Information and Software Technology

A longitudinal study on the temporal validity of software samples

--Manuscript Draft--

Manuscript Number:	INFSOF-D-23-00572R1
Article Type:	Research paper
Keywords:	Software samples; temporal validity; longitudinal study; sample evolution
Abstract:	Context In Empirical Software Engineering, it is crucial to work with representative samples that reflect the current state of the software industry. An important consideration, especially in rapidly changing fields like software development, is that a sample collected during a particular time period continues to represent the same population in the future. However, many recent studies rely on rather old datasets to conduct their investigations. Despite the fact that it is seldom the case in which a software repository sample built several years ago accurately depicts the current state of the development industry, there is not enough empirical evidence to support this claim. Objective To analyze the evolution of a population of open-source projects, determine the likelihood of detecting significant differences over time, and study the activity history of the same projects. Method We performed a longitudinal study with 72 snapshots of quality projects from Github, covering the period between July 1st 2017 and June 1st 2023. We recorded monthly values of seven repository metrics (contributors, commits, closed pull-requests, merged pull-requests, closed issues, number of stars and forks), encompassing data from a total of 1991 repositories. Results We observed significant changes in all the metrics evaluated, with most cases showing negligible to small effect sizes. Notably, merged pull-requests registered medium effect sizes. The evolution was not equal in all the metrics, however, after five years it was unlikely that a sample of projects remained representative for any of the analyzed metrics, showing probabilities below 25%. Conclusion Although the temporal validity of a sample depends on the specific data being studied, employing datasets created several years ago does not appear to be a sound strategy if the aim is to produce results that can be extrapolated to the entire population.

Powered by Editorial Manager® and ProduXion Manager® from Aries Systems Corporation

Dear Editor,

We appreciate the comments and suggestions of the reviewers on our manuscript. Their feedback has helped us to identify areas for improvement to enhance the quality and clarity of our work.

In this letter, we address each of the reviewers' comments. We divided the comments by reviewer and numbered each point for easy reference. For each comment, we provide a response and when applicable, the excerpt from the updated manuscript reflecting the changes.

Thanks once again for your time, effort, and valuable input.

Kind Regards,

Juan Andrés, Andrés and Emanuel

Reviewer #1:

1) The text suggests (Section 1, third paragraph) that the usefulness of a sample (i.e., temporal validity) depends on having significant changes over time. However, more context and motivation are required to support this statement.

   The main idea we tried to convey is that the usefulness of a sample depends on being representative of the target population, but due to the dynamic nature of the software it is quite difficult that a sample remains representative over an extended period of time. We re-wrote the third paragraph (of Section 1) to communicate this idea better and provided also an additional context on software evolution:

   "We believe that a sample (from software repositories) built several years ago often fails to represent the current state of the development industry. One reason for this is the (temporal) software evolution, in which the structure of the system undergoes modifications, as new features, fix requests and dependency updates are addressed. These modifications make the source code more complex and the structure drifts away from its original design [17]. Furthermore, the environment where the system lives changes as well. Although a software project might have had a reasonable good quality and technological relevance at its peak of development, it is unlikely that a project will remain active for more than five years [18], [19]. Since a project evolves over time; a sample of projects created seven (or more) years ago will usually have many deprecated projects, and for those projects which still remain active their data will be outdated; affecting negatively the temporal validity of the whole sample when compared to the current target population."

2) This concern is also related to the definition of "temporal validity" given in Section 2, where it seems that open-source projects are static (cf. Section 2, second paragraph, "a representative sample collected during a particular time period accurately represents the population at a future point"). I wonder if the authors could refine the text to better explain the concept of temporal validity and its relation to the usefulness of a sample.

   Our intention was not to characterize the software (or a snapshot) as being static, but we acknowledge that the definition was a bit confusing. We tackled this notion of software evolution in the excerpt cited in comment 1) and extended the explanation in the second paragraph of Section 2.1 as follows:

   "In particular, when we analyze software projects, they must be understood as the result of a dynamic process of continuous evolution [17], which involves tasks such as, implementing new features, fixing errors, testing functionalities, documentation, among others. Changes in the technologies and environments employed to build, test and deploy the software should also be

considered, as they imply the proper update of the software to remain operational. Along this line, Ait et al. [18] and Coelho et al. [19] have reported that the likelihood of a project to survive more than five years is less than 50% in both cases. The evolutionary nature of software systems compounded with its low survival rate in a relative short period of time (5 years) highlight the importance of monitoring the data collected for testing, evaluation, or analysis, and ensuring that they remain valid to accurately depict the target population. Nonetheless, several examples of Empirical Software Engineering studies published in top-ranked journals and conferences still rely on rather old datasets."

Furthermore, we worked on the definition of temporal validity given in the first paragraph of the Section 2.1, and clarified the direct relation between temporal validity and representativeness:

"To be able to generalize the results to a desired population, the experiment sample must be representative [20]. We see temporal validity as the degree to which findings or conclusions based on data collected at one point in time can be generalized to the present day. In other words, it refers to the extent to which a representative sample collected in the past is still able to represent the same population in the future, thus a sample stops being temporally valid when the data distribution of the studied variables is no longer representative of the target population."

3) An additional related work that could be considered in Sections 2 and 3.3.2 is:
   "Adem Ait, Javier Luis Cánovas Izquierdo, Jordi Cabot: An Empirical Study on the Survival Rate of GitHub Projects. MSR 2022: 365-375"

   We appreciate the provision of the reference which is relevant for this article. It is listed as the 18th reference (in the bibliography) and also cited in the Introduction, Background and Discussion of the study.

4) The study design (Section 3) describes the steps followed to perform the longitudinal study. Regarding the data collection (section 3.2), additional details could be provided:
   a) The authors should provide the full list of keywords used to exclude repositories. It would promote replicability & reproducibility,

      The full list of keywords used to exclude repositories is reported at the end of the second paragraph, which were: benchmark, conf, demo, docs, exam, sample, tutorial, and wiki. We have re-written the sentence for clarity:

"To avoid unwanted repositories that might pass the quality thresholds (e.g., demo, tutorials or configuration repositories), we excluded repositories including any of the following keywords: benchmark, conf, demo, docs, exam, sample, tutorial, and wiki."

b) T6 aims at projects with at least three collaborators, why that threshold? Could projects developed by 1-2 developers show a different pattern?

   That threshold was set to quantify the collaboration in a repository. In Section 2.2 we included it as one of the attributes to identify quality projects, and that threshold in particular to detect collaborative projects:

   "Metrics like the number of contributors is a sign of project success that influences the process lifecycle and how a project grows over time [27]. The presence of three or more contributors indicates that there is some form of collaboration and cooperation [7]."

   Along with that citation, we manually examined the retrieved datasets using a threshold of two contributors, and in most cases we found that the number of repositories meeting all the criteria and having exactly two contributors was negligible.

c) T7-12 clearly aims at famous projects, is this representative of the whole population of projects on GitHub?

   One of the potential perils of consuming data from public code sharing platforms (e.g. Github) is the amount of noisy repositories, thus it is necessary to do some kind of filtering to avoid sampling irrelevant ones. This was stated in the first paragraph of Section 2.2:

   "The rise of code sharing platforms, such as Github, Gitlab, and Bitbucket, provided researchers with abundant data to explore and uncover new findings. However, anyone can create a repository to store everything but valuable development information, for example, homework assignments, books, images, presentation slides, resumes etc. This situation compromises data quality within code sharing platforms and could lead to erroneous conclusions in research studies. Unfortunately, manually selecting a sample of repositories for a study is unfeasible given the sheer volume. Thus, researchers provided various criteria to assess the quality of a software using repository meta-data to characterize them."

   In the same section we listed some criteria to characterize quality software and provided a rationale on why it is important to evaluate them. In the 3rd column of Table 2 are the quality criteria which the threshold aimed to assess: the history of the project (T7-T9), issue resolution (T10) and popularity of the repository (T11 and T12). Our aim was not to

represent every single repository on Github, but a population of open-source projects with valuable development data.

5) The results are discussed in Section 5, in particular the temporal validity and the project update. On the one hand, I wonder whether the fact that there are significant differences implies that we are losing validity.

   What we meant was that according to the test conducted, the samples had significant differences in all the variables implying a shift in the data distribution. Since this part can be a bit misleading, we re-phrased that sentence in the first paragraph:

   "At first glance, the statistical analysis of the snapshots performed with the Kruskal-Wallis H test revealed significant differences in all the variables, meaning that the distributions of the variables can change as time passes by. These differences show how a sample can lose representativeness if it is not updated timely."

6) On the other hand, Section 5.2 (first paragraph) includes examples of repositories where having more contributors, commits or issues does not guarantee a long term survival, which may generate some confusion with the definition of "active project" given in the paper (RQ3).

   In that section, we interpreted the results reflected in the boxplots of the Fig. 7. The examples included (now presented in the second paragraph) showcase outlier projects within the group of less activity (Q1) that had large absolute numbers of contributors, commits, issues and pull requests. The argument is that, despite a repository had a lot of activity in the past it does not guarantee it will keep it in the future. We re-phrased the sentence to avoid confusion:

   "Nonetheless, a repository having a large absolute number of contributors, commits, issues or pull requests does not guarantee it will always be actively maintained."

Reviewer #2:

1) However, there is a significant omission in the early sections regarding the reasoning behind the study. A detailed explanation of why the research is necessary, how it could change the current state of the art, the potential impact of the findings, and the practical applications of the results is missing. Expanding on these aspects would greatly enhance the quality and depth of the paper, providing more clarity on its contributions and relevance.

We extended the Introduction and Background of the article and provided more context of the problem at hand, the consequences of disregarding temporal validity when conducting an empirical study and the practical implications for the research community. Below are the paragraphs added in the Introduction:

"We believe that a sample (from software repositories) built several years ago often fails to represent the current state of the development industry. One reason for this is the (temporal) software evolution, in which the structure of the system undergoes modifications, as new features, fix requests and dependency updates are addressed. These modifications make the source code more complex and the structure drifts away from its original design [17]. Furthermore, the environment where the system lives changes as well. Although a software project might have had a reasonable good quality and technological relevance at its peak of development, it is unlikely that a project will remain active for more than four years [18], [19]. Since a project evolves over time; a sample of projects created seven (or more) years ago will usually have many deprecated projects, and for those projects which still remain active their data will be outdated; affecting negatively the temporal validity of the whole sample comparing with the current target population…

The findings of the study shed light into temporal validity and the importance of monitoring the state of a sample to keep its data up-to-date with respect to the target population. The evidence presented should motivate researchers to consider not only the definition of sampling strategies to select the projects, but also the design of maintenance strategies to detect data drift signs in the population and update the sample accordingly."

A longitudinal study on the temporal validity of software samples

Structured abstract

Context: In Empirical Software Engineering, it is crucial to work with representative samples that reflect the current state of the software industry. An important consideration, especially in rapidly changing fields like software development, is that if a sample collected years ago is used, it should continue to represent the same population in the present day to produce generalizable results. However, it is seldom the case in which a software repository sample built several years ago accurately depicts the current state of the development industry. Nevertheless, many recent studies rely on rather old datasets (seven or more years of age) to conduct their investigations.

Objective: To analyze the evolution of a population of open-source projects, determine the likelihood of detecting significant differences over time, and study the activity history of the same projects.

Method: We performed a longitudinal study with 72 snapshots of quality projects from Github, covering the period between July 1st 2017 and June 1st 2023. We recorded monthly values of seven repository metrics (contributors, commits, closed pull-requests, merged pull-requests, closed issues, number of stars and forks), encompassing data from a total of 1991 repositories.

