A Time Series-Based Dataset of Open-Source Software Evolution

Bruno L. Sousa
Department of Computer Science - UFMG
Belo Horizonte, Minas Gerais, Brazil
bruno.luan.sousa@dcc.ufmg.br

Mariza A. S. Bigonha
Department of Computer Science - UFMG
Belo Horizonte, Minas Gerais, Brazil
mariza@dcc.ufmg.br

Kecia A. M. Ferreira
Department of Computing - CEFET-MG
Belo Horizonte, Minas Gerais, Brazil
kecia@cefetmg.br

Glaura C. Franco
Department of Statistics - UFMG
Belo Horizonte, Minas Gerais, Brazil
glaura@est.ufmg.br

ABSTRACT

Software evolution is the process of developing, maintaining, and updating software systems. It is known that the software systems tend to increase their complexity and size over their evolution to meet the demands required by the users. Due to this fact, researchers have increasingly carried out studies on software evolution to understand the systems' evolution pattern and propose techniques to overcome inherent problems in software evolution. Many of these works collect data but do not make them publicly available. Many datasets on software evolution are outdated, and/or are small, and some of them do not provide time series from software metrics. We propose an extensive software evolution dataset with temporal information about open-source Java systems. To build this dataset, we proposed a methodology of four steps: selecting the systems using a criterion, extracting and measuring their releases, and generating their time series. Our dataset contains time series of 46 software metrics extracted from 46 open-source Java systems, and we make it publicly available.

CCS CONCEPTS

• **Software and its engineering** → **Software evolution**; • **Information systems** → **Data mining**; **Temporal data**; • **General and reference** → *Metrics*.

KEYWORDS

dataset, software evolution, software metrics, time series, open-source software

ACM Reference Format:
Bruno L. Sousa, Mariza A. S. Bigonha, Kecia A. M. Ferreira, and Glaura C. Franco. 2022. A Time Series-Based Dataset of Open-Source Software Evolution. 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/10.1145/3524842.3528492

1 INTRODUCTION

Software evolution consists of developing, maintaining, and updating software systems [27]. This process allows the systems to adapt to the several demands and become more robust in terms of functionality. Lehman et al. [28] pointed out that as a software system evolves, it increases in growth and complexity, whereas its internal quality declines.

Many studies have investigated how the process of software evolution occurs [1, 2, 4–8, 11, 13, 16–18, 21–26, 30, 31, 33, 35, 37], but there is a lack of software evolution data to be analyzed. Some works have made efforts in providing datasets and making them public [10, 14, 19, 29, 36, 38]. Nevertheless, they are not too large, are outdated, or do not provide time series about software metrics.

This work builds a comprehensive software evolution dataset containing temporal data of open-source Java systems. We defined a methodology composed of four steps. First, we determined a selection process with requirements for selecting a system. Second, we built an approach to make releases from the software systems. Third, we extracted static software metrics for the systems' releases. Fourth, we generated the time series of the metrics values. We used GitHub[1] to collect the data of the software systems. We have applied this dataset to a study to model the software evolution process. In this work, we present the dataset and make it available so that other studies on software evolution may use it.

2 CONSTRUCTION OF THE DATASET

This section describes the steps we followed to build the dataset. Figure 1 presents an overview of these four steps.

Step 1 - Selecting the Subject Systems. We identified systems using a selection criterion composed of two phases. First, we selected nine systems from COMETS [10]. We included them because COMETS is an important time series dataset and contains relevant systems. However, it was created in 2010 and is outdated. Second, we based on the information provided by GitHub to define the characteristics a system should have to be selected. Therefore, we considered the following aspects to select a software system:

1. **Programming Language:** the system needs to be developed in Java. We concentrated on Java because it is a programming language widely used in software engineering empirical studies and is highly popular among developers.
2. **Popularity:** this requirement ensures the selection of popular software systems. We considered here the number of stars from the projects in GitHub to characterize their popularity. The higher the number of stars the higher its popularity.

