

Dataset: Copy-based Reuse in Open Source Software

Mahmoud Jahanshahi
mjahansh@vols.utk.edu
University of Tennessee
Knoxville, USA

Audris Mockus
audris@utk.edu
University of Tennessee
Knoxville, USA

ABSTRACT

In Open Source Software, the source code and any other resources available in a project can be viewed or reused by anyone subject to often permissive licensing restrictions. In contrast to some studies of dependency-based reuse supported via package managers, no studies of OSS-wide copy-based reuse exist. This dataset seeks to encourage the studies of OSS-wide copy-based reuse by providing copying activity data that captures whole-file copying that captures nearly all OSS. To accomplish that, we develop approaches to detect copy-based reuse by developing an efficient algorithm that exploits World of Code infrastructure: a curated and cross referenced collection of nearly all open source repositories. We expect this data will enable future research and tool development that support such reuse and minimize associated risks.

KEYWORDS

Reuse, Open Source Software, Software Development, Copy-based Reuse, Software Supply Chain, World of Code

1 INTRODUCTION

The cornerstone of Open Source Software (OSS) is its “openness”, including the ability to access, examine, and copy any project artifact subject to licensing restrictions that, in turn, often enforce openness of the derived work. Such ability has a potential to bring dramatic improvements in developer productivity, but it may also result in the proliferation of, potentially, poor quality code in various states of disrepair (e.g. orphan vulnerabilities [38]).

Furthermore, as such copying progresses from one project to another, potentially with modifications, critical information about the original design, authorship, copyright, and licensing may be lost [37], thus impeding further improvements, bug fixing, reducing attribution-related motivation for original authors and creating legal problems for downstream users, not only of the copied code but of software that depends on at least one package involving copied code [5].

As OSS grows, the tasks of tracking the origins of source code, finding good quality code suitable for reuse, or untangling parallel evolution of code in multiple projects are ever more daunting. Despite the longstanding interest, potentially massive benefits and risks of copying activity, the precise extent, the prevailing practices, and the potentially negative impacts of the source code copying at the scale of the entire OSS have not been investigated mainly due to the challenge of being able to track code over entire OSS.

A better understanding of code copying practices may suggest future research on approaches or tools that make productivity improvements even greater while, at the same time, helping to minimize inherent risks of copying. Specifically, we aim to provide a copy-based reuse dataset to enable further analysis of aspects concerning the extent and the nature of reuse in OSS and to provide

information necessary to investigate approaches that support this common activity, make it more efficient, and safer.

First, we create a measurement framework that tracks all versions of source code (we refer to a single version as a blob in keeping with the terminology of the version control system git) across all repositories. The time when each unique blob b was first committed to each project P is denoted as $t_b(P)$. The first repository $P_o(b) = \text{ArgMin}_P t_b(P)$ is referred to as the the originating repository for b (and the first author as the creator). Next, copy instances are identified via projects pairs: a project with the originating commit and the destination project with one of the subsequent commits producing the same blob ($P_o(b), P_d(b)$).

2 BACKGROUND

The term software reuse refers to the process of creating software systems from existing software rather than building them from scratch [26]. Coding from scratch may require more time and effort than reusing already created source code that is suitable for the task and is of good quality. Programmers, therefore, opportunistically reuse code and do so frequently [24]. Programming of well-defined problems often start by searching in code repositories followed by judicious copying and pasting [41].

Open source software (OSS) development and platforms like GitHub greatly expand opportunities to reuse by enabling the community of developers to curate software projects and by encouraging and enhancing the process of opportunistic finding and reusing artifacts. Much of OSS is specifically designed to be reused and to provide resources or functionality to other software projects [16], thus such reuse can be categorized as one of building blocks of OSS. In fact, developers not only look for the opportunity to reuse, but also advertise their own high quality artifacts for others to reuse [12]. High reuse not only brings the software project and its maintainers popularity and job prospects [39], but also may bring maintainers and corporate support.