Results: We observed significant changes in all the metrics evaluated, with most cases showing negligible to small effect sizes. Notably, merged pull-requests registered medium effect sizes. The evolution was not equal in all the metrics, however, after five years it was unlikely that a sample of projects remained representative for any of the analyzed metrics, showing probabilities below 25%.

Conclusion: Although the temporal validity of a sample depends on the specific data being studied, employing datasets created several years ago does not appear to be a sound strategy if the aim is to produce results that can be extrapolated to the current state of the population.

Manuscript
Click here to view linked References

A longitudinal study on the temporal validity of software samples

Juan Andrés Carruthers[1]; Jorge Andrés Diaz-Pace[2]; Emanuel Irrazábal[1]

[1]*Software Quality Research Group – FaCENA-UNNE, Corrientes, Argentina*

[2]*ISISTAN, CONICET-UNICEN, Tandil, Buenos Aires, Argentina*

Abstract
Context: In Empirical Software Engineering, it is crucial to work with representative samples that reflect the current state of the software industry. An important consideration, especially in rapidly changing fields like software development, is that if a sample collected years ago is used, it should continue to represent the same population in the present day to produce generalizable results. However, it is seldom the case in which a software repository sample built several years ago accurately depicts the current state of the development industry. Nevertheless, many recent studies rely on rather old datasets (seven or more years of age) to conduct their investigations. ***Objective:*** To analyze the evolution of a population of open-source projects, determine the likelihood of detecting significant differences over time, and study the activity history of the same projects. ***Method:*** We performed a longitudinal study with 72 snapshots of quality projects from Github, covering the period between July 1st 2017 and June 1st 2023. We recorded monthly values of seven repository metrics (contributors, commits, closed pull-requests, merged pull-requests, closed issues, number of stars and forks), encompassing data from a total of 1991 repositories. ***Results:*** We observed significant changes in all the metrics evaluated, with most cases showing negligible to small effect sizes. Notably, merged pull-requests registered medium effect sizes. The evolution was not equal in all the metrics, however, after five years it was unlikely that a sample of projects remained representative for any of the analyzed metrics, showing probabilities below 25%. ***Conclusion:*** Although the temporal validity of a sample depends on the specific data being studied, employing datasets created several years ago does not appear to be a sound strategy if the aim is to produce results that can be extrapolated to the current state of the population.

Keywords
Software samples, temporal validity, longitudinal study, sample evolution

Juan Andrés Carruthers * (Corresponding author)
jacarruthers@exa.unne.edu.ar
https://orcid.org/0009-0003-8229-1779

Jorge Andrés Diaz-Pace
andres.diazpace@isistan.unicen.edu.ar
https://orcid.org/0000-0002-1765-7872

Emanuel Irrazábal
eirrazabal@exa.unne.edu.ar
https://orcid.org/0000-0003-2096-5638

Competing interest. The authors have no competing interests to declare.

Acknowledgements. This work was supported by the National Council on Scientific and Technical Research (CONICET) under a PhD Fellowship (RESOL-2021-154-APN-DIR#CONICET) and the National University of the North-East (SCyT-UNNE) under Grant 21F001. It is a part of the research conducted under the Computer Science Doctorate Program at UNNE, UNaM, and UTN.

1. Introduction

The massive use of source code repositories has provided researchers and practitioners access to data from many projects, enabling the development of empirical studies [1]–[3]. To ensure these studies produce generalizable results, it is important to work with representative samples that accurately depict the current state of the software industry. By representative samples, we mean samples with properties that resemble those of a target population [4]. However, many repositories on code sharing platforms (e.g., Github, Gitlab, Bitbucket) do not always meet specific quality criteria [5], [6], such as: community support, development history, or recent activity, among others. Some quality criteria might involve checking time-related indicators that reflect recent development activity, such as: last commit date, last pull-request date, number of commits in a period of time, etc. [7]. Hence, taking a representative sample of open-source projects would imply to periodically monitor these quality indicators in order to ensure the temporal validity of the sample. By temporal validity, we refer to the accuracy and relevance of data over time [8].

As pointed out by Lewowski and Madeyski [9], relying on a static dataset (i.e., a sample that is not updated over time) might not be an appropriate strategy for conducting an empirical study with source code or repository meta-data, due to potential risks such as: using obsolete data, making incorrect assumptions about the population, and drawing non-generalizable conclusions. Therefore, since temporal validity matters, a strategy to evolve the sample becomes necessary. Secondary studies [10] have reported this is seldom the case in several publications. For instance, Jureczko and Madeyski [11] collected object-oriented product quality metrics and defect information from 48 versions of 15 open-source projects and 27 versions of 6 proprietary projects. Shepperd et al. [12] collected size and complexity metrics from NASA software projects coded in the C and C++ languages. Both datasets are publicly available and are frequently used in current empirical studies [13]–[15], but their latest versions were published more than 7 years ago. As another example, Qualitas Corpus [16] is a dataset containing the source code of 112 open-source Java projects and includes projects built with Java 1.2 and 1.5. Since the Java long-term releases are 7, 8, 11 and 17, codebases in Java 5 and 1.2 are not the most representative ones to use because they are no longer relevant.

We believe that a sample (from software repositories) built several years ago often fails to represent the current state of the development industry. One reason for this is the (temporal) software evolution, in which the structure of the system undergoes modifications, as new features, fix requests and dependency updates are addressed. These modifications make the source code more complex and the structure drifts away from its original design [17]. Furthermore, the environment where the system lives changes as well. Although a software project might have had a reasonable good quality and technological relevance at its peak of development, it is unlikely that a project will remain active for more than five years [18], [19]. Since a project evolves over time; a sample of projects created seven (or more) years ago will usually have many deprecated projects, and for those projects which still remain active their data will be outdated; affecting negatively the temporal validity of the whole sample when compared to the current target population.

To investigate temporal validity, we conducted an empirical study to assess whether a population of open-source projects undergoes significant changes over time. In particular, we performed a longitudinal study with 72 snapshots of projects from Github and analyzed the evolution of seven repository metrics. Our findings show that the temporal validity of a sample depends on the specific data being studied, however, after five years all variables showed representativeness probabilities below 25%. Furthermore, we analyzed the activity history of the projects inside the snapshots and studied the distributions of these metrics according to the repositories' recent activity.

The findings of the study shed light into temporal validity and the importance of monitoring the state of a sample to keep its data up-to-date with respect to the target population. The evidence presented should motivate researchers to consider not only the definition of sampling strategies to select the projects, but also the design of maintenance strategies to detect data drift signs in the population and update the sample accordingly.

The rest of the paper is structured as follows. Section 2 describes the background on the subject. Section 3 details the methodology for our study and presents the research questions. Section 4 reports the findings of the longitudinal study. Section 5 discusses the answers to the research questions. Section 6 discusses threats to the validity of the study. Finally, Section 7 presents the conclusion and outlines future work.

2. Background

Our study seeks to analyze the evolution of repository metrics in a population of open-source projects from Github. In this section, we shed light on certain topics used in this study, such as temporal validity, software quality evaluation and longitudinal studies in Software Engineering.

2.1. Temporal validity

In empirical research, the time when the data is allocated can influence the results of a study. This is especially the case for rapidly changing subjects, in which data distributions rarely remain stagnant over time. In the domain of online social sciences, Munger [8] presents temporal validity as an extension of external validity, arguing that quantitative research needs to be designed to ameliorate its impact. External validity concerns about the generalization of an experiment result to other environments than the one in which the study was conducted [20]. To be able to generalize the results to a desired population, the experiment sample must be representative [20]. We see temporal validity as the degree to which findings or conclusions based on data collected at one point in time can be generalized to the present day. In other words, it refers to the extent to which a representative sample collected in the past is still able to represent the same population in the future, thus a sample stops being temporally valid when the data distribution of the studied variables is no longer representative of the target population.

In particular, when we analyze software projects, they must be understood as the result of a dynamic process of continuous evolution [17], which involves tasks such as, implementing new features, fixing errors, testing functionalities, documentation, among others. Changes in the technologies and environments employed to build, test and deploy the software should also be considered, as they imply the proper update of the software to remain operational. Along this line, Ait et al. [18] and Coelho et al. [19] have reported that the likelihood of a project to survive more than five years is less than 50% in both cases. The evolutionary nature of software systems compounded with its low survival rate in a relative short period of time (5 years) highlight the importance of monitoring the data collected for testing, evaluation, or analysis, and ensuring that they remain valid to accurately depict the target population. Nonetheless, several examples of Empirical Software Engineering studies published in top-ranked journals and conferences still rely on rather old datasets.

For example, Hassouneh et al. [1] proposed a defect prediction approach and empirically validated with a subset of Jureczko & Madeyski [11] dataset. In particular, they experimented with Java apache projects ant, camel, jedit, log4j, lucene and xalan with versions 1.7, 1.6, 4.2, 1.2, 2.4 and 2.6 respectively. These versions were released between 2005 and 2009, more than 14 years ago. Ni et al. [3] compared supervised and unsupervised crossed-project defect prediction models with the datasets of D'Ambros et al. [21], Shepperd et al. [12], Wu et al. [22] and Jureczko & Madeyski [11]. All of these datasets were created or updated in 2013 or before.

Alazba & Aljamaan [2] studied machine learning approaches for code smell detection with a 74 projects from Qualitas corpus [16]. They mentioned how missing values were treated and relevant features were selected to reduce their negative effect in machine learning classifiers. Similarly, Lenarduzzi et al. [23], assessed the agreement between six static analysis tools for code quality analysis using a subset of 47 projects also from Qualitas corpus. The last release of the dataset was ten years ago.

These studies illustrate that temporal validity is not a concept broadly addressed by the scientific literature on Empirical Software Engineering. We understand these authors had valid reasons to work with these datasets, such as the scarcity of curated datasets, the dataset's popularity, limited access to the type of data needed for the study, or the effort required to retrieve the data was restrictive. Although these datasets can provide insights, they were created several years ago, we think it is necessary to stress that time can certainly pose a threat to the validity of the reported results.

Other authors came to similar conclusions and proposed approaches to tackle temporal validity. Lewowski & Madeyski [9], after discovering that within six months approximately one percent of the projects included in a dataset were either removed or relocated, developed a script to generate their dataset dynamically, rather than relying on a static dataset. More recently, Sousa et al. [24] reported the same issues with current datasets in software evolution research, and proposed a methodology to build a software dataset with temporal data.

2.2. Software quality evaluation

Software repositories offer information on the code, people, and procedures involved in software development, and can provide valuable insights into the growth and evolution of software projects. The rise of code sharing platforms, such as Github, Gitlab, and Bitbucket, provided researchers with abundant data to explore and uncover new findings. However, anyone can create a repository to store everything but valuable development information, for example, homework assignments, books, images, presentation slides, resumes etc. This situation compromises data quality within code sharing platforms and could lead to erroneous conclusions in research studies. Unfortunately, manually selecting a sample of repositories for a study is unfeasible given the sheer volume. Thus, researchers provided various criteria to assess the quality of a software using repository meta-data to characterize them. Some of these criteria are discussed below.

Collaboration. Almost all non-trivial software projects require effort and talent from many people working together [25], but the majority of GitHub users use the platform primarily for their own projects and not with the intention of collaborating with others [5]. Open-source development involves the interaction of globally dispersed developers contributing code, knowledge, and ideas over a period of time to achieve a set of common goals [26]. Metrics like the number of contributors is a sign of project success that influences the process lifecycle and how a project grows over time [27]. The presence of three or more contributors indicates that there is some form of collaboration and cooperation [7].

