How Do Code Changes Evolve in Different Platforms? A Mining-based Investigation

Markos Viggiato[1], Johnatan Oliveira[2], Eduardo Figueiredo[2], Pooyan Jamshidi[3], Christian Kästner[4]
[1]*Dept. of Electrical and Computer Engineering, University of Alberta*, Edmonton, Canada
[2]*Computer Science Department*, *Federal University of Minas Gerais*, Belo Horizonte, Brazil
[3]*Computer Science and Engineering Department*, *University of South Carolina*, Columbia, United States
[4]*Institute for Software Research*, *Carnegie Mellon University*, Pittsburgh, United States

***Abstract*—Code changes are performed differently in the mobile and non-mobile platforms. Prior work has investigated the differences in specific platforms. However, we still lack a deeper understanding of how code changes evolve across different software platforms. In this paper, we present a study aiming at investigating the frequency of changes and how source code, build and test changes co-evolve in mobile and non-mobile platforms. We developed regression models to explain which factors influence the frequency of changes and applied the Apriori algorithm to find types of changes that frequently co-occur. Our findings show that non-mobile repositories have a higher number of commits per month and our regression models suggest that being mobile significantly impacts on the number of commits in a negative direction when controlling for confound factors, such as code size. We also found that developers do not usually change source code files together with build or test files. We argue that our results can provide valuable information for developers on how changes are performed in different platforms so that practices adopted in successful software systems can be followed.**

***Index Terms*—Software evolution, code changes, platforms.**

I. Introduction

Software systems developed in different platforms have different practices due to their specific needs [1]. Software platform (e.g., mobile and desktop) refers to the underlying "structure" upon which software is built and it has specific characteristics [2]. For instance, the mobile platform, differently from desktop and Web applications, is usually used to develop sensor-, gesture-, and event-driven applications and it has memory and power consumption constraints [2]. Also, in desktop, the most frequent high-severity bugs occur due to build issues, while in Android, the cause of most problematic bugs is concurrency [3].

Ignoring differences from software platforms may be problematic. Shedding light on software evolution of specific platforms may improve the development of platform-specific tools. It may also strongly benefit developers as they can be aware of how that platform works and behaves before getting into it, mainly those professionals who are looking for a new job. For instance, if a professional is looking for a mobile developer position, a previous understanding of common patterns of changes (e.g., types of code changes usually made together) may prepare the developer for an interview.

Revealing how code changes are performed may also support newcomers who intend to contribute to Open Source Software (OSS) projects, given their important role in the survival and long-term success of community-based OSS [4]. Due to the quite independent and self-organized characteristics of working in OSS projects [4], newcomers should be provided with insights and technical support of how current contributors in fact work so that they can be prepared. Prior work has addressed this issue [4], but it has not identified significant ways to support newcomers with technical barriers.

For instance, understanding how changes are performed in a repository before sending pull requests or joining an OSS project may provide the newcomer with valuable information to support this initial phase of contribution and avoid rework or contribution rejection, which could demotivate the developer to keep contributing. We argue that providing insights at platform level is an additional useful information to the newcomer, besides other library and framework-related information.

Another important aspect of understanding changes is concerning the specific behavior of the development team depending on the platform. Few works have studied the differences in desktop and mobile platforms [2, 5]. Desktop system developers usually are not involved in reporting bugs and the bug-fixing process takes a longer period of time compared to Android and iOS [2, 5, 6]. On the other hand, bug fixers of mobile applications are more involved in reporting bugs to be discussed and the main causes of bugs are concurrency (in Android) and application logic (in iOS) [2]. We believe companies should provide targeted training for their employees to focus on specific platform' characteristics and needs, and on how developers from those platforms usually work (behavior and practices adopted).

This paper presents a study aiming at understanding how code changes evolve in mobile and non-mobile platforms. At this stage, we focus on Android projects (mobile) and Java-based desktop/Web projects (non-mobile). We hypothesize the mobile platform has different evolution patterns compared to non-mobile platforms. The analyses are performed on a dataset composed of 363 popular OSS systems from GitHub: 181 Android applications and 182 desktop and Web applications. We investigate the frequency of commits, whether being mobile significantly impacts on the frequency, and the co-evolution of three different sorts of changes: source code changes, build changes, and test changes.

