50 Years of Programming Language Evolution through the Software Heritage looking glass

Adèle Desmazières*, Roberto Di Cosmo†, and Valentin Lorentz‡

*Sorbonne University, France, adesmaz.pro@gmail.com

†Inria and University Paris Cité, France, roberto@dicosmo.org

‡Inria Foundation, Inria, France, valentin.lorentz@inria.fr

***Abstract*—Programming languages have evolved rapidly over the past five decades, reflecting broader shifts in software development practices and technological advances. Early on, entities like the U.S. Department of Defense recognized the challenges posed by diverse programming languages, leading to initiatives such as the Ada programming language. Since then, indexes like Tiobe, RedMonk, and Open Hub have attempted to track language popularity, though their metrics provide only a snapshot view, and most of them do not make available their data. We show that Software Heritage, the largest public archive of source code, makes it now possible, and easy, to address this question in a comprehensive, transparent and reproducible manner through its unified dataset, which includes over 20 billion source files and 4 billion commits. As a result of our study, we have created a dataset and pipeline that allows to analyze five decades of programming language trends, by measuring the programming activity as seen in the Software Heritage archive, confirming trends in language adoption, shifts in popularity, and significant transitions linked to technological changes. The comparison with the existing indexes shows rather good alignment for the first positions in the rankings, but differences emerge down the line, as programmer activity, and language popularity are not necessary aligned. To facilitate further research on programming language evolution, we publish the whole software pipeline as Open Source, and make available the full dataset, that will be updated bi-annually.**

I. Introduction

Over the last 50 years, the evolution of programming languages has mirrored the growth and diversification of software development practices, at an astonishing pace. Already in the late seventies, the U.S. Department of Defense (DoD) the challenge of managing over 450 different programming languages in use across various projects [1], and as a consequence funded the development of the Ada programming language as a standard for its contractors.

The importance of understanding language development and adoption has been recognized since that same decade, as evidenced by the *History of Programming Languages* conference (HOPL) [2] held in Los Angeles in 1978, and that spawned the comprehensive catalog *HOPL.info*, which documents over 8000 programming languages, illustrating the vast diversity in the programming landscape [3].

With the advent of the Internet, a sustained interest emerged in tracking how programming languages are adopted and evolve over time, with popularity indexes like *Tiobe* [4], *RedMonk* [5], and the former *Ohloh* (now *Open Hub*) [6]. These indexes provide a snapshot of language popularity based on different metrics: Tiobe assigns a score to each language by counting the number of search results for "[language name] programming" across several search engines, while PYPL measures tutorial search popularity on Google Trends; *Open Hub* publishes statistics of progamming languages based on project activity by counting the number of commits, which aligns more closely with public software repositories but does not provide access to historical data; RedMonk provides a combined score based on GitHub projects and StackOverflow activity but still focuses on more recent trends.

The proliferation of open-source repositories and

This work was made possible by Software Heritage, the great library of source code: https://www.softwareheritage.org

collaborative platforms like GitHub and GitLab provides an unprecedented volume of historical data to trace this evolution. However, this data is difficult to analyze in a consolidated manner for researchers, making it impractical to perform a large-scale, time-based analysis of programming language evolution across the entire history of software development.

Karus and Gall [7] studied language usage trends across 22 open-source software (OSS) projects, analyzing various language co-evolution patterns. While they uncovered useful insights into language combinations, such as the frequent co-evolution of JavaScript with XML, their scope was limited to a small set of projects, reflecting the lack of comprehensive data needed for broader analyses of language evolution over time.

Meyerovich and Rabkin [8] explored factors influencing language adoption, such as developer experience and library availability, across large datasets from platforms like SourceForge and Ohloh. Although they provided valuable insights into adoption drivers, their snapshot-based data did not capture the historical evolution of languages.

Ray et al. [9] examined the effect of programming languages on code quality using over 700 GitHub projects. While this study highlighted some links between language design and software quality, it was limited to GitHub, thus restricting its relevance to recent OSS trends.

Finally, Kochhar et al. [10] analyzed multi-language usage in software projects and its correlation with bug proneness. Although their findings on defect rates in multi-language projects are insightful, the GitHub-only scope restricts generalizability across the software ecosystem.

Overall, these studies underscore the limitations researchers faced due to the lack of an infrastructure providing easy access to extensive, open and well structured data to perform large scale studies.

The advent of Software Heritage (SWH) [11], [12] changes the landscape, as it is the largest public archive of source code, capturing development history across a vast array of software projects. It enables large-scale empirical studies of software evolution by offering access to over 20 billion unique source files and 4 billion commits, stored in a unified, Merkle-based graph representation that has been used in several large scale studies since `swh-graph`, a highly efficient in memory representation, has been released [13].

This study leverages Software Heritage to tackle for the first time a comprehensive longitudinal analysis of the evolution of programming languages over five decades, that is missing in all previous studies, and answer the following **Research Questions**:

- **RQ1**: Can the Software Heritage archive be used to estimate the evolution of programming language adoption over time at a global scale?
- **RQ2**: How has the usage of programming languages evolved over the last 50 years in terms of adoption and decline? What are the major shifts in programming language popularity over time, particularly around key technological transitions (e.g., the rise of web development, mobile platforms)?
- **RQ3**: How do the trends observed in Software Heritage's dataset compare with other language popularity indexes like Tiobe, PYPL, RedMonk and Open Hub?

In the following, we describe how we leveraged the SWH's comprehensive dataset and `swh-graph` to produce intermediate derived datasets used to uncover trends in programming language adoption, shifts in popularity, and a picture of the overall dynamics of the software ecosystem. Beyond the results presented in this work, we provide researchers with ready to use datasets and a pipeline to easily reuse and expand on this work.

The article is organised as follows: Section II recalls the basic concepts of the Software Heritage data structure, Section III details the method used to estimate the activity, per year, for each programming language, Section IV presents and discusses the results, Section V compares our findings with existing language popularity indexes, and then we present the answers to the research questions, discuss threats to validity and conclude with paths for future works.

