Assessing the Impact of Untraceable Bugs on the Quality of Software Defect Prediction Datasets

GORAN MAUŠA and TIHANA GALINAC GRBAC, University of Rijeka, Faculty of Engineering

The results of empirical case studies in Software Defect Prediction are dependent on data obtained by mining and linking separate software repositories. These data often suffer from low quality. In order to overcome this problem, we have already investigated all the issues that influence the data collection process, proposed a systematic data collection procedure and evaluated it. The proposed collection procedure is implemented in the Bug-Code Analyzer tool and used on several projects from the Eclipse open source community. In this paper, we perform additional analysis of the collected data quality. We investigate the impact of untraceable bugs on non-fault-prone category of files, which is, to the best of our knowledge, an issue that has never been addressed. Our results reveal this issue should not be an underestimated one and should be reported along with bugs' linking rate as a measure of dataset quality.

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: Data quality, untraceable bugs, fault-proneness

1. INTRODUCTION

Software Defect Prediction (SDP) is a widely investigated area in the software engineering research community. Its goal is to find effective prediction models that are able to predict risky software parts, in terms of fault proneness, early enough in the software development process and accordingly enable better focusing of verification efforts. The analyses performed in the environment of large scale industrial software with high focus on reliability show that the faults are distributed within the system according to the Pareto principle [Fenton and Ohlsson 2000; Galinac Grbac et al. 2013]. Focusing verification efforts on software modules affected by faults could bring significant costs savings. Hence, SDP is becoming an increasingly interesting approach, even more so with the rise of software complexity.

Empirical case studies are the most important research method in software engineering because they analyse phenomena in their natural surrounding [Runeson and Höst 2009]. The collection of data is the most important step in an empirical case study. Data collection needs to be planned according to the research goals and it has to be done according to a verifiable, repeatable and precise procedure [Basili and Weiss 1984]. The collection of data for SDP requires linking of software development repositories

This work has been supported in part by Croatian Science Foundation's funding of the project UIP-2014-09-7945 and by the University of Rijeka Research Grant 13.09.2.2.16.
Author's address: G. Mauša, Faculty of engineering, Vukovarska 58, 51000 Rijeka, Croatia; email: gmausa@riteh.hr; T. Galinac Grbac, Faculty of engineering, Vukovarska 58, 51000 Rijeka, Croatia; email: tgalinac@riteh.hr.

In: Z. Budimac, Z. Horváth, T. Kozsik (eds.): Proceedings of the SQAMIA 2016: 5th Workshop of Software Quality, Analysis, Monitoring, Improvement, and Applications, Budapest, Hungary, 29.-31.08.2016. Also published online by CEUR Workshop Proceedings (CEUR-WS.org, ISSN 1613-0073)

that do not share a formal link [D'Ambros et al. 2012]. This is not an easy task, so the majority of researchers tend to use the publicly available datasets. In such cases, researchers rely on the integrity of data collection procedure that yielded the datasets and focus mainly on prediction algorithms. Many machine learning algorithms are demanding and hence they divert the attention of researchers from the data upon which their research and results are based [Shepperd et al. 2013]. However, the datasets and their collection procedures often suffer from various quality issues [Rodriguez et al. 2012; Hall et al. 2012].

Our past research was focused on the development of systematic data collection procedure for the SDP research. The following actions had been carried out:

—We analyzed all the data collection parameters that were addressed in contemporary related work, investigated whether there are unaddressed issues in practice and evaluated their impact on the final dataset [Mauša et al. 2015a];

—We had evaluated the weaknesses of existing techniques for linking the issue tracking repository with the source code management repository and developed a linking technique that is based on regular expressions to overcome others' limitations [Mauša et al. 2014];

—We determined all the parameters that define the systematic data collection procedure and performed an extensive comparative study that confirmed its importance for the research community [Mauša et al. 2015b];

—We developed the Bug-Code Analyzer (BuCo) tool for automated execution of data collection process that implements our systematic data collection procedure [Mauša et al. 2014].

So far, data quality was observed mainly in terms of bias that undefined or incorrectly defined data collection parameters could impose to the final dataset. Certain data characteristics affect the quality characteristics. For example, empty commit messages may lead to duplicated bug reports [Bachmann and Bernstein 2010]. That is why software engineers and project managers should care about the quality of the development process. The data collection process cannot influence these issues but it may analyse to what extent do they influence the quality of the final datasets. For example, empty commit messages may also be the reason why some bug reports remain unlinked. Missing links between bugs and commit messages lead to untraceable bugs. This problem is common in the open source community [Bachmann et al. 2010].