2.1 Research that could be done using this dataset

Not always is code reuse beneficial though. Although it may reduce the development costs, it could pose some other risks that eventually would lead to increased overall costs. Security vulnerabilities, licensing and copyright issues and code quality are just a few such risks [10]. Our curated dataset can enable future research in many potential areas including but not limited to the following:

Security. The relation between security and reuse can go both ways: a system can become more secure by relying on mature dependencies, or more insecure by exposing a larger attack surface via exploitable dependencies [15]. Specifically, in copy-based reuse, extensive code copying in OSS results in an extensive spread of

potentially vulnerable code not only in inactive projects that are still publicly available for others to use that do spread the vulnerability further, but also in currently active and in highly popular projects [38].

Licensing. As software systems evolve, so do licenses. Various factors, such as changes in the legal landscape, commercial code licensed as free and open source, or code reused from other open source systems, lead to evolution of licensing, which may affect the way a system or part thereof can be subsequently used and therefore, it is crucial to monitor licensing evolution [4]. But monitoring the huge amount of data in entire OSS is not an easy task to do and thus many developers fail to adhere to licensing requirements [2, 11].

Quality. Code reuse is not only assumed to inflate maintenance costs in certain circumstances, but also considered defect-prone as inconsistent changes to code duplicates can lead to unexpected behavior [24]. Also forgetting to modify identifiers (variables, functions, types, etc.) consistently throughout the reused code will cause errors that often slip through compile-time checks and become hidden bugs that are very hard to detect [27].

Other than the bugs introduced by this kind of reuse, the source code itself could have bugs or be of low quality that can be spread in the same way explained about security vulnerabilities earlier. The future study using this dataset might suggest approaches to leverage the information obtained from multiple projects containing reused code to reduce these kind of risks.

While the described benefits and risks associated with reuse appear to be real, the extent and the types of copying in the entire OSS are not clear. In order to prioritize these risks/benefits and investigate approaches to minimize/maximize them, we first need to develop an approach to track copy-based reuse scalable to the massive size of the entire OSS as investigations of convenience samples presented in prior work do not capture the bulk of copying activity.

2.2 Contribution

To the best of our knowledge no curation system exists at the level of a blob, nor is there an easy way for anyone to determine the extent of copy-based reuse at that level and the introduced reuse identification methods (such as [25]) find reuse between given input projects and are not easily scalable to find reuse across all OSS repositories. The methods we use to identify reuse could, therefore, provide a basis for tools that expose these hard-to-obtain yet potentially important phenomenon.

Our dataset has two important aspects. *First*, we present the copying activity at the whole open source software ecosystem level. Previous provided datasets normally focus on a specific programming language (e.g. Java as in [22]) and the data used in previous works investigating copying have as well mostly concentrated on a small subset of a specific community (e.g. Java language, Android apps, etc.) [6, 16, 17, 20, 33, 42] or sampled from a single hosting platform (e.g. GitHub) [12, 13]. Even research more comprehensive in programming language coverage [28] considered only a subset of programming languages and more importantly, used convenience sampling by excluding less active repositories [18, 19]. Furthermore, almost all research only focuses on code reuse whereas our

dataset tracks all artifacts whether they are code or other reusable development resources, such as images or documentation.

Second, copy-based reuse has not been as extensively investigated as the dependency-based reuse, e.g., [3, 8, 36]. Copy-based reuse is, potentially, no less important, but much less understood form of reuse. In fact, most of the efforts in copy-based reuse domain are focused on clone detection¹ tools and techniques [1, 17, 23, 40, 44], not on the properties of files that are being reused. Clone detection tools and techniques usually take a snippet of code as input and then try to find similar code snippets in a target directory or an specific domain [21, 43] whereas in our dataset, we are finding all instances of reuse in nearly entirety of OSS.

The description and the curation methods of this dataset has not been published before. Furthermore, although the dataset is now publicly available through WoC², to the best of our knowledge, the data has not been used by authors or others in any published paper yet.