II. General Presentation of Software Heritage and the Structure of the Merkle Graph

The Software Heritage archive [11], [12] contains more than 20 billion unique source files, with the full history of development contained in 4 billion unique commits, coming from more than 300 million software projects. The software artifacts are retrieved during periodic crawls from major collaborative development platforms (for example GitHub, GitLab) and package repositories (for example PyPI, Debian, NPM). They are stored

in a uniform representation, the `swh-graph` [13], linking source files ("contents"), directories, commits ("revisions"), snapshots, and origins ("origins") of version control system (VCS) repositories.

The structure of the Merkle graph used by Software Heritage ensures the integrity and traceability of the archived files. Each file, directory, commit, and snapshot is uniquely identified by a cryptographic hash identifier called SWHID (see `https://swhid.org` for a detailed presentation), enabling efficient data verification and content deduplication.

In particular, this ensures that each file content is present only once in the archive. This feature of the data structure simplifies significantly the task of analysing the evolution of language usage by avoiding the need to deal with duplicates, which are commonplace since the creation of distributed version control systems and with the popularization of contributions via pull or merge requests.

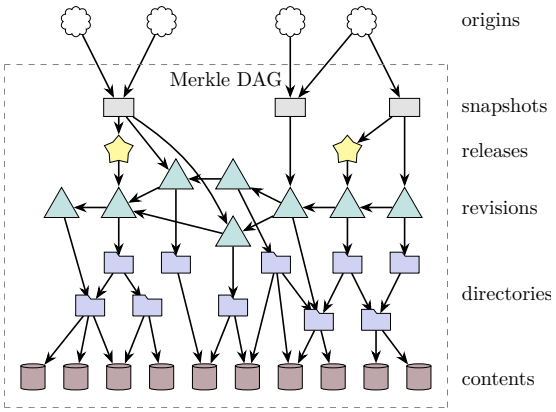


Fig. 1. *Structure of the Merkle graph underlying Software Heritage*

III. Technical approach: from contents to filenames and extensions

In order to perform our analysis on the programming language evolution, we need to identify the programming language used for each of the unique file contents in the archive. Unfortunately, Software Heritage does not compute yet the programming language information by analysing the contents of each file, and it is impractical to run a language detector on tens of billions of files, so we decided to resort to *an heuristic approach: use the file extension as an indicator of the programming language used*.

This in turns requires to find the proper filename for each file content, but due to the structure of the Merkle tree, the same file content may have different file names in different projects. This issue has been already addressed in the work originally done to build the **"Popular Content Filenames Dataset"**, a dataset that associates to each of the 20+ billion contents (identified by the SWHID) its most popular filename [14], i.e., the name that was assigned to it the most times. This file contains more than 20 billion lines, with one line per content file.

A. Computing first occurrence information

Now that we have the filename for each content, we need to know when it was first seen in the archive: this will be the date of creation or modification of the file, and provides the basic indicator of activity that we can use to build an objective view of programming language popularity at a given moment in time.

By visiting the Software Heritage graph, one can propagate the date associated to each revision or release node downwards, and associate to the file content nodes the date of its first appearance in the archive. This is performed in a single visit of the graph, and leads to a new version of the popular contents filenames dataset that now includes the following fields:

Name	Type	Explanation
id	bigint	Numeric identifier for each entry in the dataset.
length	bigint	Length or size of the file content in bytes.
filename	binary	Encoded most popular name of the file content, stored in binary format.
timestamp	bigint	Timestamp marking the first occurrence of the file in the dataset.
revrel	bigint	Numeric identifier of the revision or release associated with the first occurrence of the file.
origin	bigint	Numeric identifier for the original source where the file first appeared.

Tab. I. *Dataset Fields with Types and Explanations*

Numeric identifiers can be converted to Software Hash Identifiers (or URLs, for origins) using the "nodes" table provided alongside the dataset, but are not used in our analysis.

As a result, for each unique file content one now has at hand its most popular file name, and the date of its first appearence in the archive. As a

first application, it is quite easy to sum the number of different file seen per year, and get a picture of the cumulative growth of the original contents in the archive, as shown in Figure 2, that confirms the exponential trend already identified in [15].

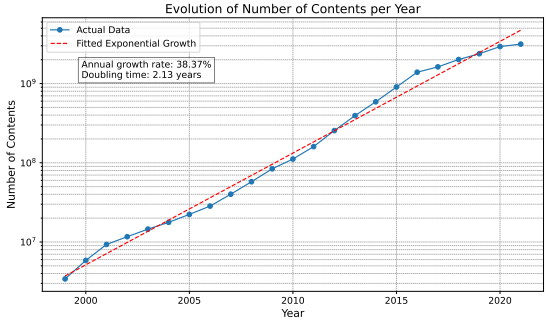


Fig. 2. *Growth of the unique contents of the Software Heritage archive over time*

The last two fields in the dataset are provided to make it easier to find the context where each file content has been found in the archive.

B. Building a Table of Occurrences by Year and Extension

Having performed this initial processing, we can now extract the file extension for each file, and construct a table that associates each file extension with its number of occurrences per year, as described in Algorithm 1:

Algorithm 1 Counting File Extensions by Year

```
Initialize by_year
for each line in agg_data do
    nm ← get(line, 'filename')
    dt ← get(line, 'timestamp')
    ext ← get_ext(nm)
    year ← get_year(dt)
    if ext not in by_year then
        Initialize by_year[ext]
    increment(by_year[ext][year])
Write by_year to file.
```

In this phase, that is computationally expensive, we use a very lax implementation of `get_ext` that considers as an *extension* any aphanumeric sequence of characters following the last dot in a filename, postponing filtering to a later phase.

IV. Presentation of the Results

We filter out irrelevant data from the raw data produced by the processing phases described above, which contains almost 3 million lines, as follows:

- disregard files dated before 1967 (there is no significant data in the archive) or after the current year (2024) when counting occurrences of extensions
- group extensions that differ only by case, summing their occurrences per year

Since other researchers may want to filter the data differently, we make available the full raw data *before filtering* in the file `nb_extensions_alphanum.csv` of the reproducibility package.