Our findings, while preliminary, show that non-mobile repositories have a higher number of commits per month compared to mobile. The trend graphs for both platforms are not similar, but both have a peculiar behavior in the holiday season (period from November to January): the number of commits sharply decreases. Our regression models suggest that being mobile significantly impacts on the number of commits in a negative direction when controlling for confound factors.

II. Methodological Procedures

Our goal in this study is to understand how code changes evolve in mobile and non-mobile platforms. Regarding the mobile platform, we consider only Android applications since they are largely present in GitHub and we were able to find several repositories. For the non-mobile platform, we consider desktop and Web applications as we intend to compare mobile platform against other platforms. That is, we do not aim at comparing all platforms against each other. We address the following initial research questions:

- *RQ1: How frequent are code changes in mobile and non-mobile platforms?*
- *RQ2: How is the co-evolution of source code changes, build changes and test changes in mobile and non-mobile platforms?*

A. Study Phases

To answer RQ1, we rely on statistical modelling (linear regression) to understand the frequency of commit activity in repositories from mobile and non-mobile platforms. To address RQ2, we make use of *Apriori* to analyze the co-evolution of code changes made to three types of files: source code, build, and test files. The study is composed of three main phases, detailed next.

Phase 1 - Software Repository Mining. We initially selected the 1000 most popular Java repositories in GitHub based on their number of stars, which is considered a reliable proxy to the repository popularity [7]. Aiming at retrieving the most relevant repositories, we filtered out repositories with less than 1000 SLOC (toy samples) and we consider only repositories with at least 24 commits in the last 2 years (active projects). This filtering process resulted in 363 repositories. We automatically classified these systems as mobile (if it contains *AndroidManifest.xml*) or non-mobile. Our final dataset contains 181 mobile systems and 182 non-mobile systems. Table I presents the aggregate statistics of our dataset. We can see the number of stars, SLOC, number of contributors, number of pull requests, and number of issues. The systems in our dataset are relevant as indicated the mean number of starts (i.e., more than 6,000 for both platforms). Furthermore, regarding SLOC, we can see that systems in non-mobile platforms are larger than systems in mobile, with means of 152K SLOC and 40K SLOC, respectively.

Phase 2 - Data Collection. Through the GitHub REST API[1], we collected the following data at repository-level: number of contributors, number of pull requests, number of issues, and SLOC. Furthermore, we collected commit-level data by the mining algorithm to perform the code change co-evolution analysis: commit date and type(s) of file(s) changed by the commit (source code, build or/and test files). In addition, we retrieved additional information (number of changed files, added lines of code, deleted lines of code, and total changed lines of code) to be used in next steps of our work.

[1]https://developer.github.com/v3/

TABLE I
Aggregate statistics of the 363 repositories

		Mean	St. Dev.	Min	Median	Max
Mobile	**Stars**	6308.32	4573.52	2451	4710	24975
	SLOC	40706.21	191940.8	1003	7807	2367689
	Contribut.	43.73	64.65	1	21	351
	Pull Req.	9.54	16.69	0	3	84
	Issues	125.98	193.72	0	65	1640
Non-Mobile	**Stars**	6490.28	6426.73	2443	4548	41653
	SLOC	152319.4	295851.9	1418	48158.5	2729887
	Contribut.	96.69	94.05	1	64	400
	Pull Req.	30.06	62.73	0	9	521
	Issues	231.83	304.37	0	120	1730

Phase 3 - Data Analysis. We have two main parts in the data analysis. First, we use statistical modelling to address the first research question. Second, we apply the *Apriori* algorithm to check whether different types of code changes co-occur in commits. Next, we detail how we proceed when building linear regression models and applying the mining algorithm.

B. Statistical Modelling

Our hypothesis is that mobile systems have a higher frequency of commits since users from that platform (in our case, Android users) expect fast bug fixes and rapid availability of new features [8, 9]. To provide evidence on whether being mobile influences the frequency of commits, we build two successive regression models: a model that contains only the control variables; and a model with the addition of the experimental (indicator) variable. We then use Cohen's f^2 measure to gauge the effect size of the indicator variable. We consider model coefficients important if they are statistically significant at a 0.05 level. In our models, the response (dependent) variable is the number of commits per month - **nCommMonth**. Based on variables that can influence the number of commits, we consider the following repository-level control (independent) variables: size of the system in terms of number of source lines of code - **sloc**, number of contributors - **nCont**, number of pull requests - **nPR**, and number of issues - **nIssues**. We also have an indicator (experimental) variable, **isMobile**, which is a binary variable that indicates whether a repository is mobile (1) or not (0).