[1] https://github.com/

MSR '22, May 23–24, 2022, Pittsburgh, PA, USA

ACM ISBN 978-1-4503-9303-4/22/05…$15.00
https://doi.org/10.1145/3524842.3528492

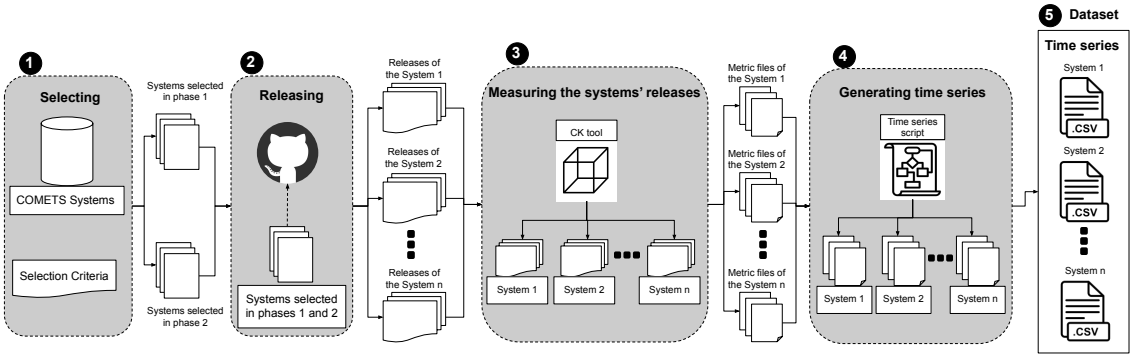

Figure 1: Dataset creation process

Therefore, we considered the software systems with the highest number of stars.

3. **Activity:** the system must have at least 5,000 commits. We defined 5,000 as the minimum limit of commits for a system because this was the lower value identified in the software systems selected in the first phase.
4. **Lifetime:** to include a system in the dataset, it must have at least five years.
5. **Not be deprecated:** the year of the last commit in the project should be 2020. This requirement avoids selecting systems which have been abandoned by the developer community.

We implemented a Python script using a REST API provided by GitHub applying the criteria defined to select the systems. We obtained 46 software systems. Nine of them belong to COMETS, and 37 were selected in the second phase. It is essential to highlight that `TV-Browser` considered in COMETS was discontinued in 2013. Therefore, as it is not updated, we did not consider it.

Step 2 - Extracting the Software Systems' Releases. We carried out a process to extract the systems' source code in a versioned way. We based on information provided by GitHub about the whole lifetime of the software systems and defined an interval to delineate a release. We formalized a release as source code information of software regarding bi-weeks, i.e., 14 days.

We implemented a script in Python. We used Python because it works efficiently with a large quantity of data, and it provides a library that deals directly with the REST API provided by GitHub. The library we used is named PyGitHub.[2]

To get the released source code of the systems, we defined the following approach. We identify the repository of a particular system on GitHub by providing the complete name of this repository for our approach. It is essential to highlight that the full name of the repository is always composed of a user's name followed by a bar ("/") and the name of the system repository. For instance, if we want to access the `Eclipse JDT CORE` repository, we need to inform "eclipse/eclipse.jdt.core" to the script. Observe that "eclipse" is the owner user's name of this repository, and "eclipse.jdt.core" is the name of the system repository we want to access.

Our script creates a folder using the project's name and clones the repository inside its respective folder. With support of PyGitHub, our script analyzes the data and extracts its release list considering the timeframe of the life of the project on GitHub. At this moment, the releases list of the project is only a sequence of bi-week periods that indicates the beginning and end of each release. The script downloads the source code of each release, considering the last commit from the bi-week interval, inside the project's folder considering its release list, creates a sub-folder for each release, and stores the respective source code inside this sub-folder.

