2019 International Conference on Electrical, Computer and Communication Engineering (ECCE), 7-9 February, 2019

Analyzing Corrective Maintenance using Change Coupled Clusters at Fix-inducing Changes

Ali Zafar Sadiq*, Ahmedul Kabir†, Pritom Saha Akash‡ and Md. Jubair Ibna Mostafa§
Institute of Information Technology, University of Dhaka
Email: *zafarsadiq120@gmail.com, †kabir@iit.du.ac.bd, ‡bsse0604@iit.du.ac.bd, §bsse0614@iit.du.ac.bd

***Abstract*—Due to changing nature of change coupled relation, corrective maintenance of any software system is costly. Existing works on change coupled relation analyzed different areas like change prediction or impact. But, none of those tried to utilize historical information to analyze system maintenance. In this paper, a methodology is proposed by using change coupled relations at fix introducing changes to analyze the corrective maintenance history of the system. This analysis helps to understand which part of the system are prone to erroneous changes.**

***Index Terms*—Bug inducing change, LDA, Software Quality Assurance, Software Change, Software Maintenance, Software Coupling**

I. Introduction

Frequently changing software artifacts have an inherent relation known as change coupling. Because of this relation, changing a software artifact requires changing others change coupled with it. A cluster of artifacts is formed from the relation of change coupling. This cluster of artifacts will be affected by any change in the artifact with which other artifacts are forming change coupled relation. So, any error introduced in the artifact will require changing affected parts of other artifacts forming cluster.

Changes in a software can introduce errors, flaws or failures into the system. Analyzing the impact of change and enforcing available resources efficiently are seen in various Software Quality Assurance works [1] [2]. Unwanted errors and bugs can be introduced through improper or careless changes. These errors and bugs are results of the contents in lines of changed or added codes [3]. Various properties of these changes like affected files, time of day, developer experience and others are now in the active field of study [4] [5] [6] [7]. In Java programming language, each of the changed files represents a public class. And a change in any class leads to change in other classes due to change coupling [8]. Maintenance cost for any Fix-Inducing Change (FIC) in the CC classes is high [9]. If corrective maintenance does not fix the errors, then for continuous errors the amount of cost and effort that will be used are useless. So reappearance of errors certain part of the system indicates that part is error-prone and needs redesigning and reorganizing. Unless these issues are dealt properly, continuous corrective maintenance cost would exceed the cost of benefit obtained.

Existing works either worked with bugs or change coupling. To analyze the evolution of software system, D'Ambros et. al proposed Evolution Radar [10]. There, developers can move through time to analyze the evolution of software system using change coupled relation. But it did not provide any information regarding how this change coupled relations appearing (i.e frequently or sparsely) in the corrective maintenance history of the system. Frequent appearance of change coupled relations in a fix-inducing changes may represent faulty design. So, in this paper a methodology is proposed to analyze the system for finding the error-prone parts from change coupled relation and to the best of authors' knowledge, this analysis was not done previously.

II. Methodology

For analysis, the version control system, git is used to retrieve historical data. Using git, firstly the fix-inducing changes are identified by tracking last modification of changed or deleted lines in fixing changes (FC). Then using a commit frame of 100 commits and analyzing the FICs in the commit window, from the 1st commit to present, the change coupled relation is analyzed. Then classes, variable and method names of change coupled relations within each commit frame are collected. Lastly, by performing LDA on available architectural information, topics are generated. Then using the information from change coupled relation of a certain commit window, the affected part of the system is identified using inference. The entire process is described below.

A. Identifying Fix-Inducing Changes (FIC):

The Fix-inducing Changes (FIC) are the changes that introduce errors into the system. These are also known as bug introducing changes. The entire process of finding FIC is shown in Figure 1. Firstly, Fixing Changes (FC) or commits are identified by examining commit history with keywords "Fix", "Bug", "Patch" or their past and "-ing" form. Also, hashtags containing any number to represent bug or patch are also considered. In the parent commit of FC, the erroneous codes can be found. The difference between a class in parent commit of FC and FC gives the changed or deleted lines that fixed an error. Those line numbers in parent commit of FC represents the erroneous code.

In parent commit of FC, those erroneous lines are tracked to their last modification like rule-based SZZ algorithm [11]. However, in this study only available commit comments from git are used to identify FC without linking it to the bug repository. The reasons are explained in the threats to validity section.