Another types of metrics are the ones associated with the pull-request development model, in which external contributors can propose changes to a software project without the need to share access to the central repository with the core team [28]. Contributors fork the repository and make their changes, when a set of changes is ready they create a pull-request, then a member of the project's core team inspects the changes and request for more modifications, refuse them (close) or accept them to the project's master branch (merge) [29].

History. In order to remain relevant, software must undergo continuous changes, such as, bug fixing, feature addition, preventive maintenance, or vulnerability resolution, among others [30]. Hence, having a rich development history can be viewed as a quality indicator, as it suggests that the project members have established effective processes (e.g. version and issue tracking, collaborative code review, or team management) for managing changes over an extended period of time. A usual approach to measure history of a project is through the number of commits [31]–[33], years of active development [34], [35], commit frequency [36], [37], among others.

Issues. In Software Engineering it is common practice for developers and end-users to report issues with software not performing as it should or could be better at what it does. Studies have demonstrated that this practice can be beneficial

for both open and closed source projects [38], and user feedback is widely accepted as a relevant information for software development [39]. Issue tracking systems aim at facilitating activities such as management of requirements, schedules, tasks, defects, and releases through issue reports, which serve as a request for improving a software system, fixing a bug, adding new features, or enhancing documentation.

Popularity. Jarczyk et al. [40] define that community reaction to the project is a proper measure of its quality, and a star in Github shows admiration and positive attitude towards the chosen repository. Another feature highlighted by the authors as highly correlated with popularity is the number of times a project is forked. Borges and Valente [41] performed a survey and a software repository mining study, and they were able to validate Jarczyk's claims about stars being a factor to assess repository quality (71% of respondents), finding a positive correlation between stars and forks. However, Munaiah et al. [6] discovered that setting a high thresholds (500 - 1123 stars) might exclude a large set of repositories containing quality projects that might not be popular.

Recent activity. Intuitively, to accurately represent the current state of the software industry it is necessary to gather the most recent development data, instead of projects without recent activity. The presence of change indicates that the software system is being modified to ensure its viability [17]. According to Coelho & Valente [42], common reasons for an open-source project to stop being maintained include: it was usurped by competitor, it became functionally obsolete, lack of time of the main contributor, lack of interest of the main contributor, or outdated technologies. Hence, many empirical studies considered recent activity as a selection criterion [7], [36], [43].

2.3. Longitudinal studies in Software Engineering

A longitudinal study consists in capturing repeated observations (or waves) of the same units on the same outcomes over a certain amount of time [44]. Changes are traced by repeatedly measuring the same phenomenon under the same circumstances (e.g., using the same assessment method). Longitudinal research in Software Engineering is not so widespread, this is particularly the case of studies about repository archival meta-data [45]–[47]. This subsection reports on longitudinal studies in Software Engineering topics that inspected project meta-data.

Dependency management is a well-studied subject in software maintenance, which focuses in reusable code (e.g., libraries, packages, components.) that can be included in other applications. Bavota et al. [48] studied the evolution of a Java subset of the Apache ecosystem for a period of 14 years, in terms of their size, dependencies among them, and other meta-data. The authors reported an exponential increase in the size of the projects and the number of dependencies. A follow-up study [49] extended on the analysis of project size and dependencies, but also included the number of projects' developers. Similarly, Kikas et al. [50] investigated the evolution of dependencies from the ecosystem of three programming languages: Javascript, Ruby and Rust, observing a growth over the years.

Code smells are symptoms of poor design and implementation choices that could affect the maintainability of a software system [51]. Tufano et al. [52] mined half-a-million commits from the change history of 200 open-source projects to study when code smells start manifesting in the evolution of code entities along change histories of software projects. Another topic related to code smells is technical debt, which is the debt that is accumulated when new features are prioritized over fixing known issues. In this regard, Molnar & Motogna [53] retrieved SonarQube metrics' from the entire lifespan of three open-source projects in order to investigate the presence, characteristics and long-term evolution of source code technical debt in open-source software. Trautsch et al. [54] studied the evolutionary trends of warnings produced by static analysis tools in 54 open-source projects from 2001-2017.

Project success prediction aims to evaluate the factors that influence a project's survival or abandonment rate over time. Chengalur-Smith et al. [55] examined the activity of 2.772 open-source projects on SourceForge for five years, focusing on predictors such as development base size, project age, and population niche size. Coelho et al. [19] analyzed the historical data of 2.927 projects, using factors such as number of forks, issues, pull-requests, commits,

contributors, and project owners to assess their level of maintenance activity. Similarly, Ait et al. [18] assessed the survival rate of 1127 projects with their commits, issues, pull requests, comments, and code reviews .In a more recent study, Xia et al. [7] investigated seven indicators: the number of contributors, commits, opened and closed pull-requests, opened and closed issues, and stars; using data from 1.159 projects between 2016 and 2020.

After reviewing the state of temporal validity in Software Engineering, we observe lack of empirical studies addressing this issue. Therefore, our main contribution is a longitudinal study to evaluate a population of quality open-source projects over six years in order to provide insights on the matter.

3. Study design

The study has a longitudinal design to investigate the evolution of temporal validity in a population of quality software projects. In this section, we present the research questions (RQ), provide details about the collected data, and describe the measures and data analysis process to answer each question.

3.1. Research Questions

RQ1. Does time produce changes in the population?

We evaluated seven repository metrics (contributors, commits, closed pull-requests, merged pull-requests, closed issues, number of stars and forks) in a sample of 1991 Java projects taken from Github for a time period of 6 years (from July 2017 until June 2023). Our aim is to identify whether the pass of time leads to significant differences on the distributions of the metrics above. If that is the case, we also aim to detect when the differences occur, quantify their size, and identify metrics being more affected by time.

RQ2. How long does it take for the population to change?

Based on the results of RQ1 (and assuming changes are detected), for RQ2 we determine how likely it is for the dataset to suffer a data distribution shift for the analyzed metrics. Our aim is to estimate the probability that a dataset remains representative as time passes by.

RQ3. How long does a project remain active?

To determine the duration of projects being actively maintained, we tracked the evolution of 1991 repositories and studied their activity history. Furthermore, we characterized and compared the repositories data from the latest activity registered (last year).

3.2. Data Collection

The project data used for the study are presented in Table 1 along with their descriptions and rationale. To characterize the repositories described in Section 2.2 and answer the RQs, we extracted meta-data D4 to D13. In particular, D1 and D2 were collected for RQ3 to identify the repositories over time.

Table 1. Retrieved data.

Id.	Meta-data	Description	Rationale
D1	id	Repository's global id	Identify the repository over the waves
D2	url	Repository's url	Access to the repository
D3	name	Repository's name	Keyword filtering
D4	contributors	Number of contributors	Quantify the collaboration in the repository
D5	commits	Number of commits	Quantify the history in the repository.
D6	closedPullReqCount	Number of closed pull-requests	
D7	mergedPullReqCount	Number of merged pull-requests	
D8	closedIssuesCount	Number of closed issues	Quantify the issues in the repository
D9	stargazerCount	Number of stars	Quantify the popularity in the repository
D10	forkCount	Number of forks	
D11	dateLastCommit	Last commit date	Detect recent activity in the repository
D12	closedPullReqLastDate	Last closed pull-request date	
D13	mergedPullReqLastDate	Last merged pull-request date	
D14	monthlyCommits	Number of commits in the month	Quantify monthly activity
D15	monthlyclosedPullReq	Number of closed pull-requests in the month	
D16	monthlymergedPullReq	Number of merged pull-requests in the month	

We retrieved the projects from Github, but filtered some results because many of repositories might not be suitable for Software Engineering research [6]. The filtering process consisted on discarding repositories that did not meet the thresholds in Table 2. The thresholds were based on the results of secondary studies [10], [56], and recommendations of Munaiah et al. [6], Kalliamvakou et al. [5], and Lewowski & Madeyski [9]. To avoid unwanted repositories that might pass the quality thresholds (e.g., demo, tutorials or configuration repositories), we excluded repositories including any of the following keywords: benchmark, conf, demo, docs, exam, sample, tutorial, and wiki.

Table 2. Quality thresholds and criteria.

Id.	Threshold	Criterion
T1	Java programming language.	Java projects.
T2	Public repositories.	Repository meta-data available.
T3	GIT repositories.	Support processes tools usage.
T4	Repositories that are not forks.	Original projects
T5	10000 lines of code or more.	Size
T6	3 or more contributors.	Collaboration
T7	1 or more years since creation.	History
T8	1000 commits o more.	
T9	50 pull-requests or more.	
T10	50 closed issues or more	Issues
T11	10 or more forks.	Popularity
T12	10 or more stars.	
T13	1 or more commits or merged/closed pull-request in the last month.	Recent activity

Once the collection instrument was built, we applied thresholds T1 to T12 to collect a dataset of 2077 repositories. Afterwards, we retrieved the historical data of these repositories with Xia et al. [7] script[1], which generates CSV files with the monthly activity for each repository in terms of commits, contributors, pull-requests, issues, stars and forks. We analyzed the data of each project and if it passed the complete set of thresholds of Table 2 for the date of the snapshot we included it. A snapshot was generated every 1st of the month (wave) since July 1st 2017 until June 1st 2023 (six years), which resulted in 72 snapshots retrieved. From the 2077 repositories, 86 were discarded because did not appeared in any snapshot, resulting on a dataset with information of 1991 repositories. The complete dataset

[1] https://github.com/arennax/Health_Indicator_Prediction/

including the collected data, scripts, supplementary tables and notebooks can be accessed here[2]. The date and number of repositories obtained from each snapshot are shown in Table 3.

Table 3. Repositories included in each snapshot

	2017	2018	2019	2020	2021	2022	2023
January	-	588	724	852	1033	1187	1300
February	-	637	768	888	1070	1243	1331
March	-	630	755	893	1077	1202	1298
April	-	640	775	916	1094	1226	1341
May	-	644	781	959	1106	1272	1312
June	-	655	802	964	1098	1271	1300
July	521	666	810	980	1120	1255	-
August	532	678	830	985	1102	1288	-
September	545	699	826	980	1134	1269	-
October	550	702	844	995	1128	1326	-
November	584	725	873	1053	1147	1308	-
December	594	725	869	1018	1158	1313	-

3.3. Data Analysis

In this section, we present the descriptive statistics, statistical tests, metrics and graphs used to analyze the data obtained from the dataset.

3.3.1. RQ1: Does time produce changes in the population?

We evaluated the seven metrics in Table 4 for the dataset mentioned in Section 3.2 to detect significant differences between waves. Descriptive statistics, such as median, standard deviation, skewness and kurtosis were computed to summarize the distributions of the analyzed variables. Furthermore, we used kernel density plots to visualize variables distributions and a time series to graphically assess the similarity between the medians of the waves for the same variable.

We considered statistical hypothesis testing and effect size measures to ensure that our results were valid from a statistical perspective. We chose a significance level of $\alpha = 0.05$ and applied Bonferroni's correction [57] when testing the hypotheses of the tests for each variable. Therefore, the threshold to reject the null hypotheses was 0.00714.

Table 4. Analyzed variables.

