Inside Commits: An Empirical Study on 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

GitHub is currently the most popular open-source software hosting platform, containing about 20 million public repositories. Many studies have relied on data mined from GitHub repositories, especially commits. However, not knowing the characteristics of commits may introduce biases and threats in those studies. This work presents an empirical study to characterize commits in terms of three aspects: categories of activities performed in the commits; co-occurrences of activities in commits; and size of commits by category. We analyzed 1M commits from the 24 most popular and most active Java-based projects hosted in GitHub. The main findings of this work show that: reengineering is the most frequent activity; 30% of commits involve more than one type of activity; the most common co-occurrence of activities in commits is reengineering with forwarding and corrective reengineering, however in a low rate, only 8%. The results of this study should be considered by empirical works to avoid threats and biases when considering commits' data.

CCS CONCEPTS

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

KEYWORDS

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

ACM Reference Format:
Mívian M. Ferreira, Diego Santos Gonçalves, Mariza A. S. Bigonha, and Kecia A. M. Ferreira. 2021. Inside Commits: An Empirical Study on Commits in Open-Source Software. In *SBES '21: 35th Brazilian Symposium on Software Engineering, August 23–25, 2021, Joinville, Brazil.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3474624.3474629

SBES 2021, 23 - 27 August, 2021, Joinville, Brazil
© 2021 Association for Computing Machinery.
ACM ISBN 978-1-4503-9061-3/21/09…$15.00
https://doi.org/10.1145/3474624.3474629

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. In many of those studies, commits appear as an essential data source to be analyzed since it is the basic unit of information about activities performed on the projects.

Studies on co-change and change impact analysis, for instance, are firmly 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 [8, 12, 14, 16]. However, such an assumption may introduce biases in the studies because it does not consider the so-called *tangled changes* problem registered in the same commit transaction [4]. Code authorship studies also rely on commit analysis. Works on this subject usually consider the number of commits the developer register per files [2, 5–7]. However, considering only the files involved in a commit or the number of times a developer committed a file may introduce bias in the analysis performed in such works when the approach does not consider the commits' patterns in the repositories. In those contexts, knowing the number of files usually committed together or the frequency in which developers commit may bring insights to be considered when proposing an approach to analyze code authorship.

Previous studies have investigated the characteristics of commits in open-source software repositories. Many of them analyzed Centralized Version Control Systems (CVCS) data, such as CVS and SVN [1, 9, 11, 15]. However, after those studies, Distributed Version Control Systems (DVCS), such Git, have emerged, and few studies were carried out to understand the commits characteristics in this environment. One of the most popular hosting platforms for Git projects is GitHub, founded in 2007. Given this scenario, research related to the structure of commits is essential to increase the accuracy of studies that use GitHub as a data source. In this work, we aim to contribute with the characterization of commits in GitHub repositories in three aspects: (i) categories of activities performed in the commits; (ii) co-occurrences of activities in commits; and (iii) size of commits by category. GitHub hosts software projects developed in several programming languages. In this work, we focused on Java, which is the third most popular language on Github and, a great deal of empirical studies usually consider it. We analyzed data of 1M commits from 24 Java-based projects hosted in GitHub.

2 STUDY DESIGN

This section presents the method applied to construct the dataset analyzed in this work.

2.1 Dataset

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 high amount of time it takes to collect the data we analyzed in this study. As shown in Table 1, 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 a high number of commits, from 22.9K up to 92K. Besides, the dataset is diverse in terms of application domains.

