2014 14th IEEE International Working Conference on Source Code Analysis and Manipulation

Supplementary Bug Fixes vs. Re-opened Bugs

Le An, Foutse Khomh, Bram Adams
SWAT–MCIS, Polytechnique Montréal, Québec, Canada
{le.an, foutse.khomh, bram.adams}@polymtl.ca

***Abstract*—A typical bug fixing cycle involves the reporting of a bug, the triaging of the report, the production and verification of a fix, and the closing of the bug. However, previous work has studied two phenomena where more than one fix are associated with the same bug report. The first one is the case where developers re-open a previously fixed bug in the bug repository (sometimes even multiple times) to provide a new bug fix that replace a previous fix, whereas the second one is the case where multiple commits in the version control system contribute to the same bug report ("supplementary bug fixes"). Even though both phenomena seem related, they have never been studied together, *i.e.*, are supplementary fixes a subset of re-opened bugs or the other way around? This paper investigates the interplay between both phenomena in five open source software projects: Mozilla, Netbeans, Eclipse JDT Core, Eclipse Platform SWT, and WebKit. We found that re-opened bugs account for between 21.6% and 33.8% of all supplementary fixes. However, 33% to 57.5% of re-opened bugs had only one commit associated, which means that the original bug report was prematurely closed instead of fixed incorrectly. Furthermore, we constructed predictive models for re-opened bugs using historical information about supplementary bug fixes with a precision between 72.2% and 97%, as well as a recall between 47.7% and 65.3%. Software researchers and practitioners who are mining data repositories can use our approach to identify potential failures of their bug fixes and the re-opening of bug reports.**

***Index Terms*—Supplementary fixes, re-opened bugs, prediction model, mining software repositories**

I. Introduction

According to a report by the US Department of Commerce [11], bug fixing accounts for up to 80% of software development costs. Part of the reason for this is that a typical bug fixing cycle includes many different phases, performed by a variety of stakeholders: reporting of the bug, production of a fix, verification of the fix, and closing of the bug definitively. In some cases, developers even have to try multiple times before fixing a bug. As a result of these several attempts, bug reports are sometimes re-opened, which requires even more time for a bug to be fixed and hence is likely to degrade the satisfaction of users and decrease the productivity of development teams, as developers have to rework the same bug multiple times: re-analyzing the context of the bug, reading previous discussions about the bug and examining previous failed fixes (proposed for the bug). Thus, it is important to identify flawed bug fixes early before they can crash in the field.

Work on failed bug fixes has focused on two areas, *i.e.*, supplementary bug fixes and re-opened bugs. Supplementary bug fixes correspond to multiple commits linked (via their commit log message) to the same bug report. Park et al. [12] investigated supplementary fixes in three open source projects: Eclipse JDT core, Eclipse SWT, and Mozilla. They conclude that supplementary fixes are typically caused by forgetting to port changes, by incorrect handling of conditional statements, or by incomplete refactorings. On the other hand, the work on re-opened bugs analyzes bug reports that have been closed at least once and re-opened again later, possibly replacing an old bug fix by a newer (possibly more correct) one. Shihab et al. [13], Zimmermann et al. [17], and Xia et al. [15] proposed models for the prediction of such re-opened bugs. Although both areas obviously are related and have spawned two active research communities, their exact relation has never been studied: are re-opened bugs a subset of supplementary bug fixes (or the other way around)?

This paper analyzes the relation between supplementary bug fixes and re-opened bugs by studying the factors that indicate whether a bug fix will require supplementary fixes and–or will be re-opened. Knowing the characteristics of fixes that require supplementary fixes will help to better focus code review activities and prevent known bugs from re-appearing in the field. Knowing the characteristics of bugs that require to be re-opened will help to predict the probability of bug re-opening in order to reduce the maintenance overhead and improve the overall quality of software. Using bug fix and bug re-opening information from five open source software projects, Mozilla, Netbeans, Eclipse JDT Core, Eclipse Platform SWT and WebKit, we address the following three research questions:

RQ1: What is the proportion of bugs among all bug reports that require supplementary bug fixes or are re-opened?

This research question replicates the work of Park et al. [12], who analyzed Eclipse JDT core, Eclipse SWT, Mozilla and found that between 22.5% to 32.8% of resolved bugs involved more than one fixing attempt. In this research, we want to verify whether supplementary fixes are related to frequent failure, and hence, whether they are worth investigating in details. We find that the proportion of bugs that required supplementary bug fixes in Mozilla[1], Netbeans[2], Eclipse JDT Core[3], Eclipse Platform SWT[4] and WebKit[5] accounts for, respectively, 23.8%, 17.2%, 26.9%, 25.9% and 10.3%

[1] http://www.mozilla.org/
[2] https://netbeans.org/
[3] http://www.eclipse.org/jdt/core/
[4] http://www.eclipse.org/swt/
[5] https://www.webkit.org

978-1-4799-6148-1/14 $31.00 © 2014 IEEE
DOI 10.1109/SCAM.2014.29

of the total number of resolved bugs reports. Only the results for Webkit are not similar to those of Park et al. We attribute the difference to the style of commit messages in this project where many commits cannot be mapped to their corresponding bug reports.

RQ2: What is the relation between supplementary bug fixes and re-opened bugs?

We want to understand whether bug fix failures are caught early during reviews and testing activities or whether they slip through these verification processes and crash in the field, prompting the re-opening of bug reports. According to our result, between 21.6% and 33.8% of supplementary fixes have been re-opened at least once. In addition, bug re-openings tend to coincide with multiple fixing attempts, long fixing period and multiple developers. Surprisingly, we also found that, contrary to intuition, 33% to 57.5% of the re-opened bugs were not detected as supplementary fixes, instead they are mostly due to premature closing of bugs.

RQ3: Can we predict the re-opening of supplementary bug fixes?

Re-opened bugs may increase maintenance costs, degrade the overall software quality and the satisfaction of users [13]. In this research question, we use GLM, C5.0, ctree, cforest and randomForest [3] algorithms with attributes about developers' working habits, commit logs, bug fix, and development teams' dynamic, to build models that can predict whether or not a bug that required supplementary fixes before initial closing of its report will be re-opened. Our models can correctly predict whether or not a supplementary fix will need to be re-opened with a precision between 72.2% and 97% and a recall between 47.7% and 65.3%. Software organizations could use our proposed models to predict potential failures of their bug fixes and the re-opening of bug reports, hence preventing these bugs from re-appearing in the field.

The rest of this paper is organized as follows. Section II describes the design of our case study. Section III describes and discusses the results of our three research questions. Section IV discusses the results of our replication study in the context of previous work. Section V discloses the threats to validity of our study. Section VI summarizes related work. Section VII concludes the paper.

