Techniques for Bug–Code Linking

GORAN MAUŠA, PAOLO PERKOVIĆ, TIHANA GALINAC GRBAC and IVAN ŠTAJDUHAR,
Faculty of Engineering, University of Rijeka

Diverse research groups have identified that analyzing the bugs-to-code fix links may generate many interesting theories. However, this kind of datasets are usually not easily available from the software development projects. Usually, the bug tracking and the related code fix activities are divided into separate processes and use separated repositories thus complicating the linking of the data. Numerous techniques have been developed but we still lack the empirical data collected with standard data collection procedures.

In this paper we examine the effectiveness of the bug linking techniques. In particular we evaluate the proposed technique based on the regular expressions and compare its effectiveness with other known bug linking techniques. Our case study shows that the proposed technique is equally effective as some other already proposed techniques in eliminating the linking bias we identified some differences that may have influence on the linking precision. However, the technique is not addressing data quality issues that may be present in software repositories.

Categories and Subject Descriptors: D.2.5 [**SOFTWARE ENGINEERING**]: Testing and Debugging—*Tracing*; D.2.9 [**SOFTWARE ENGINEERING**]: Management—*Software quality assurance (SQA)*; H.3.3 [**INFORMATION STORAGE AND RETRIEVAL**] Information Search and Retrieval

Additional Key Words and Phrases: Bug – code linking, software, defect, open source repositories

ACM Reference Format:

Goran Mauša, Paolo Perković, Tihana Galinac Grbac and Ivan Štajduhar *jn* 2, 3, Article 1 (May 2010), 9 pages.
DOI:http://dx.doi.org/10.1145/0000000.0000000

1. INTRODUCTION

Analyzing bugs-to-code fix links may generate many interesting theories. For example, new empirical findings aiming to contribute the theory of fault distributions may have significant influence on future software development methods and practices [Grbac et al. 2013; Petrić and Grbac 2014; Grbac and Huljenic 2014]. Since bugs and code commits are usually stored in different repositories with different information needs their links have to be mined. Thus, such retrieved datasets suffer from data quality as well as data collection bias. Data quality is very much affected by software developers responsible for filling the data and missing or wrongly inserted data in software repositories are a result of human error. On the other hand, retroactive data collection and linking between different repositories represent another problem that is well known as linking bias. A Link may be incorrectly established or missing.The reason may be in ineffectiveness of linking technique employed and/or weak understanding of underlaying complex software development interactions.

The work presented in this paper is supported by the University of Rijeka research grant Grant 13.09.2.2.16.
Author's address: G. Mauša, Faculty of Engineering, Vukovarska 58, HR–51000 Rijeka, Croatia; email: gmausa@riteh.hr; P. Perković, student at the Faculty of Engineering, Vukovarska 58, HR–51000 Rijeka, Croatia; email: pperkov@riteh.hr; T. Galinac Grbac, Faculty of Engineering, Vukovarska 58, HR–51000 Rijeka, Croatia; email: tgalinac@riteh.hr; I. Štajduhar, Faculty of Engineering, Vukovarska 58, HR–51000 Rijeka, Croatia; email: istajduh@riteh.hr

As part of our research in the fault distribution of complex systems we want to perform extensive explorative analysis on bug–code datasets [Galinac Grbac et al. 2013]. As software defect datasets available for such analysis have been criticized by number of authors we were motivated to develop a tool that would based on the numerous use open source available repositories build as much as possible bug-to-code fix datasets. This paper deals with definition of such a tool that we will refer to as BuCo (Bug-Code) tool.

In the last few years, research on bug–fix datasets has progressed well, since the number of open source projects and using of open source project repositories grow and reported many important results. The popularization of open source software enabled individuals to introduce different ways of software development and code analysis. However, datasets bias still remains as a huge issue for wider generalization of the results and development of sound theories. The bug–code fix linking process is extremely hard task and there is still no standard protocol that would be applicable for wider community and numerous software development repositories. Different linking techniques have been proposed in order to minimize the linking bias and some even try to minimize the impact of data quality issues. However, we believe that different linking strategies may not be equally effective for all kind of repositories and software projects. Obviously, proposing the general dataset collection procedures and linking techniques may not be relevant. Instead of proposing the best performing one for open source repositories we review several linking techniques and evaluated their effectiveness on different datasets. We discuss its applicability for data collection from complex open source software projects.