System	Age	#Commits	#Stars	#Issues
ballerina-lang/ballerina-platform	3	96,121	2,644	10,086
neo4j/neo4j	8	69,702	8,315	2,849
jdk/openjdk	2	62,947	6,553	0
elasticsearch/elastic	10	57,414	52,228	25,326
camel/apache	11	50,138	3,489	0
graal/oracle	4	53,665	13,950	2,097
languagetool/languagetool-org	7	46,224	4,114	2,893
vespa/vespa-engine	4	46,403	3,363	358
lucene-solr/apache	4	34,703	3,863	0
rstudio/rstudio	9	34,292	3,423	4,707
alluxio/Alluxio	7	31,587	4,805	884
hazelcast/hazelcast	8	30,936	4,033	6,402
jenkins/jenkinsci	9	31,136	16,463	0
sonarqube/SonarSource	9	30,480	5,272	0
beam/apache	4	30,519	4,362	0
spring-boot/spring-projects	8	30,671	51,678	19,515
bazel/bazelbuild	6	28,662	15,673	8,205
shardingsphere/apache	4	28,457	12,387	3,800
ignite/apache	5	27,401	3,518	0
selenium/SeleniumHQ	7	26,432	19,074	6,843
cassandra/apache	11	25,994	6,278	0
flink/apache	6	25,543	14,626	0
hadoop/apache	6	24,584	11,041	0
tomcat/apache	9	22,909	4,984	0

Table 1: Dataset systems sorted by number of commits.

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 (Jan 2021). We developed a Python script using *GitPython* to collect all the commits' information for each repository: author, date, description message, and the modified files. We also defined a new commit attribute to our dataset: *issue number*. The GitHub platform allows attributing an *issue number* to a commit. An *issue* is a mechanism to link a commit to a specific project context such as bug description, development tasks, merge and pull request tasks, among others. To collect the commit *issues*, the script inspected its messages looking for the pattern `#digits`.

2.3 Commits Categories

To analyze the main activities registered in the system's commits, we classified each commit into six categories: merge, corrective engineering, forward engineering, reengineering, management, and others, including commits that do not match any of the five categories. Excepted by the merge category we have included, we used the same set of categories proposed by Hattori and Lanza [9]. We considered "merge" a particular category because, in GitHub, a merge is a specific activity that differs from the other management activities. Unlike Hattori and Lanza's approach, we do not use a hierarchy to set only one category for a commit, 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., correction and reengineering. This type of commit is called *tangled commit* [3].

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, read, 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 categories of commits.

To categorize a commit, we extracted keywords from the commit messages. We choose to analyze the messages because it presents the complete description of the commits' activities. To identify the commits' activity categories, we developed a Python script 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 API counts an instance of a word only if there is an exact match in the text. Therefore, we build a dictionary with keywords that correspond to the commits' categories and include the keywords variations, e.g., add, addition, adding, added, adds. To improve the categorization rate, we manually inspected ≈ 500 commit messages to identify keywords. Table 2 shows the final primary keywords set.

3 RESULTS

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

RQ1. How often the activity types are performed in commits?

To answer this research question, we categorized the commits as described in Section 2.3. Figure 1 shows the percentage of commits corresponding to each category: Merge, Corrective Engineering, Forward Engineering, Reengineering, Management, and others, in

