Characterizing Commits in Open-Source Software

Mívian M. Ferreira
Federal University of Minas Gerais - Brazil
mivian.ferreira@dcc.ufmg.br

Diego Santos Gonçalves
Federal Center for Technological
Education of Minas Gerais - Brazil
disantosg18@gmail.com

Mariza A. S. Bigonha
Federal University of Minas Gerais - Brazil
mariza@dcc.ufmg.br

Kecia A. M. Ferreira
Federal Center for Technological
Education of Minas Gerais - Brazil
kecia@cefetmg.br

ABSTRACT

Mining software repositories has been the basis of many studies on software engineering. Many of these works rely on commits' data extracted since commit is the basic unit of information about activities performed on the projects. However, not knowing the characteristics of commits may introduce biases and threats in studies that consider commits' data. This work presents an empirical study to characterize commits in terms of four aspects: the size of commits in the total number of files; the size of commits in the number of source-code files, the size of commits by category; and the time interval of commits performed by contributors. We analyzed 1M commits from the 24 most popular and active Java-based projects hosted on GitHub. The main findings of this work show that: the size of commits follows a heavy-tailed distribution; most commits involve one to 10 files; most commits affect one to four source-code files; the commits involving hundreds of files not only refer to merge or management activities; the distribution of the time intervals is approximately a Normal distribution, i.e., the distribution tends to be symmetric, and the mean is representative; in the average, a developer proceed a commit every eight hours. The results of this study should be considered by researchers in empirical works to avoid biases when analyzing commits' data. Besides, the results provide information that practitioners may apply to improve the management and the planning of software activities.

CCS CONCEPTS

• **Software and its engineering → Software version control**.

KEYWORDS

empirical study, commit, open-source, mining software repositories, Java

ACM Reference Format:
Mívian M. Ferreira, Diego Santos Gonçalves, Mariza A. S. Bigonha, and Kecia A. M. Ferreira. 2022. Characterizing Commits in Open-Source Software. In *XXI Brazilian Symposium on Software Quality (SBQS '22), November 7–10, 2022, Curitiba, Brazil.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3571473.3571508

SBQS '22, November 7–10, 2022, Curitiba, Brazil

ACM ISBN 978-1-4503-9999-9/22/11.
https://doi.org/10.1145/3571473.3571508

1 INTRODUCTION

Several works have mined data from GitHub to investigate many subjects, such as code authorship, failure prediction, software evolution, and change impact analysis. Many of these works are based on quantitative and qualitative commits' data extracted from software systems repositories since commit is the basic unit of information about activities performed on the projects [18]. An important example is the case of the studies on co-change and changes impact analysis, which is based on commits' data. Several studies consider files registered in the same commit as a unit of co-change, i.e., they assume that if a set of files changed in the same commit, they are related [9, 13, 16, 19]. However, such an assumption may introduce biases in the studies if it does not consider the so-called *tangled changes* problem, i.e., non-related changes registered in the same commit transaction [7]. In this study, it is important to know, for instance, the types of activities each commit involves and the number of commits involving more than one type of activity.

Besides, knowing the commits' characteristics may reveal patterns of activities performed in the repositories and then may aid practitioners in planning tasks on software maintenance. Given this scenario, developing research related to the structure of commits is essential to increase the accuracy of studies that use GitHub as a data source. Some characteristics have been investigated regarding commit practices in GitHub, such as the number of files per commit, frequency of commits, and others [2–4, 11, 15, 17].

In our previous work, we characterized commits in GitHub repositories regarding categories of activities performed in the commits and co-occurrences of activities in commits [8]. In this paper, we aim to characterize commits according to four aspects: the size of commits in the total number of files, the size of commits in the number of source-code files; the size of commits by category; and the time interval of commits performed by contributors.

We concentrate our analysis on a single programming language to avoid biases since the maintenance activities might differ according to the language. We chose to focus on Java because many empirical studies usually consider Java in their analysis. We analyzed 1M commits from 24 most popular and active Java-based projects hosted on GitHub.

Specifically, this work aims to answer the following research questions:

RQ1. What is the size of commits in software system repositories? This question investigates the developers' behavior regarding the number of files they use to commit together. Such a result may guide establishing the granularity of commits when carrying out research using commit data. For instance, in research investigating co-changes, the files changed together in a commit may be considered a co-change instance. However, the number of commits involving many files may bias the research results.

RQ2. What is the size of commits involving only .java files in software system repositories? In RQ3, we consider all types of files in the commits. In RQ4, we aim to analyze only source-code files. As we focus on Java-based software systems, we considered Java source-code files to answer this research question.

RQ3. What is the size of commits according to their aims? We analyze the number of files that usually are involved in the different activities, such as reengineering, managing, Corrective maintenance, and Forward maintenance. Answering this RQ will bring insights into the proportion of each type of activity performed along the software systems' life cycle.

RQ4. What is the time interval a developer registers a commit in a repository? We analyze the interval of time the contributors usually perform commits in a repository. The results of this analysis may aid studies that define heuristics to co-change and change impact analysis.

We organized this paper as follows. Section 2 presents the study design and the creation of the dataset used in this work. Section 3 brings the results of the study and Section Section 4 discusses them. Section 5 describes the related work. Section 6 presents the threats to validity and Section 7 brings the conclusion.

2 STUDY DESIGN

This section presents the method we applied to construct the dataset analyzed in this work and the data extraction and commits categorization process. The design of this study is basically the same we applied in our previous work [8].

2.1 Dataset

As this work aims to characterize commits, we selected well-known open-source software systems with a high number of commits. To identify the corpus of systems to consider in this work, first, we identified the 900 highest-rated Java repositories. We found many repositories that do not contain source code among these projects. Those repositories were mostly used as libraries - it contains books, "how to", and similar files. Thus, we removed those repositories and obtained 846 repositories of Java software. From these 846 repositories, we selected 24 open-source systems with the highest number of commits to be the subject of this study. We restricted the number of systems to 24 due to the long time it takes to collect the data we analyzed in this study.

We developed a Python script using GraphQL API to mine GitHub to retrieve the projects. The data returned by this API contain the repositories' name, owner, age in years, URL, commits, forks, issues, and the number of stars. Java language was used as the primary selection criteria to define the projects considered in the analysis. We chose Java because studies on software engineering commonly consider this language.

Table 1 shows the name, age, number of commits, and number of stars of the systems analyzed in this study. The dataset comprises mature and well-known systems aged between 3 and 11 years and rated between 52K and 2,6K stars. All the systems have high commits, varying from 22.9K to 92K. Besides, the dataset is diverse in terms of application domains.

