GitDelver Enterprise Dataset (GDED): An Industrial Closed-source Dataset for Socio-Technical Research

Nicolas Riquet
NADI, University of Namur
Namur, Belgium
nicolas.riquet@unamur.be

Xavier Devroey
NADI, University of Namur
Namur, Belgium
xavier.devroey@unamur.be

Benoît Vanderose
NADI, University of Namur
Namur, Belgium
benoit.vanderose@unamur.be

ABSTRACT

Conducting socio-technical software engineering research on closed-source software is difficult as most organizations do not want to give access to their code repositories. Most experiments and publications therefore focus on open-source projects, which only provides a partial view of software development communities. Yet, closing the gap between open and closed source software industries is essential to increase the validity and applicability of results stemming from socio-technical software engineering research. We contribute to this effort by sharing our work in a large company counting 4,800 employees. We mined 101 repositories and produced the GDED dataset containing socio-technical information about 106,216 commits, 470,940 file modifications and 3,471,556 method modifications from 164 developers during the last 13 years, using various programming languages. For that, we used GitDelver, an open-source tool we developed on top of Pydriller, and anonymized and scrambled the data to comply with legal and corporate requirements. Our dataset can be used for various purposes and provides information about code complexity, self-admitted technical debt, bug fixes, as well as temporal information. We also share our experience regarding the processing of sensitive data to help other organizations making datasets publicly available to the research community.

CCS CONCEPTS

• **Social and professional topics → Software maintenance**; *Project staffing*; • **Software and its engineering → Software libraries and repositories**; *Programming teams.*

KEYWORDS

dataset showcase, socio-technical aspects, development teams

ACM Reference Format:
Nicolas Riquet, Xavier Devroey, and Benoît Vanderose. 2022. GitDelver Enterprise Dataset (GDED): An Industrial Closed-source Dataset for Socio-Technical Research. In *19th International Conference on Mining Software Repositories (MSR '22), May 23–24, 2022, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3524842.3528003

1 INTRODUCTION

Many academic researchers working on socio-technical aspects of software engineering do not have access to closed-source repositories data. Most (replication) studies therefore focus on open source projects [2, 15, 16, 21, 34]. Although there has been an increasing effort from the industry to support open source initiatives, solely relying on openly accessible sources of information represents a threat to the validity, generalizability, and applicability of research projects as the cultures of open and closed source communities may differ significantly [14, 19, 20, 24]. For instance, unlike for open source communities, closed source developers are usually all employees of the same organization. We believe this situation can undermine researchers' efforts to provide insights and solutions to practical problems faced by a large number of developers.

Yet, most organizations do not want to give external researchers access to their proprietary closed source repositories and infrastructures. It is considered a sensitive topic as it usually represents risks in terms of intellectual property, trade secrets protection, application and infrastructure security, and regulatory constraints. For instance, developer names are protected by the General Data Protection Regulation (GDPR) [17] in Europe. Our goal with this paper is to contribute to the effort of closing the gap between research on open and closed source communities. We report on our efforts and results in producing and sharing with the research community a dataset containing socio-technical information from industrial closed-source projects.

In summary, our contributions are: (1) a dataset containing socio-technical information about 101 software projects, 106,216 commits, 470,940 file modifications and 3,471,556 method modifications performed by 164 developers in 7 different programming languages (mostly Java, C#, and JavaScript, but also TypeScript, Python, PHP, C++) over 13 years; and (2) an experience report on collecting and anonymizing socio-technical information in an industrial closed-source context.

2 CONTEXT AND MOTIVATION

Software is critical for the efficient operations of *Anonymous*[1] and it manages more than six million lines of code, mostly written in Java, C#, and JavaScript. The company has also recently begun using SonarQube [26] to perform static analysis on its codebases. Complexity metrics, such as the number of lines of code (LOC) and cyclomatic complexity [12], are particularly monitored. Those metrics can quickly reveal bad coding practices and design mistakes

[1] Although the company helped us anonymizing and openly distributing the dataset, they explicitly asked us to not mention their name for public relations reasons and in case security vulnerabilities could still be found in the data despite our best efforts. We will refer to the company as *Anonymous*. Please contact us if you would like to have more information about the company.

potentially increasing technical debt [5, 9]. They can also be used to identify developers in need of coaching or teams rushing to meet a deadline [3], allowing to mitigate the social debt [29].