3 METHODOLOGY

We start by briefly outlining World of Code infrastructure we employed to create our dataset and then present the methods used to identify instances of copying.

3.1 World of Code Infrastructure

Finding duplicate pieces of code and all revisions of that code across all open source projects is a data and computation intensive task due to the vast number of OSS projects hosted on numerous platforms. Previous research on code reuse has, therefore, typically looked at a relatively small subset of open source software potentially missing the full extent of copying that could only be obtained with a nearly complete collection. World of Code (WoC) [31, 32] infrastructure attempts to remedy this by, on a regular basis, discovering publicly available new and updated version control repositories, retrieving complete information (or updates) in them, indexing and cross-referencing retrieved objects, conducting auto-curation involving author aliasing [9] and repository deforking [34], and provides shell, Python and web APIs to support creation of various research workflows. The source code version control systems in WoC are collected from hundreds of forges and, after complete deduplication, takes approximately 300TB of disk space for the most recent snapshot we use for our dataset³. The specific objective of WoC is to support research on three kinds of software supply chains [30]: technical dependency (traditional dependency-based package reuse), copy-based reuse, and knowledge flows [14, 29, 45] (developers working on, and learning about, projects and then using that knowledge in their work on other projects).

WoC's operationalization of copy-based supply chains is based on mapping blobs (versions of the source code) to all commits and projects where they have been created. This implies that copy is detected only if the entire file is copied intact without any modifications. Because of that, our dataset includes only the whole-file

¹identification of, often, relatively small snippets of code within a single or a limited number of projects

²It has been made available only recently

³version V

copying activity. This also means that different versions of the originally same file will be considered different objects since they are different blobs.

Furthermore, to discriminate copy-based reuse from forking (a commit uniquely identifies modified blobs, and forked projects share commits), we use project deforking (p2P map) provided in WoC [34]. Throughout the paper, even if we only use the word project, we mean deforked project with the explained definition.

Specifically, WoC uses git object indexing via sha1 signature so that each association has to store only the sha1 of the object (in this case blob), and the actual content of each object is stored exactly once. When objects are extracted from a repository, WoC associates all extracted commits with that repository (the so called c2p map). Since a commit points to a tree and to its parent commit objects, the remaining objects in a repository can be easily derived by traversing versions and trees. WoC also computes the association between commits and blobs created by a commit (new versions of existing files or entirely new files) and makes it available via c2fbb map. The map lists all the instances where a blob corresponding to one of the files in the repository changed or a new file was created. In the former case, the blob corresponding to an earlier version of the file is also provided, making it possible to trace back or forth for earlier or newer versions of a blob.

Commits have attributes, such as time of the commit and author of the commit and these attributes can be accessed via c2dat map in WoC. A few more maps provided by WoC are also used in creating this dataset.

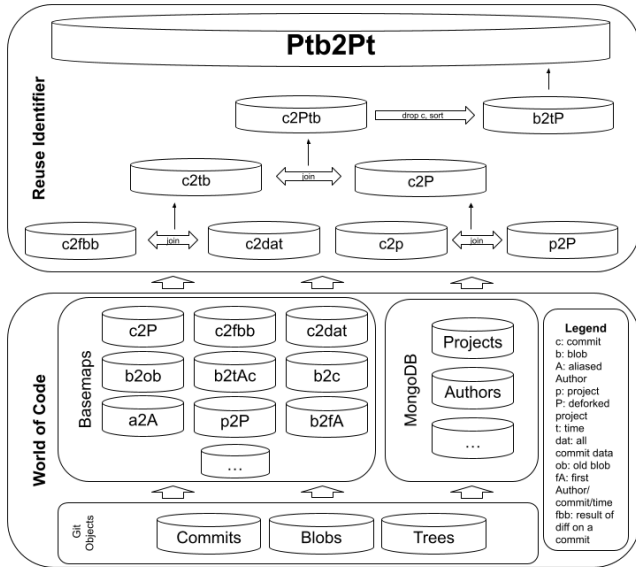


Figure 1: Schematic Architecture of Reuse Identification

3.2 Identification of reused blobs

Despite the key relationships available in WoC, we have to resolve several critical obstacles. We first need to identify the first time $t_b(P)$ each of the nearly 16B blobs landed in each of the almost 108M projects. We aim to minimize memory use and be able to