Step 3 - Collecting Metrics. We computed static metrics of the releases of the 46 selected systems. We collected a total of 46 software metrics that characterize several aspects of the software, such as size, cohesion, inheritance hierarchy, coupling, complexity, and others. To compute the set of metrics, we used a tool named CK tool. CK tool is an open-source tool hosted on GitHub that computes a broad set of code metrics for Java projects at class-level, method-level, and variable-level [3]. We decided to use it due to its ease of installation, good documentation, and completeness regarding the number of software metrics. Although the CK tool allows to extract metrics at method-level and variable-level, we decided to compute only the class-level metrics for our dataset, which already consists of a large quantity of information about the software systems. CK tool exports the metrics values in *CSV* files.

Step 4 - Generating Time Series. We considered the same pattern defined in [10, 12] for generating the time series. This pattern consists of defining *CSV* files for each metric *M* and each system *S*, where the lines represent the classes of *S*, the columns represent the versions, and each cell *(c,r)* consists of the metric value of class *c* in the release *r*. Then, each generated *CSV* contains time series of classes regarding a given metric in a specific system. We considered only classes that refer to the systems' core functionality to extract the time series. Then, we discarded "test" classes because they do not have this characteristic and may statistically invalidate the information provided by this dataset for future prediction studies. To remove these classes, we filtered the classes considering their directory when generating their time series. We did not generate time series from classes kept in a directory that started with "test", or from directories that had "test" as part of its complete name.

Table 1 presents the 46 open-source Java systems that compose our dataset. We make the dataset publicly available and provide additional information about it on our supplementary website [34].

[2]https://pygithub.readthedocs.io/en/latest/index.html

We also make the scripts used to construct the dataset publicly available as supplementary material. [3]

3 RELATED WORK

The literature has produced datasets for supporting studies on software evolution. Some of them are publicly available [10, 14, 36, 38] whereas others are not [15, 20, 24, 32, 33]. This section provides an overview of these two kind of studies and discusses how our study differs from the related works.

Public datasets. Aiming to support empirical studies on software evolution and evaluate bug prediction models, D'Ambros et al. [14] proposed a benchmark containing data of defects in software systems over their lifetime. The D'Ambros dataset comprises time series metrics values of five open-source Java systems. Vasa et al. [38] also proposed a dataset that provides temporal information on source code static metrics' values. It comprises 40 open-source Java systems and contains time series information on many software metrics. However, this dataset did not include several relevant software metrics, such as CK metrics [9] and coupling. Tempero et al. [36] built a corpus composed of 111 software systems for supporting empirical studies on software engineering, named Qualitas Corpus, containing evolution data of 14 systems and did not represent these data as time series. Couto et al. [10] proposed a dataset named COMETS, composed of time series of 17 well-known software metrics data of defects existing in 10 Java software systems. Although these datasets contain software metrics data, they were created years ago and are outdated. Gousios [19] constructed a public dataset named GHTorrent, which contains information about many open source systems. It contains the main information about a software system from GitHub, such as repository name, commits, pull request, essential for extracting insights about the software systems. However, it does not provide time series from the software systems. Ma et al. [29] proposed World of Code (WoC), a public dataset containing information about 1.6 billion of commits made by authors of repositories from GitHub. This dataset has versioned data about the software. However, it does not provide time series regarding metrics of the source code from the systems.

Not publicly datasets. Robles et al. [33] studied how open-source software systems evolve from the perspective of size. For this purpose, they created a dataset containing time series from size metrics of 20 relevant software systems. Herraiz et al. [24] investigated how software systems evolve in size from different perspectives. They built a dataset composed of time series regarding size software metrics. Their dataset contains temporal information of 13 open-source software systems. Raja et al. [32] carried out their dataset with time series regarding defects in software. Their purpose was to analyze how defects evolve in open-source systems. The proposed dataset comprises eight open-source Java systems and contains only information about defects. They spaced out the time series considering 30 days as the release interval. Darcy et al. [15] carried out an empirical study to identify the pattern that open-source software systems follow over time. They built a dataset composed of 108 software systems to perform this analysis. Their dataset contains temporal information of size and complexity metrics, which was not publicly available. Grigorio et al. [20] studied the relationship between complexity metrics and the abandonment of open-source systems. For this purpose, they created a dataset composed of 10 open-source systems and provided time series of complexity metrics. Each observation present in their dataset refers to a release defined by the project owners in SourceForge.[4]

