OpenSZZ: A Free, Open-Source, Web-Accessible Implementation of the SZZ Algorithm

Valentina Lenarduzzi,[1] Fabio Palomba,[2] Davide Taibi,[3] Damian Andrew Tamburri[4]
LUT University, Finland — SeSa Lab - University of Salerno, Italy
Tampere University, Finland — Jheronimus Academy of Data Science, The Netherlands
valentina.lenarduzzi@lut.fi,fpalomba@unisa.it,davide.taibi@tuni.fi,d.a.tamburri@uvt.nl

ABSTRACT

The accurate identification of defect-inducing commits represents a key problem for researchers interested in studying the naturalness of defects and defining defect prediction models. To tackle this problem, software engineering researchers have relied on and proposed several implementations of the well-known Sliwerski-Zimmermann-Zeller (SZZ) algorithm. Despite its popularity and wide usage, no open-source, publicly available, and web-accessible implementation of the algorithm has been proposed so far. In this paper, we prototype and make available one such implementation for further use by practitioners and researchers alike. The evaluation of the proposed prototype showed competitive results and lays the foundation for future work. This paper outlines our prototype, illustrating its usage and reporting on its evaluation in action.

KEYWORDS

Software Defect Proneness, Software Defect Prediction, Open-Source Tools, Web APIs

ACM Reference Format:
Valentina Lenarduzzi,[1] Fabio Palomba,[2] Davide Taibi,[3] Damian Andrew Tamburri[4]. 2020. OpenSZZ: A Free, Open-Source, Web-Accessible Implementation of the SZZ Algorithm. In *28th International Conference on Program Comprehension (ICPC '20), October 5–6, 2020, Seoul, Republic of Korea.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3387904.3389295

1 INTRODUCTION

State-of-the-art research in automated software engineering has proposed many studies investigating the nature of software defects [5, 18] as well as a number of approaches for predicting defects in various contexts, combining anything between rule-based [17] and machine-learning based approaches [7, 8, 19]. To support the conclusions of all these pieces of research, a common problem concerns the correct identification of so-called defect-inducing commits, i.e., the code changes in which a defect was introduced by developers. To handle this problem, a widely used solution is to rely the on Sliwerski-Zimmermann-Zeller (SZZ for short) algorithm [24] — that is, an algorithm based on the annotation/blame feature of version-control systems able to locate such defect-inducing commits.

Despite its wide usage, there exists no open-source and publicly available implementation of the SZZ algorithm which is parallelizable and highly-distributable for use in large-scale production systems, as highlighted by a recent systematic literature review on the topic [25]. This lacking entails two main consequences: (1), it is not possible for researchers to have a common ground on which to build and base their experiments; (2) there is no benchmark implementation that can be either augmented or compared to alternative solutions. For these reasons, in this paper we present OpenSZZ, our public implementation of the SZZ algorithm. Specifically, we publish and make available the full SZZ implementation project as a GitHub project [1]. Researchers can use our implementation to investigate different hypotheses or submit new versions of the algorithm. Furthermore, practitioners can make use of our implementation e.g., as part of their DevOps pipelines [2].

Overall, this paper offers two main contributions:

(1) a publicly available implementation of OpenSZZ, an open-source, scalable implementation of the SZZ algorithm;
(2) the empirical evaluation of OpenSZZ, done by conducting a manual analysis of the results yielded by the tool, which confirmed the validity of our implementation.

The remainder of this paper is organized as follows. In Section 2, we provide the background on the SZZ algorithm. Section 3 outlines our own implementation of the SZZ algorithm, highlighting its design characteristics and licensing schema. Subsequently, Section 4 outlines the evaluation, while Section 5 summarizes the impact of the proposed tool for the research community. In Section 6 we overview the tools similar to the one proposed, thus highlighting how it overcomes the state of the art. Finally, Section 7 concludes the paper.