We then use the extensive catalog of file extensions known to GitHub's `linguist` [16], that are associated to data formats, markup and programming languagues, to compute a broad series of charts and tables that highlight the evolution over time of the most popular file extensions, for data formats and programming languages, that can be found in the directory `graphs/clean` of the reproducibility package.

Due to space limitations, we present here only a selection of these charts.

A. Evolution of activity, top file extensions

As a first step, we looked at the file extensions, without distinction of kind, with the most activity over the whole period of time: the results for the top ten are shown in Figure 3

It is no surprise to observe a majority of ".c" files between 1980 and 2000, along with a significant portion of files without extensions and ".h" files: this reflects the dominance of the C language during this period.

However, there are also some noticeable anomalies in the data: for example, there are 441 ".php" files in 1971, even though PHP was created in 1994. One can examine their contents by entering their SWHID on https://archive.softwareheritage.org/ and indeed a few files we opened all start with the header `<?php` and contain PHP code. These anomalies are typically caused by erroneous conversions between version control systems that end up squashing file modification or creation date around the Unix epoch, and indeed there is a huge spike in the number of files in 1970.

To mitigate the impact of these erroneous conversions, in the following we have removed from the graphs the 1970 spike, and only counted exten-

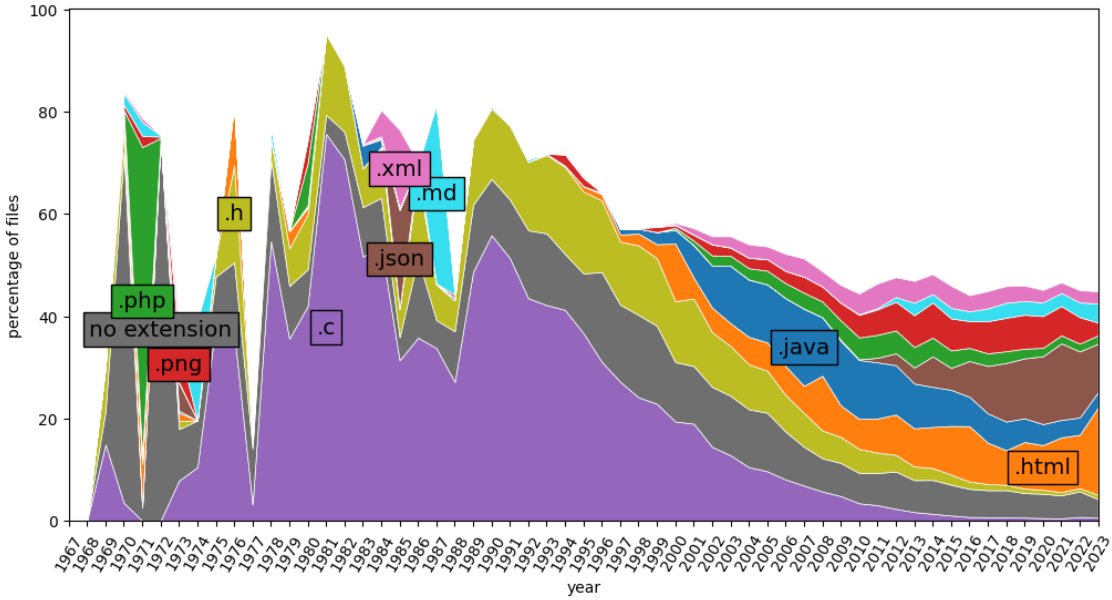


Fig. 3. *Evolution of activity percentages per file extension since 1967. The 10 extensions with the highest cumulative percentage over this period are shown.*

sions for programming languages after the year of their creation. Indeed, by retrieving the publication dates of languages from Wikipedia, we were able to ignore all occurrences of files from a language that had not yet been published. However, this correction has a negligible effect on the data for the years 2000 to 2021.

Full data, and many detailed figures about the whole 50 years time span are available in the reproducibility package. Due to lack of space, we do not present them here, but focus on the activity starting from the year 2000, where the larger amount of activity gives more stable results, as shown in Figure 4.

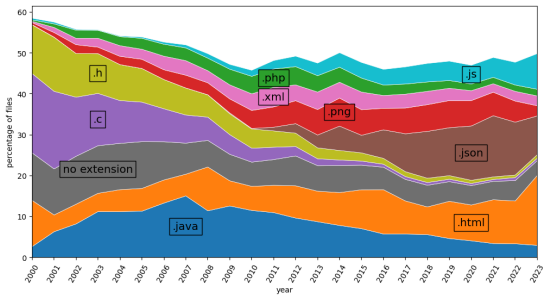


Fig. 4. *Evolution of percentage of activity per file extensions from 2000 to 2023. The 10 extensions with the highest cumulative quantity over this period are shown.*

Several extensions associated with programming languages appear: “.c”, “.h”, “.java”, “.js”, “.php”. We can observe that the share of unique “.c” and “.h” files posted each year is decreasing, while the share of “.js” files is increasing. We also observe data-related language extensions, such as “.html”, “.json”, and “.xml”, as well as files without extensions and “.png” files, which correspond to the large amount of images included in software projects.

Figure 5 shows the activity as absolute value.

