Towards a Classification of Bugs to Facilitate Software Maintainability Tasks

Mathieu Nayrolles and Abdelwahab Hamou-Lhadj
Intelligent System Logging and Monitoring (ISyLM) Lab
ECE, Concordia University, Montréal, Canada
{m_nayrol, abdelw}@ece.concordia.ca

ABSTRACT

Software maintainability is an important software quality attribute that defines the degree by which a software system is understood, repaired, or enhanced. In recent years, there has been an increase in attention in techniques and tools that mine large bug repositories to help software developers understand the causes of bugs and speed up the fixing process. These techniques, however, treat all bugs in the same way. Bugs that are fixed by changing a single location in the code are examined the same way as those that require complex changes. After examining more than 100 thousand bug reports of 380 projects, we found that bugs can be classified into four types based on the location of their fixes. Type 1 bugs are the ones that fixed by modifying a single location in the code, while Type 2 refers to bugs that are fixed in more than one location. Type 3 refers to multiple bugs that are fixed in the exact same location. Type 4 is an extension of Type 3, where multiple bugs are resolved by modifying the same set of locations. This classification can help companies put the resources where they are needed the most. It also provides useful insight into the quality of the code. Knowing, for example, that a system contains a large number of bugs of Type 4 suggests high maintenance efforts. This classification can also be used for other tasks such as predicting the type of incoming bugs for an improved bug handling process. For example, if a bug is found to be of Type 4 then it should be directed to experienced developers.

KEYWORDS

Bug Classification, Empirical Studies, Software Maintainability.

1. INTRODUCTION

The analysis of bug provides useful insight that can help with many maintenance activities such as bug fixing [1][2], bug reproduction [3]–[5], fault analysis [6], etc.

SQUADE'18, May 28–29, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5741-8/18/05. . . $15.00
https://doi.org/10.1145/3194718.3194726

This insight can, in turn, be used to build more maintainable systems by detecting (and preferably preventing) bugs before a system is released. There exist many studies that focus on investigating techniques and tools for bug prediction, detection, and reproduction (e.g., [3, [8][9]). These studies, however, treat all bugs as the same. For example, a bug that requires only one fix is analyzed the same way as a bug that necessitates multiple fixes. Similarly, if multiple bugs are fixed by modifying the exact same locations in the code, then we should investigate how these bugs are related in order to predict them in the future. Note here that we do not refer to duplicate bugs. Duplicate bugs are marked as duplicate (and not fixed) and only the master bug is fixed.

As a motivating example, consider Bugs #AMQ-5066 and #AMQ-5092 from the Apache Software Foundation bug report management system (used to build one of the datasets in this paper). Bug #AMQ-5066 was reported on February 19, 2014 and solved with a patch provided by the reporter. The solution involves a relatively complex patch that modifies MQTTProtocolConverter.java, MQTTSubscription.java and MQTTTest.java files. The description of the bug is as follows:

> *"When a client sends a SUBSCRIBE message with the same Topic/Filter as a previous SUBSCRIBE message but a different QoS, the Server MUST discard the older subscription, and resend all retained messages limited to the new Subscription QoS."*

A few months later, another bug, Bug #AMQ-5092 was reported:

> *"MQTT protocol converters does not correctly generate unique packet ids for retained and non-retained publish messages sent to clients. […] Although retained messages published on creation of client subscriptions are copies of retained messages, they must carry a unique packet id when dispatched to clients. ActiveMQ re-uses the retained message's packet id, which makes it difficult to acknowledge these messages when wildcard topics are used.*
> *ActiveMQ also sends the same non-retained message multiple times for every matching subscription for overlapping subscriptions. These messages also re-use the publisher's message id as the packet id, which breaks client acknowledgment."*

This bug was assigned and fixed by a different person than the one who fixed bug #AMQ-5066. The fix consists of modifying slightly the same lines of the code in the exact files used to fix

Bug #AMQ-5066. In fact, Bug #5092 could have been avoided altogether if the first developer provided a more comprehensive fix to #AMQ-5066 (a task that is easier said than done). These two bugs are not duplicates since, according to their description, they deal with different types of problems and yet they can be fixed by providing a similar patch. The failures are different while the root causes (faults in the code) are more or less the same.

From the bug handling perspective, if we can develop a way to detect such related bug reports during triaging then we can achieve considerable time saving in the way bug reports are processed, for example, by assigning them to the same developers. We also conjecture that detecting such related bugs can help with other tasks such as bug reproduction. We can reuse the reproduction of an already fixed bug to reproduce an incoming and related bug.

The objective of this paper is not to propose a way to detect such related bug reports or how we can take advantage of them to improve the bug handling process, but to introduce a new way of grouping bugs into types that we believe can facilitate research analysis and software maintainability. We discuss this in more detail in the next section. We identify bug types by empirically examining bugs and their fixes of more than 100 thousand bug reports from Netbeans[1] and Apache[2] systems.

This paper continues by presenting our proposed bug classification in Section 2. In Section 3, we present the study setup. The results of our empirical study are presented in Section 4. Related work is discussed in Section 5, following with a conclusion in Section 5.

