Is Refactoring Always a Good Egg? Exploring the Interconnection Between Bugs and Refactorings

Amirreza Bagheri
University of Szeged
Hungary
bagheri@inf.u-szeged.hu

Péter Hegedűs
University of Szeged
Hungary
hpeter@inf.u-szeged.hu

ABSTRACT

Bug fixing and code refactoring are two distinct maintenance actions with different goals. While bug fixing is a corrective change that eliminates a defect from the program, refactoring targets improving the internal quality (i.e., maintainability) of a software system without changing its functionality. Best practices and common intuition suggest that these code actions should not be mixed in a single code change. Furthermore, as refactoring aims for improving quality without functional changes, we would expect that refactoring code changes will not be sources of bugs. Nonetheless, empirical studies show that none of the above hypotheses are necessarily true in practice. In this paper, we empirically investigate the interconnection between bug-related and refactoring code changes using the SmartSHARK dataset. Our goal is to explore how often bug fixes and refactorings co-occur in a single commit (tangled changes) and whether refactoring changes themselves might induce bugs into the system. We found that it is not uncommon to have tangled commits of bug fixes and refactorings; 21% of bug-fixing commits include at least one type of refactoring on average. What is even more shocking is that 54% of bug-inducing commits also contain code refactoring changes. For instance, 10% (652 occurrences) of the Change Variable Type refactorings in the dataset appear in bug-inducing commits that make up 7.9% of the total inducing commits.

KEYWORDS

Bug inducing commits, refactoring, tangled code changes, empirical analysis

ACM Reference format:

Amirreza Bagheri and Péter Hegedűs. 2022. Is Refactoring Always a Good Egg? Exploring the Interconnection Between Bugs and Refactorings. In *Proceedings of MSR'22: Proceedings of the 19th International Conference on Mining Software Repositories (MSR 2022). ACM, New York, NY, USA, 5 pages.*

1 INTRODUCTION

Fixing software defects and improving code structure with refactoring are two of the most common software maintenance actions. They are inherently different in nature. A bug fix is a corrective code modification that fixes a flaw in the program. Developers correct the undesirable behavior by altering the code, database, or configuration, among other things. The method they use to fix the bug will be determined by the type of bug. Refactoring [6] on the other hand, is a code modification targeting the improvement of the internal software quality without changing its functional behavior. Since these typical software development activities are very different in their nature, researchers have extensively studied them separately.

Having high-quality and large-scale libraries of validated bugs and their concise patches collected from real-world applications are crucial for studying them. On the one hand, real bugs/patches are required for a thorough examination of a variety of automatic or semi-automatic methods for detecting problematic software programs, to finding incorrect statements [7], [8] and fixing incorrect applications [9], [10]. These methods are expected to function in real-world situations. As a result, before such approaches can be widely used in the field, they must be evaluated with a significant number of real bugs/patches from real-world applications [11]. Real bugs and fixes, on the other hand, may provide inspiration for finding, locating, and repairing software flaws. Researchers could, for example, establish which types of statements are more error-prone by evaluating genuine defects, and then try to repair those statements first during autonomous program repair [12]. The common repair patterns learned from human-written patches are another good example. Using such patterns improved the performance of automatic program repair greatly. [13]. Finally, statistics and learning-based methods to autonomous software repair [14] and bug identification [15] also rely on many real bugs/patches.

The research community has spent a lot of time looking into software refactoring as well. Observational studies investigated why and how developers perform refactoring [16], [17], [18], [19], [20] what refactorings are connected to application performance indicators [21], [22], [23], [24], developers' productivity [25], and how refactoring relates to other development tasks[26] .

Since refactorings are quality improving actions that do not alter functionality and bug fixes are targeted changes to correct functional flaws, one would expect that these activities are independent of each other. Nonetheless, despite the intuition, some researchers started to study the relationship between bugs and refactoring activities. Interestingly, according to some researchers [27], developers are typically apprehensive about refactoring efforts since they may introduce defects. Several studies have investigated the relationship between refactoring [28] and bugs, analyzing software repositories to see how much refactoring activities introduce bugs [29]. Weibgerber and Diehl [30] investigated the relationship between refactoring actions and the number of bug reports opened in the following days and found no significant link.