In order to supplement the findings provided by SonarQube and study the evolution of socio-technical indicators, the company decided to analyze historical development data by mining information from its development infrastructure. Data mining techniques on version control history and behavioral code analyses [31] can indeed help identify some of the social, organizational and higher-level engineering aspects of software development. This type of analyses can be more helpful in predicting quality problems than purely code-related metrics [34].

3 DATA COLLECTION AND ANONYMIZATION

3.1 Repository selection

The company counts hundreds of software repositories and not all of them are relevant for our research. We selected repositories according to a set of inclusion and exclusion criteria. We included repositories of applications under active development and maintenance, i.e., applications currently assigned to developers. And we excluded repositories of applications in their early phase of development, as they do not provide enough history, and various utility tools, as they are small, do not evolve, and do not represent the core business of *Anonymous*. This selection process left us 101 repositories to analyze.

3.2 Mining tool

The selected repositories were mined using GitDelver [22], an open-source tool that we developed. It is a command line tool written in Python that relies on the PyDriller Git mining framework [27], the Lizard cyclomatic complexity analyzer [32], and the Pandas data analysis library. As shown in (1) and (2) in Figure 1, it can analyze either a single repository or multiple repositories in bulk (using Python's multiprocessing), depending on the input folder path. It uses PyDriller for traversing all the commits and records and calculates modifications information on three different granularity levels: the commits, the files, and the methods. For each repository, GitDelver produces CSV files containing the collected data (Section 5 provides a detailed description of the content).

By default, during the analysis at the files and methods levels, GitDelver only keeps files for which Lizard can calculate complexity metrics. As a consequence, the data at the file and method levels do not contain information about all of the commit objects identified at the commit level as some of the latter only concern changes on non-code files (e.g., text files, images). Similarly, not all file modifications imply a change inside of methods as some of them concern changes related to format, comments, order of methods, or code outside of methods and constructors. We used the default configuration of GitDelver for our research. More details about the design and usage of the tool can be found on GitHub: https://github.com/nicolasriquet/GitDelver.

3.3 Anonymization requirements

The data collected by GitDelver contains information that the company does not want or was not allowed to make public:

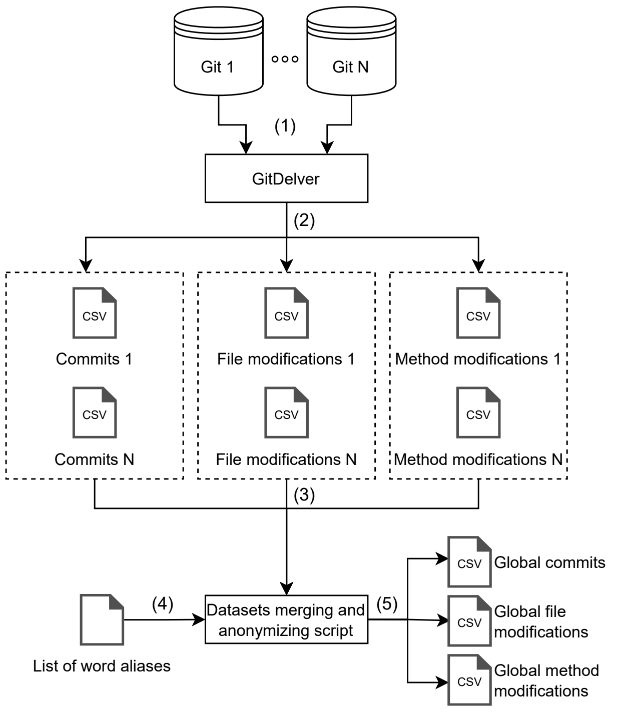

Figure 1: Dataset generation pipeline

(Anonym. 1) Personal data related to the developers, as per the GDPR. Asking for everyone's consent was not feasible as some people would not agree disclosing personal data and others had left the company.

(Anonym. 2) Code excerpts (GitDelver outputs the lines of code containing self-admitted technical debt), as the analyzed code is proprietary and the company does not want to make it public.

(Anonym. 3) The name of the company. There are two reasons for this. The first one is linked to public relations, as the company does not want its name to be used in future publications without first approving the content. Therefore, not removing the name of the company would make future uses of the dataset very impractical for all the parties involved. The second reason is that the company wants to minimize the risks of becoming the target of cyberattacks.