2.2 Data Extraction

The first step of the data extraction was to create a copy of all the 24 systems' repositories using the `git clone` command.[1] We developed a Python script using *GitPython* API to perform this cloning process. The script collects all the commits' information for each repository: author, date, description message, and the modified files, and export all the data to a .csv file.[2]

In our analysis, we considered data of the first-parent line. We support our decision by the findings of Kovalenko et al.'s study [14]. The results of their study show that considering complete file histories, i.e., including branches, may modestly increase the performance of reviewer recommendation, change recommendation, and defect prediction techniques. On the other hand, collecting the entire file history demands extra effort, e.g., the time to collect the data may be exorbitant. Therefore, the increase in performance may not justify such an effort.

2.3 Commits Categories

This work analyzes the main activities registered in the system's commits to answer the request question RQ3. For this purpose, we classified each commit into six categories:

- Merge: specific GitHub activities of merge and pull requests.
- Corrective Engineering: changes performed in the code to correct bugs, errors, or defects.
- Forward Engineering: inclusion of new features or requirements.
- Reengineering: changes performed in the code to enhance its quality.
- Management: activities not related to codification, such as documentation.
- Other: when the commit does not match any of the five categories.

We used the same set of categories proposed by Hattori and Lanza [11] and included a new one: Merge. In Hattori and Lanza's work, "merge" was a keyword of the Management category. We considered Merge a particular category because a merge is a specific activity that differs from the other management activities in GitHub. Unlike Hattori and Lanza's approach, we do not use a hierarchy to set only one category for a commit, i.e., in our approach, a commit may be classified in more than one category. We did that to cover the cases in which a developer proceeds a commit corresponding to more than one activity type, e.g., Corrective Engineering and Reengineering. This type of commit is called *tangled commit* [6].

[1]We cloned all repositories in January 2021.
[2]The data was exported as a .csv file and is available at *https://figshare.com/s/fab86b2522ded083f81c*

System	Age	#Commits	#Stars
ballerina-lang/ballerina-platform	3	96,121	2,644
neo4j/neo4j	8	69,702	8,315
jdk/openjdk	2	62,947	6,553
elasticsearch/elastic	10	57,414	52,228
camel/apache	11	50,138	3,489
graal/oracle	4	53,665	13,950
languagetool/languagetool-org	7	46,224	4,114
vespa/vespa-engine	4	46,403	3,363
lucene-solr/apache	4	34,703	3,863
rstudio/rstudio	9	34,292	3,423
alluxio/Alluxio	7	31,587	4,805
hazelcast/hazelcast	8	30,936	4,033
jenkins/jenkinsci	9	31,136	16,463
sonarqube/SonarSource	9	30,480	5,272
beam/apache	4	30,519	4,362
spring-boot/spring-projects	8	30,671	51,678
bazel/bazelbuild	6	28,662	15,673
shardingsphere/apache	4	28,457	12,387
ignite/apache	5	27,401	3,518
selenium/SeleniumHQ	7	26,432	19,074
cassandra/apache	11	25,994	6,278
flink/apache	6	25,543	14,626
hadoop/apache	6	24,584	11,041
tomcat/apache	9	22,909	4,984

Table 1: Dataset systems sorted by number of commits.

Similar to other works [3, 11], our approach to categorizing a commit is based on the analysis of keywords extracted from the commit's messages. We chose to analyze the messages because it presents a complete description of the commits' activities. We developed a Python script to identify the commits' activity categories using the *flashtext* API. Given the vast number of commits ($\approx 1M$), we used this API because its performance is better than the search using regex. The *flashtext* API counts an instance of a word only if there is an exact match in the text with the word. Therefore, it was necessary to build a dictionary containing the keywords corresponding to the commits' categories we considered and their variations, e.g., add, addition, adding, added, and adds. We started the construction of the dictionary having as basis the keywords used by Hattori and Lanza [11]. Then, we ran the classification and manually inspected the results considering a set of randomly selected $\approx$ 500 commit messages. We included new keywords extracted from the commits' messages in the dictionary based on the manual inspection. We executed such a process iteratively until we found a correct classification of the set of commits selected for manual inspection. Table 2 exhibits the final primary keywords set. It is worthwhile to note that the complete dictionary contains variations of these words.

To assess the approach used to classify the commits' activities, we calculated the precision and the recall considering a random sample containing 500 commits. In this evaluation, we manually analyzed each commit and tagged each categorization result as:

- True positive (TP): when the script indicates that a commit belongs to a category and this categorization is correct;
- False Positive (FP): when the script shows that a commit belongs to a category and this categorization is wrong; and
- True Negative (TN): when the script indicates that a commit does not belong to a category and this result is right.

Category	Keywords
Merge	merge, pull request
Corrective	bug, fix, correct, miss, proper, broken, corrupted, failure, fault, deprecate, throw/catch exception, crash, typo
Forward	implement, add, request, new, test, increase, expansion, include, initial, create, introduce, launch, define, determine, support, extend, set
Reengineering	parallelize, optimization, adjust, update, delete, remove, expunge, cut off, refactor, replace, modification, improve, is/are now, change, rename, eliminate, duplicate, obsolete, enhance, restructure, alter, rearrange, withdraw, conversion, revision, simplify, move, relocate, downgrade, exclude, reuse, revert, extract, reset, redefine, edit, readd, revamp, decouple
Management	clear, license, release, structure, integration, copyright, documentation, manual, Javadoc, migrate, review, polish, upgrade, style, standardization, TODO, migration, organization, normalize, configure, ensure, resolve conflict, bump, dump, comment, format code, do not use

Table 2: Primary keywords used to identify the activity category of commits.

Precision is given by TP/(TP+FP) and indicates how many positive classifications are correct. As shown in Table 3, all categories showed precision above 52%. Merge is the category with the highest precision (96%). A Recall is given by TP/(TP+FN) and indicates how many situations the script should detect as true positives were correctly detected. The results show that the categorization's recall reaches 99%.

	Precision	Recall
Merge	0,96	0,99
Bug	0,78	0,98
Reengeneering	0,84	0,79
Foward	0,59	0,93
Management	0,52	0,77
Others	0,74	0,70

Table 3: Categorization Results: Precision and Recall

3 RESULTS

This section presents the results of our study by answering the research questions.

RQ1. What is the size of commits in software system repositories?

The first step in answering this research question was to analyze the data distribution. Therefore, we calculated the number of files changed by each commit. Figure 1 shows the results, where a boxplot represents the distributions of the number of files per commit for each system. We marked the distributions' median as red dots in the boxplots.