In this paper, we address the issue of data quality with respect to the structure of the final datasets and the problem of untraceable bugs. This paper defines untraceable bugs as the defects that caused a loss of functionality, that are now fixed, and for which we cannot find the bug-fixing commit, i.e. their location in the source code. Our research questions tend to quantify the impact of untraceable bugs on SDP datasets. Giving answer to this question may improve the assessment of SDP datasets' quality and it is the contribution of this paper. Hence, we propose several metrics to estimate the impact of untraceable bugs on the fault-free category of software modules and perform a case study on 35 datasets that represent subsequent releases of 3 major Eclipse projects. The results revealed that the untraceable bugs may impact a significant amount of software modules that are otherwise unlinked to bugs. This confirms our doubts that the traditional approach, which pronounces the files that are unlinked to bugs as fault-free, may be lead to incorrect data.

2. BACKGROUND

Software modules are pronounced as Fault-Prone (FP) if the number of bugs is above a certain threshold. Typically, this threshold is set to zero. The software units that remained unlinked to bugs are typically declared as Non-Fault-Prone (NFP). However, this may not be entirely correct if there exists a certain amount of untraceable bugs. This is especially the case in projects of lower maturity level. No mater which linking technique is used in the process of data collection from open source projects, all the bugs from the issue tracking repository are never linked. Therefore, it is important to report the linking rate, i.e. the proportion of successfully linked bugs, to reveal the quality of the dataset. Linking rate is usually improved with the maturity of the project, but it never reaches 100%. Instead, we can expect to link between 20% and 40% bugs in the earlier releases and up to 80% - 90% of bugs in the "more mature", later releases [Mauša et al. 2015a; Mizuno et al. 2007; Gyimothy et al. 2005; Denaro and Pezze 2002]. Moreover, an Apache developer identified that a certain amount of bugs might even be left out from the issue tracking system [Bachmann et al. 2010].

Both of these data issues reveal that there is often a number of untraceable bugs present in the open source projects, i.e. a serious data quality issue. So far, the problem of untraceable bugs was considered only in studies that were developing linking techniques. For example, the ReLink tool was designed with the goal to find the missing links between bugs and commits [Wu et al. 2011]. However, our simpler linking technique based on regular expressions performed equally good or better than the ReLink tool and it did not yield false links [Mauša et al. 2014; Mauša et al. 2015b]. The bugs that remained unlinked could actually be present in the software units that remained unliked and, thus, disrupt the correctness of the dataset. Thus, it may be incorrect to declare all the software units not linked to bugs as NFP. To the best of our knowledge, this issue remained unattended so far. Nonetheless, there are indications that lead us to believe that the correctness of SDP datasets that is deteriorated by untraceable bugs can be improved. Khoshgoftaar et al. collected the data for SDP from a a very large legacy telecommunications system and found that more than 99% of the modules that were unchanged from the prior release had no faults [Khoshgoftaar et al. 2002; Khoshgoftaar and Seliya 2004].

3. CASE STUDY METHODOLOGY

We use the GQM (Goal-Question-Metrics) approach to state the precise goals of our case study. Our **goal** is to obtain high quality of data for SDP research. To achieve this goal, we have already analysed open software development repositories, investigated existing data collection approaches, revealed issues that could introduce bias if left open to interpretation and defined a systematic data collection procedure. The data collection procedure was proven to be of high quality [Mauša et al. 2015b]. However, a certain amount of untraceable bugs is always present. If such a bug actually belongs to a software module that is otherwise unlinked to the remaining bugs, than it would be incorrect to pronounce such a software module as fault-free.

3.1 Research questions

Research questions that drive this paper are related to the issue of untraceable bugs and their impact on the quality of data for SDP research. To accomplish the aforementioned goal, we need to answer the following **research questions** (**RQ**):

(1) How many fixed bugs remain unlinked to commits?

(2) How many software modules might be affected by the untraceable bugs?

(3) How important it is to distinguish the unchanged software modules from other modules that remain unlinked to bugs?