(Anonym. 4) Everything that might contain sensitive keywords related to the company's business or to security features that would attract the attention of malicious actors. Examples of sensitive security-related keywords include: *account*, *password*, *token*, *authenticate*, *authorization*, etc. We identified 140 of such keywords. We validated those keywords by manually exploring the dataset after the anonymization process.

4 DATASET GENERATION

Figure 1 shows an overview of the dataset generation pipeline. Overall, the process took 23 hours on a 2.50GHz Intel Core i5-7200U CPU with 8GB RAM. The manual steps involved in the data preparation and anonymization took about five days.

4.1 Mining code history and complexity

We cloned the selected repositories using the `git clone --bare` command to simplify the analysis of all Git branches at once. The repositories were then analyzed with GitDelver (step (1) in Figure 1). It mined and analyzed all the repositories and generated for each of them three CSV files containing information about (resp.) the commits, the file modifications, and the method modifications (step (2) in Figure 1).

4.2 Merging and anonymization

We rely on the Pandas library [13, 30] to merge and anonymize the outputs of GitDelver (step (3) in Figure 1) according to the following steps: (1) merging the CSV files related to the commits, files modifications, method modifications into three global CSV files. (2) Fixing and normalizing the Git author names (Git allows each developer to freely set their author name and e-mail address, and we discovered that most developers had used several different author names). For this, we manually defined renaming rules for each developer and simply replaced all their aliases with their normalized name. (3) Removing the `SATDLine` column, which contains code excerpts, from the `Files_history` CSV file to comply with **Anonym. 2**. (4) Removing the `Message` column from the `Commits_history` CSV file, as the commit messages tend to mention sensitive business and security-related information and could not be kept intelligible when scrambling keywords (**Anonym. 1**, **3**, **4**). It should be noted that most of the messages were not in English and so were of limited usefulness to the broad research community. (5) Manually preparing a list of more than 700 word replacements for developers' names and other keywords (step (4) in Figure 1), and feeding it to the script to remove or scramble all remaining sensitive information (step (5) in Figure 1) (**Anonym. 1**-**4**).

5 DATASET DESCRIPTION

This section describes the three CSVs contained in the dataset. It is important to note that the *file modifications* CSV references identifiers from the *commits* CSV and that the *method modifications* CSV uses identifiers referencing both of these CSVs (i.e., columns like *CommitId* or *FilePath* can be used as foreign keys). The **[Anonymized]** tag means that all the names contained in the column have been replaced with fictional ones. The **[Scrambled]** tag means that all values where scanned for sensitive keywords and that those have been replaced with other unrelated words. The scrambled values are mainly used for branches, file paths and file names. The goal here is to make it possible to study the evolution of these elements without compromising security. The dataset has been uploaded and is openly accessible on Zenodo [23].

***Commits_history* CSV.** The *commits_history* is a 59 MB CSV file that contains data about 106,216 commits performed by 164 developers on 101 repositories over 13 years. It has the following columns:

- *Repository*: the name of the repository. **[Anonymized]**
- *Branches*: the list of Git branch names in which this commit has been integrated (e.g., 'master', 'develop', 'release/V1.1'). **[Scrambled]**
- *NbBranches*: the number of branches in which this modification has been integrated.
- *CommitId*: the identifier of the commit.
- *Author*: the author of the modification. **[Anonymized]**
- *DateTime*: the date and time of the modification.
- *Date*: the date of the modification.
- *HourOfDay*: the hour of the day of the modification.
- *Merge*: flag telling if the commit is a merge commit.
- *BugFix*: flag telling if the modification is a bugfix. This relies on the analysis of all commit messages and the detection of the following keywords: `fix`, `solve`, `bug`, `defect`, `problem`. This list of keywords matches our observations at *Anonymous* and can be extended in GitDelver.
- *SATD*: flag telling if the modification contains Self-Admitted Technical Debt. This relies on the analysis of all added or modified lines of code and the detection of the following keywords in code comments: `todo`, `fixme`, `tofix`, `hack`, `workaround`. This list of keywords matches our observations at *Anonymous* and can be extended in GitDelver.
- *NbModifiedFiles*: the total number of files modified.
- *ModifiedFiles*: the list of files modified by this commit.
- *NbModifiedProdSourceFiles*: the number of production source files modified by this commit.
- *NbModifiedTestSourceFiles*: the number of test source files modified by this commit.
- *NbModifications*: the total number of modifications.
- *NbInsertions*: the number of insertions done by the commit.
- *NbDeletions*: the number of deletions done by the commit.