978-1-5386-9111-3/19/$31.00 © 2019 IEEE

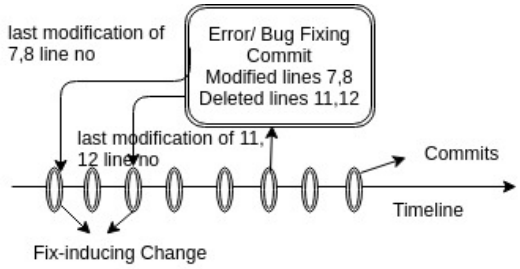
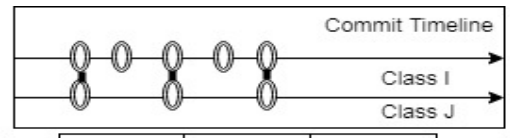


Figure 1: Identifying Fix-Inducing Changes.

B. Finding Change Coupled (CC) classes:

Co-change matrix is used to find CC classes [12]. It is a symmetric matrix where any cell [A, B] and A≠B, keeps track of how many times artifact A and artifact B changed together. As for the case of cell [A, A] keeps track of how many times artifact A changed. Using appropriate support and confidence, the change coupled relation can be found among Java classes. Support represents how many times a class changed and confidence represents the likelihood, of 2 coupled artifacts, if one artifact changes whether another artifact is going to change or not. In co-change matrix [A, A] represents support and confidence is represented by following Equation 1

$$Confidence(A \rightarrow B) = \frac{support(A \rightarrow B)}{support(A)} = \frac{support(A \cup B)}{support(A)} \quad (1)$$



	Class I	Class j
Class I	5	3
Class J	3	3

Figure 2: Example of Co-Change Matrix and Commit Timeline

In order to construct the co-change matrix, all Java files within the period of consideration are taken. Then for each Java class, the change coupled relation with others are found based on considered support and confidence. Those other Java classes that have change coupled relation with a Java class is considered to form a cluster. The entire process is shown in Figure 2.

C. Topics from Gibbs LDA on Architectural Information:

The architectural information of the repositories is collected from authenticated sources. This information gives an overview of the entire software system. By analyzing topics generated from Latent Dirichlet Allocation, basic idea about overall systems components can be inferred. However, directly estimating parameters for LDA and maximizing the whole data collection likelihood is inflexible. To address this issue Variational Methods [13], Expectation propagation [14], and Gibbs Sampling [15] is used to approximate estimation. In Markov-chain Monte Carlo (MCMC) [16], Gibbs Sampling represents a special case to produce approximate inference in high-dimensional models such as LDA [17]. In this work, Gibbs LDA is used.

D. Commit History Analysis:

CC classes were found by using different support and confidence in different works. Zimmermann et al used the support greater than 1 and confidence level 0.5 in their work [18]. However, Bavota et al considered elements that co-changing in at least 2% of the commits along with a confidence level of 0.8 [19]. Since, 0.8 confidence is high, in this paper 0.7 confidence level is used along with support 1 or more is considered.

For the analysis, firstly the commit history is traversed to find fix-inducing changes. From there the FICs are obtained. Then from the beginning to the end of commit history, first 20 commits were omitted for the files to form change relations among themselves properly. After that from 20 with a commit window of 100 commits, i.e 20-120, 120-220 and so on is considered. For each of this commit window, the FICs that are present are analyzed. Figure 3 shows the entire process.

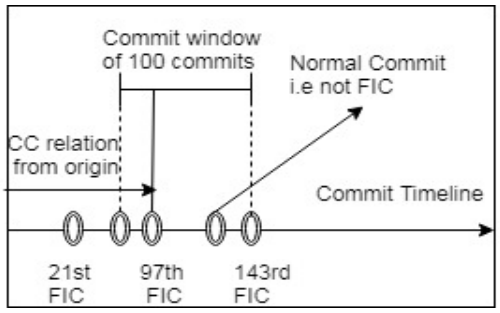


Figure 3: Analysis process of Evolution of Change Coupling relation

At first, all classes in the repository from beginning to the present are listed. Then, all FICs are sorted according to the timeline. Then the version history is traversed along with a commit window of 100 commits, and for each FIC in that commit window, the CC classes are considered. The CC relation formed from the 1st commit up to the FIC is considered. To find the corrective maintenance information of the system, information like class, method and variable names of CC classes of different CC clusters in FICs are collected. These are used in the inference process where topics were found by performing LDA on the architectural information the repositories. Analyzing the commit window and frequent appearance of the same topic means obtained topic in the

architectural information are vulnerable to errors and causing costly corrective maintenance,

E. Topics inference from available CC class information:

In order to infer topics from collected CC classes information, the method [20] proposed by Phan et al is used. According to their proposed method, information of class, method and variable names of the CC cluster is used as a new document whereas the universal data-set is the architectural information of the system. Topic inference also requires Gibb Sampling, however, the number of iterations required is around 20-30 [20]. In this report, the number of iterations used is 30. For a document or CC classes information about one cluster, the topic was inferred by taking the number of times topic appeared. This is because the name of a topic appears once or several times depending on the probability of that topic in [20]. So for a commit window of 100 commits, all CC class clusters are analyzed and the maximum percentage of a topic was noted to be dominant for that period. In this way, the maintenance history of a repository was analyzed.

III. 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. To conduct the experiment, popular Java repositories are searched and among them, 2 java repositories are selected for the study. These are as follows:

Repository Name	Source	Total Commits	Commits Analyzed	Number of Java classes	Lines of Code
Google Guava [21]	Github	4798	4000	3170	768858
Apache Tomcat 8.5.x [22]	Github	18778	7320	2316	561593

TABLE I: General Description of the repositories.

Guava is an open-source set of libraries maintained mostly by Google developers. It provides extensions to Java collections framework and other features like hashing, graph, range objects etc. This repository contains commits from June 2009 to the last commit updated a few days ago. In case of Apache Tomcat. it is a Java-based opensource servelet container. In this repository, commit history began in March 2006 and the last one was made in 2018.

In these repositories, the experiment was carried out methodologically. Firstly, the fixing changes are found in the source repositories. After that, cosmetic changes are eliminated and those lines that are modified or deleted in the FCs are identified by using diffj [23] tool. Then those identified lines are tracked to find the FIC commits where the last modifications are made. The number of FCs and FICs found in the repositories are shown in Table II

After finding the CC classes in FIC, gradual development of the Change Couple relation throughout the lifespan of a project was analyzed. For this reason, information about classes, methods and variables were collected at 100^{th} commit of the considered commit window in the history time line. Then from sorted FICs, according to their commit time, the change coupled relations were noted in each of the FICs within each commit window of 100 commits.

Repository Name	Fixing Commits	Fix-Inducing Commits
Google Guava	597	486
Apache Tomcat 8.5.x	5901	3984

TABLE II: Total Fix-Inducing Commits and Fixing Commits of each repository

From architectural information, collected from authenticated sources like in GitHub or the main website, topics were generated by using Gibbs LDA [24]. Although the number of topics in the LDA is preconfigured with the number of components found in architectural information, their description in the topics overlaps to some extent. After that, information related to different Change Couple Cluster classes were obtained and Gibbs LDA inference method was applied to identify the probability of topics in the cluster. From output maximum appearance of topics were selected as the main topic of the cluster. The average percentage of different topics in the cluster was analyzed periodically to see which topic appeared most.

IV. Result Analysis

The results are obtained by conducting an experiment on first 4000 commits of Google guava and 7320 commits of Apache Tomcat 8.5.x. From the experiment, the topics generated from LDA are manually labeled.

Topics labeled Manually	Topic words
Connector	lang,class, default, get, true, connector, path, java, specified, method, resources
Context	context, string, find, filter, tomcat, when, Catalina, cookie, ssl, description, otherwise, role, must, setting, engine
Security	value, attribute, context, host, or, remove, http, which, any, servlet, void, can, flag, no, configured, security, copied, nio, apache, standard
Server	application, that, public, parameters, java, add, session, from, void, file, false, element, servlet, the, resource, thread, mapping, loader
Hosts	set, is, to, this, used, get, returns, xml, with, are, listener, it, as, in, config, default, new, only
Service	name, by, web, specified, interface, use, an, request, on, boolean, base, this, you, server, deploy, socket, have, may

TABLE III: Tomcat 8.5.x Topics generated form LDA using Gibbs sampling and their manual labeling

The number of topics for LDA is selected depending on the architectural information of the system. In Figure 4a the

Topics labeled Manually	Topic words
Reflection	to, all, map, be, collection, new, using, domain, its, can, number, nodes, put, jdk, result, file, directly, static
Primitives	which, class, multiset, of, may, by, long, prim, instance, them, linked, from, output, where, override, enum, path, operation, either, uv
Hashing	type, cache, immutable, hash, int, event, returns, return, types, elements, map, byte, no, closed, utilities, most
Basic Utilities	get, function, some, element, at, same, ordering, need, want, length, order, builder, instead, mode, equal, just, see, value, default, check
Ranges	entries, what, used, time, bytes, cases, discrete, utility, events, include, undirected, callable, was, multiset
I/O	service, table, above, files, useful, change, makes, details, work, query, custom, strings, utf, api, writing, application
EventBus	from, set, will, have, guava, node, provides, add, bus, with, entry, out, long, after
Concurrency	be, stream, open, throw, even,propagate, collections, fluent, provide, behavior, optional, throwables, thread, listeners, filter, private, cause
Caches	of, values, map, e, one, should, contains, iterator, size, listener, mutable, support, sink, rounding, ranges, void, allows, loading, own, right
String	exception, null, future, of, source, objects, like, matcher, each, equals, two, index, implementation, foo, its, handler
Networking	iterable, edge, edges, does, view, classes, suffix, builder, forwarding, each, bi, characters, method, always, com, less, particular, connected
Math	in, for, that, as, if, string, list, the, with, range, value, integer, set, key, but, method, by, when, returns
FunctionalIdom	example, multimap, char, specified, token, than, c, operations, common, name, public, interface, big, first, throws, double, compare
Concurrency	or, sorted, empty, multimap, up, input, checked, there, true, then, corresponding, sequence, internet, supports, read, next, intersection, been, concurrent
Graphs	code, object, more, other, implementations, unsigned, create, keys, only, write, must, while, remove, count, both

TABLE IV: Guava Topics generated form LDA using Gibbs sampling and their manual labeling

x axis represents the commit history and y axis represents the probability of topic inferred from CC clusters at FICs within commit frame of 100 commits along x axis. From the table III and Figure 4a, it can be clearly stated that the security component of the Apache Tomcat in most of the cases of commit window is mostly affected. It is natural as the server requires to satisfy the customers however customers can exploit the vulnerabilities unless there are upgrade and redesign. The same happened for Tomcat repository. In the end, it can also be noted that the number of classes forming the CC cluster with topics related to the server has increased. Through this, an overview of the total maintenance activities as well as vulnerable parts of Apache Tomcat can be known.

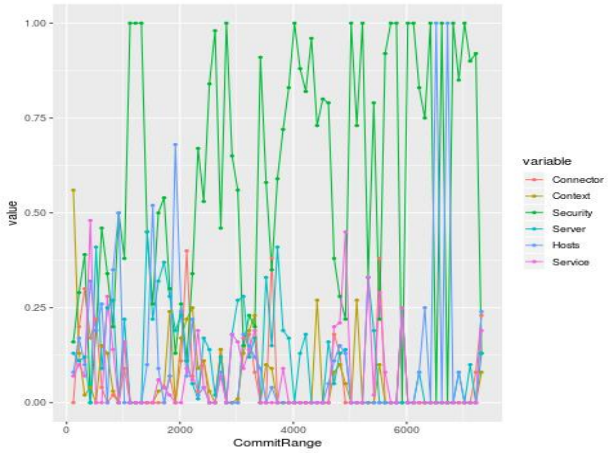
In case of Guava, from table IV and Figure 4b, it can be cleverly seen that different parts of the system are affected different times. Since the Google Guava is library extension for existing Java libraries and is maintained purely by Google developers, so it is expected that this repository will randomly correct and update different libraries extensions. However, it can be seen that among these affected parts mostly corrective maintenance is focused on I/0 and ranges. Since guava is a library is its expected that to give faster output changes are needed to be done in I/O parts. Again ranges on query and interval also need faster and errorless output. So Corrective Maintenance is seen to be more focused on them.

V. Threats to Validity

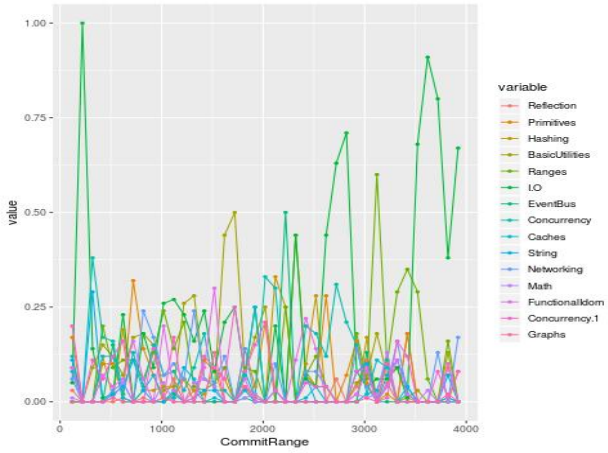
Among the construct threats, firstly, only commit messages are used to find fixing changes and bug repository is not linked. Linking bugs with bug repository has its own problem that some bugs may not have any id or report. So this creates an unfair situation [25]. Besides, the main objective of this study is to analyze the corrective maintenance history. Secondly, the commits made can have varying behavior. Unrelated classes in bug fix can lead to wrong fix-inducing changes. This is mitigated by considering large dataset consisting of 100 commits. Thirdly, all fixes may not be actually corrective maintenance [26]. However, in the two considered projects, only bug fixes are found by going through 10 random FCs in each project, and the main objective of those FCs were corrective maintenance, The replication validity is limited by topics with top words generated from LDA and manual labeling of the topics. Again to perfectly analyze the maintenance history one can go through the cluster of CC classes that is produced as a byproduct. In this study, only Java classes are considered, so the future extension will perform an elaborate analysis of different types of files and projects. Again, corrective maintenance will be analyzed with respect to different confidence level rather than confidence level at 0.7 to get more insight.

VI. Related Work

From the work of Geipel and Schweitzer, it is understood that structural dependencies do not explain the total evolvability of Java software [27]. Using historical information about co-changing classes in software evolution, Zimmerman et al proposed ROSE [9] prototype. ROSE predicts further changes depending on CC relation. Similarly using Bayesian Network, Zhou et al worked on prediction [28]. Again, Fluri et al classified changes according to tree edit operations in AST [29] based on how a change is made and Arisholm et al took into account inheritance, polymorphism, and dynamic binding and provided an operational definition of dynamic coupling measures [30]. Frequent co changes may indicate faulty design



(a) Corrective Maintenance History for Apache Tomcat



(b) Corrective Maintenance History for Google Guava Repository

Figure 4: Corrective Maintenance History

or hard coding. So, Ratzinger et al showed those changing software parts may be refactoring candidates [31].

Frequent changes, errors may cause bugs to be introduced in the system. The correlation between change and defect was shown by Marco DAmbros et al [32]. Sliwerski et al mentioned these erroneous changes, that would require a fix, later on, as fix-inducing changes [11]. To improve the findings of Sliwerski et al, a framework is proposed recently for evaluating the results of SZZ by Daniel Alencar da Costa et al [33]. Additionally, information from FIC in VCS and changed codes can be used for bug prediction [34] [35], localization [36] as well as to to identify affected parts [37].

A considerable amount of work can be seen in both change coupling and Bug/Errors, but very few focused on the evolution of software. To analyze the evolution, Beck et al showed that evolutionary data or co-changing classes have a positive impact on clustering [38]. Recently, Kirbas et al showed a positive correlation between EC and defect measures [39]. All of these focuses on either change coupling or evolution but did not combine them to obtain both change and error prone parts of the software system from the historical information.

VII. Conclusion

In this work, a methodology is proposed to analyze the corrective maintenance history of the software system using CC classes. By gaining insight in this information will allow the software developers to find which part of the software system is continuously responsible for change and producing bugs. Generated words in each topic can be related to the components of the software system. This will provide useful information about vulnerable change prone components that are responsible for creating bugs. These components can be further refactored and re-engineered for better performance.

References

[1] A. T. Misirli, E. Shihab, and Y. Kamei, "Studying high impact fix-inducing changes," *Empirical Software Engineering*, vol. 21, no. 2, pp. 605–641, 2016. [Online]. Available: https://doi.org/10.1007/s10664-015-9370-z

[2] G. Antoniol, V. F. Rollo, and G. Venturi, "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*, 2005, pp. 23–32. [Online]. Available: https://doi.org/10.1109/IWPSE.2005.11

[3] J. Sliwerski, T. Zimmermann, and A. Zeller, "When do changes induce fixes?" *ACM SIGSOFT Software Engineering Notes*, vol. 30, no. 4, pp. 1–5, 2005. [Online]. Available: http://doi.acm.org/10.1145/1082983.1083147

[4] S. Levin and A. Yehudai, "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*, 2017, pp. 97–106. [Online]. Available: http://doi.acm.org/10.1145/3127005.3127016

[5] S. Kim, E. J. W. Jr., and Y. Zhang, "Classifying software changes: Clean or buggy?" *IEEE Trans. Software Eng.*, vol. 34, no. 2, pp. 181–196, 2008. [Online]. Available: https://doi.org/10.1109/TSE.2007.70773

[6] T. Menzies, J. Greenwald, and A. Frank, "Data mining static code attributes to learn defect predictors," *IEEE Trans. Software Eng.*, vol. 33, no. 1, pp. 2–13, 2007. [Online]. Available: https://doi.org/10.1109/TSE.2007.256941

[7] T. Fukushima, Y. Kamei, S. McIntosh, K. Yamashita, and N. Ubayashi, "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*, 2014, pp. 172–181. [Online]. Available: http://doi.acm.org/10.1145/2597073.2597075

[8] R. Wang, R. Huang, and B. Qu, "Network-based analysis of software change propagation," *The Scientific World Journal*, vol. 2014, 2014.

[9] T. Zimmermann, P. Weißgerber, S. Diehl, and A. Zeller, "Mining version histories to guide software changes," *IEEE Trans. Software Eng.*, vol. 31, no. 6, pp. 429–445, 2005. [Online]. Available: https://doi.org/10.1109/TSE.2005.72

[10] M. D'Ambros, M. Lanza, and M. Lungu, "The evolution radar: Visualizing integrated logical coupling information," in *Proceedings of the 2006 international workshop on Mining software repositories*. ACM, 2006, pp. 26–32.

[11] J. Sliwerski, T. Zimmermann, and A. Zeller, "When do changes induce fixes?" *ACM SIGSOFT Software Engineering Notes*, vol. 30, no. 4, pp. 1–5, 2005. [Online]. Available: http://doi.acm.org/10.1145/1082983.1083147

[12] T. Menzies, L. L. Minku, and F. Peters, "The art and science of analyzing software data; quantitative methods," in *37th IEEE/ACM International Conference on Software Engineering, ICSE 2015,*

Florence, Italy, May 16-24, 2015, Volume 2, 2015, pp. 959–960. [Online]. Available: https://doi.org/10.1109/ICSE.2015.306

[13] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *Journal of machine Learning research*, vol. 3, no. Jan, pp. 993–1022, 2003.

[14] T. Minka and J. Lafferty, "Expectation-propagation for the generative aspect model," in *Proceedings of the Eighteenth conference on Uncertainty in artificial intelligence*. Morgan Kaufmann Publishers Inc., 2002, pp. 352–359.

[15] T. L. Griffiths and M. Steyvers, "Finding scientific topics," *Proceedings of the National academy of Sciences*, vol. 101, no. suppl 1, pp. 5228–5235, 2004.

[16] E. Gabrilovich and S. Markovitch, "Computing semantic relatedness using wikipedia-based explicit semantic analysis." in *IJcAI*, vol. 7, 2007, pp. 1606–1611.

[17] I. Bíró, J. Szabó, and A. A. Benczúr, "Latent dirichlet allocation in web spam filtering," in *Proceedings of the 4th international workshop on Adversarial information retrieval on the web*. ACM, 2008, pp. 29–32.

[18] T. Zimmermann, S. Diehl, and A. Zeller, "How history justifies system architecture (or not)," in *Software Evolution, 2003. Proceedings. Sixth International Workshop on Principles of*. IEEE, 2003, pp. 73–83.

[19] G. Bavota, B. Dit, R. Oliveto, M. Di Penta, D. Poshyvanyk, and A. De Lucia, "An empirical study on the developers' perception of software coupling," in *Proceedings of the 2013 International Conference on Software Engineering*. IEEE Press, 2013, pp. 692–701.

[20] X.-H. Phan, L.-M. Nguyen, and S. Horiguchi, "Learning to classify short and sparse text & web with hidden topics from large-scale data collections," in *Proceedings of the 17th international conference on World Wide Web*. ACM, 2008, pp. 91–100.

[21] Guava repository from github. [Online]. Available: https://github.com/google/guava

[22] Apache tomcat repository from github. [Online]. Available: https://github.com/apache/tomcat85

[23] Diffj. [Online]. Available: https://github.com/jpace/diffj

[24] "Gibbs LDA(Access date: 17-09-2018)," https://github.com/david2dai/gibbslda.

[25] C. Bird, A. Bachmann, E. Aune, J. Duffy, A. Bernstein, V. Filkov, and P. Devanbu, "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*. ACM, 2009, pp. 121–130.

[26] G. Antoniol, K. Ayari, M. Di Penta, F. Khomh, and Y.-G. Guéhéneuc, "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*. ACM, 2008, p. 23.

[27] M. M. Geipel and F. Schweitzer, "The link between dependency and cochange: Empirical evidence," *IEEE Transactions on Software Engineering*, vol. 38, no. 6, pp. 1432–1444, 2012.

[28] Y. Zhou, M. Würsch, E. Giger, H. C. Gall, and J. Lu, "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*, 2008, pp. 27–36. [Online]. Available: https://doi.org/10.1109/WCRE.2008.39

[29] B. Fluri and H. C. Gall, "Classifying change types for qualifying change couplings," in *14th International Conference on Program Comprehension (ICPC 2006), pages = 35–45, year = 2006, crossref = DBLP:conf/iwpc/2006, url = https://doi.org/10.1109/ICPC.2006.16, doi = 10.1109/ICPC.2006.16, timestamp = Mon, 22 May 2017 17:11:18 +0200, biburl = https://dblp.org/rec/bib/conf/iwpc/FluriG06, bibsource = dblp computer science bibliography, https://dblp.org*.

[30] E. Arisholm, L. C. Briand, and A. Foyen, "Dynamic coupling measurement for object-oriented software," *IEEE Transactions on software engineering*, vol. 30, no. 8, pp. 491–506, 2004.

[31] J. Ratzinger, M. Fischer, and H. C. Gall, "Improving evolvability through refactoring," *ACM SIGSOFT Software Engineering Notes*, vol. 30, no. 4, pp. 1–5, 2005. [Online]. Available: http://doi.acm.org/10.1145/1082983.1083155

[32] M. D'Ambros, M. Lanza, and R. Robbes, "On the relationship between change coupling and software defects," in *16th Working Conference on Reverse Engineering, WCRE 2009, 13-16 October 2009, Lille, France*, 2009, pp. 135–144. [Online]. Available: https://doi.org/10.1109/WCRE.2009.19

[33] D. A. da Costa, S. McIntosh, W. Shang, U. Kulesza, R. Coelho, and A. E. Hassan, "A framework for evaluating the results of the SZZ approach for identifying bug-introducing changes," *IEEE Trans. Software Eng.*, vol. 43, no. 7, pp. 641–657, 2017. [Online]. Available: https://doi.org/10.1109/TSE.2016.2616306

[34] D. D. Nucci, F. Palomba, G. D. Rosa, G. Bavota, R. Oliveto, and A. D. Lucia, "A developer centered bug prediction model," *IEEE Trans. Software Eng.*, vol. 44, no. 1, pp. 5–24, 2018. [Online]. Available: https://doi.org/10.1109/TSE.2017.2659747

[35] S. Shivaji, E. J. W. Jr., R. Akella, and S. Kim, "Reducing features to improve code change-based bug prediction," *IEEE Trans. Software Eng.*, vol. 39, no. 4, pp. 552–569, 2013. [Online]. Available: https://doi.org/10.1109/TSE.2012.43

[36] M. Wen, R. Wu, and S. Cheung, "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*, 2016, pp. 262–273. [Online]. Available: http://doi.acm.org/10.1145/2970276.2970359

[37] A. T. Misirli, E. Shihab, and Y. Kamei, "Studying high impact fix-inducing changes," *Empirical Software Engineering*, vol. 21, no. 2, pp. 605–641, 2016.

[38] F. Beck and S. Diehl, "On the impact of software evolution on software clustering," *Empirical Software Engineering*, vol. 18, no. 5, pp. 970–1004, 2013.

[39] S. Kirbas, B. Caglayan, T. Hall, S. Counsell, D. Bowes, A. Sen, and A. Bener, "The relationship between evolutionary coupling and defects in large industrial software," *Journal of Software: Evolution and Process*, vol. 29, no. 4, p. e1842, 2017.