Before building our models, we log-transformed variables aiming at stabilizing their variance and reduce heteroscedasticity [2, 10]. To compare the distribution of our raw data regarding number of commits per month for both groups (mobile and non-mobile), we adopt the non-parametric Wilcoxon Signed-Rank Test. We also report the Cliff's delta to indicate the size

of the difference of distributions. Finally, to tackle possible problems related to multicollinearity [11] in our regression analysis, we check for multicollinearity using the variance inflation factor (VIF) [12]. If it is below 3, which is a safe and conservative threshold, we confirm that our models do not suffer from multicollinearity [2, 13].

C. Mining frequent code changes and association rules

To find co-occurrences of different code changes file types along the last 2 years, we apply Apriori [14, 15]. We analyze whether there are co-occurrences of source code changes, build changes, and test changes. After obtaining the co-occurrences of code changes, we are able to find association rules also using the *Apriori* algorithm. Therefore, based on a change the developer performs, we can suggest other types of changes according to the learned association rules.

We rely on heuristics to identify build and test changes. For build changes, we look for files with names as recommended by the build automation tools we use. For Apache Maven build files, we search for *pom.xml*; for Apache Ant files, we look for *build.xml*; finally, for Graddle files, we search for *build.graddle*. Regarding changes on test files, we adapt an heuristic adopted by previous works [16, 17]. We classify a change as a test change if the name of the class begins with the word "Test" or ends with the word "Test", or "Tests", or "TestCase". We also consider a test change if the modified class is contained in a directory with the word "Test", "Tests", or "TestCase". Note that all situations in which the word is lower case are considered in the same way.

III. Results and Discussion

A. Frequency of Commits

We analyzed 465,500 commits from 363 repositories hosted in GitHub. Figure 1 presents the frequency of commits (average number of commits per month per repository) in mobile and non-mobile platforms in 2 years. We double checked the data to confirm the sharp drop in the end of the plot, which may be caused by discontinuation of mobile applications.

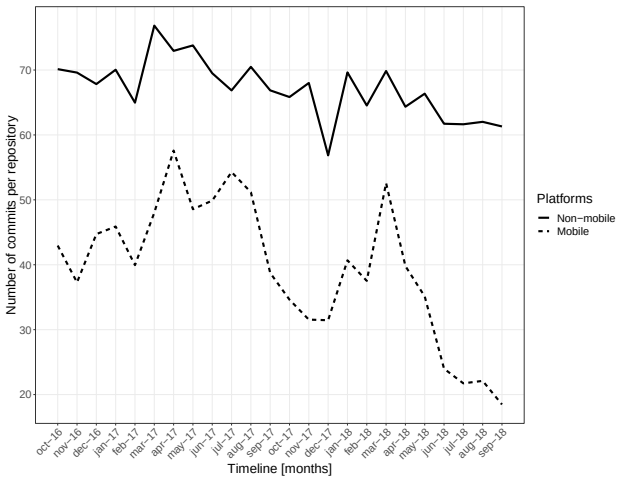

Fig. 1. Frequency of commits in a 2-year time period.

Surprisingly, the number of commits is always higher in non-mobile platform compared to mobile. For the mobile platform, we can note a regular pattern in some periods, specially in the holiday season (November to January). For instance, in the period from nov-16 to apr-17, the number of commits increased about 54%. The curve behaves similarly in the period from nov-17 to mar-18, with an increase of approximately 67%. This may suggest that some factors influence this behavior and contributions to OSS mobile projects in that period of the year. Regarding those periods in non-mobile platforms, the increase in the number of commits was much smaller, with 4.8% for nov-16 to apr-17 and 2.7% for nov-17 to mar-18. However, we can observe that non-mobile platforms had a very low average number of commits in December 2017 (56 commits), which suggests that holiday season may also influence the work activity in non-mobile projects.

This kind of temporal picture helps us to see the general trends, and how both platforms behave along these 2 years. However, we still lack an explanation regarding what factors are impacting the frequency. We then developed multiple linear regression models to understand the impact of the platforms on the frequency when controlling for confound variables.

