On the Evolutionary Relationship between Change Coupling and Fix-Inducing Changes

Ali Zafar Sadiq, Md. Jubair Ibna Mostafa and Kazi Sakib
Institute of Information Technology, University of Dhaka, Bangladesh

Keywords: Change Coupling, Software Bug, Fix-Inducing Change, Software Defect.

Abstract: Change Coupling (CC) is the implicit relation formed between two or more changing software artifacts (e.g. source code). These artifacts are found to have design issues and code smells. Existing research has revealed the relationship between the change coupled relation of a class with the number of bugs in bug repositories. However, this ignored their true relation at the creation time of bugs or erroneous changes known as Fix-Inducing Changes (FIC). This paper tries to find the actual relationship between FIC and change coupled relations with respect to considering recent and all commits. This is done by traversing the entire history of a repository with a commit window of 100 commits and collecting data about FICs and metrics related to change coupling and object oriented system. It is found from the analysis that recent CC relations at the time of error are more correlated with new errors. Besides, it is found that explanatory power for predicting future erroneous change is more in recent CC relation than the one formed by considering all commits starting from the 1st commit.

1 INTRODUCTION

Software artifacts form change coupled relation by frequently changing together. Continuous changes in software artifacts indicate that previous changes were not enough to make the software work correctly. This may be the result of design issues or erroneous code changes. Changes, where any erroneous code is introduced in the software, are known as Fix-Inducing Changes (FIC). These changes create adverse effect and produces unexpected results (Sliwerski et al., 2005) (Antoniol et al., 2005). In order to analyze these erroneous changes, various works focused on different properties of change like affected files, time of day, developer experience and others that would induce the bugs (Levin and Yehudai, 2017), (Kim et al., 2008), (Menzies et al., 2007), (Fukushima et al., 2014). However, the change coupled relation of artifacts like files or classes (in Java) with the number of erroneous changes is yet to be investigated.

CC relations depend on the number of changes considered in the commit history of a software system. This relation when considered with all commits from the origin or 1st commit and that of a small number of commits recent to FIC or an erroneous change will differ. Besides, considering the same object-oriented and change coupled metrics for bugs like (D'Ambros et al., 2009), originated at a different period, may provide wrong information. So, it is important to investigate the real relationship between change coupling and erroneous changes with those metrics recent to FICs.

Different researches have been conducted in both change coupling and software bugs. In case of change coupling, works like predicting change (Zimmermann et al., 2005), suggesting overlooked change and that of identifying design issues or code smells are seen (Palomba et al., 2013). However, all of these ignore the fact that the frequently changing relation may have a role in producing bugs or erroneous changes. This fact is later on explored (D'Ambros et al., 2009). They have investigated this relationship by considering the software bugs with their proposed change coupled metrics (like Linearly Weighted Sum of Coupling or LWSOC, Sum of Coupling or SOC). This considers the same object oriented and change coupling metrics for all bugs but certainly does not represent at the time of creation of bugs or errors known as FICs.

For conducting this work, firstly, Fixing Changes (FCs) in the repository are obtained by searching commit comment history using keywords and number indicating bug id. From fixing changes, edited and deleted lines are detected by comparing a fixing change with its parent change. Since any change is stored in the repository as commit, so fixing commit contains the corrected change from erroneous one by changing or deleting erroneous lines of codes. Then those edited or deleted lines are tracked to their last modification to find FICs. Then considering a commit window of 100 commits the entire history of the repository is traversed. The number of FICs of a class and change coupled metrics (like LWSOC and SOC) proposed by (D'Ambros et al., 2009) and other object-oriented metrics (like number of attributes) are collected for finding their relationship.

The main contribution of this paper is to propose considering the relationship between FICs and change coupling. Besides, it investigated whether recent CC relations or total CC relations influences more in producing bugs. For this correlation analysis is performed and obtained result shows recent relation is correlated more with recent errors. After that from regression analysis to predict number of FICs within the short window of 100 commits, it is seen from the obtained R^2, the percentage of the response variable variation that is explained by a linear model, is more for recent CC relations.