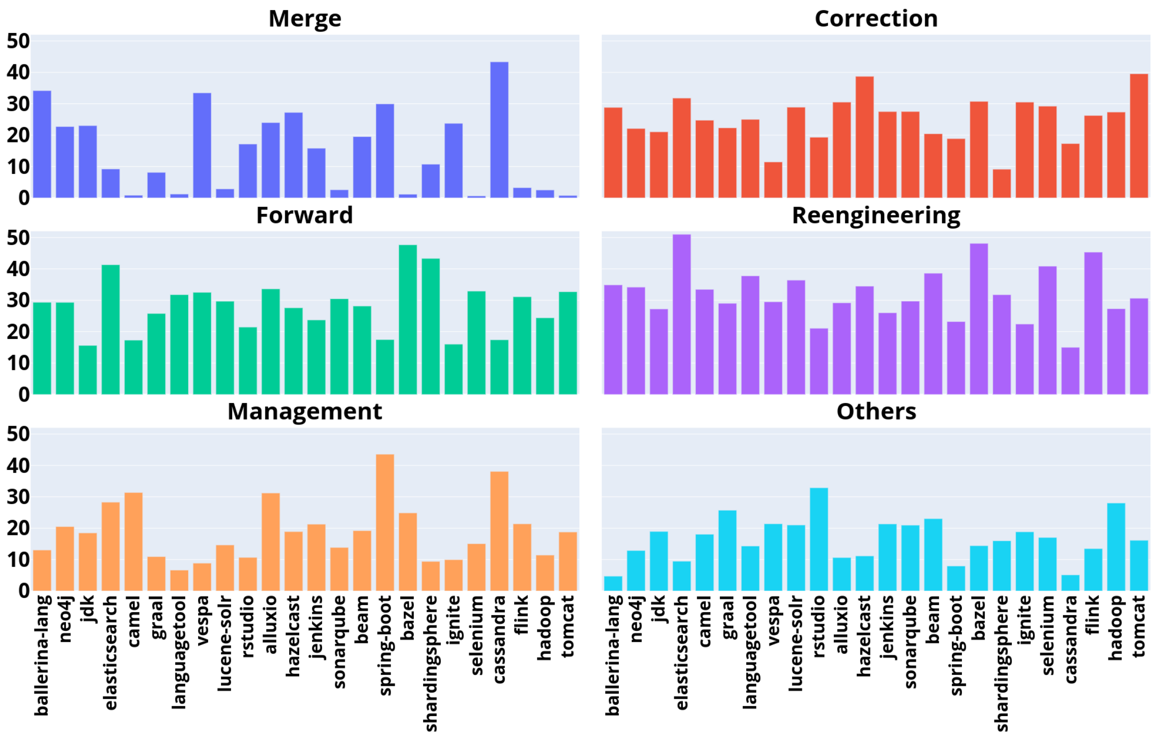

Figure 1: Percentage of commits by category.

the 24 software systems analyzed in this study, giving the percentage values by *(number of commits of a category) / (total number of commits registered in the project)*.

Reengineering is the most frequent activity, registered in 32.97% of the $\approx 1M$ commits analyzed in this work. This category has the highest percentage of commits in 13 out of the 24 systems. Besides, 83% (20/24) of the systems have at least 1/4 of the commits of Reengineering activities. *Cassandra* is the system with the lowest percentage of Reengineering activities (15%), and *elasticsearch* has the highest one (51%).

Forward Engineering is the second most frequent activity. The percentage of commits labeled as Forward Engineering ranges from 15.6% (*ignite* and *jdk*) to 47.7% (*bazel*). Forward Engineering comprises 28.2% of the commits. It is the most frequent category in three systems: *vespa*, *alluxio*, and *shardingsphere*.

Corrective Engineering is the third main activity. This category corresponds to 25% of the commits. *Shardingsphere* is the system with the lowest rate of commits of Corrective Engineering - only 9%. *Tomcat* presents the highest rate of commits tagged as Corrective Engineering and has this category as its main activity (39.6%).

Management corresponds to 18.7% of the commits. The results show that 75% (18/24) of the systems have less than 1/4 of their commits registering Management activities. The lowest number of Management activities is present in the *language tool* (6.6%), and *spring-boot* presented the highest one (43.6%).

Merge corresponds to 16,49% of the commits. This category presents a considerable disparity among the systems. The number of commits in the Merge category ranges from 0.6% (*selenium*) to 43.4% (*cassandra*). Only five systems present a percentage higher than 25%: *hazelcast* (27.3%), *spring-boot* (30%), *vespa* (33.47%), *ballerina-lang* (34.2%), and *cassandra*.

We used ***Other*** to tag messages whose content could not be categorized with any other five categories. This category corresponds to 16% of the commits. The percentages in Others category range from 4.7% (*ballerina-lang*) to 32.9%(*rstudio*). Besides *rstudio*, only two systems have more than 25% of commits tagged as Others - *graal* (25.7%) and *hadoop* (28%).

RQ2. How often co-occurrences between the activity types appear in commits?

A commit may involve more than one activity type. As described in Section 2.3, our approach allows classifying a commit with more than one category. We found that 30% of all commits analyzed in this work involve more than one activity type. We calculated the percentages of all possible co-occurrence between two categories. The results show a low rate in all cases, ranging from 1.6% to 8%.

The Merge category presented the lowest rate of co-occurrences, 1.6% to 2.8%. The highest rates of co-occurrence between the activity types are Reengineering with Corrective Engineering (8%), with Forward Engineering (8%), and with Management (6%).