The result presents some standard behaviors. All systems, observing the boxplots' shapes, show a long tail distribution because

a high concentration of commits involves few files. The boxes are placed at the chart bottom. The extensive lines ranging from the third quartile to the outliers indicate that commits registering a higher number of changed files are atypical events, i.e., there are few commits with this behavior. The median ranges from 1 to 3, with two being the median in 58% of the systems. The boxes' height (i.g, the difference between the first and the third quartiles) is not high and is very similar. They range from 1 to 3 in 41.67% of the systems. From 4 to 6 in 41.67% of the systems, and from 7 to 9 in 16.67% as well. Table 4 presents the first quartile, the median, and third quartile values.

We may take *jdk* as an example of the enormous data disparity in the number of files per commit. In *jdk*, the 80th percentile is 14, i.e., 80% of the commits register the modification of a maximum of 14 files. In contrast, Table 4 shows a commit in this system that modified 56K files. We detailed the analysis of the distributions' tails to verify whether there is a pattern of developers committing a high number of files in a single transaction. We consider outliers' values greater than the upper outer fence, i.e., values higher than $Q3 + 3 * IQR$, where Q3 is the third quartile, and IQR is given by *3rd quartile - 1st quartile*. Figure 2 shows the distribution's outliers. The outliers' distribution is also heavy-tailed. We can see that by observing the violin plots' shape: the most significant part of the plot is placed at the chart's bottom, and as the y-axes increase, the plots' shape becomes thinner. There is a big difference between the median values of the outliers. Unlike the distribution shown in Figure 1, the medians vary between 56 and 198, and the values contained in the interquartile are more dispersed.

Summary. In general, the total files per commit range between 1 and 10. Nevertheless, some commits modify a very high number of files. Among the outliers, the medians vary between 56 and 198.

RQ2. What is the size of commits involving only *.java* files in software system repositories?

To answer this research question, we carried out the same analysis of RQ1; however, observing only Java source-code files, i.e., files with the extension .java. Figure 3 exhibits the results. We observe that the number of .java files committed in a single transaction also has a long tail distribution. The medians range between 0 and 2. In 70.8% of the systems, the median is 1, 16.7% is 2, while 12.5% is 0. In 66.5% of the analyzed systems, the first quartile is 0. The third quartile has the main values of modified java files per commit: 4 files, 29.2% of the systems, and three files, 20.8% of systems.

We observed the same behavior found in the analysis of RQ1. The results show that *jdk* also presents the most considerable disparity between the number of .java files registered in a commit. The system's 80th percentile is 182, and the third quartile is 6.

Figure 4 shows the analysis of the distributions' tail of the number of .java files modified in a commit. Among the outliers, the median ranges from 25 to 104. In the same way as the previous distributions' plots, *jdk* system shows a particular behavior, with the highest median value, 104, and higher dispersion. Such characteristic is essential to be considered when performing studies about this system.

Systems	Q1	Median	Q3	80th %	Max Files
alluxio	1	1	4	5	2448
ballerina-lang	1	3	10	13	21405
bazel	1	2	5	6	2733
beam	1	2	4	6	7206
camel	1	2	4	5	17925
cassandra	1	2	4	5	645
elasticsearch	1	2	6	8	14916
flink	1	3	8	10	11013
graal	1	2	5	6	11103
hadoop	2	3	6	8	5194
hazelcast	1	2	6	8	8674
ignite	1	3	10	15	9971
jdk	1	2	9	14	56923
jenkins	1	1	3	5	8949
languagetool	1	1	2	2	1266
lucene-solr	1	2	5	6	5570
neo4j	1	2	7	9	10716
rstudio	1	2	4	5	4624
selenium	1	2	4	5	3619
shardingsphere	1	2	6	8	5259
sonarqube	1	3	7	9	9263
spring-boot	1	1	3	4	4616
tomcat	1	1	3	3	1157
vespa	1	2	6	8	18589

Table 4: Percentiles of number of files per commit, where, Q1 = 1st quartile, Q3 = 3rd quartile.

Summary. The number of .java files modified per commit follows a heavy-tailed distribution. The systems generally have between 1 and 4 .java files modified per commit. Among the outliers, the median ranges from 25 to 104.

RQ3. What is the size of commits according to their aims?

To answer this research question, we calculated the number of files modified by each commit category: Merge, Corrective Engineering, Forward Engineering, and Management.

Figure 5 shows the results of *alluxio*. The median values are low in the data distribution, ranging from 1 to 3 files. The other systems presented a similar result, except *jdk*. Due to space limitations, we do not show all graphics with the results of this research question in this paper. However, we make them available online.[3]

Figure 6 shows the results of *jdk*. Reengineering, Forward Engineering, Corrective Engineering, and Management have the same distribution pattern, and the median value is 2. The merge category presents a different result: it has the largest interquartile range:[4] 99

[3] https://figshare.com/s/ffd7b22c520abdc7129c

[4] Difference between the first and third quartiles. In *jdk*, there are, respectively, 1 and 100 files.