2. RELATED WORK

Many researchers mine open source software development repositories trying to collect bug–fix datasets. One of the most commonly applied technique is using regular expressions [Bachmann et al. 2010; Fischer et al. 2003; Schrter et al. 2006; Śliwerski et al. 2005; Čubranić and Murphy 2003]. It is a sequence of characters that forms a search pattern, most commonly used for string matching used for information retrieval [Baeza-Yates and Ribeiro-Neto 1999]. Drawing on the idea introduced by a neuroscientist and a logician in the early 1940s, it was formally introduced by a mathematician 15 years later, relating the concept to finite automata [Kleene 1956]. First implementation of a regular expression compiler was developed in 1968 by the Unix pioneer, Ken Thompson [Thompson 1968], and has since been used in various problem domains. A regular expression consists of both regular and meta–characters. Both types are combined to form an expression for identifying the pattern sequence (sequence of interest) in a sequence of characters. In other words, the pattern sequence is an expression representing prescribed targets in the most concise and flexible way to direct the automation of character sequence processing.

A regular expression processor translates into a nondeterministic finite automaton, which is then run on the sequence of characters to recognize subsequences that match the regular expression [Baeza-Yates and Ribeiro-Neto 1999]. The most commonly used linking technique is based on regular expression and is searching in source code commit messages for a specific keyword such as *fixed* or *bug* and *bugID* (such as *42233*) usually using form of regular expression [Bachmann et al. 2010].

Many researchers aiming to develop the most effective linking technique and to understand dataset bias and data quality issues have invested huge efforts to develop a reliable dataset for the benchmarking purposes. As part of the study published with [Bachmann et al. 2010] the authors work with an experienced *Apache* developer who helped them to manually develop an *ground truth* dataset. The main focus of the study was to understand the quality of the data provided in the open source software development repositories. They found out that many bug fixes are not identified within the commit messages or even not reported within the bug tracking repository and that this finding might be a serious threat to studies based on the such datasets. However, the sample size they had was not large

enough to make any statistical conclusions but provide some reasonable evidence to threats to external validity of studies performed on linked datasets from open source repositories. One solution to the data quality problem they see in improving the linking process with information available from the whole social eco–system available such as revision control system, bug tracking database, mailing systems, email discussions, etc.

That finding motivated a number of studies aiming to improve the linking techniques. Improvements may be classified into two directions; direction that involves additional information available from the open repositories and direction that introduces fancy algorithms for the link prediction purposes. One very good example is ReLink, a tool proposed in [Wu et al. 2011] that offered a 2–cycle linking process. In the first round the traditional linking technique based on the regular expression as mentioned above has been extended with new features as follows:

- bug fixing time that is close to the change–commit time,
- the change logs and the bug report share textual similarity, and
- that the developers responsible for a bug are typically the committers of the bug–fixing change.

In the second round, based on the linking dataset in the first round as training sample and some learning rules, predicts additional linking data items. They compared their results with the ground truth file prepared by [Sureka et al. 2011]. As reported in [Wu et al. 2011] they obtained the consistent results with [Sureka et al. 2011] for the Apache project. Also, they conclude that many open source projects hosted by Google are similar and follow the features they implemented in the ReLink. On the other hand, they identified that linking improvements were not equally effective for all datasets. For example linking improvement for Apache was marginal compared to other two Android projects in the study. The reason for this result lies in relatively good data quality of Apache compared to Android projects. We assume that for bigger and complex open source projects the data quality probably tends to be maximized for several reasons. Firstly, the project takes longer and community has time to learn ways of working and less human error happens and another reason is that the samples are getting bigger and thus marginalize errors.

Recently published study [Bissyande et al. 2013] has validated their results and assessed the effectiveness of ReLink quantitatively and qualitatively. Again, they confirmed that in the different project setting the effectiveness of ReLink improvements may be marginal but also they show that these improvements may lead to new errors thus significantly influencing the linking bias. They also provided a benchmark dataset collected from Apache Software Foundation with input and output files provided for 10 projects.