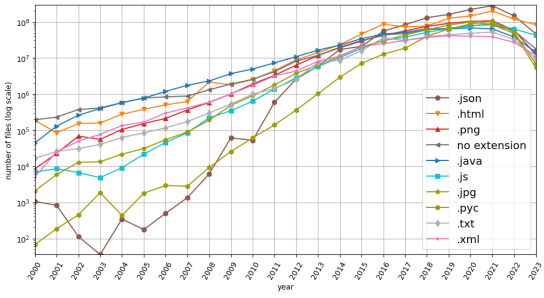


Fig. 5. *Evolution of the number of file extensions from 2000 to 2023, on a logarithmic scale. The 10 extensions with the highest cumulative quantity over this period are shown.*

While the overall quantity of files increases exponentially over time, as already seen in Figure 2, this graph shows a global decrease in the number of files between 2021 and 2023. This anomaly seems to come from the lag that Software Heritage (SWH) crawlers accumulate with respect to new

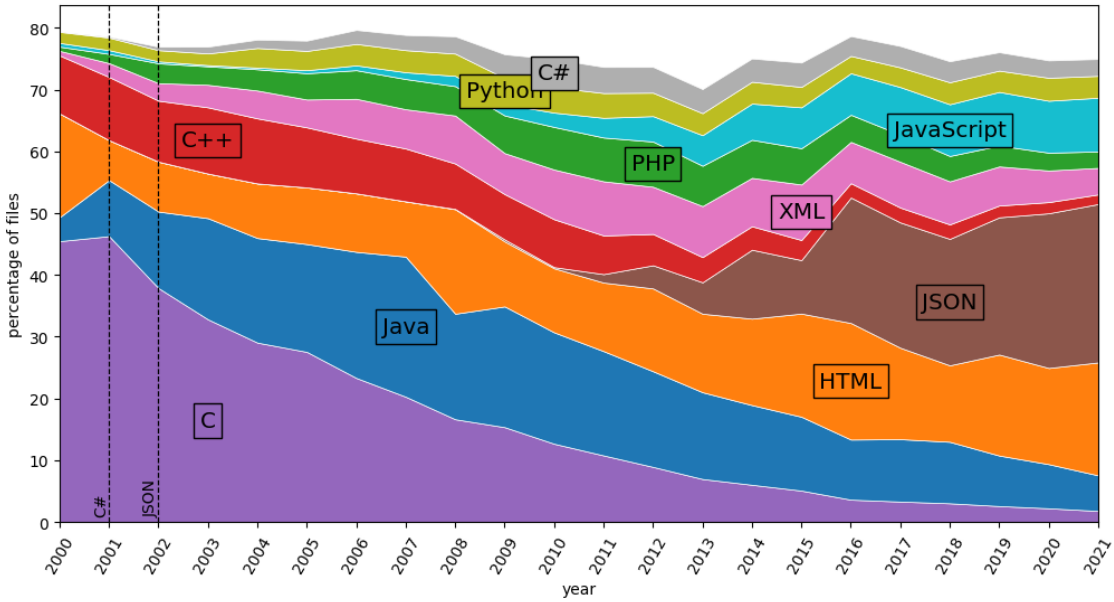


Fig. 6. *Evolution of the percentage of files for programming, markup, and data languages from 2000 to 2021. The vertical dotted lines represent the publication year of the language indicated at the bottom left of each bar.*

projects, which is overall estimated to 30 months. The archive used for our analysis was exported on August 2024, so for the remainder of this study, we will ignore data after 2021.

B. Detailed Analysis of Popular Programming Languages

We are now ready to present the results of the detailed analysis of popular languages that we conducted using all the extensions recognized by *GitHub Linguist* [16].

We proceeded as follows: each extension from the GitHub Linguist language file was associated with a unique programming language. For extensions corresponding to multiple languages, such as ".pl" for both Perl and Prolog, we decided on a case-by-case basis which language to associate. In most cases, we opted for the most popular language, such as Perl for ".pl".

From this association table, we grouped the extensions from the same language by summing their number of occurrences, in order to visualize the evolution of programming languages overall.

The GitHub Linguist file categorizes languages by type: programming (Python, Java, C...), markup (HTML, CSS...), data (SQL, CSV, Graphviz...), and prose (Text, Markdown...). In particular, this excludes plain text and binary formats, such as images and compiled files like Python bytecode (".pyc"). The tool `graph.py` has command line options that allow selecting one or more language types to display on the charts.

Figure 6 and the following ones show the evolution of programming, markup, and data languages between the years 2000 and 2021.

This confirms C's position as the most popular language in publicly available code in the first decades of compunting. Its share in the archive increased until 2000, when it constituted almost half of the file activity of that year (around 45%). C++ and HTML were also quite popular, each occupying more than 10% of the activity seen in the archive.

Between 2000 and 2010, we see the popularity of C and C++ decline, while other languages quickly gaind traction: Java, HTML, XML, PHP, Python, and C#. From 2010 to 2021, these six languages maintain a relatively stable popularity, with the exception of Java, whose popularity decreases, along with C++. Meanwhile, JavaScript rises steadily from 2005. Activity related to JSON files explodes between 2011 and 2016 and continues to grow, representing about 25% of the activity in the archive in 2021.

The activity grows exponentially for most languages, as can be more easily seen in the logarithmic scale graph in Figure 8.

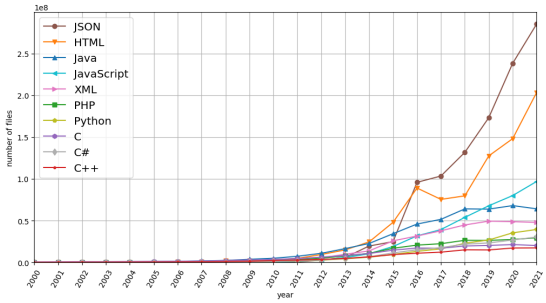


Fig. 7. *Evolution of the quantity of files for programming, markup, and data languages (excluding plain text and binary formats) from 2000 to 2021.*

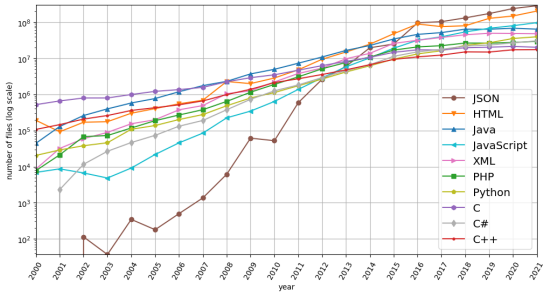


Fig. 8. *Evolution of the quantity of files for programming, markup, and data languages from 2000 to 2021, on a logarithmic scale.*

Finally, Figure 9 shows the *cumulative* amount of modified files over time since 2000, which corresponds to the number of new or modified files for a language in the archive from 2000 up to a given date. For example, in 2020, there were approximately 800 million JSON files posted since 2000 in the archive and about 400 million Java files.