2. BUG TYPES

We can reason about types of bugs in various ways depending on the purpose of the types. In the area of software testing, for example, several researchers (e.g., [38][39][40]) have proposed fault classes such as coding faults, logical faults, resource faults, data faults, etc. to group faults (bugs). The objective is to improve the testing process by identifying the fault classes that are most problematic in a given system. Eldh [39] went one step further by investigating the relationship between fault classes and crashes in telecom systems. Among her findings, she showed that bugs in a fault class can cause many crashes and that the same crash can be related to bugs from many fault classes. A more recent study is the one from Hamill et al. [40] where the authors studied fault classes (similar to the ones presented by Eldh [39]) and their relationship with crashes in critical safety systems. Similar results were found. These studies aim to prevent the occurrence of bugs from occurring by inferring better ways to test the system.

In this paper, we are interested in the broader area of bug handling by investigating how bugs can be grouped together with the goal of speeding up the provision of fixes. We look at the relationship between bugs by examining their fixes. By a fix, we mean a modification (adding or deleting lines of code) to an existing file that is used to solve the bug. With this in mind, the relationship between bugs and fixes can be modeled using the UML diagram in Figure 1. The diagram only includes bugs that are fixed. From this figure, we can think of four instances of this diagram, which we refer to as bug taxonomy or simply bug types (see Figure 2).

Fig. 1. Class diagram showing the relationship between bugs and fixed

The first and second types are the ones that we intuitively know about. Type 1 refers to a bug being fixed in one single location (i.e., one file), while Type 2 refers to bugs being fixed in more than one location. In Figure 2, only two locations are shown for the sake of clarity, but many more locations could be involved in the fix of a bug. Type 3 refers to multiple bugs that are fixed in the exact same location. Type 4 is an extension of Type 3, where multiple bugs are resolved by modifying the same set of locations.

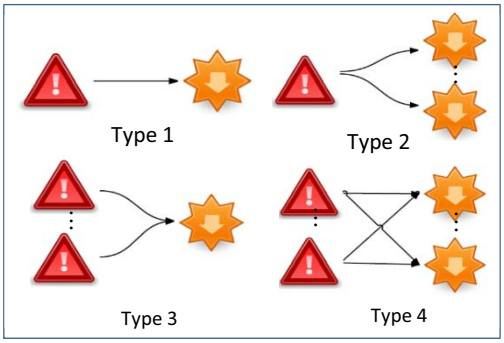

Fig. 2. Proposed taxonomy of bugs

We conjecture that knowing the proportions of each type of bugs in a system may provide insights into the quality of the system. Knowing, for example, that in a given system the proportion of Type 2 and 4 bugs is high may be an indication of poor system quality since many fixes are needed to address these bugs. In addition, the existence of a high number of Types 3 and 4 bugs calls for techniques that can effectively find bug reports related to an incoming bug during triaging. This is similar to the many studies that exist on detection of duplicates (e.g., [10]–[12]), except that we are not looking for duplicates but for related bugs (bugs that are due to failures of different features of the system, caused by the same faults). To our knowledge, there is no study that empirically examines bug data with these types in mind, which is the main objective of this paper. More particularly, the paper addresses the following research questions:

RQ1: What are the proportions of different types of bugs?

RQ2: How complex/severe is each type of bugs?

RQ3: How fast are these types of bugs fixed?

We address these questions by empirically examining bugs and their fixes of more than 100 thousand bug reports from Netbeans and Apache systems.

3. STUDY SETUP

Figure 3 illustrates our data collection and analysis process that we present here and discuss in more detail in the following subsections. First, we extract the raw data from the two bug report management systems used in this study (Bugzilla[3] and Jira[4]).

[1]https://netbeans.org/
[2]http://www.apache.org/

[3]https://netbeans.org/bugzilla/, https://www.bugzilla.org/
[4]https://issues.apache.org/jira/secure/Dashboard.jspa

Second, we extract the fix to the bugs from the source code version control system of Netbeans and Apache (Maven[5] and Git[6]). The extracted data is consolidated in one database where we associate each bug report to its fix. We mine relevant characteristics of BRs and their fixes such as opening time, number of comments, number of times the BR is reopened, number of changesets for BR and the number of files changed and lines modified for fixes or patch. Finally, we analyze these characteristics to answer the aforementioned research questions (RQ).

3.1. Bug Tracking Systems

Open source bug tracking systems allow end-users to directly create bug reports (BRs) to report on system crashes. These systems are also used by development teams to manage the BRs, and keep track of the fixes. In this study, we collect data from the Netbeans (Bugzilla/Maven) and Apache Software Foundation (Jira/Git) bug tracking and version control systems. We chose these repositories because they contain a large number of bugs.

The lifecycle of a bug in both systems is as follows: After a bug is submitted by an end-user, it have the UNCONFIRMED state until it receives enough votes or that a user with the proper permissions modifies its status to OPEN.

The bug is then assigned to a developer to fix it. When the bug is in the ASSIGNED state, developers start working on fixing the bug. A fixed bug moves to the RESOLVED state. Developers have typically five different possibilities to resolve a bug: FIXED, DUPLICATE, WONTFIX, WORKSFORME and INVALID.

For the remaining parts of this paper, we use the term 'resolved bug' to mean a bug that is revolved and fixed (i.e., marked as RESOLVED/FIXED in the bug tracking system). Note that duplicate bugs are not included in our dataset since they are marked as RESOLVED/DUPLICATE (the fix is provided when solving the master bug). Finally, the bug is closed after it is resolved. A bug can be reopened (set to the REOPENED state) and then assigned again if the initial fix was not adequate (the fix did not resolve the problem).