3. EXPERIMENT

Our goal is to identify and explore the most effective linking technique across number of different environments. In this study we define an experiment for that purpose. The ReLink tool, as a good example of combining the simple search with advanced prediction techniques is chosen for this experiment. However, it was only compared with other similarly complex approaches as well as the ground truth dataset. Moreover, because of the input data it provides to open community we decided to investigate how well does it perform comparing to the simpler search approaches. That is why we plan to conduct our experiment in several stages:

(1) **Analysis 1:** Compare the ReLink tool with the simple search on Apache HTTPD dataset given by ReLink as presented in figure 1
(2) Evaluate the results, manually investigate the incorrect links and construct a regular expression search criterion

(3) **Analysis 2:** Compare the ReLink tool with the regular expression search using the same Apache HTTPD dataset as presented in figure 1

(4) **Analysis 3:** Compare the ReLink tool and the regular expression search with the benchmark dataset for the Apache Opennlp project as presented in figure 2

The **analysis 1** consists of comparing the ReLink tool with the basic idea of linking bugs and commits searching for the Bug ID number in the commit messages. We believe this approach to be the most reliable one. The only question is how to search for the Bug_ID effectively. That is why our first approach to the linking was as simple as that. The only improvement we added was to search only for the latest commit that contains the bug ID in order to discard all the duplicated entries, bug fixing attempts and workarounds. The input bugs are taken from the ReLink webpage[1] while the source code is taken from the Github repository of the Apache HTTPD project[2]. These inputs are given to the BuCo Analyzer tool and the linking is done using simple search mechanism.

The results of **analysis 1** will indicate the weaknesses of simple search. The manual investigation of the incorrect links will look for regularities and consistent errors in order to yield a criterion for more sofisticated search. We will construct the regular expression based on this criterion and repeat the analysis. For the **analysis 2** we use the same Apache HTTPD project with the same inputs as in the **analysis 1** (figure 1).

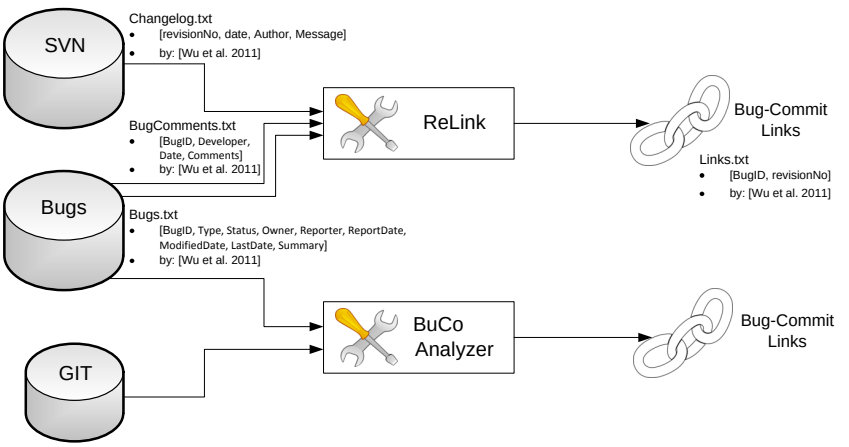

Fig. 1. The Source and Structure of Inputs and Outputs for **Analysis 1** and **2**

The **analysis 3** compares the ReLink tool and the regular expression in a controlled environment of a benchmark dataset. The Opennlp project is analyzed and given by [Bissyande et al. 2013]. The goal of the Opennlp data is to provide a benchmark dataset for comparison of studies such as this one. It offers the subset of bugs, the source code history and the correct links between the bugs and commits made by developers. Furthermore, it offers the comments on all of the given bugs extracted from bug tracking repository and the bug-commit links made by the ReLink tool. The bugs for the Opennlp project are taken from the benchmark dataset[3]. The source code is taken from the Github repository

[1] https://code.google.com/p/bugcenter/wiki/ReLink
[2] https://github.com/apache/httpd.git
[3] http://momentum.labri.fr/bugLinking/

of the Apache Opennlp project[4]. These inputs are given to the BuCo Analyzer tool and the linking is done using regular expression search mechanism. The details are presented in figure 2.