In this work, we leverage the wealth of bug related and refactoring activity data recorded in the SmartSHARK [1] dataset to empirically investigate the interconnection between bugs and refactorings. Aligned with previous works, we found that it is not uncommon to have tangled commits of bug fixes and refactorings; 21% of bug-fixing commits include at least one type of refactoring on average. What is even more shocking is that 54% of bug-

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

ACM ISBN 978-1-4503-9303-4/22/05…$15.00
https://doi.org/10.1145/3524842.3528034

inducing commits also contain code refactoring changes. We also identified the refactoring types that appear most frequently in bug inducing commits.

2 STUDY DESIGN

To explore the interconnections between bugs and refactorings, we designed a study based on the SmartSHARK [1] dataset of Java software repositories [2], which contains data for 96 projects, in particular the refactoring and bug-fixing activities, manually validated links between commits and bug issues, as well as the type of issues [4], and manually validated line labels that mark which changes contributed to a bug fix [5]. SmartSHARK is not just a dataset but a platform for replicable and reproducible software repository mining, a dataset that combines detailed information from the version control system with issue tracking data, GitHub pull request data, and Travis CI data. All the data in this database also has links to the various sources of information. We addressed the following research questions with the help of SmartSHARK:

RQ1. How common it is that a bug-fixing commit contains refactoring changes as well?

Our hypothesis is that bug-fixes should be independent changes not including any other types of code modifications. However, previous studies [27], [28] suggest that in practice, commits often contain tangled code changes. Therefore, we investigate how common it is that developers perform refactoring actions tangled together with bug-fixes.

RQ2. Do refactoring operations appear in code modifications inducing bugs?

We investigate if refactoring activities may lead to introducing bugs in the system (i.e., we can detect a bug fixing activity on the refactored code later in the commit history). For this, we analyzed if the commits marked as 'bug inducing' in the dataset also contain refactoring actions or not.

RQ3. What are the most common refactoring types appearing in bug inducing commits?

Finally, if we find that bug inducing changes may contain refactoring code modifications as well, we explore what are the most common types of refactorings we observe. It can help us understand what are the most 'dangerous' refactoring types where the developers need to pay special attention not to introduce bugs alongside with the modifications.

3 SMARTSHARK MINING

To carry out the study and answer our research questions, we analyzed the change history of 96 projects stored in the SmartSHARK dataset version 2.2. It is critical in our study to identify bug-fixing commits and those that reference the id of the issue resolved by the commit. Concerning the first point (i.e., labels for bugs), each commit in SmartSHARK has a set of labels indicating if that commit is a bug-fix or not, which are either automatically inferred by heuristics or confirmed by manual validation. In terms of the second issue, having an explicit link between commits and defects allows us to pinpoint the bug-fixing commits we require for our research. To find commits containing refactoring operations, we used SmartSHARK's RMiner detection tool results. The precision and recall of Rminer [3] are expected to be 98 percent and 87 percent, respectively. For finding fix-inducing updates, we searched for file actions related to the commits with an 'inducing' flag. All the mining scripts and collected data is available online.[1]

4 STUDY RESULTS

RQ1. How common it is that a bug-fixing commit contains refactoring changes as well?

Refactorings are behavior-preserving source code modifications, according to Fowler [6]. The fundamental goal of refactoring is to increase maintainability or comprehensibility, as well as to minimize the code footprint if necessary. Here we analyze if refactoring activity can be triggered by a bug in the code by analyzing bug fixing commits. We discovered that 41 out of 96 projects do have validated bug-fixing commits that contain refactoring activities as well. Moreover, for these projects a quite large portion of bug-fixing commits were labelled as refactoring commits as well. We relied on the label 'validated_bugfix' stored in SmartSHARK for each commit to identify. To find out if that commit contains refactoring actions, we observed the entries in the 'refactoring' collection referencing the same commit id. If a refactoring entry pointing to the same commit id had 'rMiner' value in its 'detection_tool' field, we identified this bug-fixing commit affected by refactoring as well. We found a total of 2,345 bug-fixing commits that also include refactorings in the SmartSHARK data set.