run computations in parallel. First, we join c2fbb map⁴ (that lists for each commit all the blobs it creates) with c2dat map (to obtain the date and time of the commit) and then with the c2P (which itself is the result of joining c2p with p2P maps) map to identify all projects containing that commit. WoC has each of the three maps split into 128 partitions⁵ requiring us to run a sequence of two Unix join commands (first to join c2fbb and c2dat and then the result of that join with the c2P map) on each of the 128 partitions in parallel. The result is a new c2Ptb (commit, project, time, and blob) map stored in 128 partitions $(c^i, P, t, b) : i = 0, \dots, 127$. To create the timeline for each blob we need to sort all that data by blob, time, and project. The list has hundreds of billions of rows (20B blobs often occurring in multiple commits and commits sometimes residing in multiple projects). We thus needed to break down the problem into smaller pieces to solve within a reasonable time frame. Specifically, we first split each partition (c^i, P, t, b) based on the blob into 128 sub-partitions, thus obtaining 128x128 partitions resulting from the original partitioning by commits and the secondary one by blobs $(b^j, t, P, c^i) : i, j = 0, \dots, 127$. We then sort each of the 128x128 files by blob, time, and project (using Unix sort parameterized to handle extremely large files) and drop all but the first commit creating the blob for each project⁶. In the next step we merge 128 commit-based partitions for each blob-based partition using Unix sort with a merge option and drop all but the first commit of the blob to a project. Resulting in 128 blob-based partitions (b2tP map) $(b^j, t, P) : j = 0, \dots, 127$ where we have only blob, time, and the deforked project that contain our desired timeline $t_b(P)$. Finally, the blob timelines are used to identify instances of copying $(t_b(P_o), t_b(P_d))$ (or, in the terminology of WoC, Ptb2Pt maps where the first project is originating⁷ and the second project copied the blob – the blob was created at a later time). To accomplish this we first create a list of blob origination projects and times. A sweep over b2tP by keeping only the first time and the project associated with each b and excluding blobs associated with a single project⁸ produces $(b^j, t, P_o) : j = 0, \dots, 127$. We also store never reused blobs $(b_{nc}^j, t, P_o) : j = 0, \dots, 127$ (ones that are associated with only one project as identified during the sweep mentioned above). (b^j, t, P_o) partitions containing only originating project are then joined with (b^j, t, P) to obtain the cross-product $((P_o^j, t_o, P_d, t_d, P_d, b^j) : j = 0, \dots, 127, P_o \neq P_d)$. Each of the resulting 128 partitions are then split via project name⁹, into 128 sub-partitions and each sub-partition is then sorted by the originating project: $((P_o^i, t_o, P_d, t_d, b^j) : i, j = 0, \dots, 127)$, then merging over blob-based partitions belonging to a single project-based partition. Resulting Ptb2Pt map contains all instances of blob copying: $(t_b(P_o^i), t_b(P_d))$ and is stored in 128 partitions $i = 0, \dots, 127$ with

⁴see <https://github.com/woc-hack/tutorial> for more information about WoC map naming convention

⁵Partitions are enumerated using the first seven bits of the sha1 representing the key – in this case commit – in order to obtain partitions of similar size. Each partition is a file sorted by the key and compressed.

⁶A blob is often copied within a repository.

⁷See section 5 for the limitations in identifying the originating project.

⁸Over 90% of the blobs belong to a single project, so excluding them reduces storage of the relations created downstream.