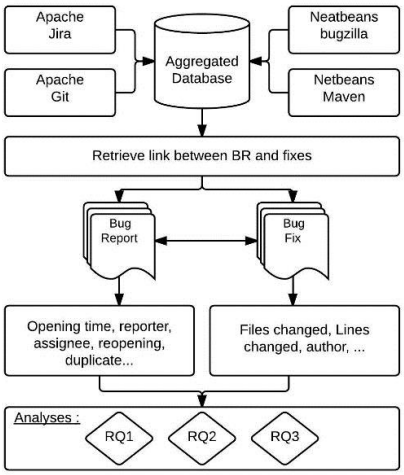

Fig. 3. Data collection and analysis process of the study.

[5]http://hg.netbeans.org/main, https://maven.apache.org/
[6]https://github.com/apache, http://git-scm.com/

The elapsed time between the time the bug is marked as a new one and the time it is resolved is known as the fixing time (in days). If the bug is reopened then the days between the time the bug is reopened and the time it is marked as RESOLVED/FIXED are accumulated. Bugs can be reopened many times.

A bug report can (and must according to [13]) have a severity. The severity is a classification of a bug to indicate the degree of negative impact on the quality of software and can evolve at any point during the lifecycle of the bug. In Bugzilla[7] (used by Netbeans), the possible severities are blocker (blocks development and/or testing work) critical (crashes, loss of data, severe memory leak), major (major loss of function), normal (regular issue, some loss of functionality under specific circumstances), minor (minor loss of function, or other problem where easy workaround is present) trivial (cosmetic problem like misspelled words or misaligned text). In Jira[8] (used by the Apache Software Foundation), the severity (also known as priority) is the same at the exception of the normal level which does not exist.

The relationship between a BR and the actual fix can be hard to establish and it has been a subject for various research studies (e.g., [14][15]). While it is considered a good practice to link each BR with the source code repository by indicating the bug id on the commit message, more than half of the bugs in our dataset are not linked to a commit. We exclude these bugs in this study and only consider the ones that have a commit. This way, we can establish a link between the bug and its fixes.

3.2. Datasets

In this study, we used two distinct datasets: Netbeans and the Apache Software Foundation projects. Netbeans is an integrated development environment (IDE) for developing with many languages including Java, PHP, and C/C++. The very first version of Netbeans, then known as Xelfi, appeared in 1996. The Apache Software Foundation is a U.S non-profit organization supporting Apache software projects such as the popular Apache web server since 1999. The characteristics of the Netbeans and Apache Software Foundation datasets used in this paper are presented in Table I.

Cumulatively, these datasets span from 2001 to 2014. In summary, our consolidated dataset contains 102,707 bugs, 229,153 changesets, 68,809 files that have been modified to fix the bugs, 462,848 comments, and 388 distinct systems. We also collected 221 million lines of code modified to fix the bugs, identified 3,284 sub-projects, and 17,984 unique contributors to these bug report and source code version management systems. Finally, the cumulated opening time for all the bugs reaches 10,661 working years (3,891,618 working days).

TABLE I. The Netbeans and Apache Datasets. R/F BR stands for RESOLVED/FIXED bug, CS Changsets, and Projects

Dataset	R/F BR	CS	Files	Projects
Netbeans	53,258	122,632	30,595	39
Apache	49,449	106,366	38,111	349
Total	102,707	229,153	68,809	388

[7]https://bugzilla.mozilla.org/page.cgi?id=fields.html#bug_severity
[8]https://confluence.atlassian.com/display/DEV/JIRA+usage+guidelines#JIRAusageguidelines-Settheprioriy

3.3. Study Design

We describe the design of our study by first stating the research questions, and then explaining the variables, and analysis methods we used to answer these questions. We formulate three research questions (RQs) with the ultimate goal to improve our understanding of each bug type. We focus, however, on Types 2 and 4. This is because these bugs require multiple fixes. They are therefore expected to be more complex.

The objective of the first research question is to analyze the proportion of each type of bugs. The remaining two questions address the complexity of the bugs and the bug fixing duration according to the type of bugs.

RQ 1: What are the proportions of different types of bugs?

Objective: The answer to this question provides insight into the distribution of bugs according to their type with a focus on Type 2 and 4 bugs. As discussed earlier, knowing, for example, that bugs of Type 2 and 4 are the most predominant ones suggests that we need to investigate techniques to help detect whether an incoming bug is of Types 2 and 4 by examining historical data. Similarly, if we can automatically identify a bug that is related to another one that has been fixed then we can reuse the results of reproducing the first bug in reproducing the second one.

Hypothesis: To answer this question, we analyze whether Type 2 and 4 bugs are predominant in the studied systems, by testing the null hypothesis:

- H_{01A}: The proportion of Types 2 and 4 does not change significantly across the studied systems

We test this hypothesis by observing both a "global" (across systems) and a "local" predominance (per system) of the different types of bugs. We must observe these two aspects to ensure that the predominance of a particular type of bug is not circumstantial (in few given systems only) but is also not due to some other, unknown factors (in all systems but not in a particular system).

Variables: We use as variables the amount of resolved/fixed bugs of each type (1, 2, 3 and 4) that are linked to a fix (commit). As mentioned earlier, duplicate bugs are excluded. These are marked as resolved/duplicate in our dataset.