Project	NRB	RNB	RB	R%
ant-ivy	440	447	128	0.23
archiva	396	1107	148	0.27
calcite	250	908	177	0.41
cayenne	686	1479	164	0.19
commons-bcel	42	130	7	0.14
commons-beanutils	51	161	8	0.14
commons-codec	48	161	11	0.19
commons-collections	75	572	13	0.15
commons-compress	165	382	41	0.20
commons-configuration	182	759	61	0.25
commons-dbcp	88	243	18	0.17
commons-digester	22	216	4	0.15
commons-imaging	20	211	6	0.23
commons-io	98	220	21	0.18
commons-jcs	53	288	19	0.26
commons-jexl	109	331	54	0.33
commons-lang	213	552	29	0.12
commons-math	316	1075	80	0.20
commons-net	138	168	38	0.22
commons-scxml	46	178	21	0.31
commons-validator	69	129	4	0.05
commons-vfs	88	313	26	0.23
deltaspike	165	413	52	0.24
directory-fortress-core	42	179	10	0.19
eagle	94	211	36	0.28
falcon	218	534	98	0.31
giraph	109	329	32	0.23
gora	91	161	11	0.11
jspwiki	209	904	25	0.11
knox	279	381	69	0.20
kylin	1052	2241	212	0.17
lens	187	480	89	0.32
mahout	269	756	59	0.18
manifoldcf	555	789	116	0.17
nutch	466	403	83	0.15
opennlp	124	283	20	0.14
parquet-mr	84	492	36	0.30
santuario-java	70	466	27	0.28
systemml	207	1427	97	0.32
tika	536	686	134	0.20
wss4j	187	764	61	0.25
Total/Avg.	**8539**	**21929**	**2345**	**0.21**

Table 1. No Refactoring and Bug-fix (NRB) commits; Refactoring and No Bug-fix (RNB) commits; Refactoring and Bug-fix (RB) commits; R% = RB/(RB+NRB) proportion of RB commits from all bug-fixing commits

Table 1 presents the detailed results for the 41 projects we found bug-fixing commits tangled with refactoring actions. For the rest of the projects, we found no bug-fixing commits or only bug-fixes not including refactoring operation at all (thus the RB count is 0). The first column (NRB) in the table shows the number of bug-fixing commits that do not include refactorings, the second column (RNB) shows the number of commits identified as refactorings but not bug-fixes, the third column is the number of commits where bug-fixing and refactoring co-occur, and the last column shows the

[1] https://doi.org/10.5281/zenodo.6381329

ratio of RB commits compared to all bug-fixing commits. As can be seen, 21% of bug-fixing commits include at least one type of refactoring on average. So, this is not uncommon to perform refactorings alongside with bug fixing. This ratio is the lowest (5%) for the 'commons-validator' project, while the highest (41%) for project 'calcite'. However, this does not mean that most of the refactoring activities happen together with bug fixes. There are 21,929 commits containing refactoring but not labelled as bug-fix. The number of refactoring commits that are bug-fix commits as well is 2345, which is slightly more than 10% of all refactoring commits.

Answer to RQ1. It is not uncommon for bug-fixes and code refactorings to co-occur in the same commit. We found that 41 out of the 96 projects had such commits. We observed the highest proportion of such commits compared to the total number of bug-fixing commits for project 'calcite', where 41% of all bug removal code changes contained at least one refactoring operation as well. Nonetheless, refactorings do not typically occur tangled with bug fixes as only 10% of all the refactoring actions in these projects were identified in bug-fixing commits.

RQ2. Do refactoring operations appear in code modifications inducing bugs?