The bug-commit linking is done using the Regex Search linking technique, implemented in the BuCo tool. This technique proved to be better than other existing techniques, like the ReLink tool [Mauša et al. 2014], and the collection procedure within the BuCo tool has shown to be more precise than other existing procedures, like the popular SZZ approach [Mauša et al. 2015a]. Using this technique

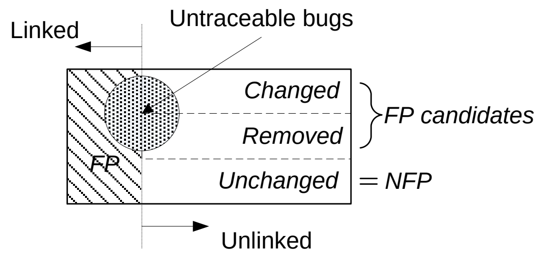

Fig. 1. Categories of files in a SDP dataset

we minimize the amount of bugs that are untraceable. Furthermore, BuCo tool uses the file level of granularity and software modules are regarded as files in the remainder of the paper.

3.2 Metrics

We propose several **metrics** to answer our research questions. The metric for the **RQ1** is the linking rate (**LR**), i.e. the ratio between the number of successfully linked bugs and the total number of relevant bugs from the issue tracking repository. The metrics for the **RQ2** and **RQ3** are defined using the following categories of software modules:

—**FP** – files linked with at least one bug

—**Unlinked** – files not linked to bugs

—**Changed** – files for which at least one of 50 product metrics is changed between two consecutive releases *n+1* and *n*

—**Removed** – files that are present in release *n*, and do not exist in release *n+1*

—**FP_Candidates** – *Unlinked* files that are *Changed* or *Removed*

—**NFP** – *Unlinked* files that are not *Changed* nor *Removed*

The relationship between these categories of files are presented in Figure 1. No previously published related research investigated the category of Non-Fault-Prone (NFP) files. It is reasonable to assume they categorized the *Unlinked* category as *NFP*. However, the linking rate that is below 100% reveals that there is a certain amount of untraceable bugs and we know that a file might be changed due to an enhancement requirement and\or a bug. Hence, we conclude that some of the *Unlinked* files that are *Changed* or *Removed* might be linked to these untraceable bugs, and categorize them as **FP_Candidates**. The *Unlinked* files that are not *Changed* are the ones for which we are more certain that they are indeed Non-Fault-Prone. Thus, we categorize only these files as *NFP*. This approach is motivated by Khoshgoftaar et al. [Khoshgoftaar et al. 2002; Khoshgoftaar and Seliya 2004] as explained in section 2. Using the previously defined categories of files, we define the following metrics:

$$C\_U = FP\_Candidates/Unlinked \tag{1}$$

The FP_Candidates in Unlinked (**C_U**) metric reveals the structure of *Unlinked* files, i.e. what percentage of *Unlinked* files is potentially affected by untraceable bugs. This metric enables us give an estimation for our **RQ2**.

$$FpB = FP/Linked\_bugs \tag{2}$$

The **F**iles per **B**ug (**FpB**) metric reveals the average number of **different** files that are affected by one bug. It should be noted that the bug-file cardinality is many-to-many, meaning that one bug may be linked to more than one file and one file may be linked to more than one bug. Hence, the untraceable bugs could be linked to files that are already FP, but we want to know how many of the *Unlinked* files they might affect. Therefore, we divide the total number of FP files (neglecting the number of established links per file) with the total number of linked bugs.

$$Ub\_U = FpB * Untraceable\_bugs/Unlinked \tag{3}$$

The **U**ntraceable **b**ugs in **U**nlinked (**Ub_U**) metric estimates the proportion of *Unlinked* files that are likely to be linked to untraceable bugs, assuming that all the bugs behave according to the FpB metric. This metric enables us give another estimation for our **RQ2**. It estimates how wrong would it be to pronounce all the *Unlinked* files as *NFP*. The greater is the value of metric Ub_U, the more wrong is that traditional approach. We must point out that there are also bugs that are not even entered into the BT repository. However, the influence of this category of untraceable bugs cannot be estimated, but it could only increase the value of Ub_U.

$$Ub\_C = FpB * Untraceable\_bugs/FP\_Candidates \tag{4}$$

The **U**ntraceable **b**ugs in FP_**C**andidates (**Ub_C**) metric estimates the percentage of *FP_Candidates* that are likely to be linked to untraceable bugs (Ub_U/C_U), assuming that all the bugs behave according to the FpB measure. This metric enables us give an estimation for our **RQ3**. It estimates how wrong it would be to pronounce all the *FP_Candidates* as *NFP*. The closer is the value of this metric to 1, the more precisely would it be not to pronounce the *FP_Candidates* as *NFP*. In other words, the Ub_C metric calculates the percentage of files that are likely to be FP among the FP_Candidates (Ub_U/C_U).