Id	Meta-data	Quality attribute
V1	contributors	Collaboration
V2	commits	History
V3	closedpullReqCount	
V4	mergedpullReqCount	
V5	closedIssuesCount	Issues
V6	stargazerCount	Popularity
V7	forkCount	

To identify the type of statistical tests most suitable for the distributions, we applied the Shapiro-Wilk test [58] to check for normality. For each variable, the null hypothesis was rejected, thus we used non-parametric tests. To compare the waves, we applied the Kruskal-Wallis H test [59], i.e., the non-parametric version of the ANOVA test. The null hypothesis states that the observations' median of all tested groups are equal. The test is applied to multiple groups simultaneously but cannot identify exactly where and how much the groups are statistically different. When

[2] https://github.com/juancarruthers/longitudinal_study

the null hypothesis was rejected, i.e., the median among all the groups is statistically different, we ran a post-hoc pairwise test to identify the pairs of groups of observations that were different. To do this, we used the Dunn's test [60], in which the null hypothesis states that there is no difference between groups.

To assess how much two groups differ, we computed the Vargha-Delaney test [61] for the effect size to characterize the magnitude of such a difference, as follows. $\hat{A}_{12} = 0.5$ if two groups (observations) are statistically indistinguishable. $\hat{A}_{12} > 0.5$ means that, on average over all observations, the first group obtains larger values than the one it is compared to, $\hat{A}_{12} < 0.5$ otherwise. The magnitude values can be summarized into 4 nominal categories, which rely on the scaled $\hat{A}_{12}$ defined as $\hat{A}_{12}^{(scaled)} = (\hat{A}_{12} - 0.5) * 2$ [62]: "negligible" ($|\hat{A}_{12}^{(scaled)}| < 0.147$), "small" ($0.147 \leq |\hat{A}_{12}^{(scaled)}| < 0.33$), "medium" ($0.33 \leq |\hat{A}_{12}^{(scaled)}| < 0.474$), and "large" ($|\hat{A}_{12}^{(scaled)}| \geq 0.474$).

3.3.2. RQ2: How long does it take for the population to change?

To answer this question, we took Dunn's test results from RQ1 and identified the snapshots in which the null hypothesis was rejected for any of the variables, and then applied a survival analysis algorithm. Survival analysis [63] is a well-known technique that models "time to event" data with the aim to estimate the survival rate of a given population, i.e., the expected time duration until a specific "event" happens. Survival analysis also takes into account the fact that some observed subjects may be "censored", either because they leave the study, or because the event of interest was not observed on them during the observation period. Survival analysis has been successfully used in a number of Software Engineering studies [36], [64], [65].

The "beginning of time" was the date of the snapshot retrieval. The event considered for the analysis was: "the snapshot is not representative", referring to the time when a significant difference is detected between a snapshot and the snapshot of a following wave. According to Baltes & Ralph [4] representativeness is dimension-specific, i.e. a sample can be representative on one parameter but not of another, therefore we provided two types of survival curves: 1) a general one that registered an event every time Dunn's test was rejected for any variable and 2) a dimension specific one that analyzed each variable individually. We calculated the duration time of the event, starting with the "beginning time" until the event occurred. Then, we use the Kaplan-Meier [66] non-parametric approximation to compute the survival curve, which is the most widely used curve for estimating survival probabilities.

3.3.3. RQ3: How long does a project remain active?

For this question, we also applied survival analysis, but in this case we performed it after inspecting the activity history of the 1991 repositories to analyze the likelihood of a project being updated over time. The "beginning of time" was the creation date of the repository and the event considered was: "the project does not show signs of activity for over a month". A sign of activity refers to a repository receiving a commit, closed pull-request or merged pull-request. To identify projects that have not been updated, we searched their commit and pull-request history and checked if they did not show any activity for a period of 30 days or more. We determined the duration as the difference between the creation date and the immediate previous date before detecting inactivity. For example, if a repository was created on '2018-09-11' and consistently logged activity at least every 30 days until '2020-01-06', we logged a duration of 482 days until the event of inactivity was detected. Then, we used the Kaplan-Meier approximation to compute the survival curve. We included a bar chart that grouped the number of projects according to their duration until the event was recorded.

Moreover, we investigated the characteristics of the projects most recent monthly activity (sum of commits, merged and closed pull-requests per month). We calculated their mean monthly activity between July 1st 2022 until June 1st 2023. Then we clustered them in two categories, lower (first quartile) and higher activity (fourth quartile). At last, we drew a series of boxplots with the groups.

4. Results

In this section we answer the research questions based on the results obtained from the data analysis.

4.1. RQ1: Does time produce changes in the population?

We initially examined the descriptive statistics from Table 5 to assess the normality of the variables. Typically, a normal distribution has skewness and kurtosis values of zero and three, respectively, regardless of the standard deviation. Therefore, all variables seem to have a non-normal distribution. Moreover, all the distributions had considerable deviations, as indicated by high kurtosis and skewness values. The kernel density plots in Fig. 1 further confirm that the variables are not normal.

Table 5. Variables descriptive statistics.

Variables	Median	Standard Deviation	Skewness	Kurtosis
V1	62	182.75	7.16	77.79
V2	2972	8787.77	5.65	48.37
V3	69	531.53	9.67	126.66
V4	332	1746.08	11.61	251.28
V5	478	1734.86	6.72	68.11
V6	413	4888.3	4.78	27.14
V7	153	1912.59	8.85	107.43

To understand the evolution of the variables over time, Fig. 2 presents a time series for each variable. The graph presents the median of the variable for each semester, where the first semester comprises the snapshots taken between July 2017 and December 2017. All the variables increased their median values over time. V7 is the one with the least increment, having a maximum value of 163 and a minimum of 134. In contrast, V4 had the biggest increase starting with a median value of 204 and finishing with 456.

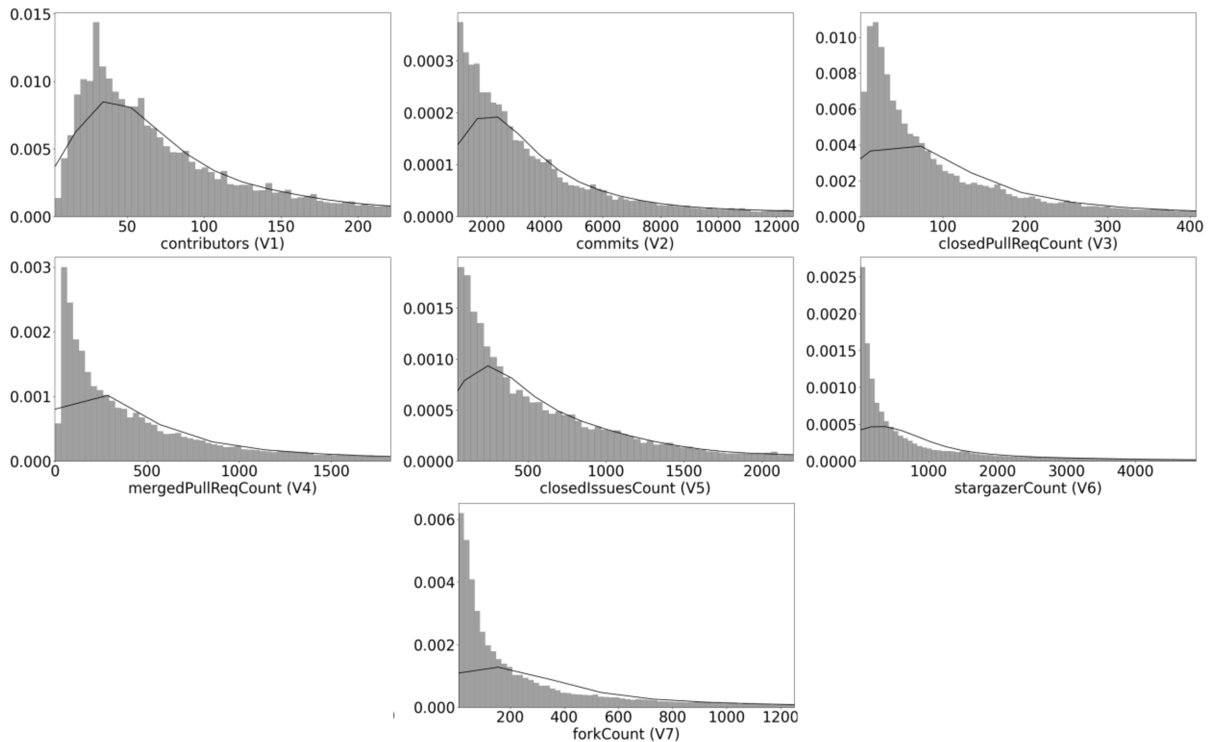

Fig. 1. Kernel density plots for each variable of interest.

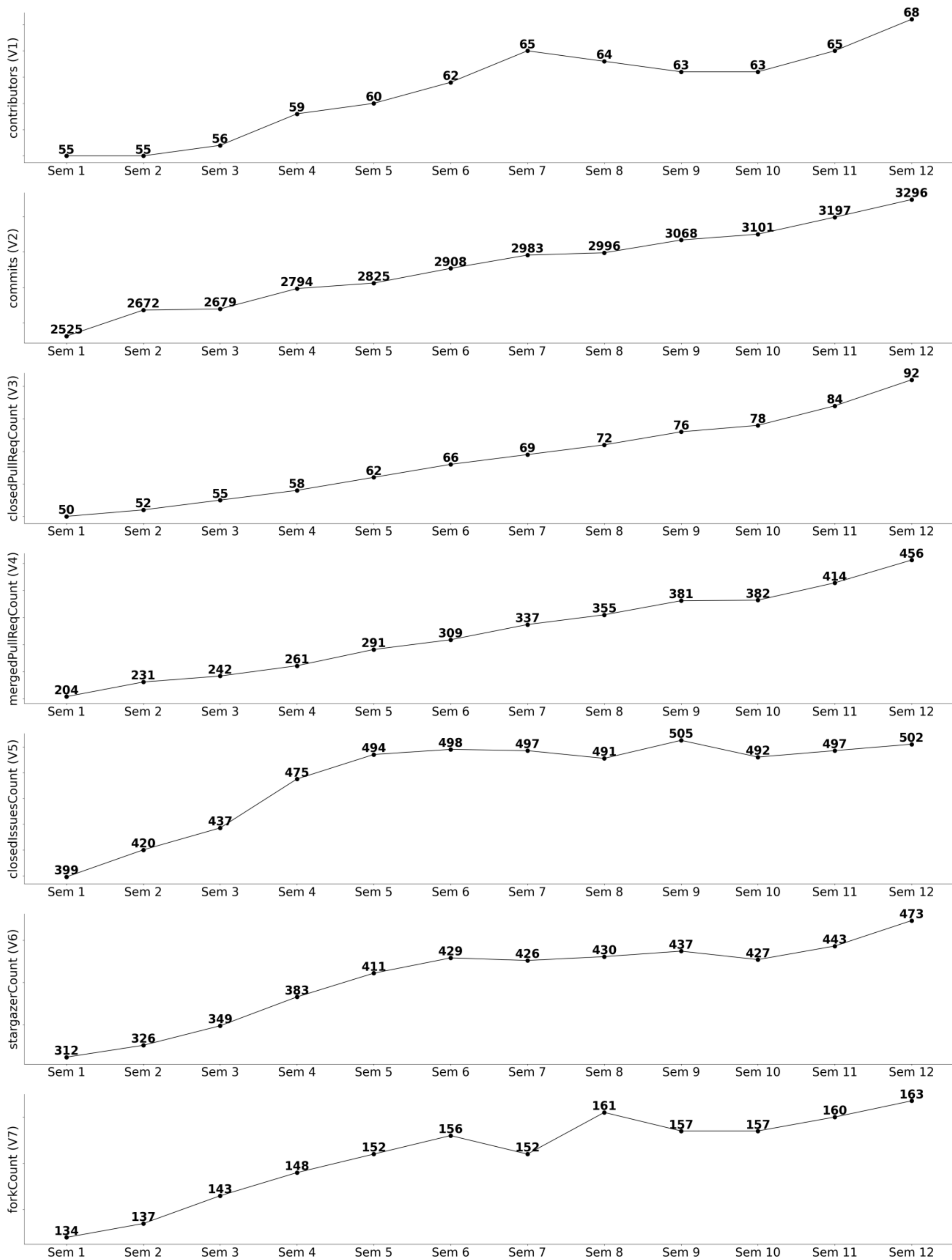

Fig. 2. Wave median evolution.