2 THE SZZ ALGORITHM

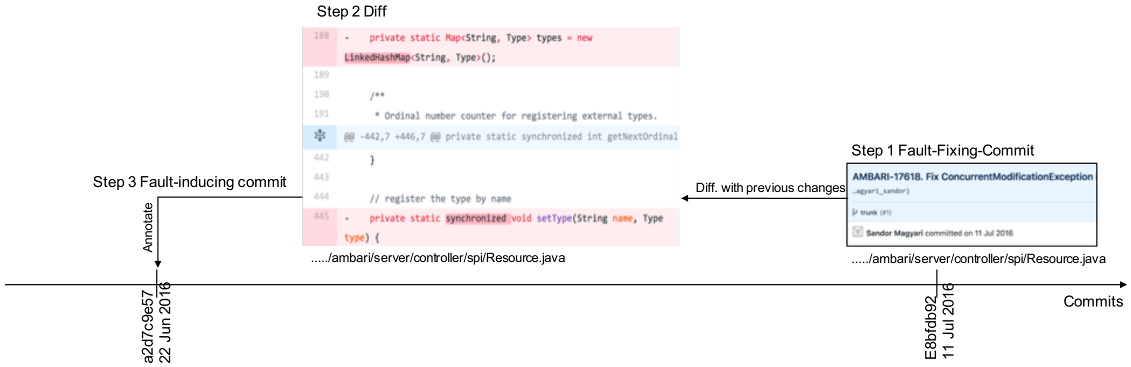
The SZZ algorithm is the most frequently used algorithm for identifying bug-introducing changes[2] [24], [10] and has been applied by the authors of 200+ research papers [20].

More specifically, the SZZ algorithm identifies sets of changes that induce bug fixes in the source code, based on historical data from versioning and issue tracking systems. The SZZ algorithm labels commits as one of three possible types: (1) **I**nducing; (2) **F**ixing; (3) **N**ot related to faults. The algorithm is based on two main steps:

ICPC '20, October 5–6, 2020, Seoul, Republic of Korea
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-7958-8/20/05…$15.00
https://doi.org/10.1145/3387904.3389295

[1] OpenSZZ Github repository: https://github.com/clowee/OpenSZZ

[2] 303 citations in Scopus and 707 in Google Scholar on 20/05/2019


Step 2 Diff
Step 3 Fault-inducing commit
Annotate
Step 1 Fault-Fixing-Commit
Diff. with previous changes
...../ambari/server/controller/spi/Resource.java
...../ambari/server/controller/spi/Resource.java
Commits
a2d7c9e57
22 Jun 2016
E8bfdb92
11 Jul 2016

```
188  -    private static Map<String, Type> types = new
LinkedHashMap<String, Type>();
189
190       /**
191        * Ordinal number counter for registering external types.
     @@ -442,7 +446,7 @@ private static synchronized int getNextOrdinal
442       }
443
444       // register the type by name
445  -    private static synchronized void setType(String name, Type
type) {
```

AMBARI-17618. Fix ConcurrentModificationException
...agyari_sandor)
trunk (#1)
Sandor Magyari committed on 11 Jul 2016

Figure 1: The OpenSZZ Approach, an outline.

Step 1. Identification of bug-fixing commits. As part of this step, the algorithm matches commits with bug reports labeled as fixed, through regular expressions that allow identifying bug numbers and keywords in the commit messages.

Step 2. Identification of bug-introducing commit(s). As part of this step, the algorithm first employs the diff functionality implemented in the control version systems to determine the lines that have been changed (to fix the bug) between the fixed commit version and its previous version.

Step 2.1. SZZ locates the commit that modified or deleted these lines the last time in previous change(s) applying annotate/blame functionalities. For example, Step 2.1 is applied in the context of Figure 1. The figure shows the differences between the commit #e8bfdb and its predecessor (#300a7e) in the *Resource.java* file. In this case, in order to fix the bug, the data structure at line 188 was changed. Therefore, SZZ identifies the changes that introduced the bug AMBARI-17618 through the history of the source configuration management system (GitHub).

Step 2.2. SZZ labels the commit *#a2d7c9* as a potential bug-introducing commit.