2 METHODOLOGY

To analyze change coupled relations, historical information is collected from the Version Control System (VCS) git. All software changes in documents or source code are stored in VCS as commits, which contain information about a changed file, author, time and comment for why the change is made. Among those changes, Fixing Change (FC) correct errors and bugs. Then changed and deleted lines in those FCs lead to identifying the commit changes that introduced errors in the system, also known as Fix-Inducing Change (FIC). After that, the temporal analysis is performed to obtain the change coupled relations. All of these are elaborated in the following subsections.

2.1 Identifying Fix-Inducing Changes (FIC)

The entire process of finding FIC, also known as Bug Introducing Change (BIC), is shown in Figure 1. To find FIC, at first, all commit comments are searched for FCs. FC commits contain comments with keywords "Fix", "Bug", "Patch" or their past and gerund form or those keywords with bug identification number represented as a number following hash-tags like "#1234". Any commit containing any one of those traits is marked as FC commit without linking those to bug repository. This is because linking with bug repository has a problem when bug ids are not found in commit messages. So it might not fairly represent all bugs (Bird et al., 2009). In these FCs, codes are either modified or deleted to correct errors. The line number of these modified or deleted codes in the immediate parent commit of FC contains the erroneous code. Therefore, by using Diffj (Diffj, 2018), the differences in file contents of an FC commit and its immediate parent commit, erroneous codes and their line numbers are found. Since Diffj compares Java files based on their code and ignores formatting, organization, comments, or whitespace, it removes the possibility of finding false FICs (Kim et al., 2006). By

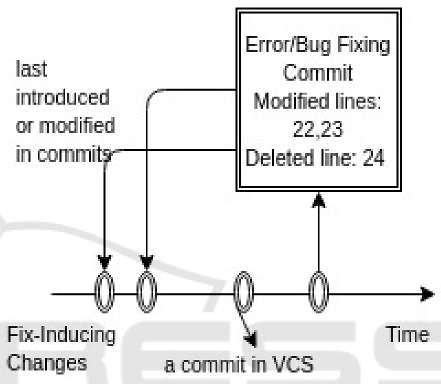

Error/Bug Fixing Commit Modified lines: 22,23 Deleted line: 24
last introduced or modified in commits
Fix-Inducing Changes
a commit in VCS
Time

Figure 1: Identifying Fix-Inducing Changes.

tracking the origin of those lines containing erroneous code in the parent commit of FCs to their last modification by using the Git blame command [1] (Fabók, 2012), FICs are found.

2.2 Analyzing Relationship

The change coupled relation is analyzed by considering a commit window rather than a time frame of month and year. This is because some software repositories might not have any commit in a month or year. Also, the first 20 commits of the repository are ignored from analyzing FICs due to the assumption that those are initial setups. So, analysis consists of commit window having commits between [20..120), [120..220) and so on. Figure 2 shows the entire process.

For analyzing change coupled relationship, firstly all java classes in the repository from 1st to the last commit are listed. Then, all obtained FICs are sorted according to the timeline. After that, the version history is traversed along with a commit window of 100

[1] git -c core.abbrev=40 blame -L(line number),+1 (FC_ParentHash)^– (filename)

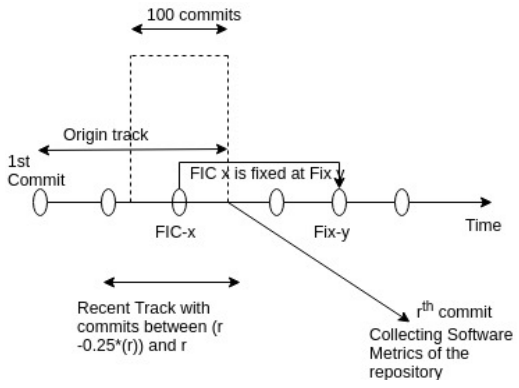

100 commits
Origin track
1st Commit
FIC x is fixed at Fix y
FIC-x
Fix-y
Time
Recent Track with commits between (r -0.25*(r)) and r
r^th commit
Collecting Software Metrics of the repository