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 category of commit: Merge, Corrective Engineering, Forward Engineering, and Management.

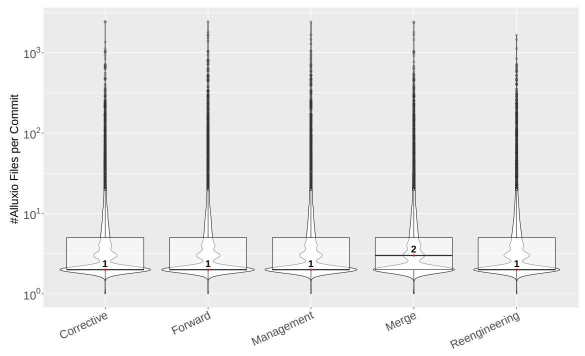

Figure 2: Alluxio distribution of files modified in commits grouped by category.

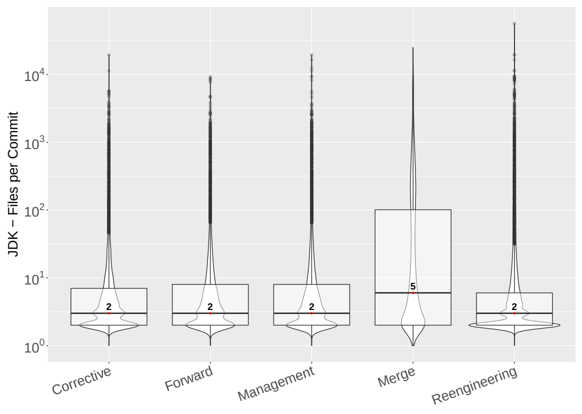

Figure 3: JDK distribution of files modified in commits grouped by category.

Figure 2 shows the results of *alluxio*. In the data distribution, the medians values are low, ranging from 1 to 3 files. The other systems presented a similar result, except *jdk*. Due to the limit of space, we do not show all graphics with the results of this research question. Figure 3 shows the results of *jdk*. The distributions of Reengineering, Forward Engineering, Corrective Engineering, and Management have the same pattern, and the median value is 2. Merge category presents a different result: it has the largest interquartile range:[1] 99 files. An important characteristic observed in *jdk* is that there are many commits that did not change any file and were categorized exclusively as Merge.

4 DISCUSSION

Understanding the characteristics of the dataset is critical for conducting a good experiment. This section discusses the main lessons learned from our study and their implications to studies that consider commits' data.

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 can report defects and failures they found in the systems continuously. This result indicates that studies on refactoring and faults may be favored by exploring commits' data. However, it is important to note the need to select the commits properly to be considered in those studies since they correspond to only 32.97% (Reengineering) and 25% (Corrective Engineering) of the commits in the systems. We should not ignore the percentage of Merge, Management, and Others activities: 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; this may be a reason for the high number of Forward Engineering.

The Quantification of the Tangled Changes Problem. We found that 30% of commits involve more than one activity type. This result indicates the need to take tangled changes into account when designing research because although tangle changes may not impact the development, they may impact the analysis of the repository data [10]. This care is critical in studies on change impact analysis. Many studies on this subject consider a commit as a basic unit of correlated changes.

Reengineering is 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. It is possible that correcting a bug or introducing a new feature in the system may cause a reengineering and, 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. Therefore, neglecting this fact may lead to significant threats to the studies.

5 RELATED WORK

Previous works investigated the characteristics of commits in CVCS [1, 9, 11, 15]. 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 seven systems and 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[9] studied the size of the commits considering the number of files and the content of their log messages. They considered nine OSS and 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 126). As we did in our work, they analyzed the commits' messages. However, they classified the commits into only two categories: development (forward engineering) and maintenance (reengineering, corrective engineering, and management). They concluded that: tiny commits are related to corrective maintenance, small and medium commits

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

are heterogeneous, large commits are more related to management activities in five projects, while forward engineering is the most frequent activity in four, management activities tend to generate larger commits, while corrective activities are related to small and tiny activities. Hindle et al.[11] concentrated on large commits. They analyzed data from nine OSS, and they concluded that most large commits are perfective, and most of the small commits are corrective. The findings of Marzban et al.[15] are similar to Hindle et al.[11]. Marzban et al. also investigated the relationship between size and type of commit. Their study considered data up to 2008 from 10 OSS, and they found that in minor categories, most activities are related to bugs (fixing bugs or files), but in large commits adding new files or data is more common.

