresh Thummalapenta, Luigi Cerulo, Lerina Aversano, and Massimiliano Di Penta. An empirical study on the maintenance of source code clones. *Empirical Software Engineering*, 15(1):1–34, 2010.

[9] Frank Li and Vern Paxson. A large-scale empirical study of security patches. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, pages 2201–2215, New York, NY, USA, 2017. ACM.

[10] Gustavo Grieco, Guillermo Luis Grinblat, Lucas Uzal, Sanjay Rawat, Josselin Feist, and Laurent Mounier. Toward large-scale vulnerability discovery using machine learning. In *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy*, CODASPY '16, pages 85–96, New York, NY, USA, 2016. ACM.

[11] Matias Martinez and Martin Monperrus. Mining software repair models for reasoning on the search space of automated program fixing. *Empirical Software Engineering*, 20(1):176–205, 2015.

[12] T. Zimmermann, R. Premraj, and A. Zeller. Predicting defects for eclipse. In *Predictor Models in Software Engineering, 2007. PROMISE'07: ICSE Workshops 2007. International Workshop on*, pages 9–9, May 2007.

[13] Thomas Zimmermann, Peter Weißgerber, Stephan Diehl, and Andreas Zeller. Mining version histories to guide software changes. In Anthony Finkelstein, Jacky Estublier, and David S. Rosenblum, editors, *26th International Conference on Software Engineering (ICSE 2004), 23-28 May 2004, Edinburgh, United Kingdom*, pages 563–572. IEEE Computer Society, 2004.

[14] Yuhao Wu, Yuki Manabe, Tetsuya Kanda, Daniel M. Germán, and Katsuro Inoue. Analysis of license inconsistency in large collections of open source projects. *Empirical Software Engineering*, 22(3):1194–1222, 2017.

[15] C. Vendome. A large scale study of license usage on github. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 2, pages 772–774, May 2015.

[16] Jean-François Abramatic, Roberto Di Cosmo, and Stefano Zacchiroli. Building the universal archive of source code. *Communications of the ACM*, 61(10):29–31, October 2018.

[17] Roberto Di Cosmo and Stefano Zacchiroli. Software Heritage: Why and how to preserve software source code. In *iPRES 2017: 14th International Conference on Digital Preservation*, 2017.

[18] Diomidis Spinellis. Version control systems. *IEEE Software*, 22(5):108–109, 2005.

[19] Ralph C. Merkle. A digital signature based on a conventional encryption function. In Carl Pomerance, editor, *Advances in Cryptology - CRYPTO '87, A Conference on the Theory and Applications of Cryptographic Techniques, Santa Barbara, California, USA, August 16-20, 1987, Proceedings*, volume 293 of *Lecture Notes in Computer Science*, pages 369–378. Springer, 1987.

[20] Michael Stonebraker and Greg Kemnitz. The POSTGRES next generation database management system. *Communications of the ACM*, 34(10):78–92, 1991.

[21] Apache Parquet. https://parquet.apache.org, 2019.

[22] Matei Zaharia, Reynold S Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J Franklin, et al. Apache Spark: a unified engine for big data processing. *Communications of the ACM*, 59(11):56–65, 2016.

[23] Amazon Athena. https://aws.amazon.com/athena/, 2019.

[24] Roberto Di Cosmo, Morane Gruenpeter, and Stefano Zacchiroli. Identifiers for digital objects: the case of software source code preservation. In *iPRES 2018: 15th International Conference on Digital Preservation*, 2018.

[25] Michael Freeman. *Programming Skills for Data Science: Start Writing Code to Wrangle, Analyze, and Visualize Data with R*. Addison-Wesley, Boston, 2019.

[26] Georgios Gousios and Diomidis Spinellis. GHTorrent: Github's data from a firehose. In Michele Lanza, Massimiliano Di Penta, and Tao Xie, editors, *9th IEEE Working Conference of Mining Software Repositories, MSR 2012, June 2-3, 2012, Zurich, Switzerland*, pages 12–21. IEEE Computer Society, 2012.

[27] Enrique Larios Vargas, Joseph Hejderup, Maria Kechagia, Magiel Bruntink, and Georgios Gousios. Enabling real-time feedback in software engineering. In Andrea Zisman and Sven Apel, editors, *Proceedings of the 40th International Conference on Software Engineering: New Ideas and Emerging Results, ICSE (NIER) 2018, Gothenburg, Sweden, May 27 - June 03, 2018*, pages 21–24. ACM, 2018.

[28] Matthieu Caneill, Daniel M. German, and Stefano Zacchiroli. The Debsources dataset: Two decades of free and open source software. *Empirical Software Engineering*, 22:1405–1437, June 2017.