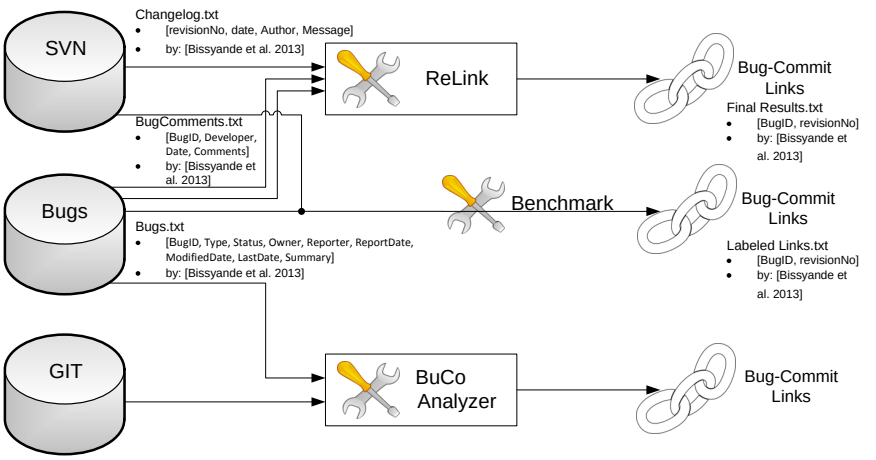

Fig. 2. The Source and Structure of Inputs and Outputs for **Analysis 3**

3.1 Approach to Data Collection

One of the first challenge encountered in our research was to find the appropriate tools for such a demanding task of data collection. We managed to find no tool that could provide us with the source code analysis of software product metrics and the number of bugs contained within every software module. That is why we developed a tool of our own and named it the Bug Code (BuCo) Analyzer. Its goal is to collect software fault's data and find its dependencies in the most efficient way possible. The tool is based upon the following technologies: Apache HTTP server, MySQL relational databases, Git and Subversion source control tools, Python and data mining. They were chosen after their reliability and experienced gained from using them in past projects.

The local database within the tool is constructed so that it can contain the bug information details, the links between the bug and the source code changes (commits) and the list of source code modules and all of their metrics for projects of three major open source communities: Eclipse, Mozilla and Apache. The bug details can be downloaded from the Bugzilla repository into our database and the complete source code history can be downloaded through the tool interface and stored locally as a GIT repository. The BuCo Analyzer also offers interface to other tools. One such tool is the ReLink. Its task is to establish connections between the bugs and the commits. However, that is just one of the techniques the tool offers for this linking process. The complete architecture of the BuCo Analyzer tool is presented in figure 3

As mentioned earlier, our intention is to analyze large and complex software products with long lasting evolution, i.e. a great number of releases. All the analyses then become demanding and slow. To optimize and gain every little boost in terms of speed some smart thinking and known techniques are used in the creation. MySQL databases which would store the data needs to be passed through normalizations steps and has to be modeled after the tool's needs. To further speed up the process

[4]https://github.com/apache/opennlp

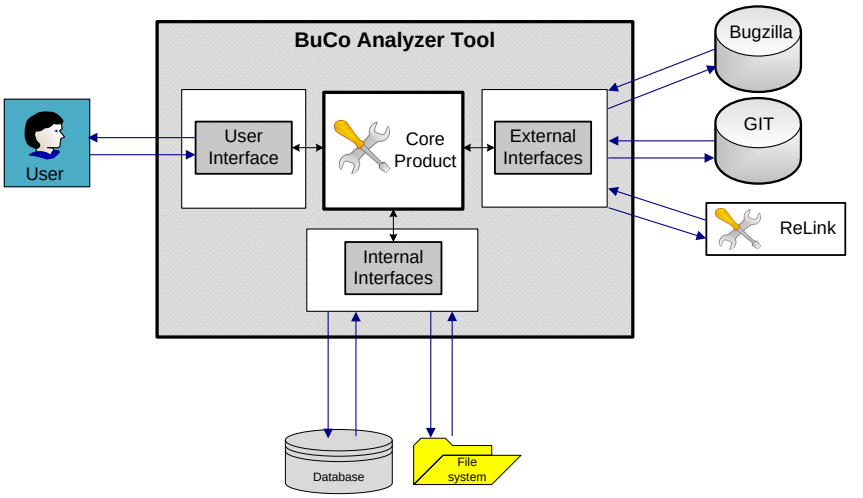

Fig. 3. The BuCo Analyzer Tool Architecture

repositories are downloaded and analyzed locally instead of analyzing them remotely over the Internet. Other optimizations are done by building efficient search patterns and compiling the patterns when they are necessary.