***Files_history* CSV.** The *files_history* is a 258 MB CSV file that contains information about 470,940 file modifications performed by 153 developers in 53,630 commits on 101 repositories over 13 years. As explained in Section 3.2, *files_history* only contains information about source code files and tracks less commits and committers than *commits_history*. It has the following columns:

- *Repository*: the name of the repository. **[Anonymized]**
- *Branches*: the list of Git branch names in which this modification has been integrated (e.g., 'master', 'develop', 'release/V1.1'). **[Scrambled]**
- *NbBranches*: the number of branches in which this modification has been integrated.
- *OldFilePath*: the old relative path to the file. **[Scrambled]**
- *FilePath*: the relative path to the file. **[Scrambled]**
- *FileName*: the name of the file. **[Scrambled]**
- *FileExtension*: the file extension.
- *FileType*: the type of the file (`Production` or `Test`).
- *ChangeType*: the type of the change (`ADD`, `COPY`, `RENAME`, `DELETE`, `MODIFY` or `UNKNOWN`).
- *NbMethods*: the number of methods in the file.
- *NbMethodsChanged*: the number of methods that have been modified in this file for this commit.
- *NLOC*: the number of lines of code of the file.
- *Complexity*: the Weighted Methods per Class complexity, i.e., the sum of the cyclomatic complexity numbers of all the methods of the file.
- *NlocDivByNbMethods*: the number of lines of code of the file divided by the number of methods of the file.
- *ComplexDivByNbMethods*: the complexity of the file divided by the number of methods of the file.

- *SATD*: flag telling if the modification contains Self-Admitted Technical Debt.
- *NbLinesAdded*: the number of lines added.
- *NbLinesDeleted*: the number of lines deleted.
- *CommitId*: the identifier of the commit.
- *Author*: the author of the modification. **[Anonymized]**
- *DateTime*: the date and time of the modification.
- *Date*: the date of the modification.
- *HourOfDay*: the hour of the day of the modification.

***Methods_history* CSV.** The *methods_history* is a 1,787 MB CSV file that contains information about 3,471,556 method modifications performed by 153 developers in 50,477 commits on 101 repositories over 13 years. *methods_history* only tracks commits involving changes in the content of methods. It has the following columns:

- *Repository*: the name of the repository. **[Anonymized]**
- *Branches*: the list of Git branch names in which this modification has been integrated (e.g., 'master', 'develop', 'release/V1.1'). **[Scrambled]**
- *NbBranches*: the number of branches in which this modification has been integrated.
- *OldFilePath*: the old relative path to the file. **[Scrambled]**
- *FilePath*: the relative path to the file. **[Scrambled]**
- *FileName*: the name of the file. **[Scrambled]**
- *FileType*: the type of the file ("Production" or "Test").
- *MethodName*: the name of the method. **[Scrambled]**
- *NbParams*: the number of parameters in the signature.
- *NLOC*: the number of lines of code of the method.
- *Complexity*: the cyclomatic complexity number of the method.
- *CommitId*: the identifier of the commit.
- *Author*: the author of the modification. **[Anonymized]**
- *DateTime*: the date and time of the modification.
- *Date*: the date of the modification.
- *HourOfDay*: the hour of the day of the modification.

6 DISCUSSION AND FUTURE WORK

Applications and research opportunities. The main value of this work is to provide researchers the opportunity to study development practices in a closed-source industrial context and conduct new and replication studies on a variety of socio-technical topics. Examples include the code branching strategies [18, 25], commit change patterns [11] and classification of commits [8], the time of the day at which certain types of commits are done (e.g., bug fixes, commits containing self-admitted technical debt) [6], the co-evolution of production and test files [33], the evolution of complexity metrics [1], etc.

Moreover, GitDelver can be used to extend the dataset by analyzing other repositories. This would allow to compare the practices of *Anonymous* to those of other companies and open-source communities. Sharing this tool allows other organizations to study their own codebases without necessarily disclosing their data.