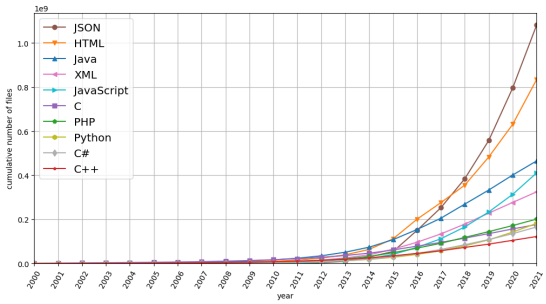


Fig. 9. *Evolution of the cumulative quantity of files for programming, markup, and data languages (excluding plain text and binary formats) from 2000 to 2021.*

The cumulative growth follows an exponential curve too, as can be seen in the lin log scale on Figure 10.

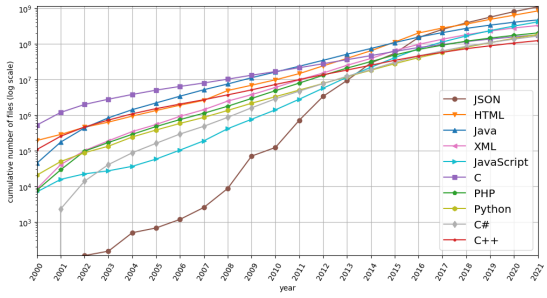


Fig. 10. *Evolution of the cumulative quantity of files for programming, markup, and data languages (excluding plain text and binary formats) from 2000 to 2021, log scale.*

V. Comparison with Other Language Ranking Indexes

The data presented above allows to compare our ranking of programming languages with four popular established indexes: Tiobe [4], PYPL [17], RedMonk [5] and Open Hub [6].

Tiobe's strategy for assigning a popularity score to a language involves counting the number of results across a selection of search engines from internet searches like "[language name] programming."

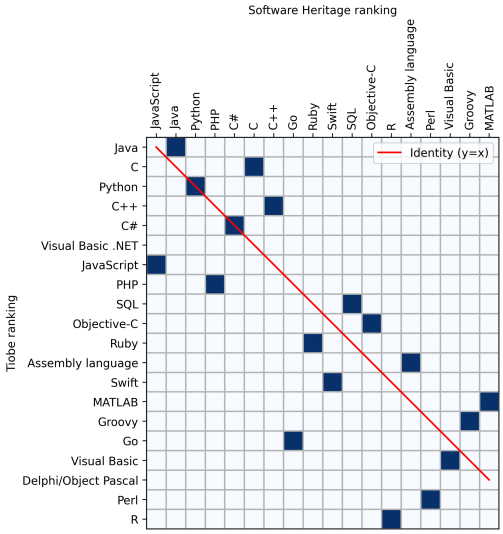
PYPL's strategy is similar, as it involves counting the number of results from search on Google Trends of the kind: "[language name] tutorial."

RedMonk's strategy uses two metrics: a score assigned to each language based on the number of projects on GitHub written primarily in that language, and the number of questions tagged with that language on StackOverflow.

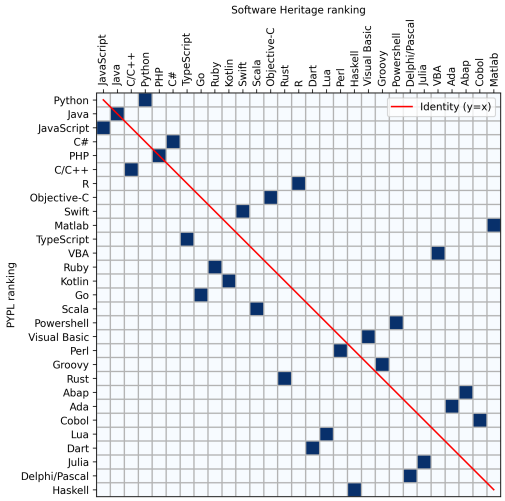
Finally, OpenHub is the one which is somewhat closer to ours, as it counts the number of commits over time per project in a given programming language.

A. Comparison Methodology of Indexes

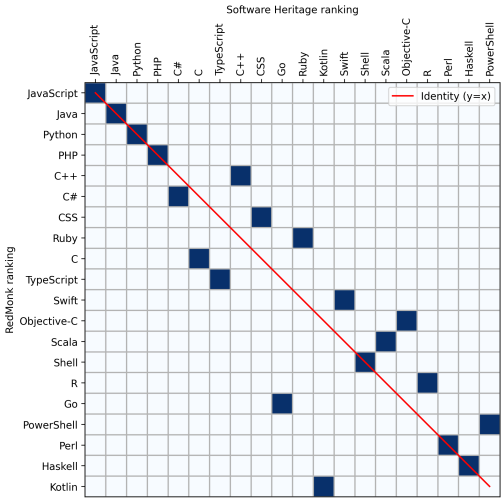
Longitudinal data is not available at reasonable conditions for all the indexes, so we could not produce a longitudinal comparison, and we resorted instead to a recent year, 2019, for which we could obtain all the data, partially by manually compiling it from the indexes websites. To compare our rankings for 2019, we created a heatmap comparing our language ranking by file quantity with each of the ranking indexes. The ascending ranking order of the x-axis is read from left to right, and the y-axis is read from top to bottom. Each cell is colored if



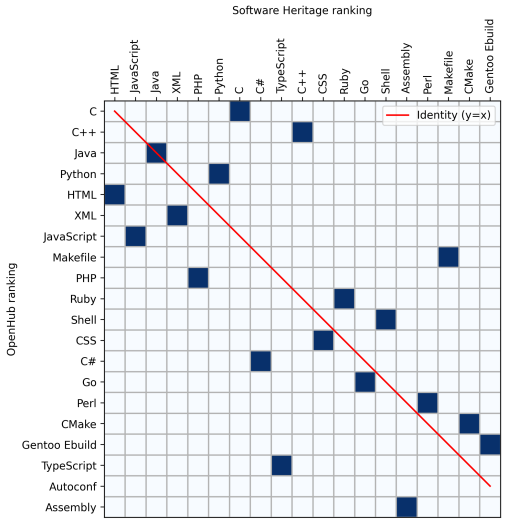
(a) *Comparison of language rankings between Tiobe and SWH in 2019.*



(b) *Comparison of language rankings between PYPL and SWH in 2019.*



(c) *Comparison of language rankings between RedMonk and SWH in 2019.*



(d) *Comparison of language rankings between OpenHub and SWH.*

the languages in the two rankings are identical for that row and column. For example, if the top-left cell is colored, it means that both rankings placed the same language in first place.