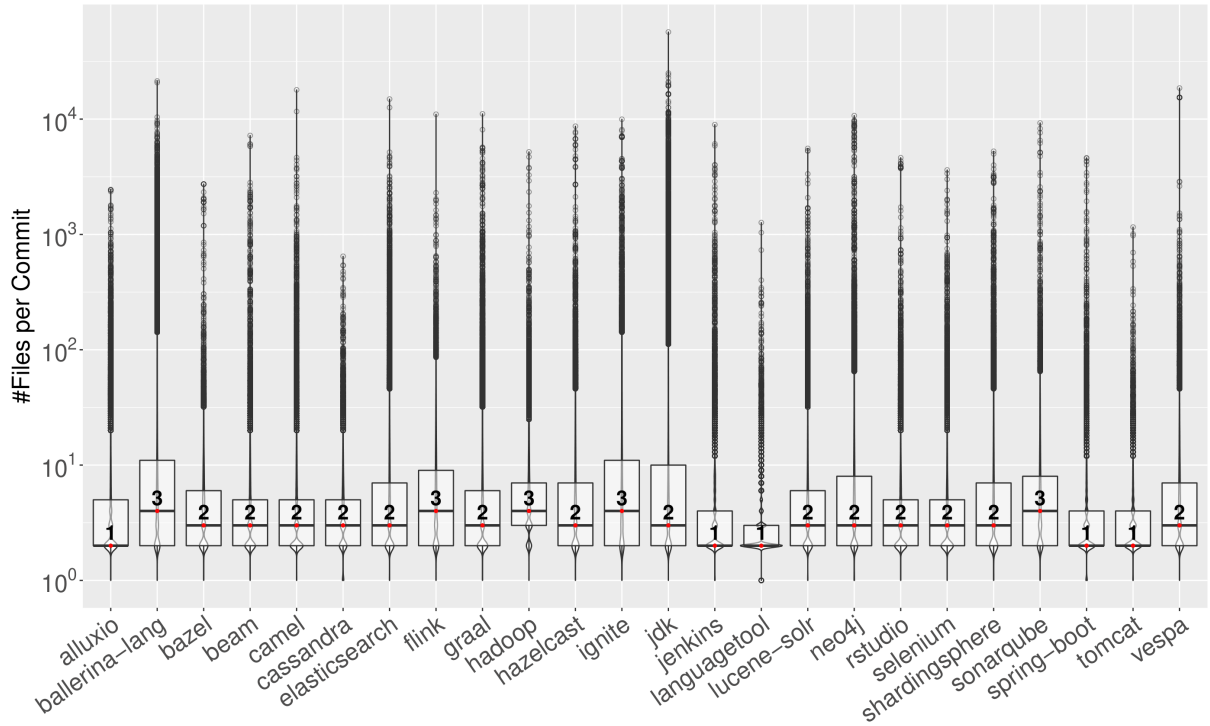

Figure 1: Distribution of files modified in commits.

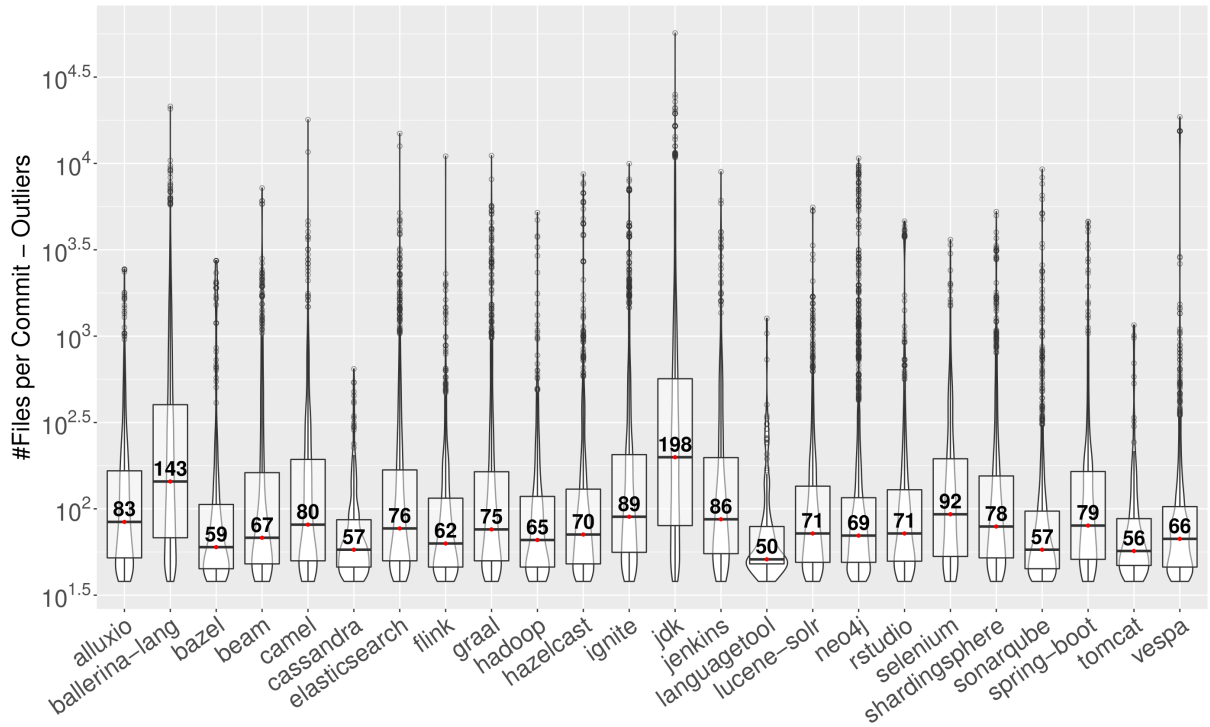

Figure 2: Distribution of files modified in commits - upper outer fence outliers.

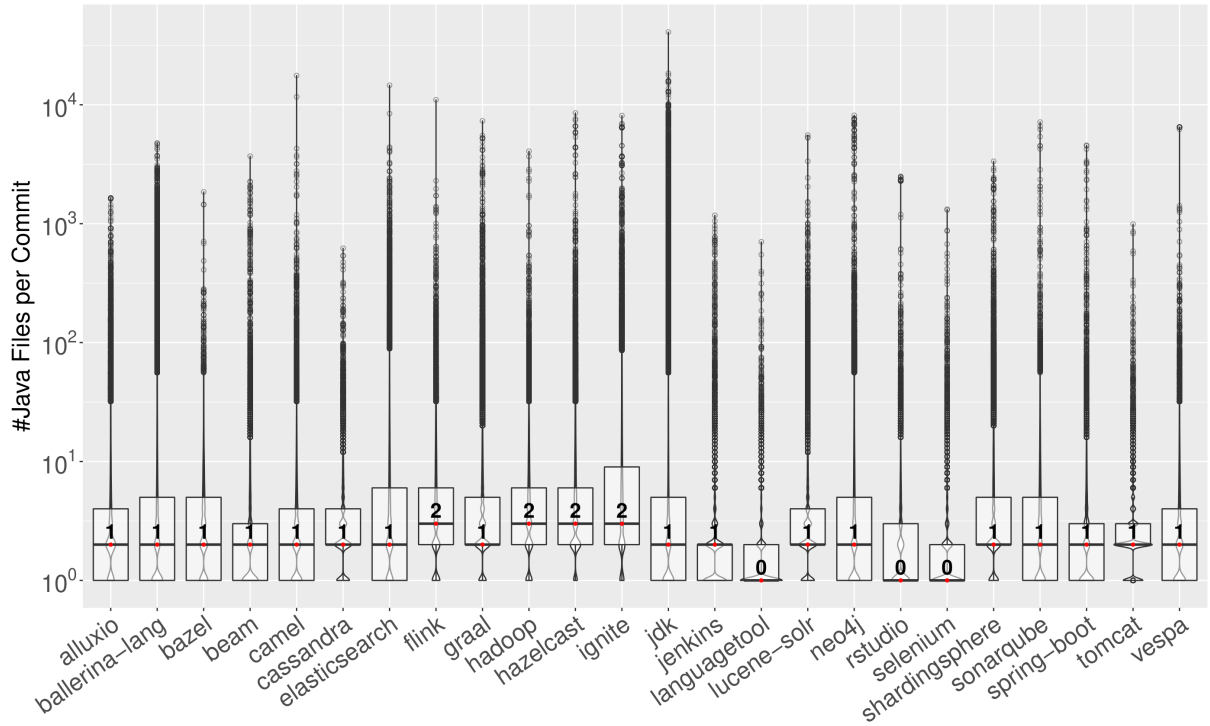

Figure 3: Distribution of Java files modified in commits.