Lessons learned on sharing industrial data. Anonymizing datasets for public sharing does not require a lot of work and can be done in a few days with freely available tools and minimal data analysis skills. The most complicated part is to take the plunge and decide which data can be shared as-is and which should be anonymized or scrambled first. Of course, someone inside the organization could easily de-anonymize the data if they have access to the analyzed software repositories. This is not a problem since they are authorized to view the information in the first place and are contractually bound to preserve its confidentiality.

The approach we followed here is straightforward. As a general lesson, it can be summed up by asking the following questions: (1) Does the dataset contain data that you have to protect because of regulatory concerns? If yes then anonymize them. (2) Does the dataset contain sensitive business or security-related information? If yes then scramble them. (3) Do you see public relations concerns with what could be done with your data? If yes then make sure to scramble everything sensitive and make the datasets public without mentioning the name of your organization.

7 RELATED WORK

Several repository mining tools have been developed over the years, such as Boa [4], which is used to analyze large-scale SVN repositories, and TNM [28] which is specialized in detailed social analyses. Our tool is built on top of the PyDriller Git mining framework [27], which offers integration with Lizard's cyclomatic complexity analyzer [32] and can easily be used in conjunction with Python data analysis libraries such as Pandas [30].

Various datasets have also been provided to perform research on commit activities and related data. For instance, the Jira Repository Dataset [16] presents data extracted from the Jira issue-tracking systems of four open-source communities and can be used for research about the social aspects of development. The GHTorent dataset [7] provides data about a large number of GitHub's public repositories and the Technical Debt Dataset [10] provides technical debt information about 33 open-source projects from the Apache Software Foundation. The GDED dataset that we present here differs in that it provides data about industrial, closed-source, software repositories and we believe it can be used in various kinds of socio-technical research and for conducting replication studies.

8 CONCLUSION

All software development organizations are different and some of them are more open to sharing development data than others. In this paper, we report our effort for building, anonymizing, and sharing GDED, a dataset containing socio-technical information about 101 closed-source industrial applications' histories, covering a period of 13 years. We built this dataset using GitDelver, an open-source tool that we developed for this purpose. GDED can be used to answer various research questions about the company's development activities.

We intend to follow up on this work by exploring various research opportunities offered by GDED to better understand the development and mitigation of socio-technical debt in the industry. We also plan to extend the generated datasets with information coming from other data sources (e.g., issues, builds, etc.) to provide a more complete picture of the situation in the organization.

REFERENCES

[1] Mamdouh Alenezi and Khaled Almustafa. 2015. Empirical analysis of the complexity evolution in open-source software systems. *International Journal of*

Hybrid Information Technology 8, 2 (2015), 257–266. https://doi.org/10.14257/ijhit.2015.8.2.24

[2] Guilherme Avelino, Leonardo Teixeira Passos, André C. Hora, and Marco Tulio Valente. 2016. A novel approach for estimating Truck Factors. In *24th IEEE International Conference on Program Comprehension, ICPC 2016, Austin, TX, USA, May 16-17, 2016*. IEEE Computer Society, 1–10. https://doi.org/10.1109/ICPC.2016.7503718

[3] Bahareh Bafandeh Mayvan, Abbas Rasoolzadegan, and Abbas Javan Jafari. 2020. Bad smell detection using quality metrics and refactoring opportunities. *Journal of Software: Evolution and Process* 32, 8 (2020), e2255. https://doi.org/10.1002/smr.2255

[4] Robert Dyer, Hoan Anh Nguyen, Hridesh Rajan, and Tien N. Nguyen. 2013. Boa: a language and infrastructure for analyzing ultra-large-scale software repositories. In *35th International Conference on Software Engineering, ICSE '13, San Francisco, CA, USA, May 18-26, 2013*, David Notkin, Betty H. C. Cheng, and Klaus Pohl (Eds.). IEEE Computer Society, 422–431. https://doi.org/10.1109/ICSE.2013.6606588

[5] Neil Ernst, Rick Kazman, and Julien Delange. 2021. *Technical Debt in Practice*. The MIT Press. https://doi.org/10.7551/mitpress/12440.001.0001

[6] Jon Eyolfson, Lin Tan, and Patrick Lam. 2011. Do time of day and developer experience affect commit bugginess. In *Proceedings of the 8th International Working Conference on Mining Software Repositories, MSR 2011 (Co-located with ICSE), Waikiki, Honolulu, HI, USA, May 21-28, 2011, Proceedings*, Arie van Deursen, Tao Xie, and Thomas Zimmermann (Eds.). ACM, 153–162. https://doi.org/10.1145/1985441.1985464