Analysis Method: We answer RQ1 in two steps. The first step is to use descriptive statistics; we compute the ratio of Types 2 and 4 bugs and the ratio of Types 1 and 3 bugs to the total number of bugs in the dataset. This shows the importance of Types 2 and 4 bugs compared to Types 1 and 3 bugs. In the second step, we compare the proportions of the different types of bugs with respect to the system where the bugs were found. We build the contingency table with these two qualitative variables (the type and studied system) and test the null hypothesis H_{01A} to assess whether the proportion of a particular type of bugs is related to a specific system or not. We use the Pearson's chi-squared test to reject the null hypothesis H_{01A}. Pearson's chi-squared independence test is used to analyze the relationship between two qualitative data, in our study the type bugs and the studied system. The results of Pearson's chi-squared independence test are considered statistically significant at $\alpha = 0.05$. If p - value < 0.05, we reject the null hypothesis H_{01A} and conclude that the proportion of types 3 and 4 bugs is different from the proportion of type 1 and 2 bugs for each system.

RQ 2: How complex is each type of bugs?

Objective: We address the relation between Types 2 and 4 bugs and the complexity of the bugs in terms of severity, duplicate and reopened. We analyze whether Types 2 and 4 bugs are more complex to handle than Types 1 and 3 bugs, by testing the null hypotheses:

- H_{02S}: The severity of Types 2 and 4 bugs is not significantly different from the severity of Types 1 and 3
- H_{02D}: Types 2 and 4 bugs are not significantly more likely to get duplicated than Types 1 and 3.
- H_{02R}: Type 2 and 4 bugs are not significantly more likely to get reopened than Types 1 and 3.

Variables: We use as independent variables for the hypotheses H_{02S}, H_{02D}, H_{02R} the bug type (if the bug is from Types 2 and 4 or if it is from Types 1 and 3). For H_{02S} we use the severity as dependent variable to assess the relationship between the bug severity and the bug type. For H_{02D} (respectively H_{02R}) we use a dummy variable duplicated (reopened) to assess if a bug has been duplicated (reopened) at least once or not. This will be used to assess the relationship between the type of the bugs and the fact that the bug is more likely to be reopened or duplicated.

Analysis Method: For each hypothesis, we build a contingency table with the qualitative variables type of bugs (2 and 4 or 1and 3) and the dependent variable duplicated (respectively reopened) and the severity variable. We use the Pearson's chi-squared test to reject the null hypothesis H_{02D} (respectively H_{02R}) and H_{02S}. The results of Pearson's chi-squared independence test are considered statistically significant at $\alpha = 0.05$. If a p - value < 0.05, we reject the null hypothesis H_{02D} (respectively H_{02R}) and conclude the fact that the bug is more likely to be duplicated (respectively reopened) is related to the bug type and we reject H_{02S} and conclude that the severity level of the bug is related to the bug type.

RQ 3 : How fast are these types of bugs fixed?

Objective: In this question, we study the relation between the different types of bugs and the fixing time. We are interested in evaluating whether developers take more time to fix Types 2 and 4 bugs than Type 1 and 3, by testing the null hypothesis

- H_{03}: There is no statistically-significant difference between the duration of fixing periods for Types 2 and 4 bugs and that of Types 1 and 3 bugs.

Variables: To compare the bug fixing time with respect to their type, we use as independent variable the type Ti of a bug Bi, to distinguish between Types 1 and 3 bugs and Types 2 and 4 bugs. We consider as dependent variable the fixing time, FTi, of the bug Bi. We compute the fixing time FTi of a bug Bi. The fixing time FTi is the time between when the bug is submitted to when it is closed/fixed.

Analysis Method: We compute the (non-parametric) Mann-Whitney test to compare the BR fixing time with respect to the BR type and analyze whether the difference in the average fixing time is statistically significant. We use the Mann-Whitney test because, as a non-parametric test, it does not make any

assumption on the underlying distributions. We analyze the results of the test to assess the null hypothesis H_{03}. The result is considered as statistically significant at $\alpha = 0.05$. Therefore, if p - value < 0.05, we reject the null hypothesis H_{03} and conclude that the average fixing time of Types 1 and 3 bugs is significantly different from the average fixing time of Types 2 and 4 bugs.

4. STUDY RESULTS AND DISCUSSION

In this section, we report on the results of the analyses we performed to answer our research questions. We then dedicate a section to discussing the results.

RQ 1 : What are the proportions of different types of bugs?

Figure 4 shows the percentage of the different types of bugs. As shown in the figure, we found that 65% of the bugs are from Types 2 and 4. This shows the predominance of this type of bugs in all the studied systems. Figure 5 shows the repartition per dataset. We can see that Netbeans and Apache have 66% and 64% bugs of Type 1and 3, respectively. To ensure that this observation is not related to a particular system, we perform Pearson's chi-squared test across the studied systems. Table II shows the contingency table for the studied systems and the result of Pearson's chi-squared test. The results show that there is statistically significant difference between the proportions of the different types of bugs.

TABLE II. Contingency table and Pearson's chi-squared tests

System	Type 1 and 3	Type 2 and 4	Pearson's chi-squared p-Value
Apache	4910	8626	p-value < 0,0001
Netbeans	9050	17586	

We can thus reject the null hypothesis **H_{01A}** and conclude that there is a predominance of Types 2 and 4 bugs in all studied systems and this observation is not related to a specific system.

TABLE III. Proportion of bug types in amount and percentage.