The tool was being improved iteratively. The first and the most notable aspects is the introduction of pattern matching using regular expressions. This version is able to collect all commits for every bug ID found and their respective messages and implements the collection of bug IDs and downloads of repositories for different foundations. The latest tool version offers a high credibility level for the bug-commit links and the software product metrics associated to the software modules.

3.2 Results

The results of **analysis 1** are presented in table I. The ReLink tool performed the linking with the same bugs but with different source code repository, the SVN. The difference between the two repositories is obvious in the number of commits they contain and it reflects on the output results as well.

Table I. Linking Bugs and Commits Using Simple Search, Regular Expression and The ReLink Tool

	Input			**Linking**	**Output**			
Analysis	Source	Commits	Bugs	**Method**	Links	Commits	Files	Bugs
1	SVN + Bugs by Relink	43867	673	ReLink	1014	957	1061	673
	GIT + Bugs by Relink	26287	673	Simple search	598	556	993	598
2	SVN + Bugs by Relink	43867	673	ReLink	1014	957	1061	673
	GIT + Bugs by Relink	26287	673	Regular Expression	703	664	495	621
3	SVN + Bugs by benchmark	847	100	Benchmark	127	125	141	81
	SVN + Bugs by benchmark	847	100	ReLink	115	113	132	76
	GIT + Bugs by benchmark	847	100	Regular Expression	128	126	141	81

The results of the simple search were then manually analyzed, compared to the ReLink results and validated. We focused on finding the links that the ReLink and the simple search had in common in order to evaluate the general appropriateness of the approach and on the links that are different in order to discover the simple search weaknesses. The results given in table II indicate that 74.1% of

the links are equal to the ones made by the ReLink and all of them were correct ones. The 20.1% of the links that were not equal to the ones made by the ReLink and were incorrect presented a valuable subset to discover patterns and consistent errors in linking process.

Table II. Analyzing Links Obtained With Simple Bug_ID Search

Equal Links	443	74.1%
Bugs with one link	196	32.8%
Bugs with multiple links	247	41.3%
Different Links	147	24.6%
Incorrect links	120	20.1%
Potentially correct links	27	4.5%
Links From Different Repository	8	1.3%

The evaluation and manual validation of the results revealed that in each of incorrect links the Bug ID was adjacent to other alphanumeric characters forming a Bug ID with more digits or a different identification code. This led us to the improved search criteria which will predetermine the bug ID surrounding characters so we constructed the following regular expression:

$$'(.*[\hat{}0-9]|\hat{})' + bug\_id +' (\backslash W|\backslash\backslash r|\$)' \quad (1)$$

The regular expression given in (1) defines:

- the preceding character to be any non digit character, including the start of string
- the number to be the exact match of the Bug ID we are looking for
- the following character to be any non alphanumerical character, including the end of string

In order to evaluate the performance of linking bugs and commits using regular expression given in (1), we performed the **analyses 2** and **3**. Unlike the HTTPD project, we managed to find the equal source code history for the Opennlp project even though we take it from different repository. Therefore, the linking is done upon the same input and it offers a good comparison basis. The correct links given by the benchmark dataset are compared with the links obtained by ReLink and with the links obtained by our regular expression search. The details are given in tableI.

The results obtained from the **analysis 2** upon the HTTPD project indicates the regular search can link approximately the same number of bugs as the ReLink tool. The difference between these two approaches is due to the different source code inputs. Analyzing the links made by the ReLink tool and not by the regular expression search reveals that all of their links do have a bug ID number in the commit message. We looked for these commits in the GIT repository but did not find them. This means that the regular expression search would have performed just as good as the ReLink does if it had the same source code input.