In theory, refactoring is described as performing simple actions in such a way that they are "unlikely to go wrong" and generate errors. To empirically investigate this hypothesis, we study the fix-inducing changes in commits overlapping with refactoring actions (i.e., search for commits that contain bug inducing changes and include at least one refactoring operation as well). We stress here, that again, we only observe the co-occurrence of bug-inducing changes and refactorings. A refactoring itself only induces a bug if a line affected by the refactoring is also modified in a bug-fixing commit later in the project history. Nonetheless, a refactoring could still influence the introduction of a bug even if it does not directly touch a line changed in the bug-fix. Therefore, it is very difficult to decide if a refactoring contributes to the bug introduction or simply co-occur with bug-inducing changes. We stress that here we only analyze whether refactoring operations appear in bug-inducing commits or not but do not track the exact modified lines back to bug-fixes. Nonetheless, it is in itself an interesting empirical question whether refactoring operations are associated with bug-inducing changes or not.

We used the bug-fix commits as a start to dig deeper and see what is happening in the files touched by these code changes. For this, we can locate bug-inducing commits inside the FileAction entries linked to the commits. In our study, we used the inducing commits labeled as 'JLMIV+' (Jira Links Manual(JLM), Issue Validation(IV), only java files(+), skip comments and empty spaces in blame(+)) that also had 'szz_type' value for the 'inducing' filed and omitted the 'hard_suspect' labels. Then, we investigated FileAction entries for 'change_file_action_id' and searched the issues for linked issues in those commits and if they had an issue type labelled as 'bug' and in the same commit refactorings were detected, we collected them. These commits are those that induce a bug in the system and contain refactoring operations as well. One of the projects ('commons-imaging') did not have any bug-inducing changes, therefore we have data for 40 projects in this analysis.

In our findings displayed in Table 2, we discovered that over 54% of the bug-inducing commits in the selected projects contain refactoring actions as well on average. The project-wise percentages range from 20% ('commons-validator' project) up to 71% (project 'calcite') of all bug-inducing commits in one project, which is very significant. Refactoring operations happen rather frequently in code changes leading to bug introduction. Interestingly, the same two projects are the two extremes as in case of RQ1. Since 'commons-validator' contains the lowest number of refactoring commits (133) among the 40 projects, it is not surprising that this is the project where the ratio of refactoring commits co-occurring with bug-fixes and bug-inducing changes are also the lowest. In the case of 'calcite', however, we observe a high number (1085) of refactoring commits but it is far not the highest. Yet, it contains the highest proportion of bug-fixing and bug-inducing commits tangled with refactorings among the 40 projects. 71% of all the bug-inducing changes do include refactoring operations as well.

Project	#Bug-Induce	RI	R%
ant-ivy	313	169	0.53
archiva	444	225	0.50
calcite	408	292	0.71
cayenne	522	317	0.60
commons-bcel	51	13	0.25
commons-beanutils	36	17	0.47
commons-codec	44	16	0.36
commons-collections	52	34	0.65
commons-compress	151	68	0.45
commons-configuration	150	76	0.50
commons-dbcp	63	28	0.44
commons-digester	20	6	0.30
commons-io	67	26	0.38
commons-jcs	128	88	0.68
commons-jexl	200	117	0.58
commons-lang	171	52	0.30
commons-math	284	128	0.45
commons-net	110	22	0.20
commons-scxml	54	29	0.53
commons-validator	35	7	0.20
commons-vfs	94	44	0.46
deltaspike	153	86	0.56
directory-fortress-core	63	33	0.52
eagle	107	71	0.66
falcon	282	180	0.63
giraph	142	104	0.73
gora	37	20	0.54
jspwiki	244	82	0.33
knox	188	106	0.56
kylin	879	525	0.59
lens	311	191	0.61
mahout	341	213	0.62
manifoldcf	440	218	0.49
nutch	270	125	0.46
opennlp	99	41	0.41
parquet-mr	124	82	0.66
santuario-java	96	60	0.62
systemml	490	307	0.62
tika	383	167	0.43
wss4j	187	105	0.56
Total/Avg.	**8233**	**4490**	**0.54**

Table 2. Total number of bug-inducing commits (#Bug-Induce), bug-inducing commits containing refactoring as well (RI), and their ratio (R%)

Answer to RQ2. The presence of code refactoring is even more significant in bug-inducing commits than in bug-fixing ones. More than half of the bug-inducing changes contain refactoring operations as well on average. For 'commons-validator' we observed the lowest ratio (20%), while for 'calcite' the highest (71%) of bug-inducing commits tangled with refactorings.