Dataset	T1	T2	T3	T4	Total
Netbeans	776 (2.90%)	240 (0.90%)	8372 (31.29%)	17366 (64.91%)	26754
Apache	1968 (14.32%)	1248 (9.08%)	3101 (22.57%)	7422 (54.02%)	13739
Total	2744 (6.78%)	1488 (3.74%)	11473 (28.33%)	24788 (61.21%)	40493

Table III shows the number of bugs for each type of bugs and the percentage of each type of bugs. We can see that Types 3 and 4 bugs represent 28.33% and 61.21% of the total of bugs, respectively. Types 1 and 2 represent only 6.78% and 3.74%. Together, Types 3 and 4 bugs represent almost 90% of the total number of bugs linked to a commit.

RQ 2 : How complex is each type of bugs?

Figure 5 and 6 show the proportion of each bug type with respect to their severity for each dataset. Table V shows the proportion of each bug type with respect to their severity and dataset. For Netbeans, the bugs we examined in our dataset are either labeled as Blocker or Normal (despite the fact that Netbeans uses Bugzilla that supports all the severity levels presented in the previous section).

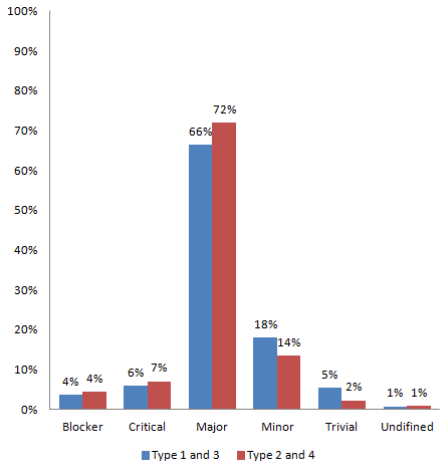

Fig. 4. Proportion of Type 1 and 3 versus Type 2 and 4 with respect to their severity in the Apache dataset.

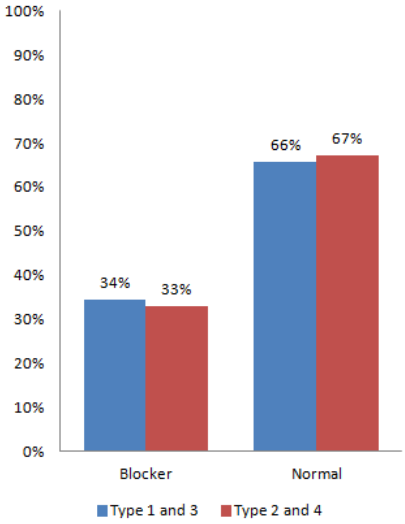

Fig. 5. Proportion of Types 1 and 3 versus Types 2 and 4 with respect to their severity in the Netbeans dataset.

For the Apache dataset, the severity levels range from Blocker to Trivial as shown in Figure 5. Figure 6 shows that in Netbeans around 67% of Types 2 and 4 bugs are normal. The same holds for Types 1 and 3 bugs (66% are considered of normal severity). This indicates that most Types 2 and 4 bugs and Types 1 and 3 bugs are not critical in the Netbeans dataset. For the Apache dataset, the results indicate that the majority of the bugs are considered of major severity (66% for Types 1 and 3 and 72% for Types 2 and 4). It is challenging to understand the discrepancy between the two datasets partly because of the way the severity is assigned to BRs. Table IV shows the result of the Pearson chi-squared tests for the H_{02S}, H_{02D} and H_{02R} hypotheses.

TABLE IV. Pearon's chi squared p-values for the severity, the reopen and the duplicate factors with respect to a dataset.

System	Factor	Pearson's chi-squared p-value
Apache	Severity	p-value < 0.005
	Reopened	p-value < 0.005
	Duplicated	p-value < 0.005
Netbeans	Severity	p-value < 0.005
	Reopened	p-value < 0.005
	Duplicated	p-value < 0.005

According to the test (**p-value < 0.005**), we reject the null hypothesis $\mathbf{H_{02S}}$ and conclude that there is a significant difference between the severity of Types 1 and 3 bugs and the severity of Types 2 and 4 bugs.

TABLE V. Proportion of each bug type with respect to severity

Severity	T1	T2	T3	T4
Netbeans				
Blocker	340 43.81%	109 45.42%	2850 34.04%	5687 32.75%
Normal	436 56.19%	131 54.58%	5522 65.96%	11678 67.25%
Total	776 100%	240 100%	8372 100%	17365 100%
Apache				
Blocker	68 3.46%	53 4.25%	115 3.71%	329 4.43%
Critical	84 4.27%	44 3.53%	213 6.87%	565 7.61%
Major	1245 63.26%	811 64.98%	2096 67.59%	5427 73.12%
Minor	408 20.73%	276 22.12%	501 16.16%	899 12.11%
Trivial	113 5.74%	31 2.48%	159 5.13%	161 2.17%
Total	1918 100%	1215 100%	3084 100%	7381 100%

TABLE VI. Percentage and occurrences of bugs duplicated by other bugs and reopenned with respect to their bug type and dataset

Type	T1	T2	T3	T4	Total
Netbeans					
Dup.	6.06% (47)	4.59% (11)	5.09% (426)	5.87% (1019)	5.62% (1503)
Reop.	4.38% (34)	7.08% (17)	4.81% (403)	7.09% (1231)	6.30% (1685)
Apache					
Dup	2.59% (51)	2.24% (28)	1.61% (50)	2.91% (216)	2.51% (345)
Reop	5.59% (110)	6.49% (81)	3.10% (96)	6.90% (512)	5.82% (799)
Total					
Dup	3.57% (98)	2.62% (39)	4.15% (476)	4.98% (1235)	4.56% (1848)
Reop	5.25% (144)	6.59% (98)	4.35% (499)	7.03% (1743)	6.13% (2484)

Table VI shows the occurrences of duplicate and reopened bugs with respect to their bug type in each dataset. In Netbeans, the proportion of Type 1 bugs that are marked as source of duplicate is 6.06%, 4.59% for Type 2 bugs, 5.09% for Type 3 bugs and 5.87% for Type 4 bugs with a total of 1503 bugs over 26754 (5.62%). In Apache, the proportion of Type 1 bugs marked as a source of a duplicate is 2.59% and 2.24%, 1.61% and 2.91% for Types 2, 3 and 4, respectively. Overall, the types that are more likely to be marked as source of duplicates at least once are Type 4 with 4.98%.