[7] Georgios Gousios. 2013. The GHTorent dataset and tool suite. In *Proceedings of the 10th Working Conference on Mining Software Repositories, MSR '13, San Francisco, CA, USA, May 18-19, 2013*, Thomas Zimmermann, Massimiliano Di Penta, and Sunghun Kim (Eds.). IEEE Computer Society, 233–236. https://doi.org/10.1109/MSR.2013.6624034

[8] Lile Hattori and Michele Lanza. 2008. On the nature of commits. In *23rd IEEE/ACM International Conference on Automated Software Engineering - Workshop Proceedings (ASE Workshops 2008), 15-16 September 2008, L'Aquila, Italy*. IEEE, 63–71. https://doi.org/10.1109/ASEW.2008.4686322

[9] Philippe Kruchten, Robert L. Nord, and Ipek Ozkaya. 2012. Technical debt: From metaphor to theory and practice. *IEEE Software* 29, 6 (2012), 18–21. https://doi.org/10.1109/MS.2012.167

[10] Valentina Lenarduzzi, Nyyti Saarimäki, and Davide Taibi. 2019. The Technical Debt Dataset. In *Proceedings of the Fifteenth International Conference on Predictive Models and Data Analytics in Software Engineering, PROMISE 2019, Recife, Brazil, September 18, 2019*, Leandro L. Minku, Foutse Khomh, and Jean Petric (Eds.). ACM, 2–11. https://doi.org/10.1145/3345629.3345630

[11] Matias Martinez and Martin Monperrus. 2019. Coming: a tool for mining change pattern instances from git commits. In *Proceedings of the 41st International Conference on Software Engineering: Companion Proceedings, ICSE 2019, Montreal, QC, Canada, May 25-31, 2019*, Joanne M. Atlee, Tevfik Bultan, and Jon Whittle (Eds.). IEEE / ACM, 79–82. https://doi.org/10.1109/ICSE-Companion.2019.00043

[12] Thomas J. McCabe. 1976. A Complexity Measure. *IEEE Trans. Software Eng.* 2, 4 (1976), 308–320. https://doi.org/10.1109/TSE.1976.233837

[13] Wes McKinney et al. 2010. Data structures for statistical computing in python. In *Proceedings of the 9th Python in Science Conference*, Vol. 445. Austin, TX, 51–56.

[14] Birendra K. Mishra, Ashutosh Prasad, and Srinivasan Raghunathan. 2002. Quality and Profits Under Open Source Versus Closed Source. (2002), 32. http://aisel.aisnet.org/icis2002/32

[15] Mathieu Nassif and Martin P. Robillard. 2017. Revisiting Turnover-Induced Knowledge Loss in Software Projects. In *2017 IEEE International Conference on Software Maintenance and Evolution, ICSME 2017, Shanghai, China, September 17-22, 2017*. IEEE Computer Society, 261–272. https://doi.org/10.1109/ICSME.2017.64

[16] Marco Ortu, Giuseppe Destefanis, Bram Adams, Alessandro Murgia, Michele Marchesi, and Roberto Tonelli. 2015. The JIRA Repository Dataset: Understanding Social Aspects of Software Development. In *Proceedings of the 11th International Conference on Predictive Models and Data Analytics in Software Engineering, PROMISE 2015, Beijing, China, October 21, 2015*, Ayse Bener, Leandro L. Minku, and Burak Turhan (Eds.). ACM, 1:1–1:4. https://doi.org/10.1145/2810146.2810147

[17] European Parliament and Council of the European Union. 2021. *Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation) (Text with EEA relevance)*. https://eur-lex.europa.eu/eli/reg/2016/679/oj

[18] Shaun Phillips, Jonathan Sillito, and Robert J. Walker. 2011. Branching and merging: an investigation into current version control practices. In *Proceedings of the 4th International Workshop on Cooperative and Human Aspects of Software Engineering, CHASE 2011, Waikiki, Honolulu, HI, USA, May 21, 2011*, Marcelo Cataldo, Cleidson R. B. de Souza, Yvonne Dittrich, Rashina Hoda, and Helen Sharp (Eds.). ACM, 9–15. https://doi.org/10.1145/1984642.1984645