RQ3. What are the most common refactoring types appearing in bug inducing commits?

As we found that a significant portion (54% on average) of the bug-inducing commits contain refactoring operations as well, we used the SmartSHARK dataset once again to collect the actual types of refactorings appearing in such commits. For the sake of this RQ, we only evaluate the case where the refactoring overlaps with the bug-induce at the commit level. Table 3 lists the different types of refactorings and shows the number of commits including at least one such refactoring operation (first column, #Refact), the number of bug-inducing commits with that type of refactoring (second column, #Bug-Induce), the proportion of bug-inducing commits containing that type of refactoring compared to the total number of commits containing that refactoring (third column, R1%), and the proportion of bug-inducing commits compared to the total number of bug-inducing commits with that refactoring operation in the full dataset (fourth column, R2%). Small refactorings that change types or rename attributes/types (change variable

type, change return type, rename variable/attribute), as well as extracting code parts (extract method/attribute/variable/class) have the highest odds of appearing in bug-inducing code changes (highest R2%). For example, the Change Variable Type refactoring occurs in 7.92% of all the bug-inducing commits. Nevertheless, exactly these are the refactorings with the most occurrences in the dataset. Looking at the R1% values, we can see that Extract Subclass is at the top with 33%. It means that one third of this refactoring operation happens in code changes that induce bugs.

Type	#Refact	#Bug-Induc	R1%	R2%
change_variable_type	6475	652	0.10	0.0792
extract_method	5257	454	0.09	0.0551
change_return_type	3529	338	0.10	0.0411
extract_attribute	3306	331	0.10	0.0402
rename_variable	2844	241	0.08	0.0293
rename_attribute	1724	213	0.12	0.0259
rename_parameter	2200	208	0.09	0.0253
extract_and_move_meth	1228	206	0.17	0.025
move_method	1534	206	0.13	0.0250
extract_variable	1697	199	0.12	0.0242
extract_class	680	162	0.24	0.0197
rename_method	2406	143	0.06	0.0174
extract_superclass	720	136	0.19	0.0165
rename_class	1148	127	0.11	0.0154
move_attribute	1224	116	0.09	0.0141
inline_method	846	111	0.13	0.0135
parametrize_variable	614	102	0.17	0.0124
move_class	1664	87	0.05	0.0106
inline_variable	734	57	0.08	0.0069
pull_up_method	705	57	0.08	0.0069
replace_variable_with_at	479	46	0.10	0.0056
extract_subclass	123	41	0.33	0.0050
pull_up_attribute	467	39	0.08	0.0047
push_down_method	230	38	0.17	0.0046
push_down_attribute	170	36	0.21	0.0044
move_and_rename_class	479	32	0.07	0.0039
extract_interface	206	28	0.14	0.0034
merge_variable	159	23	0.14	0.0028
merge_parameter	127	21	0.17	0.0026
split_attribute	58	12	0.21	0.0015
merge_attribute	48	8	0.17	0.001
split_variable	48	7	0.15	0.0009
move_and_rename_attri	18	5	0.28	0.0006
replace_attribute	14	4	0.29	0.0005
split_parameter	28	4	0.14	0.0005

Table 3: Bug-Inducing commits by refactoring types

Answer to RQ3. Type changes of variables and return statements, renaming attributes and variables, and extracting method or attribute are the most frequent refactorings appearing in bug-inducing commits. Moreover, 33% of the extract subclass refactoring appears in bug-inducing commits, even though it has only few instances altogether (123) in the dataset. Instances of move class on the other hand rarely occur in bug-inducing commits (only 5% of the cases).