This paper proposes a software evolution dataset composed of 46 relevant open-source Java systems hosted in GitHub. Our dataset contains time series of 46 static metrics for each software system included in it. Even though several studies proposed software evolution dataset, not all of them are available to be used in further empirical studies [15, 20, 24, 32, 33]. Although there are some of them made public [10, 14, 19, 29, 36, 38], they do not contain software metrics data and/or are outdated. Our present work differs from the studies described above because our dataset (i) is public, (ii) contains recent information on system evolution, (iii) provides software metrics time series, and (iv) to the best of our knowledge, it is the largest dataset of metrics time series, both in number of software metrics and software systems.

4 THREATS TO VALIDITY

This section presents the threats to the validity of this work and discusses the major decisions we made to mitigate them.

We defined a selection criterion and considered data from 46 open-source Java systems hosted on GitHub to build our time series dataset. As GitHub is a large control version platform with many software systems, our dataset may be considered limited to the number of systems. However, collecting time series from open-source Java systems requires a considerable effort. Besides, no other public time series dataset as extensive as the one we propose is public available. Therefore, our dataset can be advantageous to researchers and practitioners to carry out software evolution studies.

To carry out the data collection, we used tools and scripts. We used CK Tool [3] to extract the metrics' values from the systems' releases, and we also built Python scripts to release the source code of the systems and generate the time series. Although tools automate and facilitate the data collection process, they configure a threat to validity since tools may have errors. We decided to use a tool well evaluated and tested to mitigate this threat. We also made some manual checks in the results exported by CK Tool and the Python scripts to ensure that they produced results as expected.

5 CONCLUSION

This study provides a comprehensive software evolution dataset with temporal information of 46 static software metrics of 46 open-source Java systems. We defined a methodology composed of four steps to build the dataset reported in this work. First, we determined the aspects we consider relevant to choose the systems. Second, we made the releases from the source code of the systems. Third, we collected the static metrics from the releases. Finally, we generated the time series in the fourth step.

Although some literature efforts aim to provide data to support software evolution studies [10, 14, 19, 29, 36, 38], the datasets built so far do not encompass many systems or time series, and some of them are outdated. The dataset created in this work fills this gap

[3] https://github.com/BrunoLSousa/DSTool

[4] https://sourceforge.net/

Table 1: Overview of the software systems included in the dataset.