3 SZZ: AN OPEN IMPLEMENTATION

We implemented the SZZ algorithm as a cloud-native application provided together with a free web interface. The algorithm is implemented based on the following four concepts:

- **Transaction:** This is an object representing a commit retrieved by the git log. It contains information about the commit, such as the commit timestamp, the author, the title, the commit message, the attachments, the comments, and the changed files.
- **Issue:** This is an object representing a Jira issue. It contains information about the issue, such as the status, the priority, the reporting date, the fixing date, and the author.
- **Link:** This is an object containing a transaction and its (probably) related issue. Since a transaction can fix several issues, we can have several link objects with the same transaction, but with different issues.
- **Suspect:** For each changed link file, a suspect object is created. It contains the changed file under analysis, a commit SHA, and its time stamp. The commit is the commit that is closest to the issue-reporting day affecting the same lines of code changed by the transaction of the link.

OpenSZZ takes as input a Jira URL and a Git URL and returns a list of *Bug-Fixing Commits* and *Bug-Inducing Commits*. Below, we provide a short description of the workflow adopted:

(1) If both the Jira and the GitHub URL are correct, the tool downloads the git log and all the Jira issues;
(2) Commits containing the Jira Key (retrieved by the Jira URL) or a number or special key words (like bug(s) fix(es), defects) are saved in a transaction list;
(3) For each transaction in the list, and for each presumed Jira Key found in the description, a "Link" object is created. Each Link object is evaluated by a semantic and a syntactic analysis. Only Links achieving a certain score are kept for further analyses. We consider the remaining Links *Bug-Fixing Commits*; in other words, commits/transactions that really have closed/fixed a bug/issue;
(4) For each remaining Link for each changed file of the related Transaction, the suspect is calculated. The suspect that is closest to the issue-reporting date is kept and printed and considered the *Bug-Inducing Commit* of the issue.

The cloud-native implementation of the OpenSZZ algorithm makes it possible to execute the algorithm by invoking a web service API. The API creates a queue of requests that allows serving a higher and parallel number of requests.

The architecture of the web application version is depicted in Figure 2 and features the following architecture elements:

- The *WebApp Container* element contains the web user interface. It sends the data inserted by the user to the *Dispatcher Container* through REST web services and also displays warning or error messages in case the input data is not correct. Admin users also have the possibility to get an overview of the requests inserted by the users and the status of the requested analyses.
- The *Dispatcher Container* element checks the correctness of the input data and, if correct, inputs the data into a RabbitMQ reliable messaging queue to conduct OpenSZZ analysis. Once the analysis is completed, the Dispatcher Container component (1) receives a message from the Analyzer Container, (2) gets the results from it, and (3) sends an email to

the user communicating the completion of the analysis and providing a link for downloading the results. The results are encoded as a csv file of *BugInducingCommits*. Request data is also saved in a mongo db for monitoring purposes.

- The *Analyzer Container* is replicated n times in order to guarantee good analysis performance as well as multiple concurrent analyses; n is a parameter established by the administrator and can be instrumented with automated elasticity management following state-of-the-art approaches [1]. If it is not busy, each *Analyzer Container* listens to the same queue. Once it gets a message, it starts the analysis and will not listen to the queue anymore until it has finished the current analysis. If all containers are busy, incoming messages are queued and executed by the first available container.

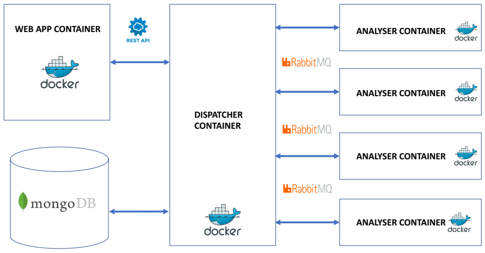
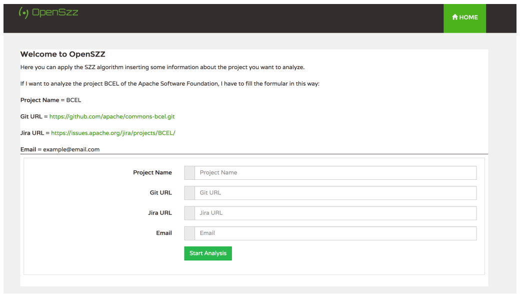