Afterwards, we proceeded with inferential statistics, and performed Shapiro-Wilk and Kruskal-Wallis H tests, as shown in Table 6. The results indicate that all variables deviate from a normal distribution, as the null hypotheses of the Shapiro-Wilk test were rejected. Additionally, the Kruskal-Wallis H test revealed significant differences for all variables, with p-values below 0.00714.

Table 6. Shapiro-Wilk and Kruskal-Wallis tests results.

Variables	Shapiro-Wilk	Shapiro-Wilk (p-value)	Normally distributed	Kruskal-Wallis H	Kruskal-Wallis H (p-value)
V1	0.44	0	No	346.56	0
V2	0.48	0	No	347.74	0
V3	0.3	0	No	1306.82	0
V4	0.38	0	No	1778.99	0
V5	0.46	0	No	215.62	0
V6	0.42	0	No	202.37	0
V7	0.3	0	No	130.29	0

We conducted Dunn's test for all the variables to identify which waves are statistically different. Then, we paired the waves with a p-value below 0.00714 and computed Vargha-Delaney to measure the magnitude of the difference. The complete tables with the test results are also in the shared dataset[2].

Starting with Dunn's test results; variables V1, V2 and V3 rejected the null hypothesis when comparing waves 1 (July 2017) and 30 (December 2019), from that point on the first wave recorded significant differences with most of the remaining waves. Similarly, V6 recorded its first difference between waves 5 (November 2017) and 31 (January 2020). V5 rejected the hypothesis sooner on waves 1 and 22 (April 2019). V4 was the variable with most differences, registering its first between waves 1 and 11 (May 2018). In contrast, V7 showed the least differences, starting on wave 6 (December 2017) and 64 (November 2022). Then, with scaled Vargha-Delaney score, we observed that all variables had negligible to small effect sizes. In particular, for V4 in waves 1 and 65 (December 2022) the effect size was medium.

After analyzing a dataset consisting of 72 waves of Java projects from Github, collected over a period of six years, we observed significant changes in all the variables. These resulted in negligible to small effect sizes in most cases and medium for V4, indicating that time is a factor that can potentially influence the distributions of repository data.

4.2. RQ2: How long does it take for the population to change?

Based on the results from Dunn's test, we generated a survival curve to analyze the representativeness of the snapshots over time. The survival curve represents the probability of a snapshot remaining representative from the time it was collected until its representativeness fades away. Fig. 3 shows the general representativeness probability of the snapshots retrieved from Github which considers all the variables. We observed that the probability of a snapshot being representative is one for the first 273 days. After a year it remains high with a probability of 82.3%. However, passing 456 days the representativeness probability drops to 51.6%. After 516 days, it is highly probable that a snapshot is not representative (24.2%). It reaches zero passing 579 days. In particular, the snapshot that remained representative the longest was the 10th one (May 2018), with 579 days. In contrast, the 59th and 61st snapshots had the shortest duration, lasting 273 days each.

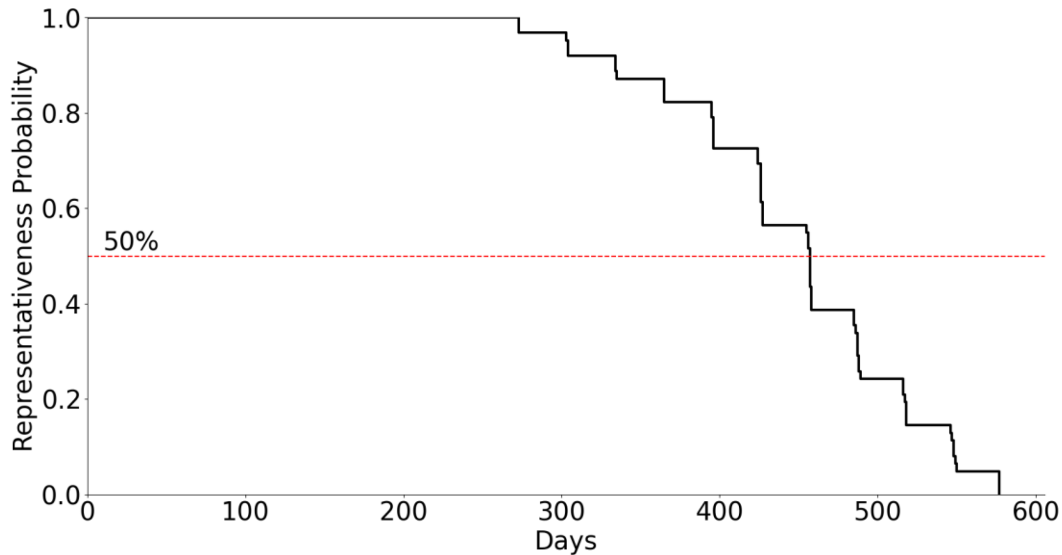

Fig. 3. General survival analysis of representativeness probability.

Moreover, in Fig. 4 we analyzed each variable individually and we detected differences regarding their temporal validity duration. For instance, V1 and V2 had a probability of 100% for over a year (577 and 669 days), remained above 50% for three years (1096 and 1005) and zero before the fourth year finished (1400 and 1280). Similarly, V5 and V6 also had more than a year with 100%, however they recorded a value above 50% for over four years. In contrast, V3 and V4 remained valid the least time, after the second year concluded they registered a value below 50%, and reached zero passing 821 and 638 days respectively. In particular, V5 was the only dimension that did not registered a probability under 25%.

The general survival analysis showed it is likely that a population of Java projects will be representative for the first year (82.3%), however it is improbable that will remain that way for the second one (0%). Regarding the dimension-specific survival curves, they exhibited a longer period of temporal validity, where depending on the variable reached as high as four years with a probability above 50%, however after five years all dropped below 25%.

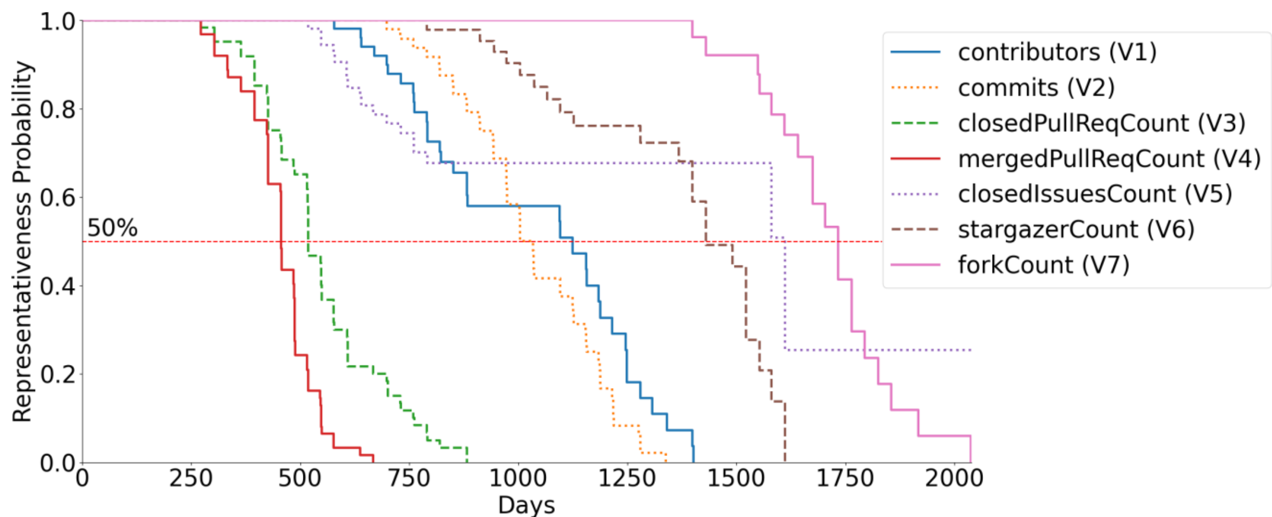

Fig. 4. Dimension-specific survival analysis of representativeness probability.

4.3. RQ3: How long does a project remain active?

Fig. 5 shows the survival plot of the 1991 repositories retrieved. The survival curve starts in 0.91 because 177 repositories did not receive a commit, closed pull-request or merged pull-request for over a month since they were created. The probability of a repository being active for the first two years is 65% and 54% respectively. After 30 months the likelihood drops to 50%, and after 161 months it declines to 14%. Repositories like *apache/jmeter*, *elastic/elasticsearch* and *spring-projects/spring-data-commons* along with other seven repositories had an uninterrupted active status for more than 150 months.

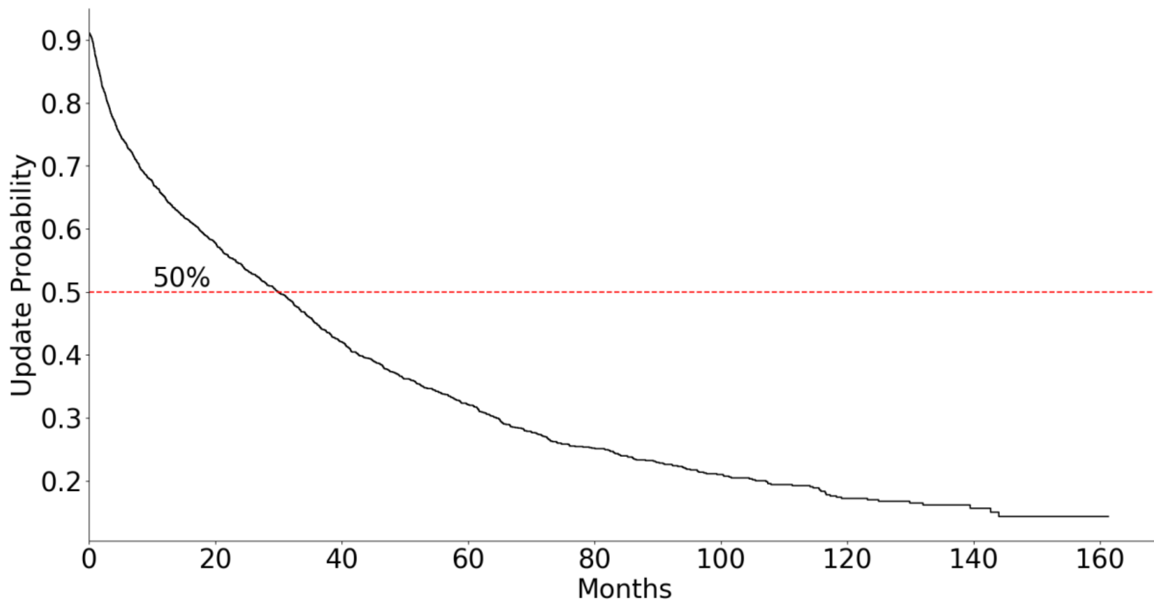

Fig. 5. Survival analysis of repository update probability.