[19] Vidyasagar Potdar and Elizabeth Chang. 2004. Open source and closed source software development methodologies. In *26th International Conference on Software Engineering*. IET, 105–109. https://doi.org/10.1049/ic:20040275

[20] S. Raghunathan, A. Prasad, B.K. Mishra, and Hsihui Chang. 2005. Open source versus closed source: software quality in monopoly and competitive markets. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans* 35, 6 (2005), 903–918. https://doi.org/10.1109/TSMCA.2005.853493

[21] Foyzur Rahman and Premkumar T. Devanbu. 2011. Ownership, experience and defects: a fine-grained study of authorship. In *Proceedings of the 33rd International Conference on Software Engineering, ICSE 2011, Waikiki, Honolulu , HI, USA, May 21-28, 2011*, Richard N. Taylor, Harald C. Gall, and Nenad Medvidovic (Eds.). ACM, 491–500. https://doi.org/10.1145/1985793.1985860

[22] Nicolas Riquet. 2022. *nicolasriquet/GitDelver: 1.7.3*. https://doi.org/10.5281/zenodo.5817559

[23] Nicolas Riquet, Xavier Devroey, and Benoît Vanderose. 2022. *GDED (GitDelver Enterprise Dataset)*. https://doi.org/10.5281/zenodo.5838537

[24] Brian Robinson and Patrick Francis. 2010. Improving industrial adoption of software engineering research: a comparison of open and closed source software. In *Proceedings of the International Symposium on Empirical Software Engineering and Measurement, ESEM 2010, 16-17 September 2010, Bolzano/Bozen, Italy*, Giancarlo Succi, Maurizio Morisio, and Nachiappan Nagappan (Eds.). ACM. https://doi.org/10.1145/1852786.1852814

[25] Emad Shihab, Christian Bird, and Thomas Zimmermann. 2012. The effect of branching strategies on software quality. In *2012 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM '12, Lund, Sweden - September 19 - 20, 2012*, Per Runeson, Martin Höst, Emilia Mendes, Anneliese Amschler Andrews, and Rachel Harrison (Eds.). ACM, 301–310. https://doi.org/10.1145/2372251.2372305

[26] SonarQube. 2021. *Code Quality and Code Security - SonarQube*. https://www.sonarqube.org/

[27] Davide Spadini, Maurício Finavaro Aniche, and Alberto Bacchelli. 2018. PyDriller: Python framework for mining software repositories. In *Proceedings of the 2018 ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2018, Lake Buena Vista, FL, USA, November 04-09, 2018*, Gary T. Leavens, Alessandro Garcia, and Corina S. Pasareanu (Eds.). ACM, 908–911. https://doi.org/10.1145/3236024.3264598

[28] Nikolai Sviridov, Mikhail Evtikhiev, and Vladimir Kovalenko. 2021. TNM: A Tool for Mining of Socio-Technical Data from Git Repositories. In *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*. IEEE, 295–299. https://doi.org/10.1109/MSR52588.2021.00041

[29] Damian A. Tamburri, Philippe Kruchten, Patricia Lago, and Hans van Vliet. 2015. Social debt in software engineering: insights from industry. *Journal of Internet Services and Applications* 6, 1 (dec 2015), 10. https://doi.org/10.1186/s13174-015-0024-6

[30] The pandas development team. 2020. *pandas-dev/pandas: Pandas*. https://doi.org/10.5281/zenodo.5574486

[31] Adam Tornhill. 2018. *Software Design X-Rays: Fix Technical Debt with Behavioral Code Analysis*. Pragmatic Bookshelf.

[32] Terry Yin. 2021. *Lizard's GitHub page*. https://github.com/terryyin/lizard

[33] Andy Zaidman, Bart Van Rompaey, Serge Demeyer, and Arie van Deursen. 2008. Mining Software Repositories to Study Co-Evolution of Production & Test Code. In *First International Conference on Software Testing, Verification, and Validation, ICST 2008, Lillehammer, Norway, April 9-11, 2008*. IEEE Computer Society, 220–229. https://doi.org/10.1109/ICST.2008.47

[34] Thomas Zimmermann, Andreas Zeller, Peter Weissgerber, and Stephan Diehl. 2005. Mining version histories to guide software changes. *IEEE Transactions on Software Engineering* 31, 6 (2005), 429–445. https://doi.org/10.1109/TSE.2005.72