Figure 2: OpenSZZ, Web Application Architecture

In terms of direct usage of the web service, OpenSZZ provides a web dashboard where users can analyze projects by entering the basic project information into the web form 3. The SZZ then starts to compute the bug-fixing and bug-inducing commits and sends an email with the link for downloading the results as soon as the computation is finished. The users should be aware that for some big projects, the computation time could be very long (e.g., it takes more than one week for the Apache Https server).



Welcome to OpenSZZ

Here you can apply the SZZ algorithm inserting some information about the project you want to analyze.

If I want to analyze the project BCEL of the Apache Software Foundation, I have to fill the formular in this way:

Project Name = BCEL

Git URL = https://github.com/apache/commons-bcel.git

Jira URL = https://issues.apache.org/jira/projects/BCEL/

Email = example@email.com

Project Name: Project Name

Git URL: Git URL

Jira URL: Jira URL

Email: Email

Figure 3: Screenshot of OpenSZZ Web Application.

3.1 Licensing Schema and Intended Follow-ups

The SZZ implementation is provided as free and open-source software. Users can redistribute and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation version 3. Furthermore, the SZZ implementation is distributed without any warranty, without even the implied warranty of merchantability or fitness for a particular purpose. More details on the GNU GPL 3 License can be found here: https://www.gnu.org/licenses/. The aforementioned schema is meant to support academics and practitioners in furthering their understanding of the algorithm as well as promoting its use in production-ready pipelines or as part of active research transfer engagements.

3.2 Current code version

Nr.	Code metadata description	Please fill in this column
C1	Current code version	R1.1
C2	Permanent link to code/repository used for this code version	*https* : *//github.com/clowee/OpenSZZ*
C3	Code Ocean compute capsule	
C4	Legal Code License	GPL-3.0
C5	Code versioning system used	Git
C6	Software code languages, tools, and services used	Java, Docker
C7	Compilation requirements, operating environments & dependencies	Docker, Docker-compose, Java JDK 1.8 or higher, availability of open port ranges (at least 10 open ports)
C8	If available Link to developer documentation/manual	*https* : *//github.com/clowee/OpenSZZ*
C9	Support email for questions	davide.taibi@tuni.fi

4 EVALUATION

Several researchers who used the SZZ algorithm have published their data. However, the links to several of these data sets are not available anymore. The vast majority of the available data sets do not report information on bug-inducing and bug-fixing commit identifiers; instead, they report aggregated information on the total number of inducing and fixing bugs.

We validated the results of our algorithm by following the same approach adopted by Kim et al. [10] and [4]. We manually inspected the commits marked as bug-inducing and bug-fixing commits.

We randomly selected 50 bug-fixing commits from the trunk of the Zookeeper project [3]. These 50 commits contained a total of 650 changed lines, which were mapped back to a bug-inducing commit.

As expected, 551 of the 650 lines changed in the bug-fixing commits appear to actually have fixed a bug. This is in line with the results obtained by [4], who reported that 83% of the lines changed in bug-fixing commits actually fixed the bugs.

Regarding false positives, several lines were related to code refactoring or cleaning. For example, in some cases, the declaration of some variables was moved to the beginning of the class or of the method.

Unfortunately, we were not able to evaluate the results against other implementations, or to compare the results with other studies [26][6], since the replication packages were not available or did not include the labeled data on the fault-fixing and fault-inducing commits.

5 PRACTICAL IMPACT AND IMPLICATIONS

As highlighted by Rodriguez et al. [20], several works adopted the SZZ algorithm, but only some limited implementations are available. Researchers need to implement the algorithm independently, and validate it internally, with the threat of using an algorithm that

[3] https://github.com/apache/zookeeper

is not externally validated and potentially buggy. Several extension of the SZZ algorithm have been proposed by different groups. However, also the extensions are not public and the algorithms are only reported on papers. Rodriguez et al. also proposed to implement a public and open source version of the SZZ algorithm, to enable researchers to propose practical improvements, and to widely validate it.

OpenSZZ, as free and open-source project, available in GitHub, is bound to enable: (1) practitioners to practically integrate the algorithm as part of automated testing, integration, and deployment pipelines; (2) researchers interested in developing and extending this algorithm as well as testing its limitations in terms of non-functional aspects such as performance, scalability, automation degree and more; (3) security and & privacy engineers to elaborate more on the SZZ' weaknesses and vulnerabilities at scale. Furthermore, researchers are enabled to fork the project proposing their new versions, or submit new improvements by means of the classical GitHub pull-request mechanism.

Beyond the current state of the art, OpenSZZ will open several research questions, including for example the public and concurrent open validation of the original SZZ algorithm as well as its further improvement. In addition, the availability of our implementation will also enable mining software repositories competitions with the goal of improving the identification accuracy of the fault-inducing commit.

Finally, on top the indication of usage of the SZZ algoritm reported in the SLR from Rodriguez et al. [20], a query in Scopus[4] reveals that more than 450 papers implemented the SZZ algorithm independently. Figure 4. OpenSZZ will enable many more works on this topic, allowing all the researchers that never implemented the algorithm to use it.

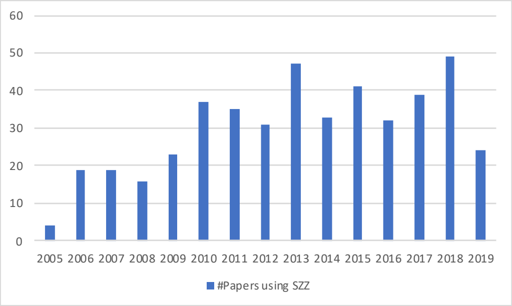


Figure 4: Number of papers using the SZZ [24] algorithm

6 RELATED WORK

In this paper, we present an open-source and publicly available tool implementing the SZZ algorithm. While we are aware of the many previous studies that employed SZZ for finding the origin of defects, we consider them out of the scope of our literature review; a comprehensive overview of these studies is available in the paper by Rodriguez et al. [20].

When it comes to the implementation of SZZ, this was originally devised by Śliwerski et al. [24]. Later on, different researchers have provided improvements in order to filter out cosmetic changes (e.g., rename refactoring operations) [9] or to increase its precision by means of line number mapping [27]. While in our work we have opted for the implementation of the original algorithm, we made our source code open with the aim of stimulating other researchers to contribute, implement, and test new versions of SZZ. At the same time, it is important to note that the evaluation conducted to verify the performance of the proposed tool showed that it is able to correctly locate most of the bug-inducing commits, meaning that our implementation already offers a valuable basis for conducting studies that require the usage of SZZ.

The closest tools to the one presented herein are those presented by Rosen et al. [21] and Borg et al. [3]. In the former work, the CommitGuru platform was introduced: this is a software analytics tool that allows researchers to specify a Github repository and outputs a number of metrics, including the information on the risky commits, i.e., those that can potentially introduce defects. Looking at the implementation details of the platform, the risky commits are identified based on the original implementation of SZZ. There are two main differences with respect to the tool we propose: contrarily to Rosen et al. [21], (1) we present an open-source platform that researchers can directly use or even improve; and (2) we empirically assessed the performance of our implementation, showing that it is suitable for mining software repository studies. As for Borg et al. [3], they presented SZZUnleashed, an open implementation of SZZ that has later used in an example scenario involving Jenkins projects. While our paper has a similar goal, the open-source nature of OpenSZZ makes it more usable and extensible. Moreover, OpenSZZ enables to scale faster, allowing to analyze several large projects in parallel thanks to its cloud-native nature. Furthermore, Borg et al. [3] did not report details about the accuracy of their implementation and the resulting performance, while we made an effort toward this direction in order to produce a tool that is as accurate as possible.

7 CONCLUSIONS

This paper presented an open implementation of the well-known SZZ algorithm. The implementation is provided as a downloadable package and container so that it can be used as a containerized web service API. The evaluation was conducted using state-of-the-art data sets and by comparing implementation performances.

The evaluation itself showed competitive performance, which reflects a highly usable and potentially production-ready piece of software. OpenSZZ has been already adopted in our previous works [22][11][12][14][16] and a dataset containing the analysis of 33 open source projects has been recently published [15]. More details on the dataset and on the diffuseness of faults can be found in [13] Moreover, the GitHub project has been forked 49 times by several researchers.

In the future, we aim to refine our implementation from several perspectives, including its usage as part of larger-scale DevOps verification and validation pipelines and analytical solutions. Moreover, we are planning to implement different extensions of the SZZ algorithm, such as [23] and [4], to allow researchers to adopt the version they need in their research work.

[4] Scopus Search String: REF (”When do changes induce fixes”)

REFERENCES

[1] Luciano Baresi, Sam Guinea, Giovanni Quattrocchi, and Damian Andrew Tamburri. 2016. MicroCloud: A Container-Based Solution for Efficient Resource Management in the Cloud.. In *SmartCloud*. 218–223.

[2] Len Bass. 2018. The Software Architect and DevOps. *IEEE Software* 35, 1 (2018), 8–10.

[3] Markus Borg, Oscar Svensson, Kristian Berg, and Daniel Hansson. 2019. SZZ Unleashed: An Open Implementation of the SZZ Algorithm-Featuring Example Usage in a Study of Just-in-Time Bug Prediction for the Jenkins Project. *arXiv preprint arXiv:1903.01742* (2019).

[4] Chadd C. Williams and Jaime Spacco. [n. d.]. SZZ revisited: verifying when changes induce fixes. In *Proceedings of the 2008 Workshop on Defects in Large Software Systems, held in conjunction with the ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2008), DEFECTS 2008, Seattle, Washington, USA, July 20, 2008*.

[5] Gemma Catolino, Fabio Palomba, Andy Zaidman, and Filomena Ferrucci. 2019. Not All Bugs Are the Same: Understanding, Characterizing, and Classifying Bug Types. *Journal of Systems and Software* (2019).

[6] 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 Transactions on Software Engineering* 43, 7 (July 2017), 641–657.

[7] Dario Di Nucci, Fabio Palomba, Giuseppe De Rosa, Gabriele Bavota, Rocco Oliveto, and Andrea De Lucia. 2017. A developer centered bug prediction model. *IEEE Transactions on Software Engineering* 44, 1 (2017), 5–24.

[8] Norman E. Fenton and Martin Neil. 1999. A Critique of Software Defect Prediction Models. *IEEE Trans. Software Eng.* 25, 5 (1999), 675–689.

[9] Sunghun Kim, Thomas Zimmermann, Kai Pan, E James Jr, et al. 2006. Automatic identification of bug-introducing changes. In *21st IEEE/ACM International Conference on Automated Software Engineering (ASE'06)*. IEEE, 81–90.

[10] S. Kim, T. Zimmermann, K. Pan, and E. J. Jr. Whitehead. 2006. Automatic Identification of Bug-Introducing Changes. In *International Conference on Automated Software Engineering (ASE'06)*. 81–90.

[11] Valentina Lenarduzzi, Francesco Lomio, Davide Taibi, and Heikki Huttunen. 2019. Are SonarQube Rules Inducing Bugs? *International Conference on Software Analysis, Evolution and Reengineering (SANER 2020). Preprint: arXiv:1907.00376*.

[12] Valentina Lenarduzzi, Antonio Martini, Davide Taibi, and Damian Andrew Tamburri. 2019. Towards Surgically-Precise Technical Debt Estimation: Early Results and Research Roadmap. In *2019 IEEE Workshop on Machine Learning Techniques for Software Quality Evaluation (MaLTeSQuE)*.

[13] V. Lenarduzzi, N. Saarimaki, and D. Taibi. 2019. On the Diffuseness of Code Technical Debt in Java Projects of the Apache Ecosystem. In *International Conference on Technical Debt (TechDebt)*. 98–107.

[14] Valentina Lenarduzzi, Nyyti Saarimäki, and Davide Taibi. 2019. Some SonarQube Issues have a Significant but Small Effect on Faults and Changes. A large-scale empirical study. *arXiv e-prints*, Article arXiv:1908.11590 (Aug 2019), arXiv:1908.11590 pages. arXiv:cs.SE/1908.11590

[15] Valentina Lenarduzzi, Nyyti Saarimäki, and Davide Taibi. 2019. The Technical Debt Dataset. In *15th conference on PREdictive Models and data analycs In Software Engineering (PROMISE '19)*.

[16] Valentina Lenarduzzi, Christian Stan, Davide Taibi, Davide Tosi, and Gustavs Venters. 2017. A Dynamical Quality Model to Continuously Monitor Software Maintenance. In *11th European Conference on Information Systems Management (ECISM2017)*.

[17] Bharavi Mishra and Kaushal K. Shukla. 2014. Software Defect Prediction Based on GUHA Data Mining Procedure and Multi-Objective Pareto Efficient Rule Selection. *IJSSCI* 6, 2 (2014), 1–29.

[18] Fabio Palomba, Gabriele Bavota, Massimiliano Di Penta, Fausto Fasano, Rocco Oliveto, and Andrea De Lucia. 2018. On the diffuseness and the impact on maintainability of code smells: a large scale empirical investigation. *Empirical Software Engineering* 23, 3 (2018), 1188–1221.

[19] Fabio Palomba, Marco Zanoni, Francesca Arcelli Fontana, Andrea De Lucia, and Rocco Oliveto. 2017. Toward a smell-aware bug prediction model. *IEEE Transactions on Software Engineering* (2017).

[20] Gema Rodriguez, Gregorio Robles, and Jesus Gonzalez-Barahona. 2018. Reproducibility and Credibility in Empirical Software Engineering: A Case Study based on a Systematic Literature Review of the use of the SZZ algorithm. *Information and Software Technology* (03 2018).

[21] Christoffer Rosen, Ben Grawi, and Emad Shihab. 2015. Commit guru: analytics and risk prediction of software commits. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. ACM, 966–969.

[22] Nyyti Saarimäki, Valentina Lenarduzzi, and Davide Taibi. 2019. On the diffuseness of code technical debt in open source projects of the Apache Ecosystem. *International Conference on Technical Debt (TechDebt 2019)* (2019).

[23] Emre Sahal and Ayse Tosun. 2018. Identifying Bug-inducing Changes for Code Additions. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM '18)*. ACM, New York, NY, USA, Article 57, 2 pages.

[24] Jacek Śliwerski, Thomas Zimmermann, and Andreas Zeller. 2005. When Do Changes Induce Fixes? *SIGSOFT Softw. Eng. Notes* 30, 4 (May 2005), 1–5.

[25] Le Hoang Son, Nakul Pritam, Manju Khari, Raghvendra Kumar, Pham Thi Minh Phuong, and Pham Huy Thong. 2019. Empirical Study of Software Defect Prediction: A Systematic Mapping. *Symmetry* 11, 2 (2019), 212.

[26] Ming Wen, Rongxin Wu, Yepang Liu, Yongqiang Tian, Xuan Xie, Shing-Chi Cheung, and Zhendong Su. 2019. Exploring and Exploiting the Correlations between Bug-Inducing and Bug-Fixing Commits. In *27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2019)*. 326–337.

[27] Chadd Williams and Jaime Spacco. 2008. Szz revisited: verifying when changes induce fixes. In *Proceedings of the 2008 workshop on Defects in large software systems*. ACM, 32–36.