Figure 2: Analysis process of recency consideration in CC relation.

commits, and for each FIC in that commit window, CC relation is considered along 2-time tracks. CC relation formed from the 1st commit up to the end of considered commit window is considered as Origin track as it considers time period from 1st commit. In order to get the recently formed relations, which will reflect recent relations, $\frac{1}{4}^{\text{th}}$ of origin distance previous to the commit window's last commit is considered as Recent track. This is done from intuition to ensure the importance of recency (Mori et al., 2015) which may be related to the erroneous codes. Consideration of $\frac{1}{4}^{\text{th}}$ of origin distance is done from following assumptions

- Recent month or year may have 0 commits. Therefore rather than tracking through time, it is best to use a suitable commit number.
- Related changes are mainly committed to together(Herzig and Zeller, 2013). So, any changes related to error will be near to the erroneous commit.
- As software evolves, different functionalities are added and the number of commits to maint=in them may increase. Therefore, as the distance from origin increases, the recent track will consider more commits.

Now, change coupling measures, as mentioned by (D'Ambros et al., 2009), are collected from these tracks for each class in FICs within the commit window. These are Number of Coupled Class (NOCC), Sum of Coupling (SOC), Exponentially Weighted Sum of Coupling (EWSOC) and Linearly Weighted Sum of Coupling (LWSOC). These are briefly described below. Here, NOCC measures the number of co-change occurrences that exceeds threshold n. SOC takes into account the total magnitude of occurrence in change coupling relationship. EWSOC and LWSOC take into account how apart change coupled relations are with respect to the considered time period.

Table 1: Repository Detail.

Repository Name	Total Commits	Analyzed Commits	Lines of Code
Google Guava (Doe, 2009)	4798	4520	768858

After that, software complexity metrics are collected from repository states at the last commits by moving commit window (e.g. 120, 220, 320 and so on). These complexity metrics include (Chidamber and Kemerer, 1994) Chidamber & Kemerer object-oriented metrics. These are Weighted Method Count (WMC), Depth of Inheritance Tree (DIT), Response For Class (RFC), Number Of Children (NOC), Coupling Between Objects (CBO) and Lack of Cohesion in Methods (LCOM). Besides it also includes some other object-oriented metrics which are Number Of Attributes (NOA), Number Of Methods (NOM), FANIN and FANOUT.

Change coupling and object-oriented metrics are collected or each of 100 commits. After that correlation and linear regression analysis are performed. Observing p and R square value, their relation and ability to predict and explain the number of actual fix-inducing changes of a class within the commit window period are analyzed. From those data, the importance of recent data about changed coupled relation can be understood.

3 EXPERIMENTATION

This experiment is carried out in Ubuntu 14.04 operating system with 8GB memory and Intel Core" i3-4130 CPU @ 3.40GHz $\times$ 4 processor. For performing this experiment, firstly, some popular Java repositories are searched and among them, 1 java repository is currently analyzed. The details of the selected repository are shown in Table 1.

In Guava repository, mostly Google developers maintain an open-source set of libraries. This project provides extensions to existing Java collections frameworks and other features like hashing, graph, range objects, and many others. This repository contains commits from June 2009 to the last commit found to be updated in August 2018.

In this repository, the experiment is carried out methodologically. Firstly, the fixing changes are found in the source repository by searching commit comments with keywords. After that, Diffj (Diffj, 2018) is used to identify changes between FC and its immediate parent commit while ignoring format and white space changes. The modified and changed lines are tracked to their last modification known as FICs using Git blame command. The number of FCs and FICs found in the repository are shown in table 2

Table 2: Total Fix-Inducing Commits and Fixing Commits of each repository.