5 CONCLUSIONS

The goal of this study was to investigate the relationship between refactorings and bug-fixing or bug-inducing code changes. As they are completely different code maintenance activities, our hypothesis was that they do not co-occur within the same commit. Nonetheless, previous works already pointed out that this might not be the case in practice.

Using the rich data available in the SmartSHARK dataset, were able to connect refactoring activities to bug-fixing and bug-inducing commits. For identifying refactoring operations, we relied on the results of the RMiner tool included in the dataset, a highly precise detection technique with a reported 98 percent precision and 87 percent recall. For locating bug-inducing commits, we used the data produced by the SZZ algorithm, which is utilized to detect fix-inducing changes in SmartSHARK. We acknowledge that the validity of our results depends on the accuracy of these tools/algorithms and data they produce. However, the many high-quality works relying on these tools and data increase the confidence in the presented results.

We found that it is not at all uncommon to have refactoring operations tangled to bug-fixes in a single commit. This suggests that developers in practice perform code structure improvement upon finding and fixing a software defect. In 41 out of 96 projects, we found that on average, 21% of bug-fixing commits contain refactorings as well. Even though the highest percentage was 41%, refactorings do not typically occur tangled with bug fixes as only 10% of all the refactoring actions in these projects were identified in bug-fixing commits.

The presence of code refactorings were even more significant in bug-inducing commits, where 54% of these commits contained at least one refactoring operation. We must note, however, that we analyzed only the co-occurrence of refactorings in bug-inducing changes not that the refactored lines directly contributed to the defect introduced. Therefore, a co-occurring refactorings might not be the root causes of the defects. Nonetheless, this high number of refactorings in bug-inducing code is alarming. This result might trigger an alert for practitioners by pointing out that refactoring might not always be behavior-preserving in practice, therefore developers must be prepared with suitable verification and validation techniques to mitigate potential hazards caused by refactorings.

We even identified that type changes of variables and return statements, renaming attributes and variables, and extracting method or attribute are the most frequent refactorings appearing in bug-inducing commits. Moreover, 33% of the extract subclass refactoring appears in bug-inducing commits, even though it has only few instances in the dataset. Instances of move class on the other hand rarely occur in bug-inducing commits (only 5% of the cases).

In this study, we provided quantitative data that shows the interconnection of refactorings and bug-fixes as well as refactorings and bug-inducing code changes. However, further qualitative studies are needed to discover the precise relationship between them. In the future, we plan to carry out such analysis on the SmartSHARK data we extracted.

ACKNOWLEDGMENT

The presented work was carried out within the SETIT Project (2018-1.2.1-NKP-2018-00004). Project no. 2018-1.2.1-NKP-2018-00004 has been implemented with the support provided from the National Research, Development and Innovation Fund of Hungary, financed under the 2018-1.2.1-NKP funding scheme.

Furthermore, Péter Hegedűs was supported by the Bolyai János Scholarship of the Hungarian Academy of Sciences and the ÚNKP-21-5-SZTE-570 New National Excellence Program of the Ministry for Innovation and Technology.

REFERENCES

[1] A. Trautsch, F. Trautsch, S. Herbold, “MSR Mining Challenge: The SmartSHARK Repository Data,” Proceedings of the International Conference on Mining Software Repositories (MSR 2022), 2022 .

[2] A. Trautsch, F. Trautsch, S. Herbold, B. Ledel, and J. Grabowski, “The smartshark ecosystem for software repository mining,” in Proc. of the 2020 Int. Conf. Softw. Eng. - Demonstrations Track, 2020.

[3] N. Tsantalis, M. Mansouri, L. M. Eshkevari, D. Mazinanian, and D. Dig, “Accurate and efficient refactoring detection in commit

history," in Proceedings of the 40th International Conference on Software Engineering, ser. ICSE '18. New York, NY, USA: ACM, 2018, pp. 483– 494.