In order to prove this theory, the **analysis 3** is done upon the Opennlp project comparing the results also to the benchmark dataset. The results show that our regular expression search approach actually outperforms the ReLink tool. We found all the correct links given by the benchmark and an incorrect one. The incorrectly linked commit is given in table III. The commit message clearly shows it has nothing to do with the Bud ID we are looking for. Furtheremore, the time elapsed between closing the bug and committing the supposed fix is greater than it usually is (from several days up to a month). Finally, the bug assignee does not correspond to the author of the commit. This link was made due to a single digit bug ID which is found in the name of Apache release. The bug IDs that are small in

value present an obstacle to our search approach because they could be found in other identification numbers, dates, release tags or similar parts of a commit message.

Table III. Commit Incorrectly Linked by the Regular Expression Search

Bug ID	Commit **Message**	Opened	**Closed**	**Commited**	Bug **Assignee**	**Commiter**
9	OPENNLP-190 Updated to Apache **9** parent pom and removed special version which we needed for the Apache 8 parent pom, namely for the rat plugin and the release plugin.	9.12.2010	13.1.2011	30.5.2011	William Colen	Joern Kottmann

The ReLink, on the other hand had no incorrect links, but managed to miss 12 of them, which is approximately 10%. They also did not link 5 bugs at all, which is approximately 6%. Several examples of missing links are given in table IV. It is unclear why the ReLink tool did not make that connection. All the presented missing links do contain the Bug ID in the commit message. The time elapsed between closing the bug and committing the supposed fix ranges from 0 up to 63. Besides the first commit linked to the Bug ID 367 which has 63 days and the commit linked to the Bug ID 471 which has 35 days, the remaining commits are within usual time interval. Finally, all the commit authors do correspond to the bug assignee role in the presented missing links.

Table IV. Examples of Links Missed by the ReLink Tool

Bug ID	Commit **Message**	Opened	**Closed**	**Commited**	Bug **Assignee**	**Commiter**
84	OPENNLP-**84** Corrected method name to sentPosDetect	25.1.2011	25.1.2011	**25.1.2011**	Joern Kottmann	**joern**
115	OPENNLP-**115** Charset should be specified before creating input stream	1.2.2011	11.7.2011	**11.7.2011**	Joern Kottmann	**joern**
471	OPENNLP-**471**: found after we find a name match, we don't jump over the found name but re-process... thanks William for pointing this out	14.3.2012	24.4.2012	19.3.2012	James Kosin	**jkosin**

The mistakes of the linking process reflect upon the files we indicate as fault prone as well. Our approach pronounces the same 141 files to be fault prone as the benchmark dataset does, because the incorrect link is connected to the already fault prone file. The ReLink failed to pronounce 9 different files as being fault prone, which is approximately 6%.

4. CONCLUSION

Integrated datasets such as Bug-Code fix datasets are very important for further development of the software engineering discipline. This datasets can provide us new insights into the software development processes and practices and the conclusions may lead to new developments.The creation of as much as possible Buge–code datasets that are reliable and comparable enough to minimize external threats to validity is an important task. However, addressing the data quality and dataset bias is not an easy task.

In this paper we show that already a simple traditional approaches with help of regular expressions may work well. However, they may not be equally effective in all environments. One should adapt the regular expression to each particular repository and even to each project. From the performed study we may observe that the data quality issues may be lower for bigger, longer and more complex projects. Therefore, we assume that very advanced linking techniques involving learning rules might

not be not only ineffective but also not applicable for the complex project environments. The very good results obtained by our regular expression bug-code linking, evident in the case of benchmark datasets, encourage us to expand this research. Our future work is to replicate it on more complex projects from different open source communities and to expand this the selection bug-code linking techniques. The comparison of techniques will include the datasets provided by other researchers and try to find statistical evidence in favor or against these statements.

The main contribution of this study is the presentation of an experiment and the preparation for more advanced and more demanding experiment that would lead to development of algorithm adaptable to open source repositories and projects aiming to build as much as reliable bug–code datasets.