Repository Name	Fixing Commits	Fix-Inducing Commits
Google Guava	597	486

After finding FICs, these are sorted according to their commit distance from the 1st commit of the repository. Then with a sliding window of 100 commits, commit history is traversed and coupling measures are collected from 2 track and object-oriented metrics from repository state at the last commit in sliding commit window. Then for each of these measures, the number of errors/FIC of a particular class found within the commit window is analyzed.

4 RESULT ANALYSIS

The data collected from the experiment is used to perform correlation and linear regression analysis. To find the relationship of change coupling measures and object-oriented metrics with the number of fix-inducing changes within the commit window, correlation analysis is done. It is observed that recent change coupling measures are more correlated to errors where co-change occurrences are small but poorly co-related where the number of co-change occurrences is large. Besides recent co-change metrics are more explanatory in predicting the number of future bugs rather than the co-change from the origin.

4.1 Correlation Analysis

Here, the correlation between coupling measures and errors introduced into the system, i.e. FICs within the commit range, is analyzed. From this analysis, following questions can be addressed.

1. Is there any correlation between FICs and change coupling measures?
2. Is there any difference between considering change coupling measures for the recent time period to that from the 1st commit?

To answer those questions Spearman correlation analysis is performed. In Figure 3, Spearman correlation is indicated on the y-axis and x-axis represents the threshold (i.e. n to get those n coupled classes with respect to considered class) used for the computation of change couplings metrics. After that plotting the points on the graph, these are connected to understand their behavior.

In Figure 3a, Spearman correlation between change coupling and object-oriented metrics of class total FICs found in the commit window period are shown. All metrics are represented with their name, except recent track metrics are named with a prefix ”r_”. The correlation analysis of object-oriented metrics gives a vague idea whereas recent track shows moderate relation during early phase or around 14-15 co-change occurrences of a class with other classes.

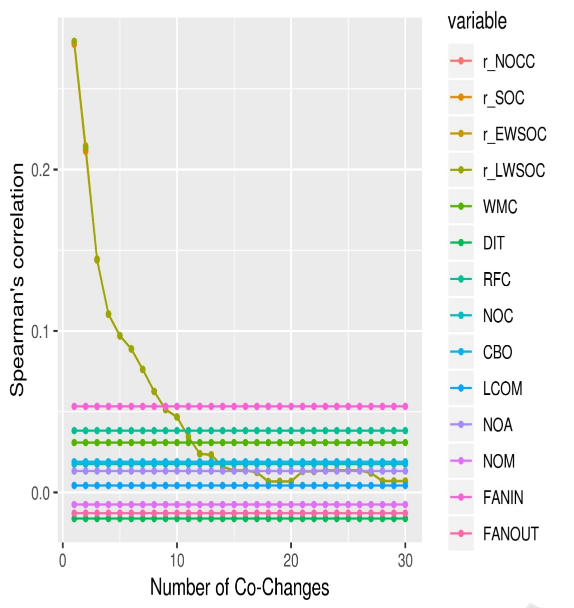
In Figure 3b, Spearman correlation between recent and origin track coupling measures with total FICs in the commit window period are shown. The correlation analysis of origin track gives the idea that change prone classes have more correlation with FICs. Since, those classes co-changing in a huge amount like 14-15 may not be present in recent track, so recent track badly correlates with FICs after 14-15 co-change occurrence of a class. However, in the early phase of co-change occurrences of a class, i.e. 1-15 recent track shows that those have more relation with errors.

Now, with respect to question 1, it can be said there is a correlation between FIC of a period with its change coupling measures. Although, it may appear moderate or weak for various reasons like Guava being a library project, considering the number of errors within commit window of 100 for prediction or the analyzed history of guava is short (i.e 4520 commits). So, it needs further analysis by changing the number of commits and exploring different projects. And for question 2, recent track coupling measures show a moderate correlation in cases where a class co-changed small number of times but weak correlation with respect to change prone classes which co-changed more than 15 changes in guava repository. This means that change prone classes like those which changed more than 15 times have been are very unlikely to produce bugs due to its co-changing relations. This indicates that newly formed relations with smaller co-change occurrences are mainly responsible for bugs.