⁹We use the first seven bits of the name's FNV digest [35] as it is faster and randomizes better short strings than sha1.

each workflow step described above capable of being run as 128 parallel processes.

4 DATASET

The created tables are stored on WoC servers and can be found at `/da?_data/basemaps/gz/Ptb2PtFullVX.s` with X ranging from 0 to 127 based on the 7 bits in the first byte of the blob sha1. The "V" in the name indicates that this dataset is based on WoC version V¹⁰(the latest at the time of this work). The format of each file is encoded in its name, that is, each line of this dataset includes the originating repository (deforked repository), the timestamp of first commit including the blob in originating project, blob sha1, destination project (deforked repository) and the timestamp of first commit including the blob in destination project, all separated by semicolon.

format :

```
originating_repo ; timestamp ; blob ;
destination_repo ; timestamp
```

example :

```
MeigeJia_ECE -364 ; 1514098666 ;
010000001b502dcb0fc8e89d4f854979c93503f8 ;
HaoboChen1887_Purdue ; 1598024605
```

This means blob `010000001b502dcb0fc8e89d4f854979c93503f8` was first seen in `MeigeJia_ECE-364`¹¹ repository at 1466402956 (Jun 20 2016) and was reused by `HaoboChen1887_Purdue` at 1551632725 (Mar 03 2019).

5 LIMITATIONS

Blob-level reuse. Our dataset is at entire blob reuse granularity and does not capture the reuse of pieces of code that form only a part of the file. Thus blob-level reuse (despite being common) does not represent the full extent of all code reuse.

Notably, different versions of the same file would have different blobs as even if two versions differ by only one character, they still produce different file hashes (are different blobs). Thus blob reuse is not the same as file reuse. File reuse is, however, difficult to define precisely as it is not clear what files should be considered equivalent in distinct projects.

Commit time. The reuse timeline (and identifying the first occurrence) of a blob is based on the commit timestamp. This time is not always accurate as it depends on the user's system time. We used suggestions by [7] and other methods to eliminate incorrect or questionable timestamps. We also used version history information to ensure time of parent commits do not postdate that of child commits.

Originating repository. The accuracy of origination estimates can be increased by the completeness of data. Even if we assume that the WoC collection is complete, some blobs may have been originated

in a private repository and then copied to a public repository, i.e., the originating repository in WoC may not be the actual creator of the blob. For example, a 3D cannon pack asset¹² was committed by 38 projects indexed by WoC. That asset, however, was created earlier in Unity Asset Store.

Copy instance. A unique combination of blob, originating project and destination project may not always reflect the actual copy pattern because some destination projects may have copied the blob not from the originating project (e.g., for projects O, A, and B in blob creation order, project B may copy either from project O or A). Also, some blobs are not copied but are created independently in each repository, e.g, an empty string, or a standard template automatically created by a common tool. We use the list of such blobs provided by WoC [31] to exclude them from all our calculations.

As was described in each paragraph, we took all the necessary steps to minimize the potential negative impact of these limitations and validated the curated data extensively to ensure its reliability within the boundaries of limitations.

6 FUTURE WORK

Dependency-based reuse. In this dataset we only introduce the network of copy-based reuse. To better understand reuse in general, it is of great importance to draw a complete picture by creating the reuse network of copy-based alongside dependency-based reuse as one is not telling the complete story without the other.

Code-snippet granularity. Another expansion area to better understand reuse is setting the granularity to code snippet level. That will produce a much more complex network of reuse that potentially offers a great opportunity to have a more in-depth analysis of reuse.

Upstream repository. As was mentioned in the limitations section, we still do not exactly know where a repository copied the blob from and consider it to be the originating repository in all copy instances. But to better understand how developers find a repository to reuse code from, meta heuristic algorithms or artificial intelligence techniques could be utilized to predict where the code was copied from in each copy instance.