Distribution comparison. Figure 2 presents the boxplots corresponding to the distribution of commits per month (response variable) for both platforms: mobile and non-mobile. We can see that the two distributions are different. In fact, the median number of commits per month for mobile is approximately 63, while for non-mobile is 84. In addition, we obtained a Cliff's Delta of -0.2181 (small), with a 95% confidence interval, indicating a small but statistically significant difference.

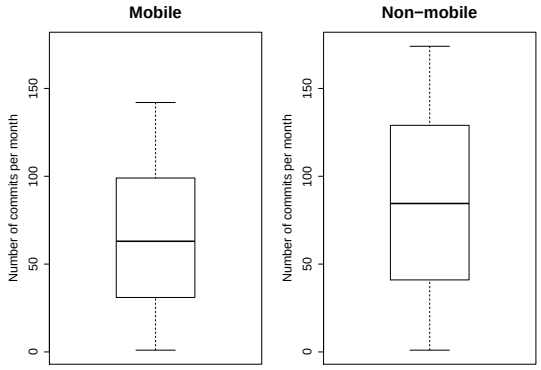

Fig. 2. Distributions of response variable for mobile and non-mobile.

Additional explanatory power. Table II presents our model coefficients along with their p-values. From the model with only control variables, we observe that most coefficients are in the positive direction (positive *T value*), as expected. For instance, we expect that more contributors result in more commits per month. The same is valid for number of pull requests and number of issues. The unexpected results occur for *sloc* coefficient, as its signal is negative. By inspecting the significance, apart from the intercept coefficient, all co-

efficients are not significant. We checked for correlation of control variables with the response variable and in fact they are not highly correlated. The highest correlation value occurs for number of contributors and number of commits per month (pearson coefficient of 0.1992).

TABLE II
MULTIPLE LINEAR REGRESSION COEFFICIENTS FOR OUR TWO MODELS.

	Variable	T value	P value (significance)
Control variables only	(Intercept)	61.02	<2e-16 ***
	nCont	1.283	0.2
	sloc	-0.224	0.823
	nPR	0.235	0.814
	nIssues	1.233	0.218
Full model, including indicator variable	(Intercept)	66.194	<2e-16 ***
	isMobile	-15.246	<2e-16 ***
	nCont	-1.19	0.235
	sloc	-1.525	0.128
	nPR	-0.117	0.907
	nIssues	0.751	0.453

***p <0.001, **p <0.01, *p <0.05

To gauge the effect of the indicator variable (*isMobile*), we build a successive regression model including the binary variable. In Table II, we can also see the coefficients of the full model. In fact, the indicator variable has a statistically significant impact on the frequency of commits in repositories (p-value <2e-16). The indicator variable also increased the explanatory power of the model, as suggested by a proportional change in R^2 of 1,900% (from 0.02 to 0.4). We adopted the Cohen's f^2 measure to estimate the effect size of the indicator variable. We computed the R^2 for both models (controls and indicator, and only controls) and obtained a Cohen's f^2 of **0.19**. The following thresholds are suggested to indicate the effect size: 0.02 (small), 0.15 (medium), and 0.35 (large). We can therefore conclude that the effect of being a mobile repository on the frequency of commits when controlling for confound variables is **medium**.

Multicollinearity diagnosis. We diagnose our models, checking for multicollinearity, since highly correlated regressors may inflate the variance. We first check the correlation between the predictors and then we get the variance inflation factor (VIF) for each predictor. We noticed that predictors are not highly correlated (the highest correlation is 0.6 between number of pull requests and number of issues). Regarding the variance inflation factor, all variables have VIF values below 3, which is a safe value and indicate that our models do not suffer from multicollinearity [2, 13].

B. Frequent Code Change Types and Association Rules

Regarding the types of frequent code changes, we analyzed the commit history using the *Apriori* algorithm and set a minimum support value of 0.05. When analyzing the association rules, we focus on the *confidence* and *lift* metrics to check the strength of the rules. Table III presents the code changes that co-occur along with their support and absolute count values.

We can note that all types of changes, when considered individually, appear in the results returned by the algorithm. However, we focus only on types of changes that occur together with other types. That is, we analyze results where at least two types of changes appear. In mobile systems, the types of code changes that occur together (i.e., in the same commit) with a support greater than the minimum support is source code and test changes. This co-occurrence happened with a support of 0.054 (i.e., in 5.4% of all commits). The low support indicates that developers do not usually perform changes in source code and test files simultaneously. Surprisingly, mobile developers do not usually change source code files together with build files [18]. We performed some tests and found that source code changes occur together with build changes only with a support of 0.03. Regarding non-mobile platforms, we see two co-occurrences of types of changes. First, source code changes occur together with test changes with a support of 0.051. Second, source code changes also co-occur with build changes with a similar support: 0.052.