Fig. 6. shows a bar chart with the number of projects that exhibited a sign of inactivity every year since its creation. The chart reveals that in the first year of development the 35% projects had at least a month of inactivity, and then dropped for 11% in the second year. The amount of projects registering inactivity events consistently decreased moving forward through the years, with an average of almost 118 projects between the 3rd and 6th year. However, after the 7th year of development the average number of projects further declined to 1% each passing year.

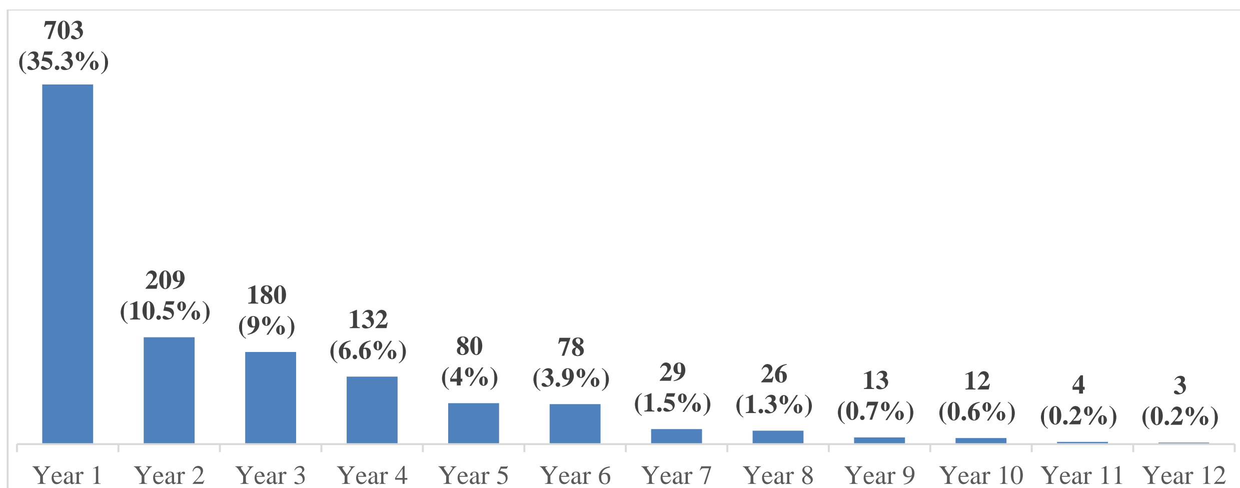

Fig. 6. Number of repositories grouped by first inactivity detection.

Afterwards, we investigated the most recent activity of these repositories, spanning from July 1st 2022 through June 1st 2023. We classified them according to their monthly activity into two groups, lower (first quartile) and higher (fourth quartile) recorded activity, and plotted boxplots for each variable with the groups (see Fig. 7). A boxplot is graph that represent the dispersion metrics, where the box encloses the first and third quartiles, while the whiskers present the lowest and highest datum within the 1.5 interquartile range. As it can be seen, there is a difference between the groups. In all cases the quartile values of the higher group are larger when compared to those of the lower group. The last graph shows the same groups distributions by project age (i.e., the period of time since the repository was created until the last activity was recorded) in months. In contrast with the variables V1 to V7, the quartile numbers of the lower quartile are larger than the higher one.

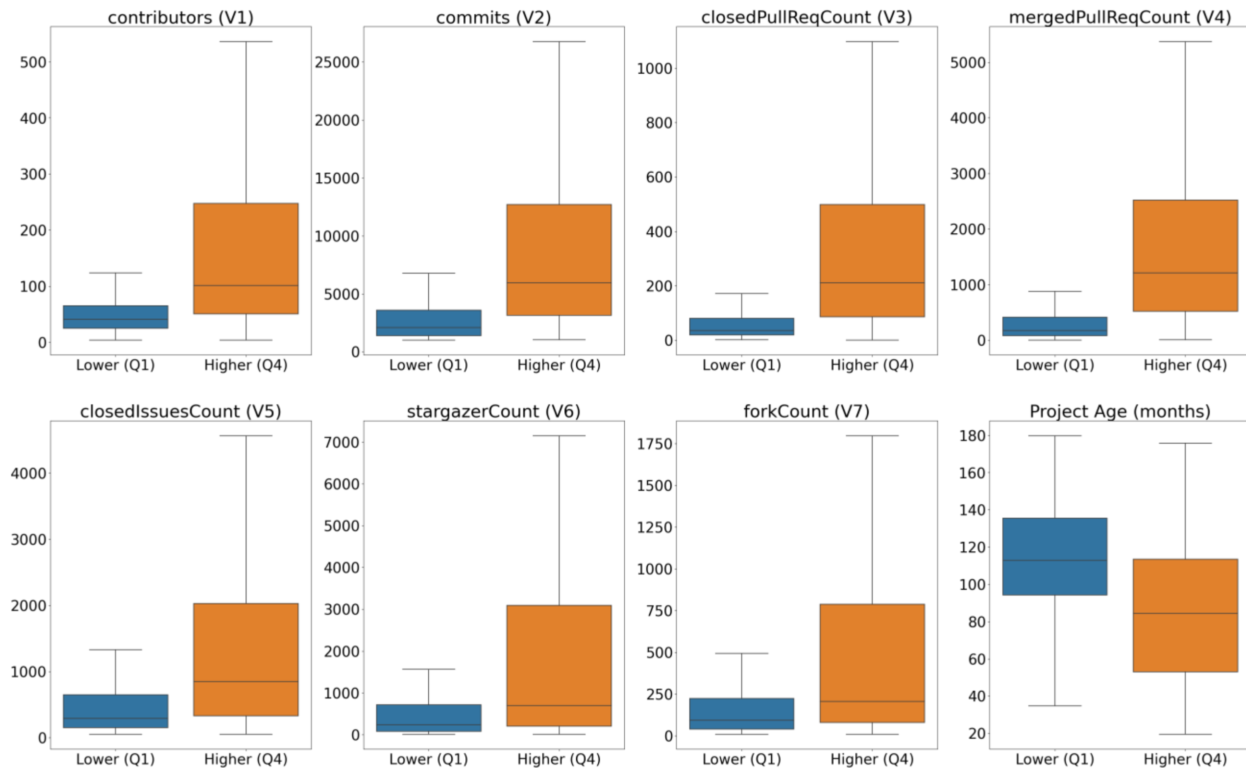

Fig. 7. Repositories grouped by update probability.

The likelihood of a repository remaining consistently updated for a year is high (65%), however after 30 months it starts to be more unlikely to happen. Furthermore, more active projects (Q4) tend to have more contributors, commits, pull-requests, issues, stars and forks than projects with less activity (Q1), and also tend to have less months of development.

5. Discussion

In this section we interpret our findings and discuss them in the context of the research questions and the literature on the subject.

5.1. Datasets temporal validity

We assessed whether a population of open-source projects undergoes shifts in terms of contributors, commits, pull-requests, issues, stars and forks distributions over the course of six years. At first glance, the statistical analysis of the

snapshots performed with the Kruskal-Wallis H test revealed significant differences in all the variables, meaning that the distributions of the variables can change as time passes by. These differences show how a sample can lose representativeness if it is not updated timely. Furthermore, from Fig. 3 these shifts become apparent in a relatively short period of time with a 0% representativeness probability after 579 days. The changes detected by hypothesis testing, while statistically significant in most cases, were accompanied by negligible to small effect sizes, which seems to indicate that the practical implications might not be substantial. The only case the effect size was medium was in merged pull-requests. Although these variable changes can be explained by their median values growth as shown in Fig. 2, they did not evolve equally.

The growth detected in the collaboration metrics, e.g. number of collaborators and pull-requests, reflects how the distributed software development process provided by Github's collaborative framework is expanding. In particular, the most impact was in pull-requests metrics, where merged pull-requests and closed pull-requests had a year average increase of 20.5% and 14% respectively. Gousios et al. [29], [67], reported on these trend shifts from classic ways of contributing, such as change sets sent to development mailing lists, to issue tracking systems, or through direct access to the version control system to the pull-based model. The inclusion of contributions external from the core development team certainly increased the number of contributors over the years.

Regarding the number of commits, it can be explained by the inherent evolution of software projects over time. This phenomenon aligns with Lehman's laws [68], which describe the progression of software and its tendency to increase in complexity over time. Recent empirical studies [69]–[71] have corroborated these laws, further highlighting that the amount of commits and source code files in open-source code doubles approximately every 30 and 22 months, respectively.

With respect to variables associated with project popularity such as fork count and stars, we detected a consistent growth year to year. Borges & Tulio Valente [41] studied popularity, specifically in terms of project stars. They observed that repositories tend to receive more stars after their public release. However, after this initial surge, the growth rate stabilizes. There is also an acceleration in the number of stars gained after subsequent releases of a project.

In contrast, closed issues did not register a constant increase all over the timespan considered, in fact, none of the snapshots retrieved after January 1st 2019 recorded significant differences with the subsequent ones. Similarly, Cosentino et al. [47] reviewed Github’s issue tracking system usage, and they reported that issues are generally stable or exhibit a slightly negative trend over time.

Although the evolution is not the same in all the variables and the temporal validity of a sample depends on the specific data being studied, as a general rule employing datasets created several years ago does not appear to be a sound strategy if the aim is to produce generalizable results. After five years, all variables showed representativeness probabilities below 25%.

5.2. Project Update

In RQ3, we analyzed the likelihood of a project being actively updated over time using a dataset of 1991 Java systems. The survival curves depicted in Fig. 5 revealed an update probability of 50% after 30 months, with subsequent decreases to 45%, 37% and 32% in the third, fourth and fifth year respectively. A comparison with Ait et al. [18] shows a 50% probability starting from the third year, based on 1127 PHP and R projects. The disparity can be mainly attributed to the survival condition used to identify “alive” systems: “it is alive, if it has shown activity of any kind in the last six months”, and secondarily to the choice of programming languages.

Furthermore, we classified the projects according to their most recent activity (from July 1st 2022 until June 1st 2023), and the boxplots in the Fig. 7 indicated that the projects with more activity (Q4) have more contributors, commits,

pull-requests, issues, stars and forks than the ones in Q1. These results are in line with previous longitudinal studies. For example, Coelho et al. [19] reported on the disparity between projects with higher and lower survivability on GitHub, specifically examining the total number of contributors, issues, and pull-requests. The authors highlighted how these variables might have a correlation on the survivability of open-source projects. Nonetheless, a repository having a large absolute number of contributors, commits, issues or pull requests does not guarantee it will always be actively maintained. For example, *ControlSystemStudio/cs-studio* is a collection of tools to monitor and operate large scale control systems created on 2012 with more than 100 contributors, 27k commits and 1k closed issues, and last year recorded only four commits. As another example, *Cuba-platform/cuba* is a development framework for enterprise applications dating from 2016 that has more than 100 contributors, 14k commits and 2k closed issues; however last year recorded 10 commits. *Facebook/buck* is a build system created on 2013 having more than 400 contributors, 22k commits and 1k closed issues, and its last commit was on April 2023.

In contrast, when considering age, projects with more activity (Q4) are younger than projects with less activity (Q1), their median ages are 80 and 108 months respectively. This pattern has also been acknowledged by Xia et al. [7] and Coelho et al. [19]. In their respective studies, they observed that older projects tend to experience reduced programmers' activity.

In summary, our findings suggest that 1) it is unlikely that a project remains actively maintained for an extended period of time (more than four years), and 2) active projects tend to be younger and attract greater participation. Younger projects, being in a stage of active development attract more attention from programmers. In contrast, as projects mature the projects' activity decline.