ID	System Name	Repository on GitHub	# of Versions	Timeframe
1	Alluxio	Alluxio/alluxio	64	2018-04-24 – 2020-12-08
2	Antlr4	antlr/antlr4	264	2010-01-28 – 2020-11-30
3	Arduino	arduino/Arduino	372	2005-08-25 – 2020-12-03
4	Bazel	bazelbuild/bazel	141	2015-02-25 – 2020-12-09
5	Bisq	bisq-network/bisq	162	2014-04-11 – 2020-12-04
6	Buck	facebook/buck	184	2013-04-18 – 2020-11-06
7	CAS	apereo/cas	252	2010-07-22 – 2020-11-25
8	CoreNLP	stanfordnlp/CoreNLP	180	2013-06-27 – 2020-11-16
9	Dbeaver	dbeaver/dbeaver	199	2012-10-03 – 2020-12-04
10	Dropwizard	dropwizard/dropwizard	223	2011-10-07 – 2020-12-02
11	Druid	alibaba/druid	232	2011-05-11 – 2020-11-18
12	Eclipse JDT Core	eclipse/eclipse.jdt.core	473	2001-06-05 – 2020-11-06
13	Eclipse PDE UI	eclipse/eclipse.pde.ui	474	2001-05-24 – 2020-11-09
14	Elasticsearch	elastic/elasticsearch	262	2010-02-08 – 2020-11-11
15	Equinox Framework	eclipse/rt.equinox.framework	414	2003-11-25 – 2020-11-24
16	FrameworkBenchmarks	TechEmpower/FrameworkBenchmarks	187	2013-03-22 – 2020-11-24
17	Gocd	gocd/gocd	162	2014-04-12 – 2020-12-05
18	Graylog	Graylog2/graylog2-server	252	2010-07-31 – 2020-12-04
19	Guava	google/guava	273	2009-09-01 – 2020-11-16
20	Hibernate Orm	hibernate/hibernate-orm	326	2007-06-29 – 2020-11-16
21	J2ObjC	google/j2objc	201	2012-09-05 – 2020-12-06
22	Jabref	JabRef/jabref	418	2003-10-14 – 2020-12-12
23	Jenkins	jenkinsci/jenkins	342	2006-11-05 – 2020-11-20
24	Jitsi	jitsi/jitsi	371	2005-07-21 – 2020-10-14
25	JMeter	apache/jmeter	86	2017-05-26 – 2020-12-05
26	JUnit 5	junit-team/junit5	125	2015-10-17 – 2020-12-03
27	K-9 Mail	k9mail/k-9	293	2008-10-28 – 2020-11-08
28	Kafka	apache/kafka	227	2011-08-01 – 2020-11-25
29	LanguageTool	languagetool-org/languagetool	73	2017-12-16 – 2020-12-14
30	Lucene	apache/lucene-solr	259	2010-03-28 – 2020-11-14
31	MinecraftForge	MinecraftForge/MinecraftForge	229	2011-07-12 – 2020-12-05
32	Neo4j	neo4j/neo4j	329	2007-05-24 – 2020-11-25
33	Netty	netty/netty	299	2008-08-08 – 2020-11-17
34	OpenRefine	OpenRefine/OpenRefine	258	2010-04-26 – 2020-11-28
35	OrientDB	orientechnologies/orientdb	260	2010-03-29 – 2020-11-30
36	Pentaho Kettle	pentaho/pentaho-kettle	329	2007-05-16 – 2020-11-17
37	Pentaho Platform	pentaho/pentaho-platform	223	2011-09-21 – 2020-11-16
38	Pinpoint	pinpoint-apm/pinpoint	153	2014-08-23 – 2020-12-03
39	PMD	pmd/pmd	449	2002-06-21 – 2020-11-27
40	Realm Java	realm/realm-java	210	2012-04-20 – 2020-12-03
41	RxJava	ReactiveX/RxJava	192	2012-12-28 – 2020-11-15
42	Spring Boot	spring-projects/spring-boot	185	2013-04-19 – 2020-11-22
43	Spring Framework	spring-projects/spring-framework	294	2008-10-23 – 2020-11-18
44	Spring Security	spring-projects/spring-security	407	2004-03-16 – 2020-12-01
45	Tomcat	apache/tomcat	358	2006-03-27 – 2020-12-07
46	Tutorials	eugenp/tutorials	184	2013-04-29 – 2020-11-17

because it contains many systems and software metrics, provides time series, and contains recent information about the systems.

Future work may (i) maintain the dataset by incorporating new open-source Java systems; (ii) extend the dataset by including evolution dataset regarding software developed in other programming languages; (iii) create a specific dataset with software evolution data regarding proprietary systems; and (iv) apply the dataset in empirical studies on software evolution based on software metrics.

ACKNOWLEDGMENTS

This work was supported by CAPES, CNPq, and FAPEMIG.

REFERENCES

[1] Arwa Abuasad and Izzat M Alsmadi. 2012. Evaluating the correlation between software defect and design coupling metrics. In *2012 International Conference on Computer, Information and Telecommunication Systems (CITS)*. IEEE, 1–5.

[2] M. Alenezi and M. Zarour. 2015. Modularity measurement and evolution in object-oriented open-source projects. *ACM International Conference Proceeding Series* 24-26-September-2015 (2015). https://doi.org/10.1145/2832987.2833013

[3] M. Aniche. 2015. *Java code metrics calculator (CK)*. Available at: https://github.com/mauricioaniche/ck/. Accessed on January, 2021.

[4] A. Capiluppi, J. Fernandez-Ramil, J. Higman, H.C. Sharp, and N. Smith. 2007. An empirical study of the evolution of an agile-developed software system. In *Proceedings of the 29th international conference on Software Engineering*. IEEE Computer Society, 511–518.

[5] A. Capiluppi, M. Morisio, and J.F. Ramil. 2004. The evolution of source folder structure in actively evolved open source systems. In *10th International Symposium on Software Metrics, 2004. Proceedings.* IEEE, 2–13.

[6] A. Capiluppi, M. Morisio, and J.F. Ramil. 2004. Structural evolution of an Open Source system: A case study. *Program Comprehension, Workshop Proceedings* 12 (2004), 172–182.

[7] A. Capiluppi and J.F. Ramil. 2004. Studying the evolution of open source systems at different levels of granularity: Two case studies. *International Workshop on Principles of Software Evolution (IWPSE)* (2004), 113–118.

[8] Michelle Cartwright. 1998. An empirical view of inheritance. *Information and Software Technology* 40, 14 (1998), 795–799.

[9] S. R. Chidamber and C. F. Kemerer. 1994. A metrics suite for object oriented design. *IEEE Transactions on Software Engineering* 20, 6 (June 1994), 476–493.

[10] C. Couto, C. Maffort, R. Garcia, and M. T. Valente. 2013. COMETS: a dataset for empirical research on software evolution using source code metrics and time series analysis. *ACM SIGSOFT Software Engineering Notes* 38, 1 (2013), 1–3.

[11] Cesar Couto, Christofer Silva, Marco Tulio Valente, Roberto Bigonha, and Nicolas Anquetil. 2012. Uncovering causal relationships between software metrics and bugs. In *2012 16th European Conference on Software Maintenance and Reengineering*. IEEE, 223–232.

[12] C. F. M. Couto. 2013. *Predicting Software Defects with Causality Tests*. Ph. D. Dissertation. UFMG, Belo Horizonte, Minas Gerais.

[13] John Daly, Andrew Brooks, James Miller, Marc Roper, and Murray Wood. 1996. Evaluating inheritance depth on the maintainability of object-oriented software. *Empirical Software Engineering* 1, 2 (1996), 109–132.

[14] M. D'Ambros, M. Lanza, and R. Robbes. 2010. An extensive comparison of bug prediction approaches. In *MSR 2010*. IEEE, 31–41.

[15] David P Darcy, Sherae L Daniel, and Katherine J Stewart. 2010. Exploring complexity in open source software: Evolutionary patterns, antecedents, and outcomes. In *2010 43rd Hawaii International Conference on System Sciences*. IEEE, 1–11.

[16] Sinan Eski and Feza Buzluca. 2011. An empirical study on object-oriented metrics and software evolution in order to reduce testing costs by predicting change-prone classes. In *2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops*. IEEE, 566–571.

[17] M.W. Godfrey and Q. Tu. 2000. Evolution in open source software: A case study. In *Proceedings 2000 International Conference on Software Maintenance*. IEEE, 131–142.

[18] J.M. Gonzalez-Barahona, G. Robles, M. Michlmayr, J.J. Amor, and D.M. German. 2009. Macro-level software evolution: a case study of a large software compilation. *Empirical Software Engineering* 14, 3 (2009), 262–285.

[19] Georgios Gousios. 2013. The GHTorent Dataset and Tool Suite. In *Proceedings of the 10th Working Conference on Mining Software Repositories* (San Francisco, CA, USA) *(MSR '13)*. IEEE Press, 233–236.

[20] F. Grigorio, D. Brito, E. Anjos, and M. Zenha-Rela. 2015. On systems project abandonment: An analysis of complexity during development and evolution of FLOSS systems. *IEEE International Conference on Adaptive Science and Technology, ICAST* 2015-January (2015). https://doi.org/10.1109/ICASTECH.2014.7068139

[21] Rachel Harrison, Steve Counsell, and Reuben Nithi. 2000. Experimental assessment of the effect of inheritance on the maintainability of object-oriented systems. *Journal of Systems and Software* 52, 2-3 (2000), 173–179.

[22] Les Hatton, Diomidis Spinellis, and Michiel van Genuchten. 2017. The long-term growth rate of evolving software: Empirical results and implications. *Journal of Software: Evolution and Process* 29, 5 (2017), e1847.

[23] I. Herraiz, J.M. Gonzalez-Barahona, and G. Robles. 2007. Towards a theoretical model for software growth. In *Fourth International Workshop on Mining Software Repositories (MSR'07: ICSE Workshops 2007)*. IEEE, 21–21.

[24] Israel Herraiz, Gregorio Robles, Jesús M González-Barahona, Andrea Capiluppi, and Juan F Ramil. 2006. Comparison between SLOCs and number of files as size metrics for software evolution analysis. In *Conference on Software Maintenance and Reengineering (CSMR'06)*. IEEE, 8–pp.

[25] C. Izurieta and J. Bieman. 2006. The evolution of FreeBSD and Linux. In *Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering*. ACM, 204–211.

[26] Stefan Koch. 2007. Software evolution in open source projects—a large-scale investigation. *Journal of Software Maintenance and Evolution: Research and Practice* 19, 6 (2007), 361–382.

[27] Manny M Lehman. 1996. Laws of software evolution revisited. In *European Workshop on Software Process Technology*. Springer, 108–124.

[28] Meir M Lehman, Juan F Ramil, Paul D Wernick, Dewayne E Perry, and Wladyslaw M Turski. 1997. Metrics and laws of software evolution-the nineties view. In *Proceedings Fourth International Software Metrics Symposium*. IEEE, 20–32.

[29] Yuxing Ma, Chris Bogart, Sadika Amreen, Russell Zaretzki, 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.

[30] Bertrand Meyer. 1996. The many faces of inheritance: A taxonomy of taxonomy. *Computer* 29, 5 (1996), 105–108.

[31] Emal Nasseri, Steve Counsell, and M Shepperd. 2008. An empirical study of evolution of inheritance in Java OSS. In *19th Australian Conference on Software Engineering (aswec 2008)*. IEEE, 269–278.

[32] U. Raja, D.P. Hale, and J.E. Hale. 2009. Modeling software evolution defects: A time series approach. *Journal of Software Maintenance and Evolution* 21, 1 (2009), 49–71. https://doi.org/10.1002/smr.398

[33] Gregorio Robles, Juan Jose Amor, Jesus M Gonzalez-Barahona, and Israel Herraiz. 2005. Evolution and growth in large libre software projects. In *Eighth International Workshop on Principles of Software Evolution (IWPSE'05)*. IEEE, 165–174.

[34] SEDataset. 2021. A Time Series-Based Dataset of Open-Source Software Evolution. https://brunolsousa.github.io/software-evolution-dataset/index.html.

[35] Garry Singh and M Daud Ahmed. 2017. Effect of Coupling on Change in Open Source Java Systems. In *Proceedings of the Australasian Computer Science Week Multiconference* (Geelong, Australia) *(ACSW '17)*. ACM, Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3014812.3014835

[36] E. Tempero, C. Anslow, J. Dietrich, T. Han, J. Li, M. Lumpe, H. Melton, and J. Noble. 2010. The Qualitas Corpus: A Curated Collection of Java Code for Empirical Studies. In *Software Engineering Conference (APSEC), 2010 17th Asia Pacific*, APSEC (Ed.). IEEE, 336–345.

[37] Ewan Tempero, James Noble, and Hayden Melton. 2008. How do Java programs use inheritance? An empirical study of inheritance in Java software. In *European Conference on Object-Oriented Programming*. Springer, 667–691.

[38] R. Vasa, M. Lumpe, and A. Jones. 2010. Helix - Software Evolution Data Set. http://www.ict.swin.edu.au/research/projects/helix.