TABLE III
FREQUENT TYPES CODE CHANGES IN ALL COMMITS.

	Frequent change types	Support	Absolute count
Mobile	{build}	0.07081202	12166
	{test}	0.07848341	13484
	{source_code}	0.67038014	115176
	{source_code,test}	0.05435169	9338
Non-mobile	{test}	0.09728867	28573
	{build}	0.11002986	32315
	{source_code}	0.70661541	207528
	{source_code,test}	0.05106353	14997
	{source_code,build}	0.05217012	15322

To obtained association rules, we rely on default values of minimum support (0.001) and minimum confidence (0.8) defined by the `arules` package in R. We then found the following rule for the mobile platform: *{build,test} => {source code}*. This association rule has a support value of 0.0062, confidence of 0.9177, lift of 1.3689, and absolute count of 1070. This rule indicates that developers commonly perform changes in source code files given they changed build and test files. However, the low value of support shows that both sides of the rule (build and test changes, and source code changes) do not occur very frequently in commits.

A similar rule was obtained for non-mobile platforms: *{build,test} => {source code}*. This association rule has a support value of 0.0134, confidence of 0.8597, lift of 1.2166, and absolute count of 3927. Although the association rule for non-mobile is the same of mobile, we obtained different metric values. For instance, the support is 2.15 times higher in non-mobile than in mobile, which indicates that both sides of the rules (build and test changes, and source code changes) occur more frequently in commits of non-mobile applications. However, the confidence is lower. This indicates that, although a source code change is likely to be necessary given that build and test changes were made, the strength of this statement is smaller compared to mobile platform.

As we observed, there is a regular pattern of change in the mobile platform, with a possible seasonal behavior. As a possible **implication for newcomers** who intend to contribute

to mobile repositories, we recommend that they look at the temporal trend before sending a contribution to an OSS project. For instance, it is not recommended to send a pull request in the end of the year as projects are not very active at that time. Instead, we recommend to send a pull requests in the beginning of the year. We also highlight that source code contributions to non-mobile repositories may require changes to test files or build configuration files as well since changes to such kinds of files usually occur together.

IV. Related Work

Several works have investigated different types of code changes and performed commit history analysis. For instance, Levin and Yehudai [17] analyzed the co-evolution of only test changes and source code changes. In this work, we also included build changes in our co-evolution analysis. In addition, Macho et al. [18] performed a study on build changes relying only on the Apache Maven build automation tool (*pom.xml* files). On the other hand, we included two other build automation tools: Ant (*build.xml*) and Graddle (*build.graddle*). Furthermore, we compare the co-evolution of changes in mobile applications against non-mobile applications. Faragó et al. [19] investigated whether modifications performed on frequently changing code have worse effect on software maintainability than those affecting less frequently modified code. Their findings indicated that modifying high-churn code is more likely to decrease the overall maintainability of a software system, which can increase the number of defects.

In summary, prior works focused on different aspects of code changes. Here, we perform a broader analysis of code changes, investigating their frequency, factors that explain them, and the co-evolution of different types of changes.

V. Final Remarks and Future Work

This paper presented a study aiming at investigating how code changes evolve in mobile and non-mobile platforms by analyzing the frequency of commits. Our statistical analysis revealed that being mobile significantly impacts the frequency of commits. We also observed that in the mobile platform source code changes occur together with test changes with a low frequency. In non-mobile platforms, we found that (i) source code changes and test changes and (ii) source code changes and build changes. This is an undergoing work, and as a next step, we plan to investigate mobile, desktop, and Web platforms separately. We also intend to perform a time-series analysis to reveal more explicit patterns of evolution of code changes and understand which factors influence the evolution, and extend our study to analyze how scattered (number of changed files) and deep (LOC) code changes are in different platforms. Finally, we plan to interview professionals from each platform.

VI. Acknowledgments

This research work was partially supported by CAPES, CNPq (grant 424340/2016-0), and FAPEMIG (grant PPM-00651-17). Kästner's work has been supported in part by the NSF (awards 1318808, 1552944, and 1717022) and AFRL and DARPA (FA8750-16-2-0042).