B. Comparison Results of Indexes

There is significant disparity between the Tiobe ranking and ours (Figure 11a). is partly due to the difference in methodology, but also to the difference in language selection. The number of languages displayed in the Tiobe ranking is limited by the cost of access to the full ranking, and on the SWH side, it is deliberately limited to make the heatmap easier to read.

The correspondence between the RedMonk ranking and ours (Figure 11c) is striking: the top 4 languages are identical, and there are few outliers in the rest of the heatmap. Firstly, the set of languages used by RedMonk is the same as ours, GitHub Linguist. Secondly, and most importantly, their heuristic involving the languages of GitHub projects is more similar to ours, which involves counting files on version control systems like GitHub, compared to the approaches of Tiobe and PYPL.

For OpenHub, we were confronted with the fact that temporal data is not available for download, and we could only get the *cumulative* number of

commits for the topmost 20 languages since 2000 by parsing their webpage. To produce some sort of meaningful comparison with this kind of data, we summed up the activity information on our side too, which leads to Figure 11d. There is clearly a misalignment, but it's difficult to draw conclusions.

VI. Answering the Research Questions

We are now in a position to answer the research questions that we set out to investigate.

A. RQ1: Use of the Software Heritage archive

We have shown that it is now possible to leverage the Software Heritage archive to perform large scale longitudinal estimation of the evolution of programming language adoption. The Merkle structure of the graph allows to precisely track the first occurrence of each new original content, which represents creation or modification of files. The **"Popular Content Filenames Dataset"** provides for each such content the reference filename, from which one can extract the extension, which is a proxy for the programming language.

B. RQ2: Evolution of Programming Languages

The analysis reveals that languages like C dominated between 1980 and 2000, while newer languages like JavaScript and Python surged in popularity post-2000. The share of C and C++ files gradually declined, while the share of web-based languages (.js, .html) and data-related formats (.json) increased, particularly in the last decade.

The data indicates clear inflection points corresponding to key technological advances. For example, the rise of Java and JavaScript coincided with the advent of web-based applications in the late 1990s, and the increased usage of Python aligns with the growth of data science in the 2010s. These shifts reflect broader changes in software development practices and platform preferences.

C. RQ3: Comparison with Existing Indexes

Data from existing indexes is not easily accessible on a year per year basis, so we could not perform a longitudinal comparison, but we did compare manually the results for specific years. We find that Tiobe and PYPL show language popularity trends that are not too different from ours, but there are some notable differences. For instance, while Python's popularity has risen sharply in Software Heritage data, its relative ranking is lower in Tiobe. The most congruent results are observed with the RedMonk index, which relies on GitHub data and selection of programming languages. We expected to find a similar alignment with Open Hub, but the lack of available longitudinal data prevents a meaningful comparison. Overall, the longitudinal data provided openly by this study is much broader than any other existing data, and we could not find indexes covering all the 50 years covered in this study.

VII. Threats to validity

To the best of our knowledge, this is the first large scale longitudinal study on language popularity spanning decades, and it is important to highlight here the hypothesis and limitations of the approach, that are of different nature.

a) Identification of the file names: The heuristic used by the *Popular Filename Dataset* is to associate to each file content the filename that is most frequent in the directories that contain it. While this is a very reasonable approach, it may not be 100% accurate.

b) Identification of Programming Languages: The best way to identify the programming language used in a file is to run a programming language detector on its contents, but such a processing is prohibitively expensive when handling tens of billions of file contents. Hence we resorted to use the file extension as a proxy for the real programming language, but this comes with several limitations:

- The file content may not match the extension.
- Several languages use the same extension (e.g., Prolog and Perl both use ".pl").
- Sometimes the last extension is not enough. For example, "file.antlers.html" will be classified as an HTML file because we extract only the last ".html" extension, instead of classifying it as an Antlers file.

c) Measuring the Relative Importance of Languages: Some languages encourage the use of many files, while others use fewer, which can bias measures of the relative importance of languages. For example, a Java project contains one file per class and often several classes, while an equivalent project in Python may be written in a single script file. This results in an overrepresentation of Java compared to Python in this case.

d) Exhaustiveness of Software Heritage: Software Heritage archives a broad range of code but is not exhaustive. It includes only publicly

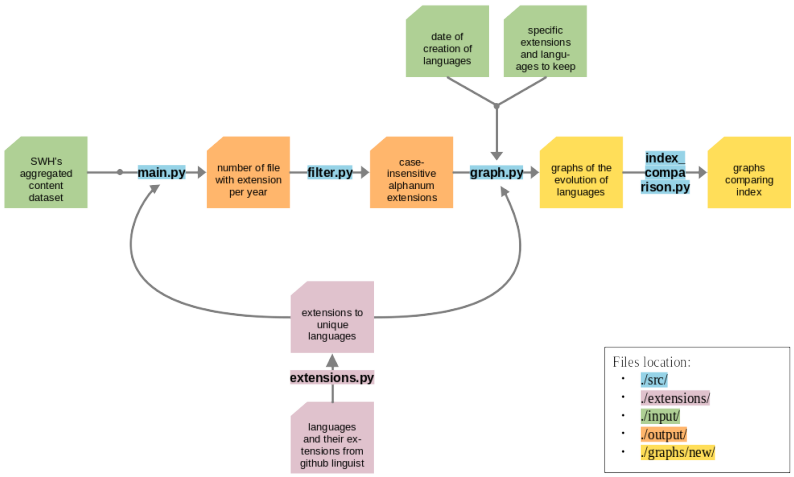


Fig. 12. *Data pipeline and location of project files.*

available projects (often open-source), leading to an overrepresentation of programming languages used in public projects and an underrepresentation of those used in proprietary software. For instance, languages used for UNIX system development (e.g., Rust) may be more represented than those for Windows development (e.g., C++).