3.3 Data

The source of data are three major and long lasting open source projects from the Eclipse community: JDT, PDE and BIRT. The bugs that satisfy following criteria are collected from the Bugzilla repository: status - closed, resolution - fixed, severity - minor or above. The whole source code management repositories are collected from the GIT system. Bugs are linked to commits using the BuCo Regex linking technique and afterwards the commits are to files that were changed. The cardinality of the link between bugs and commits is many-to-many, and the duplicated links between bugs and files are counted only once. The file level of granularity is used, test and example files are excluded from final datasets, and the main public class is analyzed in each file. A list of 50 software product metrics is calculated for each file using the *LOC Metrics*[1] and *JHawk*[2] tools.

4. RESULTS

Table I shows the total number of releases and files we collected; FP, NFP, *Changed* and *Removed* files we identified; the total number of relevant bugs from the issue tracking repository; the linking rate obtained by BuCo Regex linking technique; and the total number of commits in the source code management repository. The results of our linking technique are analysed for each project release and presented in Table II. The LR exhibits a rising trend in each following release and reaches stable and high values (80% - 90%) in the "middle" releases. A slight drop in LR is possible in the latest releases.

[1]http://www.locmetrics.com/
[2]http://www.virtualmachinery.com/

	JDT	**PDE**	**BIRT**
Releases analyzed	12	12	6
Files analyzed	52,033	23,088	31,110
FP	4,891	5,307	5,480
NFP	27,891	12,059	14,760
Changed	13,443	4,208	10,803
Removed	917	1,930	67
Bugs collected	18,404	6,698	8,000
Bugs linked	13,193	4,189	4,761
Commits	151,408	23,427	**75,216**

Table I. Raw Data Analysis

	JDT			**PDE**			**BIRT**		
Release	Overall	Linked Bugs		Overall	Linked Bugs		Overall	Linked Bugs	
1	4276	2068	48.4%	561	125	22.3%	2034	887	43.6%
2	1875	1208	64.4%	427	117	27.4%	596	314	52.7%
3	3385	2401	70.9%	1041	350	33.6%	2630	1590	60.5%
4	2653	2137	80.6%	769	397	51.6%	1761	1201	68.2%
5	1879	1595	84.9%	546	378	69.2%	807	649	80.4%
6	1341	1189	88.7%	727	620	85.3%	172	120	69.8%
7	989	890	90.0%	963	779	80.9%	25	8	32.0%
8	595	546	91.8%	879	764	86.9%	28	5	17.9%
9	492	436	88.6%	454	391	86.1%	51	16	31.4%
10	549	399	72.7%	204	174	85.3%			
11	329	288	87.5%	62	48	77.4%			
12	41	36	87.8%	65	46	70.8%			
13	348	312	89.7%	131	125	95.4%			

Table II. Bug Linking Analysis

However, observing the absolute value of bugs in those releases, we notice the difference is less severe. As these releases are still under development, new bugs are being fixed and new commits still arrive so this rates are expected to change. These results show that a considerable amount of bugs is untraceable and indicate that their influence may not be insignificant.

The distributions of four file categories are computed for each release of every project and presented in stacked column representations in Figures 2, 3 and 4. We confirm that the problem of class imbalance between FP and *Unlinked* files (*Changed*, *Removed* and NFP) is present in all the releases. The percentage of FP files is usually below 20%, on rare occasions it rises up to 40% and in worst case scenarios it drops even below 5%. The trend of FP files is dropping as the project becomes more mature in the later releases. The NFP files are rare in the earlier releases of the projects showing that the projects are evidently rather unstable then. Their percentage is rising with almost every subsequent release and it rises to rates comparable to the FP category in the ”middle” releases. The *Removed* files are a rather insignificant category of files. The *Changed* files are present in every release and they exhibit a more stable rate than the other categories.