[4] S. Herbold, A. Trautsch, and F. Trautsch, "Issues with szz: An empirical assessment of the state of practice of defect prediction data collection," 2019.

[5] S. Herbold, A. Trautsch, B. Ledel, A. Aghamohammadi, T. A. Ghaleb, K. K. Chahal, T. Bossenmaier, B. Nagaria, P. Makedonski, M. N. Ahmadabadi, K. Szabados, H. Spieker, M. Madeja, N. Hoy, V. Lenarduzzi, S. Wang, G. Rodr´ıguez-P´erez, R. Colomo-Palacios, R. Verdecchia, P. Singh, Y. Qin, D. Chakroborti, W. Davis, V. Walunj, H. Wu, D. Marcilio, O. Alam, A. Aldaeej, I. Amit, B. Turhan, S. Eismann, A.- K.Wickert, I. Malavolta, M. Sulir, F. Fard, A. Z. Henley, S. Kourtzanidis, E. Tuzun, C. Treude, S. M. Shamasbi, I. Pashchenko, M. Wyrich, J. Davis, A. Serebrenik, E. Albrecht, E. U. Aktas, D. Str¨uber, and J. Erbel, "Large-scale manual validation of bug fixing commits: A finegrained analysis of tangling," 2020.

[6] FOWLER, M., & BECK, K. (1999). Refactoring: improving the design of existing code. Reading, MA, Addison-Wesley

[7] W. E. Wong, R. Gao, Y. Li, R. Abreu, and F. Wotawa, "A survey on software fault localization," IEEE Transactions on Software Engineering, vol. 42, no. 8, pp. 707–740, 2016.

[8] J. Lee, D. Kim, T. F. Bissyand´e, W. Jung, and Y. Le Traon, "Bench4BL: reproducibility study on the performance of ir-based bug localization," in Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis, 2018, pp. 61–72.

[9] D. Jeffrey, M. Feng, N. Gupta, and R. Gupta, "BugFix: A learning-based tool to assist developers in fixing bugs," in 2009 IEEE 17th International Conference on Program Comprehension. IEEE, 2009, pp. 70–79.

[10] B. Daniel, V. Jagannath, D. Dig, and D. Marinov, "ReAssert: Suggesting repairs for broken unit tests," in 2009 IEEE/ACM International Conference on Automated Software Engineering. IEEE, 2009, pp. 433–444.

[11] R. Just, D. Jalali, and M. D. Ernst, "Defects4J: A database of existing faults to enable controlled testing studies for java programs," in Proceedings of the 2014 International Symposium on Software Testing and Analysis, 2014, pp. 437–440.

[12] J. Xuan, M. Martinez, F. DeMarco, M. Clement, S. Marcote, T. Durieux, D. L. Berre, and M. Monperrus, "Nopol: Automatic repair of conditional statement bugs in java programs," IEEE Transactions on Software Engineering, vol. 43, no. 01, pp. 34–55, jan 2017.

[13] D. Kim, J. Nam, J. Song, and S. Kim, "Automatic patch generation learned from human-written patches," in 2013 35th International Conference on Software Engineering (ICSE), 2013, pp. 802–811.

[14] Y. Xiong, J. Wang, R. Yan, J. Zhang, S. Han, G. Huang, and L. Zhang, "Precise condition synthesis for program repair," in Proceedings of the 39th International Conference on Software Engineering, ICSE 2017, Buenos Aires, Argentina, May 20-28, 2017, S. Uchitel, A. Orso, and M. P. Robillard, Eds. IEEE / ACM, 2017, pp. 416–426.

[15] H. Zhong, X. Wang, and H. Mei, "Inferring bug signatures to detect real bugs," IEEE Transactions on Software Engineering, pp. 1–1, 2020.

[16] Emerson Murphy-Hill, Chris Parnin, and Andreaw P. Black. 2011. How We Refactor, and How We Know It. Transactions on Software Engineering 38, 1 (2011), 5–18.