Additionally, Software Heritage archives certain package repositories, which may bias file counts toward their languages, if the source package distribution make changes to the upstream code. For example, the PyPI repository contains over 500,000 Python projects, which may increase Python's prevalence in our analysis.

VIII. Reproducibility Package

The data pipeline is shown in Figure 12, and processed data and images are available alongside the source code [18], distributed under GPL3+ licence from its repository on Software Heritage's Gitlab. It is archived for the long term in the Software Heritage archive[1].

The Popular Content Dataset has been enriched for analysis, with the version used available at: s3://softwareheritage/derived_datasets/2024-08-23/contents/. Updated versions are provided alongside new Software Heritage Graph Dataset releases, usually every six months.

To ensure reproducibility of the results, all graphs can be regenerated by downloading the project code and following the README.md instructions. Command lines are customizable for data processing and display based on dates, extensions, language grouping, and types.

IX. Conclusion and Future Work

We built an open dataset and pipeline to track programming language activity over more than 50 years, leveraging the structure of the Software Heritage archive.

One can easily verify in this data some folklore knowledge, like the fact that C dominated programming languages in popularity from the 1970s to 2000, and was then progressively replaced by other languages such as Java, PHP, JavaScript, and Python. One can also easily monitor the emergence and rapid adoption of markup languages like HTML, and data languages like JSON and XML.

When Software Heritage will provide information on the programming language for each of its file contents, it would be interesting to rebuild the dataset, which will be more precise. It will also be interesting to look at other measures, like number of projects or number of lines of code, over the same period of time.

[1] swh:1:dir:7347b84a900f6f07457f22be1cbb4c9b5ca9e9dc

References

[1] "A common programming language for the department of defense: Background and technical requirements," Institute for Defense Analyses, Tech. Rep. AD-A028 297, Jun. 1976, Prepared for Defense Advanced Research Projects Agency. [Online]. Available: https://apps.dtic.mil/sti/tr/pdf/ADA028297.pdf.

[2] R. L. Wexelblat, "The history of programming languages: Proceedings of the hopl conference," in *Proceedings of the ACM SIGPLAN conference on The History of Programming Languages (HOPL)*, Los Angeles, CA, USA: ACM, 1978.

[3] D. J. Pigott, B. M. Axtens, and S. Poulson, *Hopl - the history of programming languages*, https://hopl.info/, Accessed: 2024-10-23.

[4] TIOBE, *Tiobe programming community index*, Available from https://www.tiobe.com/tiobe-index/programminglanguages_definition/. Visited on 2024-06-24. [Online]. Available: https://www.tiobe.com/tiobe-index/programminglanguages_definition/ (visited on 07/30/2024).

[5] S. O'Grady, *The redmonk programming language rankings: June 2019*, Available from https://redmonk.com/sogrady/2019/07/18/language-rankings-6-19/. Visited on 2024-06-24. [Online]. Available: https://redmonk.com/sogrady/2019/07/18/language-rankings-6-19/ (visited on 07/30/2024).

[6] O. Hub, *Open hub: The open source network*, https://www.openhub.net/, Accessed: 2024-10-23.

[7] S. Karus and H. Gall, "A study of language usage evolution in open source software," in *Proceedings of the 8th Working Conference on Mining Software Repositories*, ser. MSR '11, New York, NY, USA: Association for Computing Machinery, 2011, pp. 13–22, ISBN: 978-1-4503-0574-7. DOI: 10.1145/1985441.1985447.

[8] L. A. Meyerovich and A. S. Rabkin, "Empirical analysis of programming language adoption," in *Proceedings of the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages and Applications*, ser. OOPSLA '13, New York, NY, USA: Association for Computing Machinery, 2013, pp. 1–18, ISBN: 978-1-4503-2374-1. DOI: 10.1145/2509136.2509515.

[9] B. Ray, D. Posnett, V. Filkov, and P. Devanbu, "A large scale study of programming languages and code quality in github," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. FSE 2014, New York, NY, USA: Association for Computing Machinery, 2014, pp. 155–165, ISBN: 978-1-4503-3056-5. DOI: 10.1145/2635868.2635922.

[10] P. S. Kochhar, D. Wijedasa, and D. Lo, "A large scale study of multiple programming languages and code quality," in *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, vol. 1, Mar. 2016, pp. 563–573. DOI: 10.1109/SANER.2016.112.

[11] R. Di Cosmo and S. Zacchiroli, "Software heritage: Why and how to preserve software source code," in *Proceedings of the 14th International Conference on Digital Preservation, iPRES 2017, Kyoto, Japan*, Available from https://hal.archives-ouvertes.fr/hal-01590958, Sep. 2017. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01590958.

[12] J.-F. Abramatic, R. Di Cosmo, and S. Zacchiroli, "Building the universal archive of source code," *Commun. ACM*, vol. 61, no. 10, pp. 29–31, Sep. 2018, ISSN: 0001-0782. DOI: 10.1145/3183558. [Online]. Available: http://doi.acm.org/10.1145/3183558.

[13] A. Pietri, D. Spinellis, and S. Zacchiroli, "The software heritage graph dataset: Large-scale analysis of public software development history," in *MSR 2020: The 17th International Conference on Mining Software Repositories*, IEEE, May 1, 2020, pp. 1–5. DOI: 10.1145/3379597.3387510. [Online]. Available: https://arxiv.org/abs/2011.07824, published.