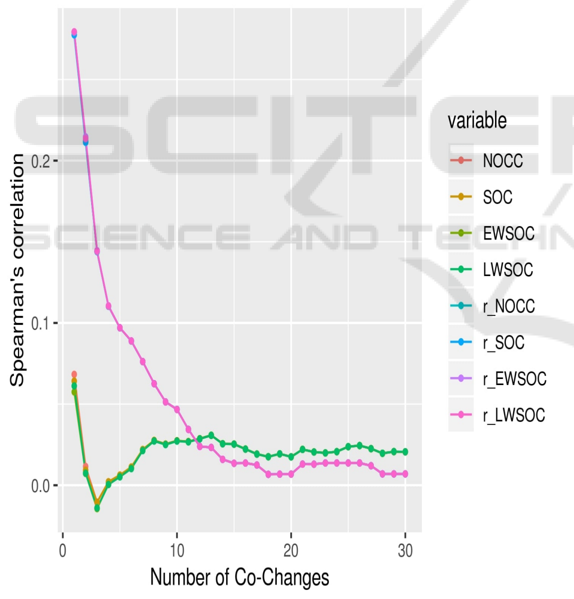
4.2 Linear Regression Analysis

After finding correlation between number of FICs with change coupling metrics, here, two questions are addressed. These are

1. Does use of change coupling increase explanatory power for predicting FICs or erroneous changes based on software metrics?



(a) Spearman correlation analysis between recent track coupling measures and metrics



(b) Spearman correlation analysis between recent track and origin coupling measures

Figure 3: Spearman Correlation analysis showing a correlation between recent, origin track change coupling measures and metrics with FIC.

2. Does use of recent track change coupling measures improve the model further?

For this, linear regression models are made. Here the independent variables are change coupling measures in origin and recent track and the object-oriented software metrics as described in the methodology section. The dependent variable is the number of errors or FICs found within the commit window considered. In Figure 4, the obtained results of regression analysis are described. Here, the x-axis represents the threshold (i.e. n in Figure 4 to get the change coupling measures as considered by in D'Ambros et al proposed change couple metrics) and the y-axis represents the R^2 found in regression analysis. After that plotted points are connected. In Figure 4, all means every metrics and change coupling measures are considered. CC represents only change coupled relations, CCAR means change coupled measures with recent and origin both track, CCOR means only recent change coupled measures and Metrics means the considered object-oriented metrics. In regression analysis, p-values help to determine whether the observed relationships also exist in the larger population. All p values for the regression models, for predicting errors when the number of co-change occurrences is from 1 to 30, are found significant i.e. $p < 0.01$. In case of independent variables, all 4 change couple metrics proposed by (D'Ambros et al., 2009) are found significant in all cases $p < 0.01$. In case of recent track, most of them are found significant in all cases with an exception in a small number of cases.

Based on R^2 from Figure 4, which explains the variability of the response data around the mean of regression line, it can be said that recent change coupling measures improve the explanatory power of existing models with object-oriented metrics. This answers question 1. Question 2 is also answered from the analysis it is clear that recent information of change coupled relations can further improve the explanatory power of prediction model for predicting future FICs. It is also noticed here that R^2 is very small, this may be either due to the project's nature or

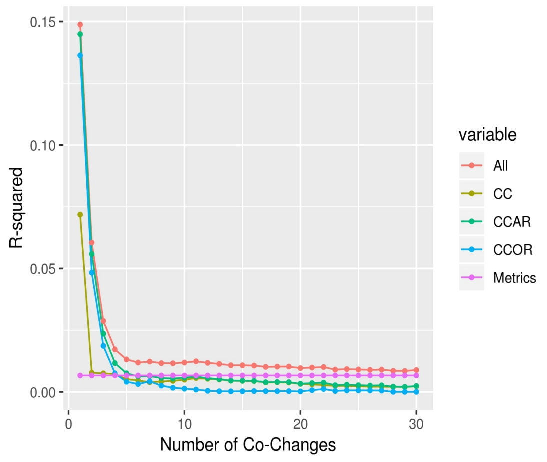


Figure 4: Linear Regression analysis for Guava Repository.

that of considered FICs within commit window of 100 commits. This requires further analysis by changing the size of the commit window and investigating other projects.