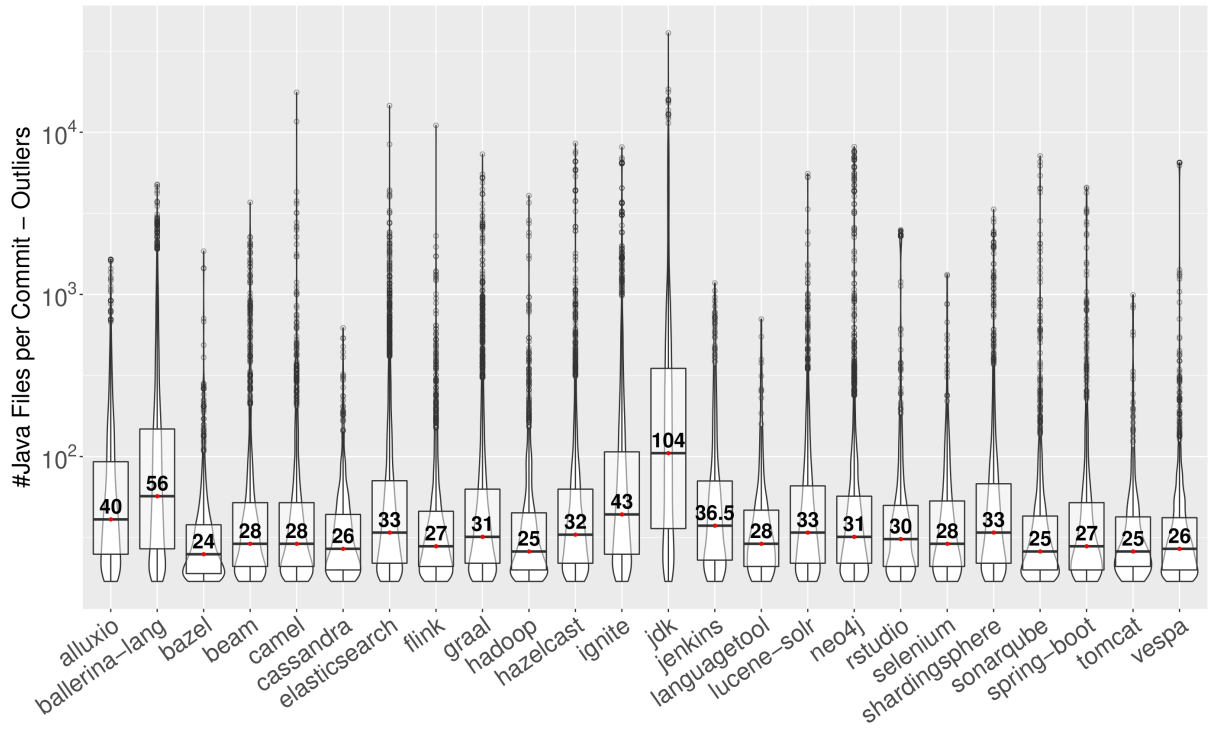

Figure 4: Distribution of Java files modified in commits - Upper outer fence outliers.

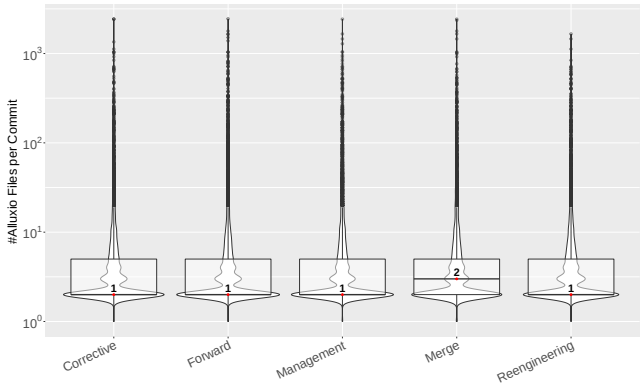

Figure 5: Alluxio distribution of files modified in commits grouped by category.

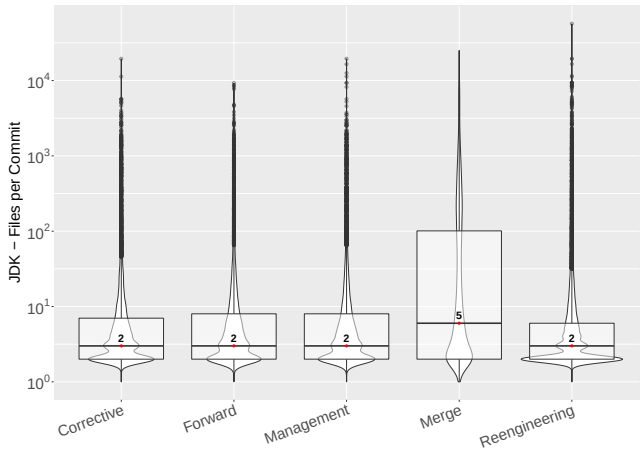

Figure 6: JDK distribution of files modified in commits grouped by category.

files. An important characteristic observed in *jdk* is that commits that did not change files were categorized exclusively as merge.

Summary. The number of files modified in a commit does not significantly differ regarding the activity type.

RQ4. What is the time interval a developer registers a commit in a repository?

With this research question, we aim to investigate if developers have a pattern of time to proceed with commits in the repositories. To identify the contributor, we considered the field "author" of the commit's data retrieved from GitHub. For each project and each contributor, we calculated the time intervals between the subsequent commits he/she performed in the repository. For example, if a contributor performed three subsequent commits at 12 a.m., 13 a.m, and 13:20 a.m, such an activity will result in two-time intervals, 60 and 20 minutes, respectively. For each project contributor, we computed the average time intervals he/she registered a commit in the repository. For the easiness of calculation, we considered seconds as a unit of time. We then analyzed the distribution of the mean time intervals of commits in each project, considering all its contributors. Figure 7 shows the box-plots of these distributions.

The analysis of these results indicates that the distributions are approximately Normal distributions. This result indicates that the mean value obtained in a given sample is representative, i.e., it may be used to infer the population means. In this context, the population corresponds to the time intervals computed for a given project, and a sample corresponds to a subset of these time intervals.