Second, we analyze the reopened bugs to see the link between the reopening and the type of bugs. We perform Pearson's chi-squared test to reject the null hypothesis H_{02R}.

According to the results of the test (**p-value < 0.005**), we reject the null hypothesis $\mathbf{H_{02R}}$ and conclude that there is a significant relationship between the reopening of a bug and its type.

Third, we analyze the duplicated bugs to see if there is a link between the bug type and the fact duplication. We perform Pearson's chi-squared test to reject the null hypothesis H_{02D}.

According to the results of the test (**p-value < 0.005**), we reject the null hypothesis $\mathbf{H_{02D}}$ and conclude that there is a significant relationship between the duplication of a bug and its type.

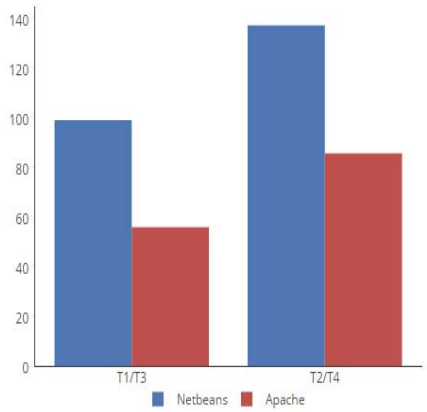

Fig. 6. Fixing time of Types 1 and 3 versus fixing time of Types 2 and 4 with respect to their datasets.

TABLE VII. Average Fixing time with respect to bug type

Dataset	T1	T2	T3	T4	Average
Netbeans	97.66	117.42	100.26	156.67	118.00
Apache	73.48	118.12	38.04	52.83	70.62
Average	85.57	117.77	69.15	104.75	94.31

RQ 3 : How fast are these types of bugs fixed ?

Figure 7 shows the fixing time for Types 1 and 3 versus Types 2 and 4 for Netbeans and the Apache Software Foundation. In Netbeans, 98.96 and 137.05 days are required to fix Types 1 and 3 and Types 2 and 4, respectively. In Apache, 55.76 and 85.48 days are required to fix Types 1 and 3 and Types 2 and 4, respectively. Table VII shows the average fixing time of bugs with respect to their bug type in each dataset.

We analyze the difference in the fixing time of bugs with respect to their bug type by conducting a Mann-Whitney test to assess H_{03}.The results show that the difference between the fixing time of Types 2 and 4 and Types 1 and 3 is statistically significant (p-value < 0,005).

> Therefore, we can reject the null hypothesis $\mathbf{H_{03}}$ and conclude that the fixing of Types 2 and 4 bugs takes more time than the fixing of Types 1 and 3 bugs.

Discussion:

Repartition of bug types: One important finding of this study is that there is significantly more Types 2 and 4 bugs than Types 1 and 3 in all studied systems. Moreover, this observation is not system-specific. The traditional one-bug/one-fault way of thinking about bugs only accounts for 35% of the bugs. We believe that, recent triaging algorithms [13][16]–[18] can benefit from these findings by developing techniques that can detect Type 2 and 4 bugs. This would result in better performance in terms of reducing the cost, time and efforts required by developers in the bug fixing process.

Severity of bugs: We discussed the severity and the complexity of a bug in terms of its likelihood to be reopened or marked as duplicates (RQ2). Although clear guidelines exist on how to assign the severity of a bug, it remains a manual process performed by the bug reporters. In addition, previous studies, notably those by Khomh et al. [16], showed that severity is not a consistent/trustworthy characteristic of a BR, which leads to the emergence of studies for predicting the severity of bugs (e.g., [19]–[21]). Nevertheless, we discovered that there is a significant difference between the severities of Types 1 and 3 compared to Types 2 and 4.

Complexity of bugs: At the complexity level, we use the number of times a bug is reopened as a measure of complexity. We found that there is a significant relationship between the number of reopenings of bugs and the types. In our datasets, Types 1 and 3 bugs are reopened in 1.88% of the cases, while Types 2 and 4 are reopened in 5.73%. Assuming that the reopening is a representative metric for the complexity of bug, Types 2 and 4 are three times more complex than Types 1 and 3. Finally, if we consider multiple reopenings, Types 2 and 4 account for almost 80% of the bugs that are reopened more than once. While current approaches aiming to predict which bug will be reopen use the number of modified files [22]–[24], we believe that they can be improved by taking into account the bug type. For example, if we can detect that an incoming bug if of Type 2 or 4 then it is more likely to be reopened than a bug of Type 1 or 3.