II. Study Design

This section presents the design of our case study, which aims to address the following three research questions:

RQ1: What is the proportion of bugs among all bug reports that require supplementary bug fixes or are re-opened?

RQ2: What is the relation between supplementary bug fixes and re-opened bugs?

RQ3: Can we predict the re-opening of supplementary bug fixes?

A. Data Collection

Since our study replicates existing work on supplementary bug fixes [12] and re-opened bugs [13], we selected the following five open source software projects: Mozilla, Netbeans, Eclipse JDT Core, Eclipse Platform SWT, and WebKit. Mozilla, which was also used by Park et al. [12], is a web project that includes several sub-products, such as the Firefox Internet browser and the Thunderbird e-mail client. Eclipse, which was used by both Park et al. and Shihab et al. [13], is an integrated development environment (IDE) supporting various programming languages. In addition, to compare with the results in [12] and [13], we introduced two other projects: Netbeans and WebKit. Similar to Eclipse, Netbeans is another commonly used IDE. WebKit is a layout engine software component for rendering web pages that powers Apple's safari browser.[6]

B. Data Processing

Figure 1 shows an overview of our analysis approach. First, we extract bug fix information from version control systems (*i.e.*, Mercurial and Git) and apply the algorithm of Park et al. to identify supplementary bug fixes [12]. Then, we mine the bug repositories (*i.e.*, Bugzilla) of our five subject projects to identify re-opened bugs. Using these data, we compute several metrics and build statistical models to predict the re-opening probability of supplementary bugs fixes. The remainder of this section elaborates on each of these steps.

1) Identification of bug fixes: We extract the revision history of each subject project from the *Mercurial* (for Mozilla and Netbeans) and *Git* (for Eclipse projects, and WebKit) repositories. We obtained the data of the three repositories Mozilla, Netbeans and Eclipse from the MSR 2011 challenge, which respectively cover the period from March 2007 to August 2010, from January 1999 to June 2010, and from October 2001 to June 2010. The WebKit data cover the period from August 2001 to June 2014. Next, we parse the files' revision logs to extract the following commit information: revision numbers, committer names, commit dates, commit messages, number of changed files, and number of inserted/deleted lines. We apply heuristics from Fischer et al. [6] to identify bug fixing commits. More specifically, we apply the following regular expressions incrementally to match bug report identifiers:

```
(bug|issue)[:#\s_]*[0-9]+
(b=|#)[0-9]+
[0-9]+\b
\b[0-9]+
```

Finally, we cross-check the bug IDs obtained from commit logs with the Bugzilla repository to ensure that they represent actual bug reports. *i.e.*, check whether the extracted bug IDs exist in the corresponding Bugzilla repository.

2) Identification of supplementary bug fixes: We apply the algorithm proposed by Park et al. [12] to track supplementary

[6] All our studied data repositories, and analysis scripts are available here: https://github.com/anlepoly/supplementary_fixes

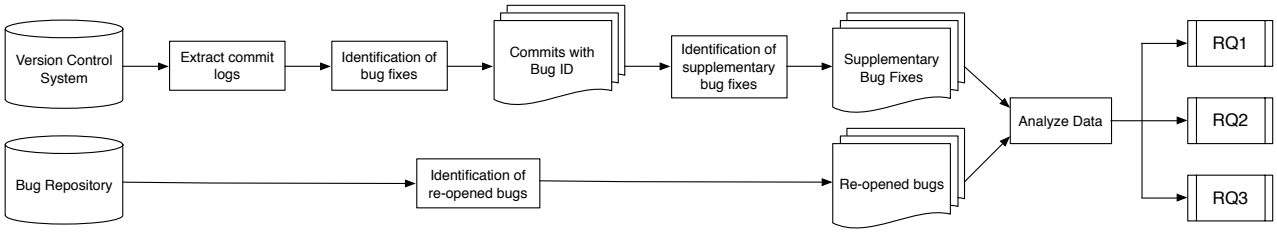

Figure 1: OVERVIEW OF OUR APPROACH TO STUDY THE RELATION BETWEEN SUPPLEMENTARY FIXES AND RE-OPENED BUGS

bug fixes. This algorithm considers as a supplementary bug fix, any fix where the commit message contains the bug ID of a previous bug fixing commit. Therefore, among all detected bug fixing commits, we search for revisions where the bug ID is repeated. During this process, we observed that in some commit messages, committers just mentioned the revision number of a previous bug fix instead of the bug ID. Hence, we enhance Park et al.'s heuristic by also matching these revisions to the corresponding bugs.

Table I presents an example of supplementary bug fixes. In this table, there are three revisions that mention the same bug ID #462381. Revision 21149 is the initial bug fix, while revisions 34890 and 34902 are supplementary bug fixes.

Table I: SUPPLEMENTARY BUG FIXES OF BUG #462381

changeset	21149:7aeaf064ad9f
date	Fri Oct 31 09:07:15 2008 -0700
summary	**Bug 462381** - Build layout directories in parallel r=ted sr=roc
churn	12 files changed, 16 insertions(+), 464 deletions(-)
... ...	
changeset	34890:fae81b8a5648
date	Fri Nov 13 14:40:00 2009 -0500
summary	**bug 462381** - sprinkle magic PARALLEL_DIRS fairy dust about the build system r=ted.mielczarek
churn	12 files changed, 191 insertions(+), 173 deletions(-)
... ...	
changeset	34902:827d8651799e
date	Mon Nov 16 07:57:15 2009 -0500
summary	bustage fix from **bug 462381**
churn	1 files changed, 4 insertions(+), 2 deletions(-)

After the identification of supplementary bug fixes, we organize all bug fixes into two groups (similarly to [12]):

- *Type I bug fix* - bug fixes that definitively solve the bug in the first attempt (*i.e.*, no supplementary fix is needed)
- *Type II bug fix* - bug fixes that require supplementary fixes before the bug can be solved.

3) Identification of Re-opened Bugs: In Bugzilla, a re-opened bug may be marked "REOPENED" in two places: in the "status" field, when it is currently re-opened and not yet solved; and in its "history" list, if it was once re-opened but afterwards the status had been changed to something else (*e.g.*, again "CLOSED"). Instead of just looking at the final status of a bug, we check the bug's "history" list and find whether there is at lease one "REOPENED" tag. In the case of Mozilla, Netbeans, Eclipse JDT Core and Eclipse Platform SWT, we extract this information directly from the Bug SQL databases that were provided for MSR 2011 Mining Challenge. In the case of WebKit, we concatenate the Bugzilla URL with each detected bug ID to download "history" pages of the bug. Then, we check whether the tag "REOPENED" exists in the bug's history. For example, to check whether bug #32698 in WebKit was once re-opened, we combine the history link of *WebKit Bugzilla* and the target bug ID as follows:

https://bugs.webkit.org/show_bug.cgi?id=32698

III. CASE STUDY RESULTS

This section presents and discusses the results of our three research questions.

RQ1: What is the proportion of bugs among all bug reports that require supplementary bug fixes or are re-opened?

Motivation. This question is preliminary to the other questions. It provides quantitative data on the proportion of bugs that required supplementary bug fixes and bugs that have been re-opened in our five subject systems. In this research question, as in the study of Park et al. [12], we determine whether bug fixes fail frequently, how fast the bugs are fixed for good and how many developers are needed for this. These results will clarify the prevalence (and hence importance) of supplementary bug fixes, and allow us to compare our findings with those from [12].

Approach. We identify supplementary bug fixes by classifying bug fixes from the five systems into two categories: Type I bug fix and Type II bug fix, as discussed in Section II-B2. We identify re-opened bugs following the heuristics described in Section II-B3, and compute the proportion of bug reports that have been re-opened. For each bug report, we also compute the number of fixing attempts required for the bug, the duration (in days) of the fixing period, and the number of developers that contributed to fix the bug. Since type II bugs contain multiple fixes, we respectively calculate their number of bug fixes and number of bug reports (*i.e.*, all fixes corresponding to the same bug ID count for one).

Findings. **Overall, in the five studied projects, type II bug**

Table II: DESCRIPTIVE STATISTICS OF THE SUBJECT SYSTEMS

	Mozilla	Netbeans	JDT Core	Platform SWT	WebKit
Studied period	03/2007 - 08/2010	01/1999 - 06/2010	10/2001 - 06/2010	10/2001 - 06/2010	08/2001 - 06/2014
# commits	51500	173559	18099	20744	152296
# detected bug fixing commits	41227	53599	7744	8504	49388
# Type II bug fixing commits	20389 (49.5%)	19111 (35.7%)	3960 (51.1%)	4523 (53.2%)	10530 (21.3%)
# bug reports	27349	41633	5176	5374	43326
# Type I bug reports	20838 (76.2%)	34488 (82.8%)	3784 (73.1%)	3981 (74.1%)	38858 (89.7%)
# Type II bug reports	6511 (23.8%)	7145 (17.2%)	1392 (26.9%)	1393 (25.9%)	4468 (10.3%)
# re-opened bug reports	2876 (10.5%)	5681 (13.6%)	707 (13.7%)	653 (12.2%)	2311(5.3%)
Max # of fixing attempts for a bug report	97	56	24	45	36
Max # of fixing days for a bug report	1125	3781	1616	1947	889
Max # of involved developers for a bug report	6	7	14	12	6

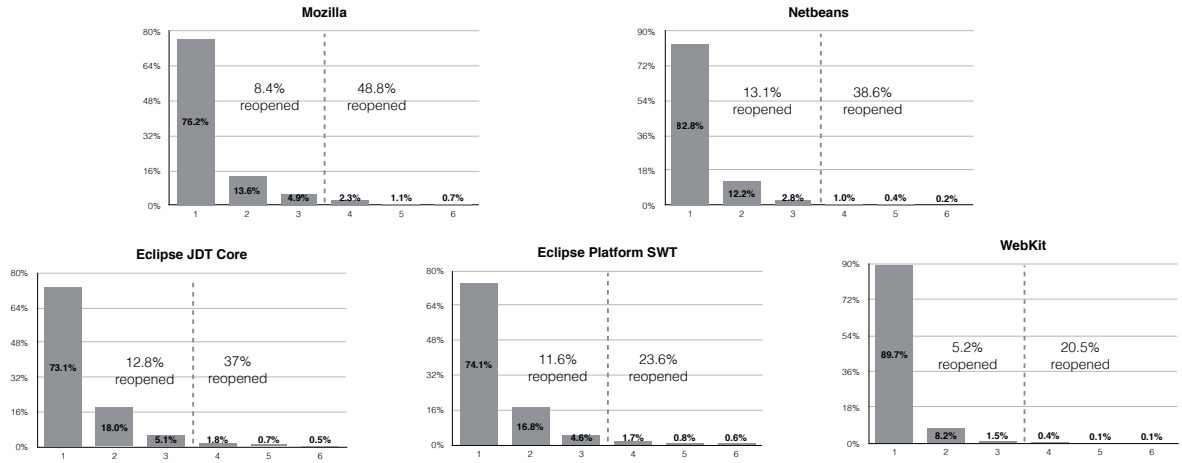

Figure 2: NUMBER OF FIXES REQUIRED FOR BUGS AS WELL AS PERCENTAGE OF BUGS THAT ARE RE-OPENED WITHIN 3 FIXING ATTEMPTS AND WITH MORE THAN 3 ATTEMPTS

reports account for 10.3% to 26.9% of all the bug reports. Table II shows descriptive statistics about our subject systems.

Netbeans has the lowest percentage of commits that fix a bug. This seems counterintuitive, because Netbeans has the highest number of commits. However, further manual analysis shows that more than 20% of commit messages only mentioned the product repository links instead of bug IDs (*e.g.*, Automated merge with http://hg.netbeans.org/cnd-main/). In other words, these commits cannot be identified as fixing a bug. There are also many very short commit messages from which we can not extract any useful information about bug fixes with the heuristic introduced in the Section II-B1. This result reveals a limitation of the current identification algorithm for supplementary bug fixes.

On average, more than one tenth of bug fixes have been re-opened. Since our re-opened bugs are detected from both VCS and bug repositories, we can guarantee that any bug fix that has been re-opened can be identified. The proportion of re-opened bugs over all detected bug fixes are similar between projects, *i.e.*, from 5.3% to 13.7%.

Most bugs required only 1 to 2 fixing attempts and less than 24 hours to get fixed. Figure 2 shows the distribution of fixing attempts required for bugs. In the worst case, in Mozilla, a bug can require up to 97 attempts before getting fixed. In other projects, we also found bugs fixed with 24 to 56 attempts. To understand the period of time needed to make the supplementary fixes, Figure 3 presents the distribution of fix duration required for bugs. Overall, most bugs are solved within 24 hours (*i.e.*, 1 day). The maximum time taken for fixing bugs is 889 to 3781 days. Some of those outliers (*e.g.*, bug #3875 in Netbeans) correspond to cases where developers forgot to close a fixed bug report (this is a threat to validity), whereas others (*e.g.*, bug #55701 in Netbeans) really took such a long time to get fixed.

In Mozilla, Netbeans, Eclipse JDT Core and Eclipse SWT, the proportion of bugs that required supplementary bug fixes is between 17.2% and 26.9%. This result is similar

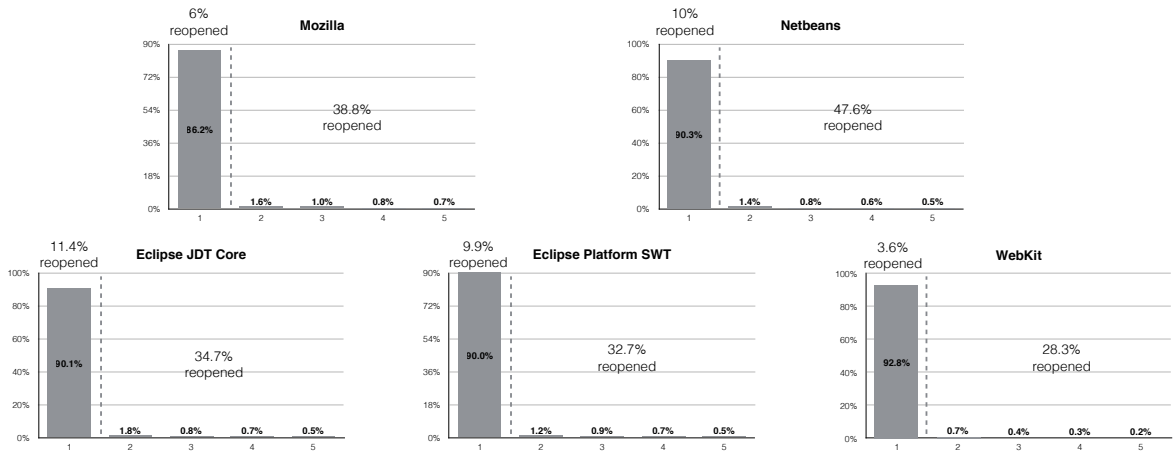

Figure 3: NUMBER OF FIXING DAYS OF BUGS AS WELL AS PERCENTAGE OF RE-OPENED BUGS THAT ARE FIXED WITHIN 1 DAY AND MORE THAN 1 DAY

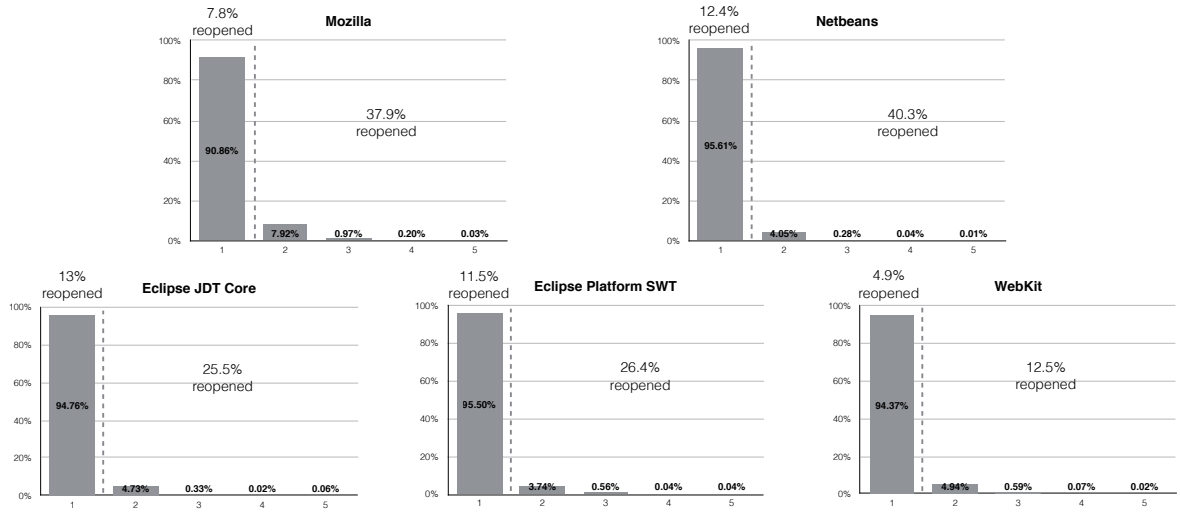

Figure 4: NUMBER OF DEVELOPERS PARTICIPATING IN FIXING BUGS AS WELL AS PERCENTAGE OF RE-OPENED BUGS THAT ARE FIXED BY ONE DEVELOPER AND BY MULTIPLE DEVELOPERS

to the finding of Park et al. [12] in which supplementary fixes account for 22.5% to 32.8% of all detected bug fixes. In Webkit, supplementary fixes account only for 10.3%. With a manual check, we found that Webkit allows developers to use both SVN and Git clients to access the source code. As a result, many commit messages mention an SVN style revision number instead of a Git revision number or a bug ID, making it difficult to track all commits related to a bug. For example, the following message could not be mapped to a bug report: "Rebaseline compositing/geometry/horizontal-scroll-composited.html after r107389". The latter number is an SVN style revision number.

Overall, in our five studied projects, supplementary bug fixes account for 10.3% to 26.9%, while re-opened bugs account for 5.3% to 13.7%.

RQ2: What is the relation between supplementary bug fixes and re-opened bugs?

Motivation. Many factors can explain the supplementary fixes found in our subject systems. A first explanation could be agile development and continuous integration practices that advocate for incremental development, in particular those that solved bugs within 24 hours, since developers may have just submitted their bug fixes incrementally (*i.e.*, through successive chunks of commits). A second explanation is suggested by the Type II bugs that experienced multiple bug fixing attempts over long period of time (up to 3781 days). It is possible that long fixing period may increase the probability of bug re-opening. A third explanation is that multiple failing attempts at fixing a bug (many supplementary bug fixes) increases the odds that the bug will be re-opened in the future. A fourth

explanation is that a bug fixing process may involve multiple committers. Multiple reasons can explain why different committers would contribute fixes for a same bug, such as the turnover in development teams, or the complexity of a bug that may require the collaboration of several developers. To verify these hypotheses, this research question investigates the relation between Type II bug fixes and re-opened bugs.

Approach. To verify the above mentioned hypotheses regarding the relation between supplementary bug fixes and re-opened bugs, we split the results in Figure 2 and 3 in two parts (by dashed lines), to distinguish bugs that required less than three fixing attempts, and those that required more than three fixing attempts (respectively bugs fixed within 24 hours, and those that required more than 24 hours). We choose these thresholds because they correspond to the modes of the distributions of the number of fixing attempts (respectively the number of fixing days). Also, bugs fixed by less than three successive commits, within 24 hours are more likely to be linked to agile development rather than incorrect bug fixes. In each figure, we then calculate the percentage of bugs below and over the above thresholds (on the left and right side of the dashed line) that are re-opened. The resulting percentages show where re-opened bugs are concentrated the most.

Figure 4 shows the distribution of the number of developers involved in fixing each bug. Dashed lines separate fixes by single developers and multiple developers. For each Type II bug, we count all different names or emails that appeared in the same Type II bug fix group (*i.e.*, all the fixing commits of a Type II bug) to identify the number of developers involved in fixing the bug. Since we extract this information from commit logs, it is possible that these developers (*i.e.*, the committers) are not the authors of the bug fixes, but are instead reviewers with commit privileges [2]. A re-opened bug may also be assigned to a different (more experienced) developer in an attempt to avoid further fixing failures. To evaluate this last hypothesis, we investigate the distribution of re-opened bugs among the groups of bugs fixed by single versus multiple developers.

Figure 5 shows the relation between supplementary bug fixes and re-opened bugs. The green circles represent type II bugs, blue circles represent re-opened bugs, pink circles represent "invalid reports" (*i.e.*, bug reports that have been closed by the following resolutions: "invalid", "wontfix", "duplicate", or "worksforme", which have a strong probability of being re-opened [13]). The overlapped parts are their intersection. For example, in Mozilla, there are 6511 type II bug reports (4583+1757+171), from which 1928 are also re-opened bugs (1757+171). Also, 171 of these re-opened bugs were "invalid reports" before being re-opened. We also found 948 re-opened bugs (324+624) with only one commit, among which 324 were "invalid reports" before being re-opened.

Findings. **Re-opened bugs are more concentrated respectively in the areas above 24 hours, three fixing attempts, or by multiple developers. Overall, between 21.6% and 33.8% of Type II bugs have been re-opened at least once. However, almost half of the re-opened bugs were not detected as Type II bug fixes (*i.e.*, we did not find more than one fix for these re-opened bugs)**. This outcome was quite a surprise for us because we expected re-opened bugs to be a subset of supplementary fixes. At first sight, this finding could be explained by limitations in our data set, such as developers forgetting to mention a bug ID in their commit message. However, closer analysis shows that 22.8% to 49.1% of re-opened bugs with only one fix tend to be linked with invalid reports, *i.e.*, not all re-opened bugs address previously fixed bugs. This seems counterintuitive, but in many cases the original bug fix was prematurely closed because developers considered that: the problem described is not a bug (marked *invalid* in Bugzilla), the bug do not need to be fixed (marked as *wontfix*), the problem is a duplicate of an existing bug (marked as *duplicate*), or all attempts at reproducing this bug were futile (marked as *worksforme*). To validate whether the invalid reports are significantly associated with single re-opened bugs, we applied Chi-squared test and Fisher's exact test to compare the four types of invalid reports in re-opened bugs with only one commit and in those with multiple commits. The result shows that in all studied systems, the p-value is less than 0.05, *i.e.*, invalid reports have a significant association with single re-opened bugs. This finding also explains that not all bug re-openings have a negative impact on software development, contrary to the conclusion of earlier works like [13]. In our subject systems, 22.8% to 49.1% of single re-opened bugs (*i.e.*, re-opened bugs with only one commit associated) have at least one of these invalid closed status. Those bugs have less impact on software quality than the re-opened bugs previously closed by the "fixed" status. Therefore, instead of building predictive models for bug re-opening over all bug fixes, like in [13], we only predict bug re-opening for supplementary bug fixes.

Therefore, bugs fixed during long period, with multiple attempts or with multiple developers tend to be re-opened. Counterintuitively, almost half of the re-opened bugs have only one fixing attempt. 22.8% to 49.1% of these single re-opened bugs are due to prematurely closed reports.

RQ3: Can we predict the re-opening of supplementary bug fixes?

Motivation. In **RQ1** and **RQ2**, we observed that between 10.3% and 26.9% of bugs required at least one supplementary fix before they were resolved for good. Among these bugs that required supplementary fixes, between 21.6% and 33.8% were re-opened. Bug fix failures, and most of the re-opened bugs are not desirable since they increase maintenance costs, degrade software quality and users' satisfaction [13]. For example, the average time from bug report to bug closing in one of the Eclipse projects for re-opened bugs was found to be as much as twice the average time to resolve a non-reopened bug [13]. In this research question, we replicate the work of Shihab et al. [13] to explore statistical models to predict whether or not a bug that required supplementary fixes will be re-opened. Using a prediction model, development teams will be able to target faulty/incomplete bug fixes for more thorough reviews, preventing re-opened bugs.

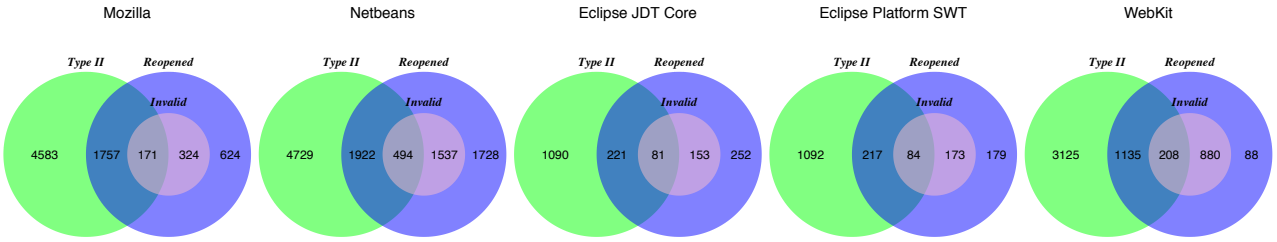

Figure 5: RELATIONSHIP BETWEEN SUPPLEMENTARY BUGS AND RE-OPENED BUGS

Table III: WORK HABIT DIMENSION

Attribute	Explanation and Rationale
Hour	Hour (0-24). Fix committed at certain hours may induce bug re-opening (*e.g.*, hours around quitting time).
Week day	Day of week (from Mon to Sun). Fix committed on certain week days may induce bug re-opening (*e.g.*, Friday) [14], [1].
Month day	Day in month (1-31). Fix committed on certain days may induce bug re-opening (*e.g.*, some dates before holidays).
Month	Month of year (1-12). Fix committed in some months may induce bug re-opening (*e.g.*, December, when we have Chrismas)
Day of year*	Day of year (1-366). Combining the rationales of month day and month.
Commit Size	Words in commit message. Too short (due to hasty work) or too long message (due to the difficulty) may lead to bug re-opening.

** this attribute was eliminated according to the VIF result*

Table IV: BUG REPORT DIMENSION

Attribute	Explanation and Rationale
Platform	Platform (*e.g.*, PC, Mac) on which the bug was reported. Bugs on some platforms are difficult to be solved, which may induce bug re-opening.
Severity	Severity of a bug report. Developers may mark a difficult bug as higher severity.
Priority	Priority of a bug report. Developers may mark a difficult bug as higher priority.
CC Number	Number of users who may not have a direct role to play on the bug, but who are interested in its progress. Bugs followed by many people may have a higher re-opening probability.
Description Size	Words in bug description. Too short (due to hasty work) or too long message (due to the difficulty) may lead to bug re-opening.
Invalid Status	Boolean value, *i.e.*, whether it exists an invalid status (see Section 3) before a commit. Invalid status may be followed by bug re-opening.

Table V: BUG FIX DIMENSION

Attribute	Explanation and Rationale
Changed files	Number of changed files in a commit. Large number of changed files may increase the risk of bug re-opening.
Churn	Total number of inserted and deleted lines. Large number of changed LOC may increase the risk of bug re-opening.
Fixing time	Time span since the first fix. Long fixing time may induce bug re-opening.
Keywords	Some keywords (*e.g.*, crash, error, incorrect) in the commit message may imply bug re-opening.

Table VI: PEOPLE DIMENSION

Attribute	Explanation and Rationale
Reporter experience	The number of prior reported bugs. Inexperienced reporters are likely to introduce buggy report.
Assignee experience	The number of prior assigned bugs. Inexperienced assignee are likely to introduce buggy fixes.
Committer experience	The number of prior committed patches. Inexperienced committers are likely to introduce buggy fixes.

Approach. Based on the approach of Shihab et al. [13], we extract 19 attributes from commit logs and bug repositories along the four dimensions shown in Table III to Table VI.

We choose several regression and classification algorithms in R to build predictive models: General Linear Model (GLM), C5.0, ctree, cforest and randomForest. GLM is an extension of multiple linear regression for a single dependent variable. It is extensively used in regression analyses. The model C4.5 obtained a good prediction score in the work of Shihab et al. [13]. As a comparison, we use two Decision Tree models, C5.0 and ctree. C5.0 is an improved version of C4.5. The two algorithms are respectively derived from R packages "C50" and "party". In addition, we apply two implementations of the Random Forest algorithm, *i.e.*, randomForest from the R package "randomForest" and cforest from the R package "party". Random Forest was developed by Leo Breiman and Adele Cutler [3]. It uses a majority voting of decision trees to generate classification (predicting, often binary, class labels) or regression (predicting numerical values) results. Random Forest offers good out-of-the-box performance and has performed very well in different defect prediction benchmarks [9]. The algorithm yields an ensemble that can achieve both low bias and low variance [4]. In our configuration, we build 50 trees with five randomly selected attributes in each tree.

Before building our models, we use Variance Inflation Factor (VIF) analysis to remove correlated variables. We set the correlation threshold to 5. Variables with VIF result over the threshold are considered as correlated, and hence are not included in our models. Among the selected attributes, "day of year" was eliminated, since its VIF result is higher than 5. We do not use reporter and assignee names like in Shihab et al.'s work [13], because these variables may lead to overfitted models.

Table VII: ACCURACY, PRECISION, RECALL AND F-MEASURE (IN %) OBTAINED FROM GLM, C5.0, CTREE, CFOREST AND RANDOMFOREST

	Algo.	Acc.	Re-op. Pre.	Re-op. Rec.	Re-op. F-m.	Non Re-op. Pre.	Non Re-op. Rec.	Non Re-op. F-m.
Mozilla	GLM	64.2	69.6	6.6	12	64	**98.3**	77.5
	C5.0	74.3	70	53.9	60.9	76	86.4	80.8
	ctree	68.9	62.8	40.2	49.1	70.8	85.9	77.6
	cforest	76.4	79.4	49.3	60.8	75.5	92.4	83.1
	randomForest	**82.1**	**82.8**	**65.3**	**73.1**	**81.8**	92	**86.6**
Netbeans	GLM	69.9	**87.7**	13.2	22.9	69	**99.1**	81.3
	C5.0	74.4	67.9	46.2	55	76.3	88.8	82.1
	ctree	71	75	21.8	33.8	70.6	96.3	81.5
	cforest	74.1	82.8	29.9	44	72.9	96.8	83.2
	randomForest	**78.3**	80.2	**47.7**	**59.8**	**77.8**	94	**85.1**
JDT Core	GLM	77.5	76.6	15.1	25.3	77.5	98.4	86.7
	C5.0	83.3	72.7	53.8	61.8	85.7	93.2	89.3
	ctree	81	78.4	34.2	47.6	81.4	96.8	88.4
	cforest	83.3	**92.2**	36.8	52.6	82.3	**99**	89.9
	randomForest	**87.7**	89.9	**57.7**	**70.2**	**87.3**	97.8	**92.2**
Plat. SWT	GLM	78.9	71.4	5.9	10.8	79.1	99.3	88.1
	C5.0	88	80.9	58.9	68.1	89.3	96.1	92.6
	ctree	82.2	73	29.3	41.8	83.1	97	89.5
	cforest	86.2	**97**	37.8	54.4	85.2	**99.7**	91.8
	randomForest	**91.6**	95.4	**64.4**	**76.9**	**90.9**	99.1	**94.8**
WebKit	GLM	71.9	51.7	5.2	9.4	72.4	**98.1**	83.3
	C50	76.6	62	44	51.5	80.2	89.4	84.6
	ctree	74.9	58.4	38.5	46.4	78.7	89.2	83.6
	cforest	77.8	**72.2**	34.7	46.9	78.7	94.7	86
	randomForest	**80.5**	69.8	**54.4**	**61.1**	**83.5**	90.8	**87**

To evaluate the importance of the different attributes (prediction variables), we applied the *MeanDecreaseGini* criteria, in which a higher value represents higher importance.

We applied 10-fold cross validation [5] to calculate the accuracy as well as the precision, recall and F-measure for respectively re-opened and non re-opened bugs. In the cross validation, each data set is randomly split into ten folds. Nine folds are used as the training set, and the remaining one fold is used as the testing set. We repeat the 10-fold cross validation for ten times and report the average results obtained.

Findings. **In all studied projects, randomForest outperforms C5.0 and other algorithms in accuracy, re-opened F-measure and non re-opened F-measure. When predicting re-opened bugs with randomForest, we can achieve an average accuracy of 84%, a precision of 83.6%, and a recall of 57.9%.** Table VII presents accuracy, precision, recall, and F-measure results for the five models predicting whether or not a bug that required supplementary fixes will be re-opened. Table VIII shows the top and second important attributes as well as their frequency in randomForest. As we executed 10 times the validation, the maximum frequency is 10. **Overall, assignee experience, commit month, churn, reporter experience and committer experience are evaluated as the top or second attributes in different projects.**

In summary, our predictive models for re-opened bugs can achieve a precision between 72.2% and 97% and a recall between 47.7% and 65.3%.

Table VIII: TOP AND SECOND ATTRIBUTES AND THEIR FREQUENCY IN RANDOMFOREST

Project	Top attribute	Freq.	Second attribute	Freq.
Mozilla	commit month	5	commit month	5
	assignee exp.	5	assignee exp.	5
Netbeans	assignee exp.	10	reporter exp.	6
			commit month	3
			committer exp.	1
JDT	assignee exp.	10	commit month	10
SWT	assignee exp.	10	commit month	10
WebKit	churn	10	assignee exp.	7
			commit month	3

IV. DISCUSSION

This section discusses some of the key aspects of our study that differ from the works of Park et al. and Shihab et al.

Identification of supplementary bug fixes. During our data collection and processing, we have uncovered some limitations of the algorithm proposed by Park et al. [12] to

track supplementary bug fixes. We have proposed an enhanced heuristic that can identify supplementary fixes with higher precision. Indeed, the new heuristic can track bug IDs that cannot be tracked by the algorithm proposed by Park et al. and it cross-checks all bug IDs mentioned in commit logs with the Bugzilla repositories to eliminate false bug IDs. Compared to the results of Park et al., the new heuristic have reported a higher percentage of supplementary bug fixes in Eclipse Platform SWT (25.9% vs. 24%) and Eclipse JDT core (26.9% vs. 22.5%), but a lower percentage in Mozilla (23.8% vs. 32.8%).

Prediction models. In **RQ2**, we observed that almost half of the re-opened bugs are fixed by only one commit. 22.8% to 49.1% of these single re-opened bugs were due to prematurely closed reports. These prematurely closed bugs do not necessarily have a negative impact on software development, since they are not related to failed bug fixes. For this reason, we decided in this study to focus our prediction of bug re-openings on supplementary bug fixes rather than on all bug reports as in the work of Shihab et al. [13]. Compared to their results (although our prediction models have a different dependent variable), our prediction models have a higher precision (72.2-97% vs. 52.1%-78.6%) and a lower recall (47.7%-65.3% vs. 70.5%-94.1%).

V. Threats to Validity

This section discusses the threats to validity of our study following the guidelines for case study research [16].

Construct validity threats concern the relation between theory and observation. We answered RQ1, RQ2, and RQ3 by carefully choosing the experimental measures, *i.e.*, identification technique and prediction algorithms. Concerning the proportion of supplementary fixes in a project, since our results for Webkit are different from those obtained by Park et al. [12], we have manually verified 200 commit messages of each project to validate the correctness of the results. Compared to Park et al., we enhanced the identification heuristic and cross-checked all the bug IDs obtained from commit logs with Bugzilla repositories to ensure that all detected bug IDs represent actual bug reports. In addition, we have the lowest type II bug reports percentage in Netbeans, since in this project, many report messages are either non bug fixing related or too brief, so it is difficult to map a fix to a certain bug ID. In WebKit, the re-opened bugs only account for 5.3% of all bug reports. In this project, many bug reports are only available to the internal staff (marked by "Access Denied"). Therefore, we cannot judge whether these bugs have been re-opened.

Internal validity threats concern factors that may affect a dependent variable and were not considered in the study. Theoretically, one would expect that all re-opened bugs are fixed more than once (*i.e.*, a fix before re-opening and other fixes afterwards), yet we obtain 33% to 57.5% re-opened bugs in the type I bug set. Although we found that a large part of these bugs had been closed prematurely without any fix, another explanation could be a limitation of the identification technique by regular expressions. Even though we used not only bug IDs to trace a bug, but also revision numbers to map revisions and bug fixes. Some software organizations do not explicitly mark bug IDs in the revision history (or at least do not enforce this). So, we can not track these bug fixes in VCS. In future work, novel identification heuristics need to be explored. Another threat is related to the computation of bug fixing time values. It is possible that some developers forgot to close a fixed bug report.

Conclusion validity threats concern the relation between the treatment and the outcome. We paid attention not to violate assumptions of the constructed statistical models. According to the bug identification technique, we improve the existing heuristic, considered commits referring to an earlier commit's revision number, and compare the identified numbers with bug repositories. We manually checked the number sequences that were not detected by our mentioned regular expressions and found that none of those numbers were related to bug IDs. In the prediction, our best model, randomForest, can achieve a precision between 69.8% and 95.4%, a recall between 47.7% and 65.3%. Due to the state of art of the bug identification technique from VCS, many bug fixes are not mapped to their corresponding bug reports. This may affect the recall of the prediction for bug re-opening.

External validity threats concern the possibility to generalize our results. Besides the project used by Park et al. [12] and Shihab [13] et al., we introduced two other projects in this study, *i.e.*, Netbeans and WebKit. They have a similar type II bug percentage and a similar prediction accuracy. In future work, we plan to expand this study by analyzing other open source projects and applying novel identification techniques. For example, we could compare the bug fix committed time with the time in the attachments of bug reports to map a bug fix to its corresponding bug report. In addition, manual analysis of commit information and re-opening distribution will help us to determine the failure-prone fixes over all supplementary bug fixes. We provide our data and script in Github (https://github.com/anlepoly/supplementary_fixes). Researchers and software practitioners can verify our results or apply our approach to other projects.

VI. Related Work

Supplementary fixes. Park et al. [12] used delimiter and integer sequences to identify bug reports and supplementary fixes. In contrast, Kim et al. [8] and Mockus et al. [10] use keywords to match bug fix revisions. Śliwerski et al. combine syntactic analysis and semantic analysis to identify bug fixes [14]. Fischer et al. use both version control system and bug tracking database to populate a release history database [6]. We applied their approaches and also took revision numbers into account to extract supplementary fixes. To eliminate false bug IDs, we cross checked all identified IDs with Bugzilla databases or Bugzilla website. Park et al. also did a manual analysis to find the rationale behind supplementary fixes. However, they did not investigate whether supplementary fixes are related to bug re-opening, neither did they attempt to predict potential failures of supplementary fixes.

Bug re-opening. Previous work considered re-opened bugs primarily as a negative factor since repeated work increases maintenance overhead, degrading the software quality. Shihab et al. [13] discussed the risk of re-opened bugs and built prediction models to prevent bug re-opening. Zimmermann et al. [18] and Xia et al. [15] also proposed models for the prediction of re-opened bugs. However, neither of these studies has considered the relation between supplementary bug fixes and bug re-openings. We link supplementary bug fixes and re-opened bugs, and found that bug re-opening is an important reason of supplementary fixes, that 21.6% to 33.8% supplementary fixes have been re-opened at least once. On the other hand, a lot of re-opened bugs do not belong to supplementary fixes, *i.e.*, they do not have any prior bug fix in the studied repositories.

Prediction. Bug re-opening is not favorable to software development. It is worth building predictive models to prevent these bugs from re-appearing in the field. Researchers found that some factors might be linked to failure, such as, the changes committed on Fridays [14], or by certain inexperienced developers [7]. In previous studies, Hassan et al. used C4.5 decision tree algorithm to predict the certification result of a build for a large software project in IBM Toronto Lab [7], Shihab et al. compared C4.5, Zero-R, Naive Bayes and Logistic Regression algorithms to predict the re-opened bugs in three open source projects [13], and Zimmermann et al. used Logistic Regression model to predict re-opened bugs in Windows [18]. We selected C5.0 (the improved implementation of C4.5, which is the best algorithm in the study of Shihab et al. [13]) and applied other classification algorithms in our study. According to our results, randomForest outperforms other algorithms in the prediction for bug re-opening.

VII. Conclusion

In software development, bug fixing is a dominant activity for developers and testers. A typical bug fixing cycle includes the reporting of the bug, the production of a fix, the verification of the fix, and the closing of the bug. However, sometimes a closed bug later may be re-opened by developers. Previous studies show that such bug re-opening can increase the maintenance costs as well as degrade the software quality and the satisfaction of users. To discover the relation between supplementary bug fixes and re-opened bugs, we investigate supplementary bug fixes where more than one fix are associated with the same bug and re-opened bugs in five open source projects, and found that supplementary bug fixes account for 10.3% to 26.9% of total bug reports. In addition, in the subject systems, a high percentage (*i.e.*, from 21.6% to 33.8%) of the supplementary fixes have been re-opened. To help development teams target faulty/incomplete bug fixes (for more thorough reviews) and prevent re-opened bugs, we have explored the possibility of predicting bug re-openings over supplementary bug fixes, using GLM, C5.0, ctree, cforest and randomForest models. Results show that these models can achieve between 72.2% and 97% precision as well as between 47.7% and 65.3% recall. Moreover, we found between 33% to 57.5% of re-opened bugs with only one commit associated to them. These re-opened bugs have a strong association with invalid bug reports in all our five studied systems. In fact, they were prematurely dismissed as "invalid" before being re-opened. These bugs are not as risky as re-opened bugs with more than one commit to the software development. In other words, contrary to claims by existing works on re-opened bugs, they will not affect the quality of the software product. Future researchers and practitioners who are mining data repositories can use our models to identify fault-prone bug fixes.

References

[1] P. Anbalagan and M. Vouk. Days of the week effect in predicting the time taken to fix defects. In *Proceedings of the 2nd International Workshop on Defects in Large Software Systems: Held in conjunction with the ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2009)*, pages 29–30. ACM, 2009.

[2] C. Bird, P. C. Rigby, E. T. Barr, D. J. Hamilton, D. M. German, and P. Devanbu. The promises and perils of mining git. In *Proceedings of the 2009 6th IEEE International Working Conference on Mining Software Repositories*, MSR '09, pages 1–10, Washington, DC, USA, 2009. IEEE Computer Society.

[3] L. Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[4] R. Díaz-Uriarte and S. A. De Andres. Gene selection and classification of microarray data using random forest. *BMC bioinformatics*, 7(1):3, 2006.

[5] B. Efron. Estimating the error rate of a prediction rule: improvement on cross-validation. *Journal of the American Statistical Association*, 78(382):316–331, 1983.

[6] M. Fischer, M. Pinzger, and H. Gall. Populating a release history database from version control and bug tracking systems. In *Software Maintenance, 2003. ICSM 2003. Proceedings. International Conference on*, pages 23–32. IEEE, 2003.

[7] A. E. Hassan and K. Zhang. Using decision trees to predict the certification result of a build. In *Automated Software Engineering, 2006. ASE'06. 21st IEEE/ACM International Conference on*, pages 189–198. IEEE, 2006.

[8] S. Kim, E. J. Whitehead, and Y. Zhang. Classifying software changes: Clean or buggy? *Software Engineering, IEEE Transactions on*, 34(2):181–196, 2008.

[9] T. Mende and R. Koschke. Effort-aware defect prediction models. In *Software Maintenance and Reengineering (CSMR), 2010 14th European Conference on*, pages 107–116. IEEE, 2010.

[10] A. Mockus and L. G. Votta. Identifying reasons for software changes using historic databases. In *Software Maintenance, 2000. Proceedings. International Conference on*, pages 120–130. IEEE, 2000.

[11] N. I. of Standards & Technology. The economic impacts of inadequate infrastructure for software testing, May 2002. US Dept of Commerce.

[12] J. Park, M. Kim, B. Ray, and D.-H. Bae. An empirical study of supplementary bug fixes. In *Mining Software Repositories (MSR), 2012 9th IEEE Working Conference on*, pages 40–49. IEEE, 2012.

[13] E. Shihab, A. Ihara, Y. Kamei, W. M. Ibrahim, M. Ohira, B. Adams, A. E. Hassan, and K.-i. Matsumoto. Studying re-opened bugs in open source software. *Empirical Software Engineering*, pages 1–38, 2012.

[14] J. Śliwerski, T. Zimmermann, and A. Zeller. When do changes induce fixes? In *ACM sigsoft software engineering notes*, volume 30, pages 1–5. ACM, 2005.

[15] X. Xia, D. Lo, X. Wang, X. Yang, S. Li, and J. Sun. A comparative study of supervised learning algorithms for re-opened bug prediction. In *Software Maintenance and Reengineering (CSMR), 2013 17th European Conference on*, pages 331–334, March 2013.

[16] R. K. Yin. *Case Study Research: Design and Methods - Third Edition*. SAGE Publications, 3rd edition, 2002.

[17] T. Zimmermann, N. Nagappan, P. J. Guo, and B. Murphy. Characterizing and predicting which bugs get reopened. In *Proceedings of the 2012 International Conference on Software Engineering*, ICSE 2012, pages 1074–1083, Piscataway, NJ, USA, 2012. IEEE Press.

[18] T. Zimmermann, N. Nagappan, P. J. Guo, and B. Murphy. Characterizing and predicting which bugs get reopened. In *Software Engineering (ICSE), 2012 34th International Conference on*, pages 1074–1083. IEEE, 2012.