REFERENCES

Adrian Bachmann, Christian Bird, Foyzur Rahman, Premkumar Devanbu, and Abraham Bernstein. 2010. The Missing Links: Bugs and Bug-fix Commits. In *Proceedings of the Eighteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE '10)*. ACM, New York, NY, USA, 97–106. DOI:http://dx.doi.org/10.1145/1882291.1882308

Ricardo A. Baeza-Yates and Berthier Ribeiro-Neto. 1999. *Modern Information Retrieval*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Tegawende F. Bissyande, Ferdian Thung, Shaowei Wang, David Lo, Lingxiao Jiang, and Laurent Reveillere. 2013. Empirical Evaluation of Bug Linking. In *Proceedings of CSMR '13*. IEEE Computer Society, Washington, DC, USA, 89–98. DOI:http://dx.doi.org/10.1109/CSMR.2013.19

Michael Fischer, Martin Pinzger, and Harald Gall. 2003. Analyzing and Relating Bug Report Data for Feature Tracking. In *Proceedings of the 10th Working Conference on Reverse Engineering (WCRE '03)*. IEEE Computer Society, Washington, DC, USA, 90–. http://dl.acm.org/citation.cfm?id=950792.951355

Tihana Galinac Grbac, Goran Mauša, and Bojana Dalbelo Bašić. 2013. Stability of Software Defect Prediction in Relation to Levels of Data Imbalance.. In *Proceedings of SQAMIA '13*. Novi Sad, Serbia, 1–10.

Tihana Galinac Grbac and Darko Huljenic. 2014. On the probability distribution of faults in complex software systems. *Information and Software Technology* 0 (2014), –. DOI:http://dx.doi.org/10.1016/j.infsof.2014.06.014

Tihana Galinac Grbac, Per Runeson, and Darko Huljenic. 2013. A Second Replicated Quantitative Analysis of Fault Distributions in Complex Software Systems. *IEEE Trans. Software Eng.* 39, 4 (2013), 462–476. http://dblp.uni-trier.de/db/journals/tse/tse39.html#GrbacRH13

S. C. Kleene. 1956. Representation of Events in Nerve Nets and Finite Automata. *Automata Studies* (1956).

Jean Petrić and Tihana Galinac Grbac. 2014. Software Structure Evolution and Relation to System Defectiveness. In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering (EASE '14)*. ACM, New York, NY, USA, Article 34, 10 pages. DOI:http://dx.doi.org/10.1145/2601248.2601287

Adrian Schrter, Thomas Zimmermann, Rahul Premraj, and Andreas Zeller. 2006. If your bug database could talk. In *In Proceedings of the 5th International Symposium on Empirical Software Engineering, Volume II: Short Papers and Posters*. 18–20.

Jacek Śliwerski, Thomas Zimmermann, and Andreas Zeller. 2005. When Do Changes Induce Fixes? *SIGSOFT Softw. Eng. Notes* 30, 4 (May 2005), 1–5. DOI:http://dx.doi.org/10.1145/1082983.1083147

Ashish Sureka, Sangeeta Lal, and Lucky Agarwal. 2011. Applying Fellegi-Sunter (FS) Model for Traceability Link Recovery between Bug Databases and Version Archives.. In *APSEC*, Tran Dan Thu and Karl R. P. H. Leung (Eds.). IEEE, 146–153. http://dblp.uni-trier.de/db/conf/apsec/apsec2011.html#SurekaLA11

Ken Thompson. 1968. Programming Techniques: Regular Expression Search Algorithm. *Commun. ACM* 11, 6 (June 1968), 419–422. DOI:http://dx.doi.org/10.1145/363347.363387

Davor Čubranić and Gail C. Murphy. 2003. Hipikat: Recommending Pertinent Software Development Artifacts. In *Proceedings of the 25th International Conference on Software Engineering (ICSE '03)*. IEEE Computer Society, Washington, DC, USA, 408–418. http://dl.acm.org/citation.cfm?id=776816.776866

Rongxin Wu, Hongyu Zhang, Sunghun Kim, and Shing-Chi Cheung. 2011. ReLink: Recovering Links Between Bugs and Changes. In *Proceedings of ESEC/FSE '11*. ACM, New York, NY, USA, 15–25. DOI:http://dx.doi.org/10.1145/2025113.2025120