The results also indicate that the means vary among the projects. The lowest mean is of *shardingsphere* (986), whereas the highest one is of *jdk* (162,225). Computing the average of the mean values of the 24 projects, we find 27,903 seconds, which corresponds to 7.75 hours. This fact indicates that, on average, a developer proceeds a commit every eight hours.

Summary. The time intervals between the developers' commits follow approximately a Normal distribution. On average, a developer proceeds a commit every eight hours.

LESSONS LEARNED FROM OUR PREVIOUS WORK

In our previous work [8], we analyzed the characteristics of commits in the following aspects: categories of activities performed in the commits and co-occurrences of activities in the commits. As in the present paper, we analyzed the size of commits by category of activity. This section summarizes the main results of our previous work for contextualization purposes.

The commits *nature* should be considered by the studies. This study found that most commits register Reengineering activities, followed by Forward Engineering and Corrective Engineering. A possible explanation for this characteristic is that as open-source software projects are developed collectively, it may demand refactoring the system more often. Besides, as the systems are publicly available, their users may continuously report defects and failures in the systems. This result indicates the need to properly select the commits in studies on refactoring and faults in software since Reengineering commits correspond to only 32.97% and Corrective Engineering to 25% of the commits in the systems.

The percentage of Merge, Management, and Other activities should not be ignored: 18.7%, 16.49%, and 16%, respectively. If these activity types can impact the analysis in a study, they need to be identified when collecting the data.

The systems analyzed in this study are popular and very active, which may be a reason for the high number of Forward Engineering. Therefore, the sample analyzed in this work may be considered for future work concentrated on Forward Engineering.

The Quantification of the Tangled Changes Problem. We found that 30% of commits involve more than one activity type, indicating the extent of tangled changes in software repositories. Therefore, works threatened by tangled changes should perform characterization of commits in terms of activities because the amount of co-occurrence of activities is expressive. For example, this care is critical in studies on change impact analysis, and many studies on

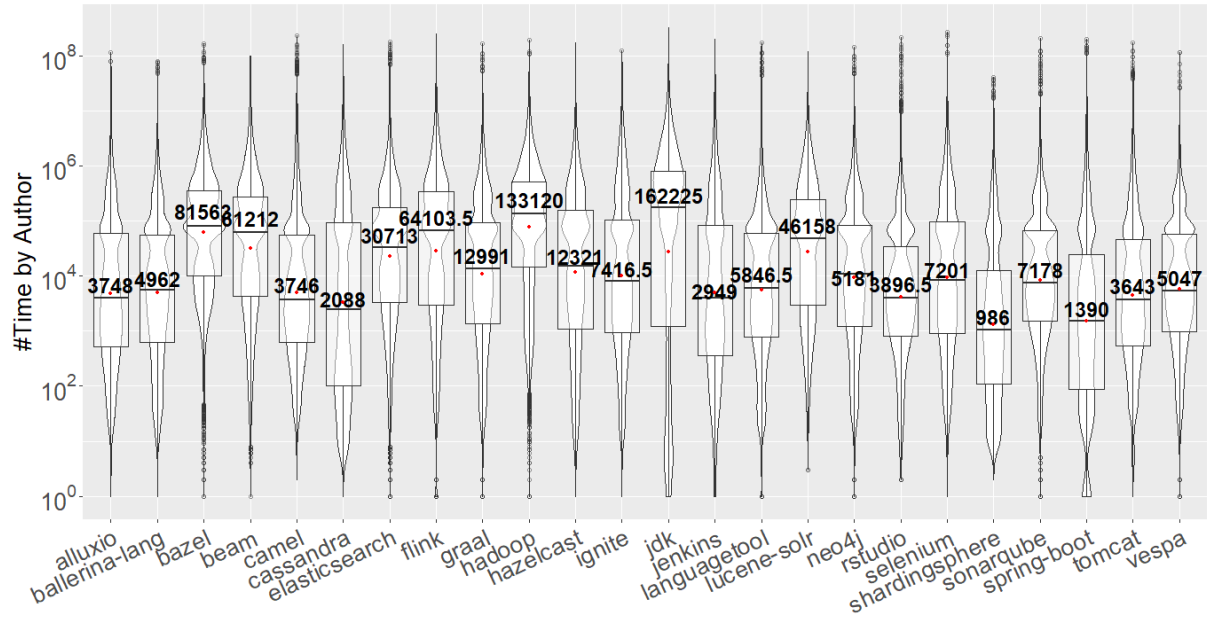

Figure 7: Time-interval of commits by developers - represented in seconds.

this subject consider a commit as a basic unit of correlated changes. In the face of the results found in this work, the analysis performed in those works may be biased.

4 DISCUSSION

Understanding the dataset's characteristics is critical for conducting a good experiment, and working with an inadequate dataset will lead any well-designed study to inaccurate results. This section discusses the main lessons learned from our study and their implications for studies that consider commits' data.

Reengineering has the highest co-occurrence with other activity types, but this does not happen too often. The incremental software development methodologies, such as the Agile methodologies, favor Reengineering, Corrective Engineering, and Forward Engineering to occur in parallel. For example, it is possible that correcting a bug or introducing a new feature in the system may cause a reengineering. Then both types of activities may be committed together. However, the results of this study show that these co-occurrences do not happen very often. The highest frequency of co-occurrences is between Reengineering and Corrective Engineering, and between Reengineering and Forward Engineering, 8% in both cases. Studies on refactoring that are based on commit analysis should verify if this amount of co-occurrence introduced bias in their results.

Intuitively, we may consider that when a system is well constructed, making changes to it will be more comfortable; therefore, it will demand fewer refactoring activities. Consequently, we raise two hypotheses that may explain the low percentage of co-occurrence between reengineering with corrective and forward engineering: (i) or fixing bugs and changing a piece of system usually demands few refactoring in the system, (ii) or the practice of developers is to commit the refactor of the system before fixing a bug or changing the system.

The size of the commit matters. The results show that the size of commits follows a heavy-tailed distribution. Therefore, although most commits involve just a few files, a relevant number of them involve many files. A single commit may include hundreds of files. In contrast with what one may intuitively assume, large commits do not occur only in Merge or Management activities.

This result is significant to studies that consider the set of files in a commit, which is the case of studies on change impact analysis and code authorship. In these works, disregarding that a relevant number of commits (more than 50%) involve a very high number of files may introduce bias in the analysis. In change impact analysis studies, the files in large commits may be more likely not to relate to a common cause of the change. In authorship analysis, a commit by a contributor involving a large number of files may not express authorship.

JDK is an exception. Many empirical studies have considered JDK, and our results revealed that JDK is an exception regarding the number of files per commit. The commits of JDK involve a higher number of files per commit than the other systems. Therefore, the study design based on commit analysis should consider this characteristic if JDK is part of its analysis.