[17] Anthony Peruma, Mohamed Wiem Mkaouer, Michael J. Decker, and Christian D. Newman. 2018. An Empirical Investigation of How and Why Developers Rename Identifiers. In Proceedings of the 2Nd International Workshop on Refactoring (IWoR 2018). 26–33.

[18] Danilo Silva, Nikolaos Tsantalis, and Marco Tulio Valente. 2016. Why we refactor? confessions of GitHub contributors. In Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2016. 858–870.

[19] Carmine Vassallo, Giovanni Grano, Fabio Palomba, Harald Gall, and AlbertoBacchelli. 2019. A large-scale empirical exploration on refactoring activities in open-source software projects. Science of Computer Programming 180, 1 (2019) ,1–15.

[20] Yi Wang. 2009. What motivate software engineers to refactor source code? Evidences from professional developers. In Software Maintenance, 2009. ICSM 2009. IEEE International Conference on. 413 –416.

[21] Mohammad Alshayeb. 2009. Empirical investigation of refactoring effect on software quality. Information and Software Technology 51, 9 (2009), 1319 – 1326.

[22] Alexander Chávez, Isabella Ferreira, Eduardo Fernandes, Diego Cedrim, and Alessandro Garcia. 2017. How Does Refactoring Affect Internal Quality Attributes?: A Multi-project Study. In Proceedings of the 31st Brazilian Symposium on Software Engineering (SBES'17). 74–83.

[23] Konstantinos Stroggylos and Diomidis Spinellis. 2007. Refactoring–Does It Improve Software Quality? In Proceedings of the 5th International Workshop on Software Quality (WoSQ '07). IEEE Computer Society, Washington, DC, USA,10–.

[24] Gábor Szoke, Gábor Antal, Csaba Nagy, Rudolf Ferenc, and Tibor Gyimóthy.2014. Bulk Fixing Coding Issues and Its Effects on Software Quality: Is It Worth Refactoring? In Source Code Analysis and Manipulation (SCAM), 2014 IEEE 14th International Working Conference on. IEEE, 95–104.

[25] Raimund Moser, Pekka Abrahamsson, Witold Pedrycz, Alberto Sillitti, and Giancarlo Succi. 2008. Balancing Agility and Formalism in Software Engineering. Chapter A Case Study on the Impact of Refactoring on Quality and Productivity in an Agile Team, 252–266.

[26] Mehran Mahmoudi, Sarah Nadi, and Nikolaos Tsantalis. 2019. Are Refactorings to Blame? An Empirical Study of Refactorings in Merge Conflicts. In 26th IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2019. 151–162.

[27] Gabriele Bavota, Bernardino De Carluccio, Andrea De Lucia, Massimiliano Di Penta, Rocco Oliveto, and Orazio Strollo. 2012. When Does a Refactoring Induce Bugs? An Empirical Study. In 12th IEEE International Working Conference on Source Code Analysis and Manipulation, SCAM 2012, Riva del Garda, Italy, September 23-24, 2012. 104–113.

[28] Isabella Ferreira, Eduardo Fernandes, Diego Cedrim, Anderson Uchôa, Ana Carla Bibiano, Alessandro Garcia, João Lucas Correia, Filipe Santos, Gabriel Nunes,Caio Barbosa, and et al. 2018. The Buggy Side of Code Refactoring: Understanding the Relationship between Refactorings and Bugs. In Proceedings of the 40 th International Conference on Software Engineering: Companion Proceeedings (ICSE'18). 406-407.

[29] Miryung Kim, Thomas Zimmermann, and Nachiappan Nagappan. 2012. A Field Study of Refactoring Challenges and Benefits. In Proceedings of the 20th International Symposium on Foundations of Software Engineering (Research Triangle Park, NC, USA).

[30] Peter Weißgerber and Stephan Diehl. 2006. Are refactorings less error-prone than other changes?.In Proceedings of the 2006 International Workshop on Mining Software Repositories, MSR 2006, Shanghai, China, May 22-23, 2006. 112–118.