References

[1] E. Murphy-Hill, T. Zimmermann, and N. Nagappan, "Cowboys, ankle sprains, and keepers of quality: How is video game development different from software development?" in *36th Int'l Conference on Software Engineering*, 2014.

[2] Y. Zhang, Y. Yu, H. Wang, B. Vasilescu, and V. Filkov, "Within-ecosystem issue linking: a large-scale study of rails," in *Proceedings of the 7th International Workshop on Software Mining*, 2018, pp. 12–19.

[3] B. Zhou, I. Neamtiu, and R. Gupta, "A cross-platform analysis of bugs and bug-fixing in open source projects: Desktop vs. android vs. ios," in *Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering*, 2015, p. 7.

[4] I. Steinmacher, T. U. Conte, C. Treude, and M. A. Gerosa, "Overcoming open source project entry barriers with a portal for newcomers," in *Proceedings of the 38th International Conference on Software Engineering*, 2016, pp. 273–284.

[5] P. Bhattacharya, L. Ulanova, I. Neamtiu, and S. C. Koduru, "An empirical analysis of bug reports and bug fixing in open source android apps," in *17th European Conference on Software Maintenance and Reengineering (CSMR)*, 2013, pp. 133–143.

[6] S. Breu, R. Premraj, J. Sillito, and T. Zimmermann, "Information needs in bug reports: improving cooperation between developers and users," in *Proceedings of the ACM conference on Computer supported cooperative work*, 2010, pp. 301–310.

[7] H. Borges, A. Hora, and M. T. Valente, "Understanding the factors that impact the popularity of github repositories," in *International Conference on Software Maintenance and Evolution (ICSME)*, 2016, pp. 334–344.

[8] J. Oliveira, M. Viggiato, M. Santos, E. Figueiredo, and H. Marques-Neto, "An empirical study on the impact of android code smells on resource usage," in *International Conference on Software Engineering & Knowledge Engineering (SEKE)*, 2018.

[9] A. Banerjee and A. Roychoudhury, "Automated re-factoring of android apps to enhance energy-efficiency," in *International Conference on Mobile Software Engineering and Systems (MOBILESoft)*, 2016, pp. 139–150.

[10] P. Cohen, S. G. West, and L. S. Aiken, *Applied multiple regression/-correlation analysis for the behavioral sciences*. Psychology Press, 2014.

[11] D. E. Farrar and R. R. Glauber, "Multicollinearity in regression analysis: the problem revisited," *The Review of Economic and Statistics*, pp. 92–107, 1967.

[12] P. D. Allison, *Multiple regression: A primer*. Pine Forge Press, 1999.

[13] A. Trockman, S. Zhou, C. Kästner, and B. Vasilescu, "Adding sparkle to social coding: an empirical study of repository badges in the npm ecosystem," in *Proceedings of the 40th International Conference on Software Engineering*, 2018, pp. 511–522.

[14] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, A. I. Verkamo *et al.*, "Fast discovery of association rules." *Advances in knowledge discovery and data mining*, vol. 12, no. 1, pp. 307–328, 1996.

[15] R. Agrawal, T. Imieliński, and A. Swami, "Mining association rules between sets of items in large databases," in *Acm sigmod record*, vol. 22, no. 2. ACM, 1993, pp. 207–216.

[16] A. Zaidman, B. Van Rompaey, A. van Deursen, and S. Demeyer, "Studying the co-evolution of production and test code in open source and industrial developer test processes through repository mining," *Empirical Software Engineering*, vol. 16, no. 3, pp. 325–364, 2011.

[17] S. Levin and A. Yehudai, "The co-evolution of test maintenance and code maintenance through the lens of fine-grained semantic changes," in *International Conference on Software Maintenance and Evolution (ICSME)*, 2017, pp. 35–46.

[18] C. Macho, S. McIntosh, and M. Pinzger, "Extracting build changes with builddiff," in *14th International Conference on Mining Software Repositories (MSR)*, 2017, pp. 368–378.

[19] C. Faragó, P. Hegedűs, and R. Ferenc, "Cumulative code churn: Impact on maintainability," in *15th International Working Conference on Source Code Analysis and Manipulation (SCAM)*, 2015, pp. 141–150.