Commits' size is not Normal. All results shown in this study lead us to a simple but not so obvious conclusion: one of the essential characteristics regarding commits' size is that we cannot apply the Normal distribution statistical analysis methodologies to them. The number of files modified in a commit has a long tail distribution, and besides, there is no standard distribution for the number of commits considering the activity type.

The time intervals of commits by developers. Understanding the developers' behavior when registering commits in the

repositories may aid practitioners, especially in management tasks. We investigated if there is a pattern of time intervals in which developers register commits in the repositories. The results indicate that, in a project, the distribution of the time intervals is approximately a Normal distribution, i.e., the distribution tends to be symmetric, and the mean is representative. The results also show that the time intervals vary among the projects. In this work, we do not investigate the causes of the behavior of developers when performing commits. However, we presume that the projects' nature, application domain, and the number of contributors may influence the frequency of commits by developers.

5 RELATED WORK

Previous works investigated the characteristics of commits in CVCS [1, 11, 12, 17]. Alali et al.[1] analyzed nine OSS from Subversion to characterize commits regarding the number of files, number of lines, number of hunks committed together, and the top 25 words used in the systems' log messages. They analyzed GCC, Collab, JEdit 6.1, Ruby, LinuxBoss, Phpmyadmin, MySql-Administrator, and Python Debian-installer. They found that 75% of the commits are very small and that the largest commits usually encompass all the system's files or add/modify a large file.

Hattori and Lanza[11] studied the size of the commits considering the number of files and the content of their log messages. They considered nine OSS: aMSN, ArgoUML, Firebird, JEdit, JHotdraw, Mantis, Miranda, Spring, and Swig. They found that the number of files in commits follows a Pareto distribution. They classified the commits into four groups according to the number of files: tiny (1 to 5), small (6 to 25), medium (26 to 125), and large (up to 126). They concluded that these categories correspond to 80%, 15%, up to 5%, and less than 1% of the commits in a project. They concluded that: tiny commits are related to Corrective Maintenance, small and medium commits are heterogeneous, large commits are more related to Management activities in five projects, while Forward Engineering is the most frequent activity in four, and Management activities tend to generate larger commits, while Corrective activities are related to small and tiny activities.Hindle et al.[12] concentrated on large commits. They analyzed data from nine OSS: Boost, MySQL, Firebird, Samba, Egroupware, Enlightenment, Spring Framework, PostgreSQL, and Evolution. They concluded that most large commits are perfective, and most small ones are Corrective. The findings of Marzban et al.[17] are similar to Hindle et al.[12]. Marzban et al. also investigated the relationship between size and type of commit. Their study considered data up to 2008 from 10 OSS: Bug-buddy, Epiphany, Gconf-editor, Gedit, Gnome-desktop, Gnome-terminal, Metacity, Nautilus-cd-burner, Sound-juicer, and Yelp. in small categories, most activities are related to bugs - fixing bugs or file-, but in large commits adding new files or data is more common.

With the advent of Distributed Version Control Systems (DVCS), such Git, the research community's interest in commit characterization has increased. Zafar et al. [21] developed an automatic approach to classify a commit as a "bug-fix commit" or not. However, their approach covers only the corrective engineering category, whereas our approach covers five categories. Casalnuovo et al. [3] created a tool called GitcProc for mining and processing commit data given the URL of the git repository. Among other data, the tool indicates which commits are involved in bug fixes. To do so, the tool uses regular expression matching to find error related-keywords in the messages associated with each commit. Similarly, our study also analyzed keywords in the commits' messages. Levin and Yehudai [15] proposed a method based on source code changes to classify commits into three maintenance activities: corrective, perfective, and adaptive. Differently from the works of Casalnuovo et al. and Levin and Yehudai, we considered five types of activities. Dey et al. [4, 5] investigated commits performed by bots and found that most bot commits involve a single file. Yan et al. [20] proposed a model to identify commits that will be reverted. Goyal et al. [10] proposed a method to identify unusual commits in GitHub, i.e., commits whose characteristics differ from the majority commits of the same repository, such as large commits, commits in files that are rarely changed, and commits registered at unusual times. Their method is based on data distribution analysis, which we also applied in this study. However, they did not consider the same characteristics we considered in our analysis. Moreover, their work differs from ours since they define a model that compares commits of the same system, and we aim to identify general characteristics of commits. Brindescu et al.[2] conducted the first in-depth, large-scale study to analyze the differences between CVCS and DVCS empirically. Their study considered 132 repositories and showed that the commit size tends to decrease, and the quantity of issues tends to increase over the project evolution. Commits in DVCS are 32% smaller than in CVCS. Commits in DVCS are more likely to have references to issue tracking labels, and 81% of the developers split commits in DVCS and 76% in CVCS. Their survey with 820 developers revealed that most developers commit several times a day (65.96%) or once an hour (19.66%). In our study, we also investigated commits' size and frequency of commits by developers. However, our analysis is concentrated on Java-based software systems. Moreover, we considered three types of analysis regarding commits' size: all files, only Java files, and activity categories involved in the commits. Regarding the frequency of commits by developers, differently from Brindescu et al., we analyzed the distribution of time intervals in which the developers register commits.

Another difference in our work from the previous ones is that the GitHub repositories they analyzed were selected based on their popularity, i.e., the repositories most favorite or forked by developers. As we are interested in analyzing commit data, we based our sampling on the projects' number of commits.

6 THREATS TO VALIDITY

To answer RQ3, we relied on the approach defined in our previous work to categorize commits [8]. That approach is based on the automatic search for keywords in the commits' messages. Therefore, as described in Section 2, it was necessary to build a dictionary containing the keywords and their variations. We constructed the dictionary manually, which may cause us to forget some keywords. To mitigate this threat, we built the dictionary based on the keywords described by a previous work [11] and added new words that we found in the manual inspection of $\approx$ 500 commits. We also evaluated the approach via manual inspection and found high precision and recall.

We considered the field "author" of the commit's data to identify the developers when calculating the time interval between commits. Depending on the type of study, such an approach may introduce substantial bias in the results, which is the case, for instance, of studies on code ownership because a developer may have more than one GitHub username. However, this is not the case in the present work because we are interested in analyzing sequential commits performed by a user whose name may be properly identified.

This work focused on Java-based software systems and considered data from 24 Java-based systems hosted on GitHub. GitHub has about 20 million public repositories. Therefore, it is not possible to ensure the generalization of the results found in this study. However, as this study concentrates on commit's data, we selected the most rated systems containing the highest number of commits, resulting in a dataset containing $\approx 1M$ commits of mature systems from well-known owners, such as Apache.