6. Threats to Validity

In this section, we discuss the threats to validity identified for our work, according to the four basic types of validity suggested by Wohlin et al. [20].

6.1. Construct Validity

Construct validity is concerned with the relation between theory and observation. We used the repository metrics number of contributors, commits, closed pull-requests, merged pull-requests, closed issues, stars, forks and total size of code as a proxy of collaboration, history, issues, popularity and size in a project, as reported in the relevant literature (Section 2.2). Another threat is related to the projects selected for analysis. We defined the thresholds for quality repositories in Section 3.2 to assure no toy projects were collected, and also, we manually assessed a test snapshot retrieved by the extraction instrument.

6.2. Internal Validity

Internal validity is threatened by external influences that we did not, or were not able to consider when trying to infer cause-effect relationships. Since this study spans a long duration with multiple data collection points over time, there is a possibility that external factors might have influenced the observed outcomes. These external factors could introduce changes in the dataset independently of the variables being studied. For example, changes in the software development industry, technological advancements, or shifts in user preferences. Although time itself may not be the direct cause of the observed changes, we believe that these external factors are intrinsically linked to the passage of time. Therefore, to some extent, time can be considered a contributing factor to these changes.

6.3. External Validity

External validity is concerned with the generalizability of the conclusions of the study. We tackled this threat using a whole frame sampling approach [4] and considering multiple snapshots at different points in time. It should be noted that we collected 72 months of data (six years), if the time interval had been longer we could have detected greater effect sizes. However, sample sizes before July 1st 2017 were considerably smaller in comparison, which would have compromised tests' statistical power, e.g. the number of subjects for January 2017 and July 2016 snapshots would have been 442 and 353 respectively. Moreover, our dataset only contains open-source Java projects hosted on GitHub, based on findings of our secondary studies [10], [56]. Thus, the generalization to other languages and projects is limited, and further studies might be needed to confirm our results.

6.4. Conclusion Validity

Threats to the conclusion validity are concerned with issues that affect the ability to draw the correct conclusion about relations between the treatment and the outcome of an experiment. We addressed low statistical power using a large sample for each snapshot (more than 520 repositories). In the case of statistical tests, we graphically and quantitatively analyzed variables and used non-parametric hypothesis tests. Furthermore, we included effect size measures to quantify the difference in rejected hypothesis. To avoid error rate, i.e. the distortion of significance level due to multiple analysis, we adjusted α applying the Bonferroni's correction.

7. Conclusions

In this article, we conducted a longitudinal study to analyze the evolution of a population of Java projects from Github. Since July 1st 2017, we generated 72 snapshots comprising all the repositories characterized as quality projects. The complete dataset contains data from 1991 repositories. In the analysis, we tackled three research questions about whether the population exhibits data shifts, when this occurs, and how much time repositories remain actively maintained.

The study revealed statistical differences in the number of contributors, commits, pull-request, closed issues, stars and forks due to their growth over time. Analyzing these facts jointly, we observed that the probability of a snapshot being representative after a year is rather high with a probability of 82.3%, and zero before reaching its second year (579 days). Studying their growth individually, we noticed that they did not evolve in the same way, depending on the variable they reached as high as four years with a probability above 50%. However, after five years all variables get values under 25%.

Regarding project updates, the likelihood of a repository being active consistently for more than 30 months is close to 50%. After 161 months it declines further to 14%, with a small set of projects recording more than 150 months of active development. Moreover, we characterized projects in terms of their last year monthly activity and discovered that more active projects tend to have more contributors, commits, pull-requests, issues, stars and forks. Furthermore, when considering the age of the projects, younger projects have more activity than older ones. These findings suggest that projects with more recent activity tend to be younger, and also receive more attention, registering more contributors, commits, pull-request, issues, stars and forks.

Our main contribution in this study is to provide evidence of the significance of temporal validity and the importance of using up-to-date datasets when conducting empirical studies in Software Engineering. We shed light on the role of temporal validity to draw generalizable results from open-source software samples, and emphasize the importance of capturing the dynamics of project evolution in current software development. Additionally, we provide insights into

the characteristics of open-source projects in terms of their update activity. As future work, we aim to develop a dataset maintenance process that can help researchers keep their datasets up-to-date and mitigate temporal validity loss.

References

[1] Y. Hassouneh, H. Turabieh, T. Thaher, I. Tumar, H. Chantar, and J. Too, "Boosted Whale Optimization Algorithm With Natural Selection Operators for Software Fault Prediction," *IEEE Access*, vol. 9, pp. 14239–14258, 2021, doi: 10.1109/ACCESS.2021.3052149.

[2] A. Alazba and H. Aljamaan, "Code smell detection using feature selection and stacking ensemble: An empirical investigation," *Inf. Softw. Technol.*, vol. 138, p. 106648, Oct. 2021, doi: 10.1016/j.infsof.2021.106648.

[3] C. Ni, X. Xia, D. Lo, X. Chen, and Q. Gu, "Revisiting Supervised and Unsupervised Methods for Effort-Aware Cross-Project Defect Prediction," *IEEE Trans. Softw. Eng.*, vol. 48, no. 3, pp. 786–802, Mar. 2022, doi: 10.1109/TSE.2020.3001739.

[4] S. Baltes and P. Ralph, "Sampling in software engineering research: a critical review and guidelines," *Empir. Softw. Eng.*, vol. 27, no. 4, p. 94, Jul. 2022, doi: 10.1007/s10664-021-10072-8.

[5] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, and D. Damian, "The promises and perils of mining GitHub," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, 2014, pp. 92–101, doi: 10.1145/2597073.2597074.

[6] N. Munaiah, S. Kroh, C. Cabrey, and M. Nagappan, "Curating GitHub for engineered software projects," *Empir. Softw. Eng.*, vol. 22, no. 6, pp. 3219–3253, Dec. 2017, doi: 10.1007/s10664-017-9512-6.

[7] T. Xia, W. Fu, R. Shu, R. Agrawal, and T. Menzies, "Predicting health indicators for open source projects (using hyperparameter optimization)," *Empir. Softw. Eng.*, vol. 27, no. 6, p. 122, Nov. 2022, doi: 10.1007/s10664-022-10171-0.

[8] K. Munger, "The Limited Value of Non-Replicable Field Experiments in Contexts With Low Temporal Validity," *Soc. Media + Soc.*, vol. 5, no. 3, p. 205630511985929, Apr. 2019, doi: 10.1177/2056305119859294.

[9] T. Lewowski and L. Madeyski, "Creating Evolving Project Data Sets in Software Engineering," in *Studies in Computational Intelligence*, vol. 851, Springer Verlag, 2020, pp. 1–14.

[10] J. A. Carruthers, J. A. Diaz-Pace, and E. A. Irrazabal, "How are software datasets constructed in Empirical Software Engineering studies? A systematic mapping study," in *2022 48th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, 2022, pp. 442–450, doi: 10.1109/SEAA56994.2022.00075.

[11] M. Jureczko and L. Madeyski, "Towards identifying software project clusters with regard to defect prediction," in *Proceedings of the 6th International Conference on Predictive Models in Software Engineering - PROMISE '10*, 2010, p. 1, doi: 10.1145/1868328.1868342.

[12] M. Shepperd, Q. Song, Z. Sun, and C. Mair, "Data quality: Some comments on the NASA software defect datasets," *IEEE Trans. Softw. Eng.*, vol. 39, no. 9, pp. 1208–1215, 2013, doi: 10.1109/TSE.2013.11.

[13] P. Afric, L. Sikic, A. S. Kurdija, and M. Silic, "REPD: Source code defect prediction as anomaly detection," *J. Syst. Softw.*, vol. 168, p. 110641, Oct. 2020, doi: 10.1016/J.JSS.2020.110641.

[14] I. H. Laradji, M. Alshayeb, and L. Ghouti, "Software defect prediction using ensemble learning on selected features," *Inf. Softw. Technol.*, vol. 58, pp. 388–402, Feb. 2015, doi: 10.1016/J.INFSOF.2014.07.005.

[15] A. Boucher and M. Badri, "Software metrics thresholds calculation techniques to predict fault-proneness: An empirical comparison," *Inf. Softw. Technol.*, vol. 96, pp. 38–67, Apr. 2018, doi: 10.1016/J.INFSOF.2017.11.005.

[16] E. Tempero *et al.*, "The Qualitas Corpus: A curated collection of Java code for empirical studies," in *Proceedings - Asia-Pacific Software Engineering Conference, APSEC*, 2010, pp. 336–345, doi: 10.1109/APSEC.2010.46.

[17] M. M. Lehman, "Programs, life cycles, and laws of software evolution," *Proc. IEEE*, vol. 68, no. 9, pp. 1060–1076, 1980, doi: 10.1109/PROC.1980.11805.

[18] A. Ait, J. L. C. Izquierdo, and J. Cabot, "An empirical study on the survival rate of GitHub projects," in *Proceedings of the 19th International Conference on Mining Software Repositories*, 2022, pp. 365–375, doi: 10.1145/3524842.3527941.

[19] J. Coelho, M. T. Valente, L. Milen, and L. L. Silva, "Is this GitHub project maintained? Measuring the level of maintenance activity of open-source projects," *Inf. Softw. Technol.*, vol. 122, p. 106274, Jun. 2020, doi: 10.1016/j.infsof.2020.106274.

[20] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering*, vol. 9783642290. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012.

[21] M. D'Ambros, M. Lanza, and R. Robbes, "An extensive comparison of bug prediction approaches," in *Proceedings - International Conference on Software Engineering*, 2010, pp. 31–41, doi: 10.1109/MSR.2010.5463279.

[22] R. Wu, H. Zhang, S. Kim, and S. C. Cheung, "ReLink: Recovering links between bugs and changes," in *SIGSOFT/FSE 2011 - Proceedings of the 19th ACM SIGSOFT Symposium on Foundations of Software Engineering*, 2011, pp. 15–25, doi: 10.1145/2025113.2025120.

[23] V. Lenarduzzi, F. Pecorelli, N. Saarimaki, S. Lujan, and F. Palomba, "A critical comparison on six static analysis tools: Detection, agreement, and precision," *J. Syst. Softw.*, vol. 198, p. 111575, Apr. 2023, doi: 10.1016/j.jss.2022.111575.

[24] B. L. Sousa, M. A. S. Bigonha, K. A. M. Ferreira, and G. C. Franco, "A time series-based dataset of open-source software evolution," in *Proceedings of the 19th International Conference on Mining Software Repositories*, 2022, pp. 702–706, doi: 10.1145/3524842.3528492.

[25] J. Whitehead, I. Mistrík, J. Grundy, and A. van der Hoek, "Collaborative Software Engineering: Concepts and Techniques," in *Collaborative Software Engineering*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 1–30.

[26] K. Crowston, Q. Li, K. Wei, U. Y. Eseryel, and J. Howison, "Self-organization of teams for free/libre open source software development," *Inf. Softw. Technol.*, vol. 49, no. 6, pp. 564–575, Jun. 2007, doi: 10.1016/j.infsof.2007.02.004.

[27] B. Gezici, A. Tarhan, and O. Chouseinoglou, "Internal and external quality in the evolution of mobile software: An exploratory study in open-source market," *Inf. Softw. Technol.*, vol. 112, pp. 178–200, Aug. 2019, doi: 10.1016/j.infsof.2019.04.002.