Impact of a bug: To measure the impact of bugs on end users and developers, we use the number of times a bug is duplicated. This is because if a bug has many duplicates, it means that a large number of users have experienced the corresponding failure. We found that there is a significant relationship between the bug type and the fact that a bug gets duplicated. Types 1 and 3 bugs are duplicated in 1.41% of the cases while Types 2 and 4 are duplicated in 3.14%. Using duplication as a metric of bug impact, Types 2 and 4 have more than two times bigger impact than Types 1 and 3. Similarly to the reopening metric, if we consider multiple duplications, Types 2 and 4 account for 75% of the bugs that get duplicated more than once and more than 80% of the bugs that get duplicated more than twice. We believe that approaches targeting the identification of duplicates [27][10][28][29] could leverage this classification to detect duplicates BRs with higher recall and precision.

Fixing time: Our third research question aimed to determine if the type of a bug impacts its fixing time. Not only we found that the type of a bug does significantly impact its fixing time, but we also found that, in average Types 2 and 4, stay open 111.26 days while Types 1 and 3 last for 77.36 days. Types 2 and 4 are 1.4 time longer to fix than Types 1 and 3.We therefore believe that, approaches aiming to predict the fixing time of a bug (e.g., [7], [30], [31]) can highly benefit from accurately predicting the type of a bug and therefore better plan the required man-power to fix the bug.

5. RELATED WORK

Researchers have been studying the relationships between the bug and source code repositories since more than two decades. To the best of our knowledge the first ones who conducted this type of study on a significant scale were Perry and Stieg [32]. In the last two decades, many aspects of these relationships have been studied in depth. For example, researchers were interested in improving the bug reports themselves by proposing guidelines [13], and by further simplifying existing bug reporting models [33]. Another field of study consist of assigning these bug reports, automatically if possible, to the right developers during triaging [34][17][18][35]. Another set of approaches focus on how long it takes to fix a bug [7][31][2] and where it should be fixed [25][26]. With the rapidly increasing number of bugs, the community was also interested in prioritizing bug reports [36], and in predicting the severity of a bug [19]. Finally, researchers proposed approaches to predict which bug will get reopened [23][24], which bug report is a duplicate of another one [27]–[29] and which locations are likely to yield new bugs [8][37].

However, to the best of our knowledge, there are not many attempts to classify bugs the way we present in this paper. In her Ph.D. thesis dissertation [39], Sigrid Eldh discussed the classification of trouble reports with respect to a set of fault classes that she identified. Fault classes include computational logical faults, resource faults, function faults, etc. She conducted studies on Ericsson systems and showed the distributions of trouble reports with respect to these fault classes. A research paper was published on the topic in [39]. or safety critical [40]. Hamill et al. [40] proposed a classification of faults and failures in critical safety systems. They proposed several types of faults and show how failures in critical safety systems relate to these classes. They found that only a few fault types were responsible for the majority of failures. They also compare on pre-release and post-release faults and showed that the distributions of fault types differed for pre-release and post-release failures. Another finding is that coding faults are the most predominant ones.

Our study differs from these studies in the way that we focus on the bugs and their fixes across a wide range of systems, programming languages, and purposes. This is done indecently

from a specific class of faults (such as coding faults, resource faults, etc.). This is because our aim is not to improve testing as it is the case in the work of Eldh [39] and Hamill et al. [40]. Our objective is to propose a classification that can allow researchers in the field of mining bug repositories to use the taxonomy as a new criterion in triaging, prediction, and reproduction of bugs.

6. CONCLUSION

In this paper, we proposed a taxonomy of bugs and performed an empirical study on two large open source datasets: the Netbeans IDE and the Apache Software Foundation's projects. Our study aimed to analyse: (1) the proportion of each type of bugs; (2) the complexity of each type in terms of severity, reopening and duplication; and (3) the required time to fix a bug depending on its type. The key findings are: Types 2 and 4 account for 65% of the bugs; Types 2 and 4 have a similar severity compared to Types 1 and 3; Types 2 and 4 are more complex (reopening) and have a bigger impact (duplicate) than Types 1 and 3; it takes more time to fix Types 2 and 4 than Types 1 and 3.

REFERENCES

[1] C. Weiss, R. Premraj, T. Zimmermann, and A. Zeller, "How Long Will It Take to Fix This Bug?," in Proc. of the 4th International Workshop on Mining Software Repositories (MSR), 2007.

[2] R. K. Saha, S. Khurshid, and D. E. Perry, "An empirical study of long lived bugs," in 2014 Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE), 2014, pp. 144–153.

[3] N. Chen, "Star: stack trace based automatic crash reproduction," IEEE Transactions on Software Eng., 41(2), 2014, pp. 198-220.

[4] S. Artzi, S. Kim, and M. D. Ernst, "Recrash: Making software failures reproducible by preserving object states," in Proc. of the 22nd European conference on Object-Oriented Programming, 2008, pp. 542–565.

[5] W. Jin and A. Orso, "BugRedux: Reproducing field failures for in-house debugging," in Proc. of the 34th Intern. Conf. on Software Engineering (ICSE), 2012, pp. 474–484.

[6] S. Nessa, M. Abedin, W. E. Wong, L. Khan, and Y. Qi, "Software Fault Localization Using N -gram Analysis," in Proc. of the 3rd Conference on Wireless Algorithms, Systems, and Applications, 2008, pp. 548–559.

[7] H. Zhang, L. Gong, and S. Versteeg, "Predicting bug-fixing time: an empirical study of commercial software projects," in Proc. of the International Conference on Software Engineering (ICSE), pp. 1042–1051, 2013.