7 CONCLUSION

The system's data hosted in GitHub have been profusely used in software engineering works. Commits data are one of the most used analysis sources in such works. However, not knowing or not considering the characteristics of commits may introduce biases in research. Besides, investigating the characteristics of commits may bring insight into the developers' practices and, hence, provide important information to practitioners to improve the management and the planning of the software activities.

We carried out an empirical study to characterize commit data in this work. We evaluated the 24 most popular and active Java-based projects hosted on GitHub. We analyzed $\approx 1M$ commits.

The main findings of this work revealed that:

- the size of commits follows a heavy-tailed distribution;
- most commits involve one to 10 files;
- most commits involve one to four source-code files;
- the commits involving hundreds of files not only refer to Merge or Management activities.
- the distribution of the time intervals is approximately a Normal distribution, i.e., the distribution tends to be symmetric, and the mean is representative.
- on average, a developer proceeds a commit every eight hours.

The results of this study lead to some lessons that should be considered by researchers in empirical studies based on commit analysis. In particular, the activity types involved in the commits and the number of files in a commit should be considered when designing a study.

Other analyses are essential to bringing more insights into the characteristics of commits, such as analyzing data of software systems developed in other programming languages; and investigating the relationship between issues and commits. Also, we can use artificial intelligence techniques such as natural language processing to improve the categorization of commits.

REFERENCES

[1] A. Alali, H. Kagdi, and J. I. Maletic. 2008. What's a Typical Commit? A Characterization of Open Source Software Repositories. In *2008 16th IEEE International Conference on Program Comprehension*. 182–191. https://doi.org/10.1109/ICPC.2008.24

[2] Caius Brindescu, Mihai Codoban, Sergii Shmarkatiuk, and Danny Dig. 2014. How do centralized and distributed version control systems impact software changes?. In *Proceedings of the 36th International Conference on Software Engineering*. 322–333.

[3] Casey Casalnuovo, Yagnik Suchak, Baishakhi Ray, and Cindy Rubio-González. 2017. GitcProc: A Tool for Processing and Classifying GitHub Commits. In *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis* (Santa Barbara, CA, USA) *(ISSTA 2017)*. Association for Computing Machinery, New York, NY, USA, 396–399. https://doi.org/10.1145/3092703.3098230

[4] Tapajit Dey, Sara Mousavi, Eduardo Ponce, Tanner Fry, Bogdan Vasilescu, Anna Filippova, and Audris Mockus. 2020. Detecting and characterizing bots that commit code. In *Proceedings of the 17th International Conference on Mining Software Repositories*. 209–219.

[5] Tapajit Dey, Bogdan Vasilescu, and Audris Mockus. 2020. An Exploratory Study of Bot Commits. In *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops* (Seoul, Republic of Korea) *(ICSEW'20)*. Association for Computing Machinery, New York, NY, USA, 61–65. https://doi.org/10.1145/3387940.3391502

[6] M. Dias, A. Bacchelli, G. Gousios, D. Cassou, and S. Ducasse. 2015. Untangling fine-grained code changes. In *22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. 341–350.

[7] Jon Eyolfson, Lin Tan, and Patrick Lam. 2014. Correlations between bugginess and time-based commit characteristics. *Empirical Software Engineering* 19, 4 (2014), 1009–1039.

[8] Mívian Ferreira, Diego Golçalves, Kecia Ferreira, and Mariza Bigonha. 2021. Inside Commits: An Empirical Study on Commits in Open-Source Software. In *Brazilian Symposium on Software Engineering* (Joinville, Brazil) *(SBES '21)*. Association for Computing Machinery, New York, NY, USA, 11–15. https://doi.org/10.1145/3474624.3474629

[9] Markus Michael Geipel and Frank Schweitzer. 2012. The link between dependency and cochange: Empirical evidence. *IEEE Transactions on Software Engineering* 38, 6 (2012), 1432–1444.

[10] Raman Goyal, Gabriel Ferreira, Christian Kästner, and James Herbsleb. 2018. Identifying unusual commits on GitHub. *Journal of Software: Evolution and Process* 30, 1 (2018), e1893.

[11] L. P. Hattori and M. Lanza. 2008. On the nature of commits. In *23rd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 63–71.

[12] Abram Hindle, Daniel M. German, and Ric Holt. 2008. What Do Large Commits Tell Us? A Taxonomical Study of Large Commits. In *Proceedings of the 2008 International Working Conference on Mining Software Repositories*. 99–108.

[13] Huzefa Kagdi, Malcom Gethers, and Denys Poshyvanyk. 2013. Integrating conceptual and logical couplings for change impact analysis in software. *Empirical Software Engineering* 18, 5 (2013), 933–969.

[14] Vladimir Kovalenko, Fabio Palomba, and Alberto Bacchelli. 2018. Mining file histories: should we consider branches?. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. 202–213.

[15] Stanislav Levin and Amiram Yehudai. 2017. Boosting Automatic Commit Classification Into Maintenance Activities By Utilizing Source Code Changes. In *Proceedings of the 13th International Conference on Predictive Models and Data Analytics in Software Engineering* (Toronto, Canada) *(PROMISE)*. Association for Computing Machinery, New York, NY, USA, 97–106. https://doi.org/10.1145/3127005.3127016

[16] C. Macho, S. McIntosh, and M. Pinzger. 2016. Predicting Build Co-changes with Source Code Change and Commit Categories. In *23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, Vol. 1. 541–551.

[17] Maryam Marzban, Zahra Khoshmanesh, and Ashkan Sami. 2012. Cohesion between size of commit and type of commit. In *Computer Science and Convergence*. Springer, 231–239.

[18] M. Vidoni. 2022. A systematic process for Mining Software Repositories: Results from a systematic literature review. *Information and Software Technology* 144 (2022), 106791. https://doi.org/10.1016/j.infsof.2021.106791

[19] Chengcheng Wan, Zece Zhu, Yuchen Zhang, and Yuting Chen. 2016. Multi-perspective change impact analysis using linked data of software engineering. In *Proceedings of the 8th Asia-Pacific Symposium on Internetware*. 95–98.

[20] Meng Yan, Xin Xia, David Lo, Ahmed E Hassan, and Shanping Li. 2019. Characterizing and identifying reverted commits. *Empirical Software Engineering* 24, 4 (2019), 2171–2208.

[21] S. Zafar, M. Z. Malik, and G. S. Walia. 2019. Towards Standardizing and Improving Classification of Bug-Fix Commits. In *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. 1–6. https://doi.org/10.1109/ESEM.2019.8870174