There are some essential differences between these works and ours. The GitHub repositories they analyzed were selected based on their popularity, i.e., the repositories indicated as the favorite by most developers. As we are interested in analyzing commit data, we based our sampling on the number of commits the projects have. In our analysis, we considered data of the first-parent line. We support our decision by the findings of Kovalenko et al.'s study [13]. 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. Some previous works have raised the problem of the tangled changes [4, 10]. Eyolfson et al. [4] found that 15% of the bug fixes correspond to *tangled changes*. Our work quantified these problems straightforwardly in terms of terms of co-occurrences between activity types committed in single operations. We found that 30% of the commits involve more than one activity type. We also found that Reengineering is the activity that most occurs in association with the other ones.

6 THREATS TO VALIDITY

The main threats to the validity of this work is the commits categorization.

To indicate that the script found a keyword, the API needs to find a perfect match in the analyzed text. Therefore, it was necessary to build a dictionary containing the keywords and their possible variations. We constructed the dictionary manually, we cannot claim completeness. To mitigate this problem, we built the dictionary based on the keywords described by Hattori and Lanza [9] and added new words that we found in the manual inspection of a sample of the commits' messages.

We analyzed data from 24 Java-based systems hosted on GitHub. We selected the most rated systems containing the highest number of commits, resulting in a dataset containing more than 496K commits. We also analyzed mature systems from well-known owners, such as Apache. However, it is not possible to state that the results shown in this work can be generalized to any proprietary software.

7 CONCLUSION

Commits data are one of the most used source of analysis in software engineering research. However, not knowing or not considering the characteristics of commits may introduce biases in research. In this work, we carried out an empirical study to characterize commit data. We considered the 24 most popular and active Java-based projects hosted in GitHub. We analyzed $\approx 1M$ commits.

The main findings of this work revealed that: (i) Reengineering is the most frequent activity, followed by Forward Engineering, and Corrective Engineering; (ii) although low, the frequency of Merge and Management activities are relevant; (iii) 30% of commits involves more than one type of activity; (iv) the most common co-occurrences are between Reengineering and Forward Engineeringn and between Reengineering and Corrective Engineering; (v) many commits involve hundreds of files and those commits not only refer to Merge or Management. The results of this studies lead to some lessons that should be considered by 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.

Further research should analyze the data of software systems developed in other programming languages, besides investigating patterns of contributors' commit practice and the characteristic of the relationship of issues and 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.

[2] Christian Bird, Nachiappan Nagappan, Brendan Murphy, Harald Gall, and Premkumar Devanbu. 2011. Don't touch my code! Examining the effects of ownership on software quality. In *19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*. 4–14.

[3] 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.

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

[5] Mívian Ferreira, Marco Tulio Valente, and Kecia Ferreira. 2017. A comparison of three algorithms for computing truck factors. In *25th International Conference on Program Comprehension (ICPC)*. 207–217.

[6] Matthieu Foucault, Jean-Rémy Falleri, and Xavier Blanc. 2014. Code ownership in open-source software. In *18th International Conference on Evaluation and Assessment in Software Engineering*. 1–9.

[7] Thomas Fritz, Gail C Murphy, Emerson Murphy-Hill, Jingwen Ou, and Emily Hill. 2014. Degree-of-knowledge: Modeling a developer's knowledge of code. *ACM Transactions on Software Engineering and Methodology* 23, 2 (2014), 1–42.

[8] 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.

[9] 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.

[10] K. Herzig and A. Zeller. 2013. The impact of tangled code changes. In *10th Working Conference on Mining Software Repositories (MSR)*. 121–130.

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

[12] 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.

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

[14] 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.

[15] 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.

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