Tables III, IV and V present the evaluation metrics which we proposed in section 3.2. Metric C_U reveals the relative amount of files that are not linked to bugs, but have been changed in the following release. Because of the untraceable bugs, we cannot be certain about their fault proneness. We

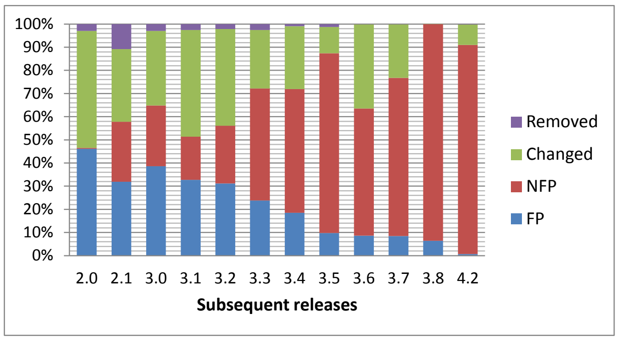

Fig. 2. Distribution of files in JDT releases

Release	C_U	FpB	Ub_U	Ub_C
2.0	99.2%	0.54	91.5%	92.2%
2.1	61.9%	0.73	25.9%	41.8%
3.0	57.2%	0.55	25.7%	45.0%
3.1	72.3%	0.60	11.8%	16.3%
3.2	63.8%	0.89	8.1%	12.6%
3.3	36.5%	0.97	4.0%	10.9%
3.4	34.4%	1.03	2.5%	7.4%
3.5	14.0%	0.90	1.0%	6.9%
3.6	39.9%	0.99	1.2%	3.0%
3.7	25.4%	1.07	3.5%	13.7%
3.8	0.0%	1.14	1.0%	2334.7%
4.2	9.0%	1.03	0.1%	1.1%

Table III. Evaluation Metrics for JDT

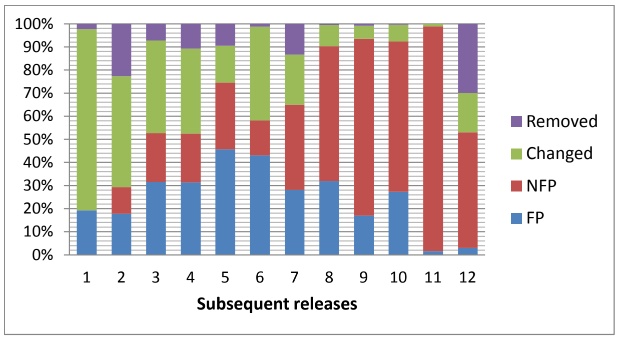

Fig. 3. Distribution of files in PDE releases

Release	C_U	FpB	Ub_U	Ub_C
2.0	100.0%	0.92	83.4%	83.4%
2.1	86.0%	1.20	57.5%	66.8%
3.0	69.1%	0.83	91.0%	131.8%
3.1	69.2%	0.90	42.8%	61.8%
3.2	46.9%	1.66	37.4%	79.7%
3.3	73.4%	1.22	13.1%	17.8%
3.4	48.7%	0.79	9.2%	18.9%
3.5	14.2%	0.98	7.1%	49.7%
3.6	7.9%	1.07	3.3%	42.0%
3.7	10.5%	3.83	6.5%	61.8%
3.8	1.1%	1.21	0.5%	43.4%
4.2	48.4%	2.30	1.3%	2.6%

Table IV. Evaluation Metrics for PDE

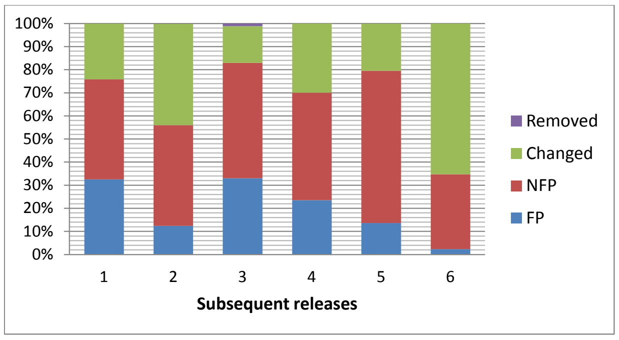

Fig. 4. Distribution of files in BIRT releases

Release	C_U	FpB	Ub_U	Ub_C
2.0.0	35.7%	0.92	62.2%	174.3%
2.1.0	50.2%	1.33	12.7%	25.2%
2.2.0	25.4%	1.09	32.2%	126.8%
2.3.0	39.1%	1.19	14.3%	36.6%
2.5.0	23.7%	1.41	3.8%	16.2%
2.6.0	66.9%	1.40	1.0%	1.6%

Table V. Evaluation Metrics for BIRT

notice a significant amount of such data. Metric FpB reveals the average number of distinct files that are changed per bug. The metric is based upon the number of bugs that were successfully linked to commits. Considering that multiple bugs may affect the same files, it is not unusual that one bug on average affects less than 1 distinct file. Later releases have less bugs in total, there is less chance that they affect the same files and there is a slight increase in the value of FpB. The FpB metric is used to estimate the amount of files prone to bugs that were untraceable from the bug tracking repository, expressed in the metric Ub_U. The Ub_U metric varies between releases, from very significant in earlier releases to rather insignificant in the later releases. The Ub_C metric reveals how important would it be to distinguish *Changed* and *Removed* files from the NFP files. With its values close to 0%, we expect little bias in the category of NFP files. However, with its greater value, the bias is expected to rise and the necessity to make such a distinction is becoming greater. In several cases, its value exceeds 100%. This only shows that the impact of untraceable bugs is assessed to be even greater than affecting just the FP_Candidates. In the case of JDT 3.8 its value is extremely high because this release contains almost none FP_Candidates. This metric was developed on our own so we cannot define a significance threshold. Nevertheless, we notice this value to be more emphasized in earlier releases that we described as immature and in later releases that are still under development.

4.1 Discussion

Linking rate (LR) enables us to answer our **RQ1**. We noticed that the LR is very low in the earliest releases of analyzed projects (below 50%). After a couple of releases, the LR can be expected to be between 80% and 90%. We also observe that the distribution of FP files exhibits a decreasing trend as the product evolves through releases. That is why we believe that developer are maturing along with the project and, with time, they become less prone to faults and more consistent in reporting the Bug IDs in the commit titles when fixing bugs. The latest releases are still under development and exhibit extreme levels of data imbalance, with below 1% of FP files. Therefore, these datasets might not be the proper choice for training the predictive models in SDP.

Our results enable us to give an estimate for the **RQ2**. The *Unlinked* files contain a rather significant ratio of files that are FP candidates, spanning from 10% up to 50% for the JDT and BIRT projects and above 50% in several releases of the PDE project. Among the FP candidates, we expect to have a more significant amount of files that are FP due to the untraceable bugs in earlier releases because of low LR. According to the Ub_C metric, we may expect that the majority of *FP Candidates* actually belong to the FP category in the earliest releases. According to the Ub_U metric, the untraceable bugs affect a rather insignificant percentage of all the *Unlinked* files after a couple of releases.

The metrics we proposed in this paper enable us to answer our **RQ3**. The difference between the Ub_U and Ub_C values confirm the importance of classifying the *Unlinked* files into *Changed*, *Removed* and NFP. In the case of high Ub_C values (above 80%) it may be prudent to categorize *FP Candidates* as FP and in the case where Ub_C is between 20% and 80% it may be prudent to be cautious and not to use the *FP Candidates* at all. In the case of high difference between the Ub_U and Ub_C metrics, we may expect to have enough of NFP files in the whole dataset even if we discard the *FP Candidates*. This is confirmed in the distribution of NFP files which displays an increasing trend which becomes dominant and rather stable in the ”middle” releases.

The process of data collection and analysis is fully repeatable and verifiable but there are some threats to validity of our exploratory case study. The construction validity is threatened because the data do not come from industry and the external validity is threatened because only one projects come from only one community. However, the chosen projects are large and long lasting ones and provide a good approximation of the projects from the industrial setting and they are widely analyzed in related research. Internal validity is threatened by assumptions that all the bugs affect the same quantity of different files and that *Unchanged* files are surely NFP.

5. CONCLUSION

The importance of having accurate data is the initial and essential step in any research. This paper is yet another step in achieving that goal in the software engineering area of SDP. We noticed that untraceable bugs are inevitable in data collection from open source projects and that this issue remained unattended by the researchers so far. This exploratory case study revealed that it may be possible to evaluate the impact of untraceable bugs on the files that are unlinked to bugs. The results show that the earliest and the latest releases might not be a good source of data for building predictive models. The earliest releases are more prone to faults (containing higher number of reported bugs), radical changes (containing almost no unchanged files) and suffer from low quality of data (lower linking rate). On the other hand, the latest releases suffer from no previously mentioned issues, but are evidently still under development and the data are not stable.

The future work plans to investigate the impact of the explored issues and the proposed solutions to the problem of untraceable bugs on the performance of predictive models. Moreover, we plan to expand this study to other communities using our BuCo Analyzer tool.

REFERENCES

A. Bachmann and A. Bernstein. 2010. When process data quality affects the number of bugs: Correlations in software engineering datasets. In *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*. 62–71. DOI:http://dx.doi.org/10.1109/MSR.2010.5463286

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

Victor R. Basili and David Weiss. 1984. A methodology for collecting valid software engineering data. *IEEE Computer Society Trans. Software Engineering* 10, 6 (1984), 728–738.

Marco D'Ambros, Michele Lanza, and Romain Robbes. 2012. Evaluating Defect Prediction Approaches: A Benchmark and an Extensive Comparison. *Empirical Softw. Engg.* 17, 4-5 (2012), 531–577.

Giovanni Denaro and Mauro Pezze. 2002. An empirical evaluation of fault-proneness models. In *Proceedings of the Int'l Conf. on Software Engineering*. 241–251.

Norman E. Fenton and Niclas Ohlsson. 2000. Quantitative Analysis of Faults and Failures in a Complex Software System. *IEEE Trans. Softw. Eng.* 26, 8 (2000), 797–814.

Tihana Galinac Grbac, Per Runeson, and Darko Huljenić. 2013. A Second Replicated Quantitative Analysis of Fault Distributions in Complex Software Systems. *IEEE Trans. Softw. Eng.* 39, 4 (April 2013), 462–476.

Tibor Gyimothy, Rudolf Ferenc, and Istvan Siket. 2005. Empirical Validation of Object-Oriented Metrics on Open Source Software for Fault Prediction. *IEEE Trans. Softw. Eng.* 31, 10 (Oct. 2005), 897–910.

Tracy Hall, Sarah Beecham, David Bowes, David Gray, and Steve Counsell. 2012. A Systematic Literature Review on Fault Prediction Performance in Software Engineering. *IEEE Trans. Softw. Eng.* 38, 6 (2012), 1276–1304.

Taghi M. Khoshgoftaar and Naeem Seliya. 2004. Comparative Assessment of Software Quality Classification Techniques: An Empirical Case Study. *Empirical Software Engineering* 9, 3 (2004), 229–257.

Taghi M. Khoshgoftaar, Xiaojing Yuan, Edward B. Allen, Wendell D. Jones, and John P. Hudepohl. 2002. Uncertain Classification of Fault-Prone Software Modules. *Empirical Software Engineering* 7, 4 (2002), 295–295.

Goran Mauša, Tihana Galinac Grbac, and Bojana Dalbelo Bašić. 2014. Software Defect Prediction with Bug-Code Analyzer - a Data Collection Tool Demo. In *Proc. of SoftCOM '14*.

Goran Mauša, Tihana Galinac Grbac, and Bojana Dalbelo Bašić. 2015a. Data Collection for Software Defect Prediction an Exploratory Case Study of Open Source Software Projects. In *Proceedings of MIPRO '14*. Opatija, Croatia, 513–519.

Goran Mauša, Tihana Galinac Grbac, and Bojana Dalbelo Bašić. 2015b. A Systemathic Data Collection Procedure for Software Defect Prediction. 12, 4 (2015), to be published.

Goran Mauša, Paolo Perković, Tihana Galinac Grbac, and Ivan Štajduhar. 2014. Techniques for Bug-Code Linking. In *Proc. of SQAMIA '14*. 47–55.

Osamu Mizuno, Shiro Ikami, Shuya Nakaichi, and Tohru Kikuno. 2007. Spam Filter Based Approach for Finding Fault-Prone Software Modules.. In *MSR*. 4.

D. Rodriguez, I. Herraiz, and R. Harrison. 2012. On software engineering repositories and their open problems. In *Proceedings of RAISE '12*. 52–56. DOI: http://dx.doi.org/10.1109/RAISE.2012.6227971

Per Runeson and Martin Höst. 2009. Guidelines for Conducting and Reporting Case Study Research in Software Engineering. *Empirical Softw. Engg.* 14, 2 (April 2009), 131–164. DOI: http://dx.doi.org/10.1007/s10664-008-9102-8

Martin J. Shepperd, Qinbao Song, Zhongbin Sun, and Carolyn Mair. 2013. Data Quality: Some Comments on the NASA Software Defect Datasets. *IEEE Trans. Software Eng.* 39, 9 (2013), 1208–1215.

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.