[28] Y. Yu, H. Wang, G. Yin, and T. Wang, "Reviewer recommendation for pull-requests in GitHub: What can we learn from code review and bug assignment?," *Inf. Softw. Technol.*, vol. 74, pp. 204–218, Jun. 2016, doi: 10.1016/j.infsof.2016.01.004.

[29] G. Gousios, M. Pinzger, and A. van Deursen, "An exploratory study of the pull-based software development model," in *Proceedings of the 36th International Conference on Software Engineering*, 2014, pp. 345–355, doi: 10.1145/2568225.2568260.

[30] S. G. Eick, T. L. Graves, A. F. Karr, J. S. Marron, and A. Mockus, "Does code decay? Assessing the evidence from change management data," *IEEE Trans. Softw. Eng.*, vol. 27, no. 1, pp. 1–12, 2001, doi: 10.1109/32.895984.

[31] C. Laaber, M. Basmaci, and P. Salza, "Predicting unstable software benchmarks using static source code features," *Empir. Softw. Eng.*, vol. 26, no. 6, p. 114, Nov. 2021, doi: 10.1007/s10664-021-09996-y.

[32] D. J. Kim, T.-H. Chen, and J. Yang, "The secret life of test smells - an empirical study on test smell evolution and maintenance," *Empir. Softw. Eng.*, vol. 26, no. 5, p. 100, Sep. 2021, doi: 10.1007/s10664-021-09969-1.

[33] C. Macho, S. Beyer, S. McIntosh, and M. Pinzger, "The nature of build changes," *Empir. Softw. Eng.*, vol. 26, no. 3, pp. 1–53, May 2021, doi: 10.1007/S10664-020-09926-4/FIGURES/13.

[34] L. P. Lima, L. S. Rocha, C. I. M. Bezerra, and M. Paixao, "Assessing exception handling testing practices in open-source libraries," *Empir. Softw. Eng.*, vol. 26, no. 5, pp. 1–39, Jun. 2021, doi: 10.1007/S10664-021-09983-3.

[35] Z. A. Kermansaravi, M. S. Rahman, F. Khomh, F. Jaafar, and Y. G. Guéhéneuc, "Investigating design anti-pattern and design pattern mutations and their change- and fault-proneness," *Empir. Softw. Eng.*, vol. 26, no. 1, pp. 1–47, Jan. 2021, doi: 10.1007/S10664-020-09900-0.

[36] G. A. A. Prana *et al.*, "Out of sight, out of mind? How vulnerable dependencies affect open-source projects," *Empir. Softw. Eng.*, vol. 26, no. 4, pp. 1–34, Apr. 2021, doi: 10.1007/S10664-021-09959-3.

[37] E. A. AlOmar, M. W. Mkaouer, A. Ouni, and M. Kessentini, "On the Impact of Refactoring on the Relationship between Quality Attributes and Design Metrics," in *International Symposium on Empirical Software Engineering and Measurement*, 2019.

[38] L. Grammel, H. Schackmann, A. Schröter, C. Treude, and M.-A. Storey, "Attracting the community's many eyes," in *Human Aspects of Software Engineering*, 2010, pp. 1–6, doi: 10.1145/1938595.1938601.

[39] N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, and T. Zimmermann, "What makes a good bug report?," in *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, 2008, pp. 308–318, doi: 10.1145/1453101.1453146.

[40] O. Jarczyk, B. Gruszka, S. Jaroszewicz, L. Bukowski, and A. Wierzbicki, "GitHub Projects. Quality Analysis of Open-Source Software," 2014, pp. 80–94.

[41] H. Borges and M. Tulio Valente, “What’s in a GitHub Star? Understanding Repository Starring Practices in a Social Coding Platform,” *J. Syst. Softw.*, vol. 146, pp. 112–129, Dec. 2018, doi: 10.1016/j.jss.2018.09.016.
[42] J. Coelho and M. T. Valente, “Why Modern Open Source Projects Fail,” *Proc. 2017 11th Jt. Meet. Found. Softw. Eng.*, vol. Part F1301, pp. 186–196, Jul. 2017, doi: 10.1145/3106237.3106246.
[43] I. Scholtes, P. Mavrodiev, and F. Schweitzer, “From Aristotle to Ringelmann: a large-scale analysis of team productivity and coordination in Open Source Software projects,” *Empir. Softw. Eng.*, vol. 21, no. 2, pp. 642–683, Apr. 2016, doi: 10.1007/s10664-015-9406-4.
[44] J. D. Singer and J. B. Willett, *Applied Longitudinal Data Analysis: Modeling Change and Event Occurrence*. Oxford University Press, 2003.
[45] K. Crowston, K. Wei, J. Howison, and A. Wiggins, “Free/Libre open-source software development,” *ACM Comput. Surv.*, vol. 44, no. 2, pp. 1–35, Feb. 2012, doi: 10.1145/2089125.2089127.
[46] V. Cosentino, J. Luis, and J. Cabot, “Findings from GitHub,” in *Proceedings of the 13th International Conference on Mining Software Repositories*, 2016, pp. 137–141, doi: 10.1145/2901739.2901776.
[47] V. Cosentino, J. L. Canovas Izquierdo, and J. Cabot, “A Systematic Mapping Study of Software Development With GitHub,” *IEEE Access*, vol. 5, pp. 7173–7192, May 2017, doi: 10.1109/ACCESS.2017.2682323.
[48] G. Bavota, G. Canfora, M. Di Penta, R. Oliveto, and S. Panichella, “The Evolution of Project Inter-dependencies in a Software Ecosystem: The Case of Apache,” in *2013 IEEE International Conference on Software Maintenance*, 2013, pp. 280–289, doi: 10.1109/ICSM.2013.39.
[49] G. Bavota, G. Canfora, M. Di Penta, R. Oliveto, and S. Panichella, “How the Apache community upgrades dependencies: an evolutionary study,” *Empir. Softw. Eng.*, vol. 20, no. 5, pp. 1275–1317, Oct. 2015, doi: 10.1007/s10664-014-9325-9.
[50] R. Kikas, G. Gousios, M. Dumas, and D. Pfahl, “Structure and Evolution of Package Dependency Networks,” in *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, 2017, pp. 102–112, doi: 10.1109/MSR.2017.55.
[51] M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts, *Refactoring: Improving the Design of Existing Code*. 2002.
[52] M. Tufano *et al.*, “When and Why Your Code Starts to Smell Bad (and Whether the Smells Go Away),” *IEEE Trans. Softw. Eng.*, vol. 43, no. 11, pp. 1063–1088, Nov. 2017, doi: 10.1109/TSE.2017.2653105.
[53] A.-J. Molnar and S. Motogna, “Long-Term Evaluation of Technical Debt in Open-Source Software,” in *Proceedings of the 14th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2020, pp. 1–9, doi: 10.1145/3382494.3410673.
[54] A. Trautsch, S. Herbold, and J. Grabowski, “A longitudinal study of static analysis warning evolution and the effects of PMD on software quality in Apache open source projects,” *Empir. Softw. Eng.*, vol. 25, no. 6, pp. 5137–5192, Nov. 2020, doi: 10.1007/s10664-020-09880-1.
[55] I. Chengalur-Smith, A. Sidorova, and S. Daniel, “Sustainability of Free/Libre Open Source Projects: A Longitudinal Study,” *J. Assoc. Inf. Syst.*, vol. 11, no. 11, pp. 657–683, Nov. 2010, doi: 10.17705/1jais.00244.
[56] J. A. Carruthers, J. A. Diaz-Pace, and E. A. Irrazabal, “A Systematic Mapping Study of Empirical Studies Performed with Collections of Software Projects,” *Comput. y Sist.*, vol. 26, no. 4, Dec. 2022, doi: 10.13053/cys-26-4-3981.
[57] O. J. Dunn, “Multiple Comparisons Among Means,” *J. Am. Stat. Assoc.*, vol. 56, no. 293, p. 52, Mar. 1961, doi: 10.2307/2282330.
[58] S. S. Shapiro and M. B. Wilk, “An Analysis of Variance Test for Normality (Complete Samples),” *Biometrika*, vol. 52, no. 3/4, p. 591, Dec. 1965, doi: 10.2307/2333709.
[59] W. H. Kruskal and W. A. Wallis, “Use of Ranks in One-Criterion Variance Analysis,” *J. Am. Stat. Assoc.*, vol. 47, no. 260, pp. 583–621, Dec. 1952, doi: 10.1080/01621459.1952.10483441.
[60] O. J. Dunn, “Multiple Comparisons Using Rank Sums,” *Technometrics*, vol. 6, no. 3, pp. 241–252, Aug. 1964, doi: 10.1080/00401706.1964.10490181.
[61] A. Vargha, H. D. Delaney, and A. Vargha, “A Critique and Improvement of the ‘CL’ Common Language Effect Size Statistics of McGraw and Wong,” *J. Educ. Behav. Stat.*, vol. 25, no. 2, p. 101, 2000, doi: 10.2307/1165329.
[62] M. R. Hess and J. D. Kromrey, “Robust confidence intervals for effect sizes: A comparative study of cohen’s d and cliff’s delta under non-normality and heterogeneous variances,” in *Annual Meeting of the American Educational Research Association*, 2004.
[63] D. R. Cox and D. Oakes, *Analysis of Survival Data*. Chapman and Hall/CRC, 2018.
[64] R. Coelho, L. Almeida, G. Gousios, A. van Deursen, and C. Treude, “Exception handling bug hazards in Android: Results from a mining study and an exploratory survey,” *Empir. Softw. Eng.*, vol. 22, no. 3, pp.

1264–1304, Jun. 2017, doi: 10.1007/s10664-016-9443-7.

[65] E. Iannone, R. Guadagni, F. Ferrucci, A. De Lucia, and F. Palomba, "The Secret Life of Software Vulnerabilities: A Large-Scale Empirical Study," *IEEE Trans. Softw. Eng.*, vol. 49, no. 1, pp. 44–63, Jan. 2023, doi: 10.1109/TSE.2022.3140868.

[66] E. L. Kaplan and P. Meier, "Nonparametric Estimation from Incomplete Observations," *J. Am. Stat. Assoc.*, vol. 53, no. 282, p. 457, Jun. 1958, doi: 10.2307/2281868.

[67] G. Gousios, M.-A. Storey, and A. Bacchelli, "Work practices and challenges in pull-based development," in *Proceedings of the 38th International Conference on Software Engineering*, 2016, pp. 285–296, doi: 10.1145/2884781.2884826.

[68] M. M. Lehman, "On understanding laws, evolution, and conservation in the large-program life cycle," *J. Syst. Softw.*, vol. 1, pp. 213–221, Jan. 1979, doi: 10.1016/0164-1212(79)90022-0.

[69] M. Caneill, D. M. Germán, and S. Zacchiroli, "The Debsources Dataset: two decades of free and open source software," *Empir. Softw. Eng.*, vol. 22, no. 3, pp. 1405–1437, Jun. 2017, doi: 10.1007/s10664-016-9461-5.

[70] L. Hatton, D. Spinellis, and M. van Genuchten, "The long-term growth rate of evolving software: Empirical results and implications," *J. Softw. Evol. Process*, vol. 29, no. 5, p. e1847, May 2017, doi: 10.1002/smr.1847.

[71] G. Rousseau, R. Di Cosmo, and S. Zacchiroli, "Software provenance tracking at the scale of public source code," *Empir. Softw. Eng.*, vol. 25, no. 4, pp. 2930–2959, Jul. 2020, doi: 10.1007/s10664-020-09828-5.

This piece of the submission is being sent via mail.

Declaration of interests

☒The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

☐The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: