Identifying Bug-Inducing Changes for Code Additions

Emre Sahal
Faculty of Computer and Informatics Engineering,
Istanbul Technical University
Istanbul, Turkey
sahale@itu.edu.tr

Ayse Tosun
Faculty of Computer and Informatics Engineering,
Istanbul Technical University
Istanbul, Turkey
tosunay@itu.edu.tr

ABSTRACT

Background. SZZ algorithm has been popularly used to identify bug-inducing changes in version history. It is still limited to link a fixing change to an inducing one, when the fix constitutes of code additions only. Goal. We improve the original SZZ by proposing a way to link the code additions in a fixing change to a list of candidate inducing changes. Method. The improved version, A-SZZ, finds the code block encapsulating the new code added in a fixing change, and traces back to the historical changes of the code block. We mined the GitHub repositories of two projects, Angular.js and Vue, and ran A-SZZ to identify bug-inducing changes of code additions. We evaluated the effectiveness of A-SZZ in terms of inducing and fixing ratios, and time span between the two changes. Results. The approach works well for linking code additions with previous changes, although it still produces many false positives. Conclusions. Nearly a quarter of the files in fixing changes contain code additions only, and hence, new heuristics should be implemented to link those with inducing changes in a more efficient way.

CCS CONCEPTS

• **Software and its engineering** → **Software defect analysis**; **Empirical software validation**;

KEYWORDS

Bug inducing changes, SZZ, repository mining

ACM Reference Format:
Emre Sahal and Ayse Tosun. 2018. Identifying Bug-Inducing Changes for Code Additions. In *ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM) (ESEM '18), October 11–12, 2018, Oulu, Finland.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3239235.3267440

1 INTRODUCTION

Software bugs, their causes and fixing costs are important, yet costly challenges of software development. One research direction in this context is the automatic identification of changes that induced these bugs into the source code. Inspecting bug-inducing changes help practitioners identify common mistakes leading multiple bug injections, collect information on when and by whom bugs are injected, and define preventive actions. Identifying bug-inducing changes is not a straightforward task. The most well-known algorithm, *SZZ* designed for identifying bug-inducing changes was proposed by Sliwerski et al. [4]. Many improvements on SZZ were also proposed. [6] and [3] suggested annotation graphs to avoid problems due to file name changes and moved code, and non-functional changes. SZZ was later improved in [5], in which line number mapping algorithm was proposed instead of annotation graphs to match individual lines with probabilities. [2] used dependency approach instead of text-based approaches to find bug inducing changes. [1] evaluated five different SZZ implementations, and stated that some of these implementations inflate the number of incorrectly identified bug-introducing changes. They stated that a single bug-inducing change is mapped to hundreds of future bugs, and almost half of the bugs are caused by bug-inducing changes that are years away from one another [1].

The current SZZ implementations still have one shortcoming: They cannot link a bug-fixing change with its corresponding inducing change when only additions were made to the code during the bug fix [1]. If a bug's fixing change contains new code additions only, rather than modifications or deletions on the previous source code, SZZ is not sufficient to identify the inducing change for this bug. In this paper, we propose an improved version of the original SZZ [4], namely A-SZZ, to identify the inducing changes of a fixing change consisting of newly added code. A-SZZ is based on the idea that a fixing change that adds new lines to the source code may indicate missing logic in the structural flow of the program. This missing logic could then be linked to a code block inside which the code additions were made. The previous change(s) contributed to the functionality of this code block can be traced back in the version control system, and later marked as bug-inducing changes. We do not pick the latest/earliest change to the corresponding code block as the bug-inducing change, since we believe all previous changes done on this code block may have collectively contributed to the bug. We evaluate the effectiveness of A-SZZ on two open source software projects, *Angular.js* and *Vue*, based on the three criteria proposed in [1]: earliest bug appearance, future impact of changes and realism of bug introduction.

2 A-SZZ IMPLEMENTATION

Different than prior research on SZZ in CVS with Bugzilla [3, 4] and Subversion with JIRA [1], we did our empirical analyis on two projects hosted on GitHub. The projects were cloned from GitHub, and their bugs were collected from GitHub's own issue tracking system. We also extracted the logs of the pull requests. The bugs are linked to the fixing changes through merging the pull request via SZZ. The remaining bugs that are not linked to fixing changes via the proposed semantics of SZZ are linked using candidate analysis: The highest textual similarity between the bug description and the pull request message, which occurred between the dates that the bug was created and fixed, is marked as the fixing change candidate. Then, git specific commands are used to traverse through the history of the project repository. The **git log** command was used to find the unique hash of a bug fixing commit made prior to the opening date of the bug. The parameters *-S* to search a specific line of code and *–until* for the commits before the issue was opened are used.

Code Additions. In *git*, any update to a line in a source file is counted as deletion of an existing line and adding the updated version of that line to the file. These updates are linked to where the existing line was first merged. To correctly identify the bug fixes which cannot be traced back in the history of the project, we propose five change categories. We then categorize every file that is changed in a bug fixing commit based on these.

Only Addition (A): New lines of code (LOC) were added to the file.

Addition by Modification (MA): New functionalities were added to existing LOC.

Modification (M): Some functionalities were removed from and added to existing LOC.

Deletion by Modification (MD): Some functionalities were removed from existing LOC.

Only Deletion (D): Existing LOC were removed from the file.

The changes that were made through the commit are determined by *diff* command. The changes are separated as added lines and deleted lines. If there are no lines removed, this change is labeled as *only addition*, and vice versa for *only deletion*. For the remaining changes, the added and deleted lines are tokenized using a Javascript parser, and compared with respect to the number of similar tokens between the previous and current change. If there are additional added/deleted tokens, the change is marked as *added/deleted by modification* respectively. If there are both added and deleted tokens, the change is categorized as *modifications*.

All fixing changes except *only additions* can be linked to inducing changes through SZZ. For the category *only additions*, we have to identify the code block encapsulating the added lines. To achieve this, the code is tokenized with the line numbers. The lines between the first left bracket above and the first right bracket below (the added lines) are considered as one block. In this block, each line of code needs to be checked whether the line is relevant to the functionality of the code, i.e., non-functional lines, namely comments and cosmetic lines, are discarded. For the remaining lines in the block, the original SZZ is used to find previous changes that added these lines. These changes are also added to the inducing changes.

3 EVALUATION OF A-SZZ

We assessed the effectiveness of A-SZZ on Angular.js and Vue. Table 1 presents the fixing change classifications of the two projects. It is seen that nearly a quarter of the files that contributed to a bug fix contains additions only, and they cannot be linked to an inducing change using SZZ. We applied A-SZZ to link these 251 fixing changes with only additions, in addition to the rest of the fixing changes. We did manual analysis to confirm that the code blocks were correctly identified through our approach. We also measured the effectiveness of A-SZZ in terms of da Costa et al.[1]'s criteria. Table 2 presents the average and median (in parantheses) statistics based on these criteria. It can be observed that a fixing change is linked with an average of two inducing changes in the version history, while one inducing change is blamed for introducing 12 fixing changes. These ratios are not as high as mentioned in [1]. When we look at the time span of bug inducing changes, we observe that in Angular.js an average of 403 days has been passed to fix a bug, which is relatively high, and arguable whether bug inducing changes reflect the actual causes of the bugs. The time span in Vue is more realistic. We think these long time spans are due to false positives.

The command *git log* searches the code either by line –L or string –S through the logs. If a file has multiple lines of the exact same code, every commit that has inserted that specific line to the repository is found. This means that for a little part of the modified or deleted lines, irrelevant commits are also returned by the command. We accepted results retrieving at most four inducing changes to a single line in A-SZZ. More than five inducing changes suggested by A-SZZ are discarded to avoid more false positives. We believe there are still, irrelevantly many inducing changes linked to bugs. In the future, we plan to improve A-SZZ by tracing back the code blocks surrounding newly added lines using semantic analysis rather than a syntactic match.

Table 1: Categorization of fixing changes based on types

Repository	A	MA	M	MD	D	Total
Angular.js	36	19	82	2	7	146
Vue	215	136	445	26	15	837

Table 2: Effectiveness of A-SZZ

Repository	inducing per fixing	fixing per inducing	days between ind. and fix.
Angular.js	1.7 (1)	12.6 (4.5)	403 (324)
Vue	2.1 (2)	12.3 (7)	163 (147)

ACKNOWLEDGMENTS

This work is partly funded by Turkish Scientific and Technological Research Council no. 5170048 and Scientific Research Projects Division, Istanbul Technical University no. MGA-2017-40712.

REFERENCES

[1] D.A. da Costa, S. McIntosh, W. Shang, U. Kulesza, R. Coelho, and A. E. Hassan. 2017. A Framework for Evaluating the Results of the SZZ Approach for Identifying Bug-Introducing Changes. *IEEE Trans. Software Eng.* 43, 7 (2017), 641–657.

[2] S. Davies, M. Roper, and M. Wood. 2014. Comparing text-based and dependence-based approaches for determining the origins of bugs. *Journal of Software: Evolution and Process* 26, 1 (2014), 107–139.

[3] S. Kim, T. Zimmermann, K. Pan, and E.J. Whitehead. 2006. Automatic Identification of Bug-Introducing Changes. In *21st Int. Conf. on Automated Soft. Eng. (ASE)*. 81–90.

[4] J. Sliwerski, T. Zimmermann, and A. Zeller. 2005. When Do Changes Induce Fixes?. In *Int. Workshop on Mining Software Repositories (MSR)*. 1–5.

[5] C. Williams and J. Spacco. 2008. SZZ Revisited: Verifying when Changes Induce Fixes. In *Workshop on Defects in Large Software Systems*. 32–36.

[6] T. Zimmermann, S. Kim, A. Zeller, and E.J. Whitehead. 2006. Mining Version Archives for Co-changed Lines. In *Int. Workshop on Mining Software Repositories (MSR)*. 72–75.