5 THREATS TO VALIDITY

The construct validity of the work is threatened by the following facts. Firstly, only commit messages are searched for fix-inducing changes without linking it with the bug repository. Linking with bug repository has a problem when bug ids are not found in commit messages. Therefore it might not fairly represent all bugs (Bird et al., 2009). Besides, the main objective of this study is to analyze, in the evolution of software, the relation between CC classes with errors at FIC. Secondly, varying commit behavior like changing multiple classes not related to fix in an FC might lead to wrong FICs. This is mitigated by considering a large dataset. This will be done in the future with repositories containing more FIC commits. Thirdly, all fixes are taken into account but these all may not represent corrective maintenance (Antoniol et al., 2008). However, in the considered project, only bug fixes are found by going through 10 random FCs. It is found that those FCs are indeed intended for corrective maintenance.

Although this study considered only Java classes. The main reason behind this limitation is Diffj which finds the modified and deleted lines between two versions of a file. In order to make it language independent, ANTLR grammars may be used but those are not as efficient as abstract syntax tree made for each language. So, the future extension may contain an elaborate analysis of different types of files and projects. Further, evolution will be analyzed with respect to different confidence level will be considered to get more insight.

6 RELATED WORKS

Although a considerable number of works are visible in both the fields of CC and Bug/Error, works combining those two are rare. All of those tried to address different issues with problems of a particular field. Among these works, some of the worth mentioning works are mentioned below.

To analyze the importance of change coupling, it is found that structural dependencies are not enough to explain the evolvability of Java software (Geipel and Schweitzer, 2012). In the evolution of the software system, historical data about co-changing classes can be used like ROSE (Zimmermann et al., 2005) prototype to predict further changes. Further, the Bayesian network can be used for change prediction(Zhou et al., 2008). Based on how changes are done in the source code....., these can be classified according to tree edit operations in AST (Fluri and Gall,). By taking into account inheritance, polymorphism, and dynamic binding, there is an operational definition of dynamic coupling measures (Arisholm et al., 2004). Continuous co-changes may indicate flawed design or rigid coding. To validate this claim, it is found that frequently changing software parts or change couplings may be candidates for refactoring (Ratzinger et al., 2005).

By using commit comments in the source repositories, fixing changes and fix-inducing changes are identified by (Sliwerski et al., 2005). However, due to considering format change and comments, it increases the number of false fix-inducing changes which is later on addressed by (Kim et al., 2006). Information from fixing change and fix introducing changes in version control commits and changed codes can be used for bug prediction (Nucci et al., 2018) (Shivaji et al., 2013), localization (Wen et al., 2016) as well as to to identify affected parts (Misirli et al., 2016). Further, to analyze whether any change can be predicted to introduce a bug, (Aversano et al., 2007) used software changes as elements of n-dimensional space. Similar work is done by (Shivaji et al., 2013) by reducing the number of features. Besides, fixing any erroneous change may not fix the bug, to address this issue, research of (Yin et al., 2011) provided useful guidelines. To investigate the characteristics of change types in bug fixing code, (Zhao et al., 2017) classified the change types into 5 categories.

The relationship between change coupling and software defect is first brought up by (D'Ambros et al., 2009). It showed that frequently changing software artifacts has a strong correlation with software bugs. For finding the relationship, this considered total bugs of a class in bug repository with collected metrics from repositories considering all transactions. Recently, similar work is performed and it is found that there is a positive correlation between change coupling and defect measures (Kirbas et al., 2017). However all of these considered bugs from bug repositories. All of these motivated this work to look at the evolutionary relationship between change coupling and FICs and to find their relationship.

7 CONCLUSIONS

This paper tries to find out whether or not change coupling has a relationship with the origin of bugs known as fix-inducing changes from source repositories rather than considering bugs from bug repositories like (D'Ambros et al., 2009). For this, change coupling measures and FICs are collected from the source repository of Google Guava. This is done by considering a commit window of 100 commits and traversing the history using the version control system git. To analyze their relationship, both correlation and regression analysis is performed. It is seen from the obtained results that recent change coupling measures are more correlated with errors rather than considering total relation. By considering the explanatory power for predicting erroneous changes within the commit window, the use of recent change coupling measures seemed to improve the model as it represents the recent interactions.

The main achievement of this work is to consider the relationship between FICs a software defect with change coupling measures. This analysis is based on Google Guava repository and by considering FICs within 100 commits of the commit window, so the obtained results seem to show moderate and weaker relation. To address this issue more repositories will be explored and commit window of different size will be taken in the future to strengthen the claim. Recently, from analysis, it is observed that total fix-inducing changes obtained from all commits under observation correlates strongly with the change coupling measures used in this study.

Various works are possible from the relationship between change coupling and fix-inducing changes. These may include prediction, automatic bug fixing, analyzing change impact and others. However, in future works focus will be given on finding how this relationship is influenced by considering different confidence level and size of commits. Besides, this work is based on java projects and in future projects of other programming languages will be analyzed to find the actual difference.

ACKNOWLEDGMENT

This research is supported by the fellowship from ICT Division, Ministry of Posts, Telecommunications and Information Technology, Bangladesh. No-56.00.0000.028.33.002.19.3; Dated 09.01.2019.

REFERENCES

Antoniol, G., Ayari, K., Di Penta, M., Khomh, F., and Guéhéneuc, Y.-G. (2008). Is it a bug or an enhancement?: a text-based approach to classify change requests. In *Proceedings of the 2008 conference of the center for advanced studies on collaborative research: meeting of minds*, page 23. ACM.

Antoniol, G., Rollo, V. F., and Venturi, G. (2005). Detecting groups of co-changing files in CVS repositories. In *8th International Workshop on Principles of Software Evolution (IWPSE 2005), 5-7 September 2005, Lisbon, Portugal*, pages 23–32.

Arisholm, E., Briand, L. C., and Foyen, A. (2004). Dynamic coupling measurement for object-oriented software. *IEEE Transactions on software engineering*, 30(8):491–506.

Aversano, L., Cerulo, L., and Del Grosso, C. (2007). Learning from bug-introducing changes to prevent fault prone code. In *Ninth international workshop on Principles of software evolution: in conjunction with the 6th ESEC/FSE joint meeting*, pages 19–26. ACM.

Bird, C., Bachmann, A., Aune, E., Duffy, J., Bernstein, A., Filkov, V., and Devanbu, P. (2009). Fair and balanced?: bias in bug-fix datasets. In *Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, pages 121–130. ACM.

Chidamber, S. R. and Kemerer, C. F. (1994). A metrics suite for object oriented design. *IEEE Transactions on software engineering*, 20(6):476–493.

D'Ambros, M., Lanza, M., and Robbes, R. (2009). On the relationship between change coupling and software defects. In *16th Working Conference on Reverse Engineering, WCRE 2009, 13-16 October 2009, Lille, France*, pages 135–144.

Diffj (2018). https://github.com/jpace/diffj.

Doe, R. (2009). Guava repository from github. https://github.com/google/guava.

Fabók, Z. (2012). Learn more about the history of a line with git blame. https://zsoltfabok.com/blog/2012/02/git-blame-line-history/.

Fluri, B. and Gall, H. C. Classifying change types for qualifying change couplings. In *14th International Conference on Program Comprehension (ICPC 2006), pages = 35–45, year = 2006, bibsource = dblp computer science bibliography, https://dblp.org*.

Fukushima, T., Kamei, Y., McIntosh, S., Yamashita, K., and Ubayashi, N. (2014). An empirical study of just-in-time defect prediction using cross-project models. In *11th Working Conference on Mining Software Repositories, MSR 2014, Proceedings, May 31 - June 1, 2014, Hyderabad, India*, pages 172–181.