¹⁰<https://bitbucket.com/swsc/overview>

¹¹Slash symbols are substituted with underscores in WoC repository naming convention, that is, `MeigeJia_ECE-364` means `github.com/MeigeJia/ECE-364`. Furthermore, the project is hosted on Github unless the domain is mentioned at the beginning of project name.

¹²<https://assetstore.unity.com/packages/3d/props/weapons/stylish-cannon-pack-174145>

REFERENCES

- [1] Qurat Ul Ain, Wasi Haider Butt, Muhammad Waseem Anwar, Farooque Azam, and Bilal Maqbool. 2019. A systematic review on code clone detection. *IEEE access* 7 (2019), 86121–86144.
- [2] Le An, Ons Mlouki, Foutse Khomh, and Giuliano Antoniol. 2017. Stack overflow: a code laundering platform?. In *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 283–293.
- [3] Russ Cox. 2019. Surviving Software Dependencies: Software reuse is finally here but comes with risks. *Queue* 17, 2 (2019), 24–47.
- [4] Massimiliano Di Penta, Daniel M German, Yann-Gaël Guéhéneuc, and Giuliano Antoniol. 2010. An exploratory study of the evolution of software licensing. In *2010 ACM/IEEE 32nd International Conference on Software Engineering*, Vol. 1. IEEE, 145–154.
- [5] Muyue Feng, Weixuan Mao, Zimu Yuan, Yang Xiao, Gu Ban, Wei Wang, Shiyang Wang, Qian Tang, Jiahuan Xu, He Su, Binghong Liu, and Wei Huo. 2019. Open-Source License Violations of Binary Software at Large Scale. In *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 564–568. <https://doi.org/10.1109/SANER.2019.8667977>
- [6] Felix Fischer, Konstantin Böttinger, Huang Xiao, Christian Stransky, Yasemin Acar, Michael Backes, and Sascha Fahl. 2017. Stack Overflow Considered Harmful? The Impact of Copy&Paste on Android Application Security. In *2017 IEEE Symposium on Security and Privacy (SP)*. 121–136. <https://doi.org/10.1109/SP.2017.31>
- [7] Samuel W. Flint, Jigyasa Chauhan, and Robert Dyer. 2021. Escaping the Time Pit: Pitfalls and Guidelines for Using Time-Based Git Data. In *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*.
- [8] William B Frakes and Giancarlo Succi. 2001. An industrial study of reuse, quality, and productivity. *Journal of Systems and Software* 57, 2 (2001), 99–106.
- [9] Tanner Fry, Tapajit Dey, Andrey Karanach, and Audris Mockus. 2020. A dataset and an approach for identity resolution of 38 million author ids extracted from 2b git commits. In *Proceedings of the 17th international conference on mining software repositories*. 518–522.
- [10] Daniel M German, Massimiliano Di Penta, Yann-Gael Gueheneuc, and Giuliano Antoniol. 2009. Code siblings: Technical and legal implications of copying code between applications. In *2009 6th IEEE International Working Conference on Mining Software Repositories*. IEEE, 81–90.
- [11] Daniel M German and Ahmed E Hassan. 2009. License integration patterns: Addressing license mismatches in component-based development. In *2009 IEEE 31st international conference on software engineering*. IEEE, 188–198.
- [12] Mohammad Gharehyazie, Baishakhi Ray, and Vladimir Filkov. 2017. Some from here, some from there: Cross-project code reuse in github. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. IEEE, 291–301.
- [13] Mohammad Gharehyazie, Baishakhi Ray, Mehdi Keshani, Masoumeh Soleimani Zavosht, Abbas Heydarnoori, and Vladimir Filkov. 2019. Cross-project code clones in GitHub. *Empirical Software Engineering* 24, 3 (2019), 1538–1573.
- [14] Shahla Ghobadi. 2015. What drives knowledge sharing in software development teams: A literature review and classification framework. *Information & Management* 52, 1 (2015), 82–97.
- [15] Antonios Gkortzis, Daniel Feitoso, and Diomidis Spinellis. 2021. Software reuse cuts both ways: An empirical analysis of its relationship with security vulnerabilities. *Journal of Systems and Software* 172 (2021), 110653.
- [16] Stefan Haeffliger, Georg Von Krogh, and Sebastian Spaeth. 2008. Code reuse in open source software. *Management science* 54, 1 (2008), 180–193.
- [17] Steve Hanna, Ling Huang, Edward Wu, Saung Li, Charles Chen, and Dawn Song. 2012. Juxtapp: A scalable system for detecting code reuse among android applications. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 62–81.
- [18] Hideaki Hata, Raula Gaikovina Kula, Takashi Ishio, and Christoph Treude. 2021. Research Artifact: The Potential of Meta-Maintenance on GitHub. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. 192–193. <https://doi.org/10.1109/ICSE-Companion52605.2021.00084>
- [19] Hideaki Hata, Raula Gaikovina Kula, Takashi Ishio, and Christoph Treude. 2021. Same File, Different Changes: The Potential of Meta-Maintenance on GitHub. In *Proceedings of the 43rd International Conference on Software Engineering (Madrid, Spain) (ICSE '21)*. IEEE Press, 773–784. <https://doi.org/10.1109/ICSE43902.2021.00076>
- [20] Lars Heinemann, Florian Deissenboeck, Mario Gleirscher, Benjamin Hummel, and Maximilian Irlbeck. 2011. On the extent and nature of software reuse in open source java projects. In *International Conference on Software Reuse*. Springer, 207–222.
- [21] Katsuro Inoue, Yuya Miyamoto, Daniel M German, and Takashi Ishio. 2021. Finding code-clone snippets in large source-code collection by CCGrep. In *Open Source Systems: 17th IFIP WG 2.13 International Conference, OSS 2021, Virtual Event, May 12–13, 2021, Proceedings 17*. Springer, 28–41.
- [22] Werner Janjic, Oliver Hummel, Marcus Schumacher, and Colin Atkinson. 2013. An unabridged source code dataset for research in software reuse. In *2013 10th Working Conference on Mining Software Repositories (MSR)*. 339–342. <https://doi.org/10.1109/MSR.2013.6624047>
- [23] Lingxiao Jiang, Ghassan Misherghi, Zhendong Su, and Stephane Gloudu. 2007. Deckard: Scalable and accurate tree-based detection of code clones. In *29th International Conference on Software Engineering (ICSE'07)*. IEEE, 96–105.
- [24] Elmar Juergens, Florian Deissenboeck, Benjamin Hummel, and Stefan Wagner. 2009. Do code clones matter?. In *2009 IEEE 31st International Conference on Software Engineering*. IEEE, 485–495.
- [25] Naohiro Kawamitsu, Takashi Ishio, Tetsuya Kanda, Raula Gaikovina Kula, Coen De Roover, and Katsuro Inoue. 2014. Identifying Source Code Reuse across Repositories Using LCS-Based Source Code Similarity. In *2014 IEEE 14th International Working Conference on Source Code Analysis and Manipulation*. 305–314. <https://doi.org/10.1109/SCAM.2014.17>
- [26] Charles W Krueger. 1992. Software reuse. *ACM Computing Surveys (CSUR)* 24, 2 (1992), 131–183.
- [27] Zhenmin Li, Shan Lu, Suvda Myagmar, and Yuanyuan Zhou. 2006. CP-Miner: Finding copy-paste and related bugs in large-scale software code. *IEEE Transactions on Software Engineering* 32, 3 (2006), 176–192.
- [28] Cristina V Lopes, Petr Maj, Pedro Martins, Vaibhav Saini, Di Yang, Jakub Zitny, Hitesh Sajani, and Jan Vitek. 2017. DéjàVu: a map of code duplicates on GitHub. *Proceedings of the ACM on Programming Languages* 1, OOPSLA (2017), 1–28.
- [29] Elena Lyulina and Mahmoud Jahanshahi. 2021. Building the Collaboration Graph of Open-Source Software Ecosystem. In *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*. 618–620. <https://doi.org/10.1109/MSR52588.2021.00086>
- [30] Yuxing Ma. 2018. Constructing supply chains in open source software. In *2018 IEEE/ACM 40th International Conference on Software Engineering: Companion (ICSE-Companion)*. IEEE, 458–459.
- [31] Yuxing Ma, Chris Bogart, Sadika Amreen, Russell Zaretski, and Audris Mockus. 2019. World of code: an infrastructure for mining the universe of open source VCS data. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. IEEE, 143–154.
- [32] Yuxing Ma, Tapajit Dey, Chris Bogart, Sadika Amreen, Marat Valiev, Adam Tutko, David Kennard, Russell Zaretski, and Audris Mockus. 2021. World of code: Enabling a research workflow for mining and analyzing the universe of open source vcs data. *Empirical Software Engineering* 26, 2 (2021), 1–42.
- [33] Audris Mockus. 2007. Large-scale code reuse in open source software. In *First International Workshop on Emerging Trends in FLOSS Research and Development (FLOSS'07: ICSE Workshops 2007)*. IEEE, 7–7.
- [34] Audris Mockus, Diomidis Spinellis, Zoe Kotti, and Gabriel John Dusing. 2020. A complete set of related git repositories identified via community detection approaches based on shared commits. In *Proceedings of the 17th International Conference on Mining Software Repositories*. 513–517.
- [35] Landon Curt Noll. 2012. Fowler/noll/vo (fnv) hash. *Accessed Jan* (2012).
- [36] Joel Osher, Sushil Bajracharya, and Cristina Lopes. 2010. Automated dependency resolution for open source software. In *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*. IEEE, 130–140.
- [37] Shi Qiu, Daniel M German, and Katsuro Inoue. 2021. Empirical study on dependency-related license violation in the javascript package ecosystem. *Journal of Information Processing* 29 (2021), 296–304.
- [38] David Reid, Mahmoud Jahanshahi, and Audris Mockus. 2022. The Extent of Orphan Vulnerabilities from Code Reuse in Open Source Software. In *2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE)*. 2104–2115. <https://doi.org/10.1145/3510003.3510216>
- [39] Jeffrey A. Roberts, Il-Horn Hann, and Sandra A. Slaughter. 2006. Understanding the motivations, participation, and performance of open source software developers: A longitudinal study of the apache projects. *Management Science* 52, 7 (July 2006), 984–999.
- [40] Chanchal K Roy, James R Cordy, and Rainer Koschke. 2009. Comparison and evaluation of code clone detection techniques and tools: A qualitative approach. *Science of computer programming* 74, 7 (2009), 470–495.
- [41] Susan Elliott Sim, Charles LA Clarke, and Richard C Holt. 1998. Archetypal source code searches: A survey of software developers and maintainers. In *Proceedings. 6th International Workshop on Program Comprehension. IWPC'98 (Cat. No. 98TB100242)*. IEEE, 180–187.
- [42] Manuel Sojer and Joachim Henkel. 2010. Code reuse in open source software development: Quantitative evidence, drivers, and impediments. *Journal of the Association for Information Systems* 11, 12 (2010), 2.
- [43] Jeffrey Svajlenko, Iman Keivanloo, and Chanchal K Roy. 2013. Scaling classical clone detection tools for ultra-large datasets: An exploratory study. In *2013 7th International Workshop on Software Clones (IWSC)*. IEEE, 16–22.
- [44] Martin White, Michele Tufano, Christopher Vendome, and Denys Poshyvanyk. 2016. Deep learning code fragments for code clone detection. In *2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 87–98.
- [45] Hai Zhuge. 2002. Knowledge flow management for distributed team software development. *Knowledge-Based Systems* 15, 8 (2002), 465–471.