[14] V. Lorentz, R. Di Cosmo, and S. Zacchiroli, "The Popular Content Filenames Dataset: Deriving Most Likely Filenames from the Software Heritage Archive," Software Heritage, Tech. Rep., Jul. 2023, preprint. [Online]. Available: https://inria.hal.science/hal-04171177.

[15] G. Rousseau, R. Di Cosmo, and S. Zacchiroli, "Software provenance tracking at the scale of public source code," *Empirical Software Engineering*, vol. 25, no. 4, pp. 2930–2959, 2020. DOI: 10.1007/s10664-020-09828-5. [Online]. Available: https://doi.org/10.1007/s10664-020-09828-5.

[16] G. Linguist, *Languages known to github*, Available from https://github.com/github-linguist/linguist/blob/master/lib/linguist/languages.yml. Visited on 2024-07-04. [Online]. Available: https://github.com/github-linguist/linguist/blob/master/lib/linguist/languages.yml (visited on 07/04/2024).

[17] P. Carbonnelle, *Pypl - popularity of programming language*, Available from https://pypl.github.io/PYPL.html and https://pypl.github.io/PYPL/All.js. Visited on 2024-06-24. [Online]. Available: https://pypl.github.io/PYPL.html (visited on 07/30/2024).

[18] [SW Rel.] A. Desmazières, *Computer Language Evolution* version 1.0.0, Jan. 15, 2025, LIC: GPL-3.0-or-later, VCS: https://gitlab.softwareheritage.org/teams/interns/evolution-prog-langs.git, SWHID: ⟨swh:1:dir:7347b84a900f6f07457f22be1cbb4c9b5ca9e9dc⟩.

Tab. II. *Number of occurrences for the 50 topmost files extensions of type program, data and markup, years 2000 to 2021 (excerpt from the open dataset). MVSS is the acronym of Microsoft Visual Studio Solution.*

rank	language	2000	2001	2002	2003	...	2018	2019	2020	2021
1	JSON	1060	835	114	37		131709358	173215073	238284890	285249981
2	HTML	192451	92293	169910	175743		79635859	127463415	148120221	203364463
3	Java	44275	129465	260647	397908		64039989	63746438	67966967	64077725
4	JavaScript	6946	8682	6704	4848		53891803	67761796	79990257	96862528
5	XML	8543	32008	59688	88685		44836779	49356207	48713987	47878960
6	PHP	7742	21264	67711	72681		26436495	26297783	27468494	29432604
7	Python	20494	28970	38019	45894		22907202	26846018	35372913	39404127
8	C	520591	660636	797256	797897		19619801	20300120	21347995	20192306
9	C#	22271	2316	11627	26661		21866472	23287720	26804639	30369753
10	C++	107379	145937	207661	260394		15053071	14908307	17210797	17323133
11	CSV	215	473	145	955		16724214	15979066	24129002	24683212
12	TypeScript	266	1138	485	3043		13104547	18719605	26188007	36141709
13	YAML	2586	295	260	2472		8827140	10504680	15316555	23237659
14	CSS	1902	3617	4067	5345		8968753	9971232	13369552	16621747
15	Ruby	292	2844	5565	5415		6856404	6656268	6378353	5490214
16	Unity3D Asset	20	65	30	136		7632857	8184324	11168025	13093450
17	Go	264	14	1	1		5459329	7267855	9676219	11577210
18	SVG	474	1416	930	2486		4900289	7445684	9436813	10522885
19	SCSS	141	37	41	11		4515693	5144852	5825765	6862150
20	INI	1698	4681	15728	15837		4191401	4109370	4619512	4410862
21	Kotlin	2	3	0	0		3200391	4812935	7084337	8592139
22	MATLAB	4475	6000	28292	13953		2953681	2438980	2311697	1780448
23	Shell	10250	8930	13228	15492		3403528	3436022	3874332	3829830
24	Vue	3	90	9	0		3316385	5054133	5946405	6340824
25	Swift	12	5	0	0		3270740	3542345	3980549	4185671
26	TSX	144	0	0	0		1126382	2705573	5574116	9841126
27	HTML+ERB	40	251	80	138		2033535	1849032	1660191	1614181
28	SQL	2453	2861	7400	8659		3591905	2576106	2706208	2717682
29	Scala	93	161	321	547		1863921	3048838	4816112	2469399
30	Java Server Pages	328	1407	6032	10367		2360832	2138504	1858304	1891463
31	Jupyter Notebook	22	7	75	0		1768107	2676168	4293264	4473764
32	R	5677	7282	7339	10986		1636908	1919109	2164659	2133413
33	Dart	0	0	1	1		450107	1617238	3862318	6111386
34	HTML+Razor	0	0	0	1		1889915	1969369	1953684	2216843
35	Gradle	7	9	11	0		1855756	2110910	2481872	2604283
36	Checksums	23	14	129	38		1579183	1449272	1941528	2229101
37	D	768	1760	2053	2087		1833539	1594886	1528147	1628156
38	RenderScript	11	25	0	33		1309661	2082934	2842431	3573786
39	Roff	31001	27605	43855	37483		1194557	1202387	1246823	1442475
40	Assembly	7024	8777	17765	15751		1428676	1038270	1216906	918041
41	Smali	0	0	0	0		1070236	1434705	1295712	1097705
42	TeX	4825	5906	12452	12411		1233801	1500283	1311354	1232805
43	Perl	31418	53652	82520	108439		712522	772995	751497	538140
44	Lua	136	150	370	715		997956	1053898	1342153	1602720
45	Wavefront Object	318	163	640	528		1030227	1099301	1551620	1676208
46	Makefile	5777	7495	8819	10252		1014984	1080002	1189728	1314293
47	MVSS	35	33	331	770		995249	1042807	1127279	1181571
48	Diff	3580	4659	10793	11113		734855	990512	699103	972737
49	Pickle	0	0	4	92		1289022	779832	1029589	2184203
50	OpenStep Property List	108	214	579	760		704706	716922	744749	685411