Geipel, M. M. and Schweitzer, F. (2012). The link between dependency and cochange: Empirical evidence. *IEEE Transactions on Software Engineering*, 38(6):1432–1444.

Herzig, K. and Zeller, A. (2013). The impact of tangled code changes. In *Proceedings of the 10th Working*

Conference on Mining Software Repositories, pages 121–130. IEEE Press.

Kim, S., Jr., E. J. W., and Zhang, Y. (2008). Classifying software changes: Clean or buggy? *IEEE Trans. Software Eng.*, 34(2):181–196.

Kim, S., Zimmermann, T., Pan, K., and Jr., E. J. W. (2006). Automatic identification of bug-introducing changes. In *21st IEEE/ACM International Conference on Automated Software Engineering (ASE 2006), 18-22 September 2006, Tokyo, Japan*, pages 81–90.

Kirbas, S., Caglayan, B., Hall, T., Counsell, S., Bowes, D., Sen, A., and Bener, A. (2017). The relationship between evolutionary coupling and defects in large industrial software. *Journal of Software: Evolution and Process*, 29(4):e1842.

Levin, S. and Yehudai, A. (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, PROMISE 2017, Toronto, Canada, November 8, 2017*, pages 97–106.

Menzies, T., Greenwald, J., and Frank, A. (2007). Data mining static code attributes to learn defect predictors. *IEEE Trans. Software Eng.*, 33(1):2–13.

Misirli, A. T., Shihab, E., and Kamei, Y. (2016). Studying high impact fix-inducing changes. *Empirical Software Engineering*, 21(2):605–641.

Mori, T., Hagward, A., and Kobayashi, T. (2015). Effects of recency and commits aggregation on change guide method based on change history analysis. In *Proceedings of the Tenth International Conference on Software Engineering Advances*, pages 96–101.

Nucci, D. D., Palomba, F., Rosa, G. D., Bavota, G., Oliveto, R., and Lucia, A. D. (2018). A developer centered bug prediction model. *IEEE Trans. Software Eng.*, 44(1):5–24.

Palomba, F., Bavota, G., Di Penta, M., Oliveto, R., De Lucia, A., and Poshyvanyk, D. (2013). Detecting bad smells in source code using change history information. In *Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering*, pages 268–278. IEEE Press.

Ratzinger, J., Fischer, M., and Gall, H. C. (2005). Improving evolvability through refactoring. *ACM SIGSOFT Software Engineering Notes*, 30(4):1–5.

Shivaji, S., Jr., E. J. W., Akella, R., and Kim, S. (2013). Reducing features to improve code change-based bug prediction. *IEEE Trans. Software Eng.*, 39(4):552–569.

Sliwerski, J., Zimmermann, T., and Zeller, A. (2005). When do changes induce fixes? *ACM SIGSOFT Software Engineering Notes*, 30(4):1–5.

Wen, M., Wu, R., and Cheung, S. (2016). Locus: locating bugs from software changes. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, ASE 2016, Singapore, September 3-7, 2016*, pages 262–273.

Yin, Z., Yuan, D., Zhou, Y., Pasupathy, S., and Bairavasundaram, L. (2011). How do fixes become bugs? In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, pages 26–36. ACM.

Zhao, Y., Leung, H., Yang, Y., Zhou, Y., and Xu, B. (2017). Towards an understanding of change types in bug fixing code. *Information and software technology*, 86:37–53.

Zhou, Y., Würsch, M., Giger, E., Gall, H. C., and Lu, J. (2008). A bayesian network based approach for change coupling prediction. In *WCRE 2008, Proceedings of the 15th Working Conference on Reverse Engineering, Antwerp, Belgium, October 15-18, 2008*, pages 27–36.

Zimmermann, T., Weißgerber, P., Diehl, S., and Zeller, A. (2005). Mining version histories to guide software changes. *IEEE Trans. Software Eng.*, 31(6):429–445.