[8] S. Kim, T. Zimmermann, E. J. Whitehead Jr., and A. Zeller, "Predicting Faults from Cached History," in Proc. of the 29th Inter. Conf. on Software Eng. (ICSE), 2007, pp. 489–498.

[9] M. Nayrolles, A. Hamou-Lhadj, T. Sofiene, and A. Larsson, "JCHARMING : A Bug Reproduction Approach Using Crash Traces and Directed Model Checking," in Proc. of the 22nd International Conference on Software Analysis, Evolution and Reengineering (SANER), 2015, pp. 101–110.

[10] C. Sun, D. Lo, X. Wang, J. Jiang, and S.-C. Khoo, "A discriminative model approach for accurate duplicate bug report retrieval," in Proc. of the 32nd ACM/IEEE International Conference on Software Engineering (ICSE), 2010.

[11] A. T. Nguyen, T. T. Nguyen, T. N. Nguyen, D. Lo, and C. Sun, "Duplicate bug report detection with a combination of information retrieval and topic modeling," in Proc. of the 27th IEEE/ACM International Conference on Automated Software Engineering(ASE), 2012, pp. 70-79.

[12] P. Runeson, M. Alexandersson, and O. Nyholm, "Detection of Duplicate Defect Reports Using Natural Language Processing," in Proc. of the 29th International Conference on Software Engineering, 2007, pp. 499–510.

[13] N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, and T. Zimmermann, "What makes a good bug report?," in Proc. of the 16th ACM SIGSOFT Intern Symposium on Foundations of Software Engineering, 2008.

[14] R. Wu, H. Zhang, S. Kim, and S. Cheung, "Relink: recovering links between bugs and changes," in Proc. of the 19th Symposium and the 13th European Conference on Foundations of Software Engineering (FSE), 2011, pp. 15–25.

[15] G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, and E. Merlo, "Recovering traceability links between code and documentation," IEEE Tranactions on Software Engineering, vol. 28, no. 10, pp. 970–983, 2002.

[16] F. Khomh, B. Chan, Y. Zou, and A. E. Hassan, "An Entropy Evaluation Approach for Triaging Field Crashes: A Case Study of Mozilla Firefox," in Proc. of the 18th Working Conference on Reverse Engineering, 2011, pp. 261–270.

[17] G. Jeong, S. Kim, and T. Zimmermann, "Improving bug triage with bug tossing graphs," in Proc. of the 7th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Eoftware Engineering (FSE), 2009, pp. 111-120.

[18] A. Tamrawi, T. T. Nguyen, J. Al-Kofahi, and T. N. Nguyen, "Fuzzy set-based automatic bug triaging," in Proceeding of the 33rd international conference on Software engineering (ICSE), 2011, pp. 884-887.

[19] A. Lamkanfi, S. Demeyer, E. Giger, and B. Goethals, "Predicting the severity of a reported bug," in Proc. of the 7th IEEE Working Conference on Mining Software Repositories (MSR), 2010, pp. 1–10.

[20] A. Lamkanfi, S. Demeyer, Q. D. Soetens, and T. Verdonck, "Comparing Mining Algorithms for Predicting the Severity of a Reported Bug," in Proc. of the 15th European Conference on Software Maintenance and Reengineering (CSMR), 2011, pp. 249–258.

[21] Y. Tian, D. Lo, and C. Sun, "Information Retrieval Based Nearest Neighbor Classification for Fine-Grained Bug Severity Prediction," in Proc. of the 19th Working Conference on Reverse Engineering (WCRE), 2012, pp. 215–224.

[22] E. Shihab, A. Ihara, Y. Kamei, W. M. Ibrahim, M. Ohira, B. Adams, A. E. Hassan, and K. Matsumoto, "Predicting Re-opened Bugs: A Case Study on the Eclipse Project," in Proc. of the 17th Working Conference on Reverse Engineering (WCRE), 2010, pp. 249–258.

[23] T. Zimmermann, N. Nagappan, P. J. Guo, and B. Murphy, "Characterizing and predicting which bugs get reopened," in Proc. of the 34th International Conference on Software Engineering (ICSE), 2012, pp. 1074–1083.

[24] D. Lo, "A Comparative Study of Supervised Learning Algorithms for Re-opened Bug Prediction," in Proc. of 17th European Conference on Software Maintenance and Reengineering (CSMR), 2013, pp. 331–334.

[25] J. Zhou, H. Zhang, and D. Lo, "Where should the bugs be fixed? More accurate information retrieval-based bug localization based on bug reports," in Proc. of the 34th International Conference on Software Engineering (ICSE), 2012, pp. 14–24.

[26] D. Kim, Y. Tao, S. Member, S. Kim, and A. Zeller, "Where Should We Fix This Bug? A Two-Phase Recommendation Model," IEEE Transactions on Softwware Engineering, vol. 39, no. 11, 2013, pp. 1597–1610.

[27] N. Bettenburg, R. Premraj, and T. Zimmermann, "Duplicate bug reports considered harmful … really?," in Proc. of the IEEE International Conference on Software Maintenance (ICSM), 2008, pp. 337–345.

[28] Y. Tian, C. Sun, and D. Lo, "Improved Duplicate Bug Report Identification," in Proc. of the 16th European Conference on Software Maintenance and Reeng., 2012, pp. 385–390.

[29] N. Jalbert and W. Weimer, "Automated duplicate detection for bug tracking systems," in Proc. of IEEE International Conference on Dependable Systems and Networks With FTCS and DCC (DSN), 2008, pp. 52–61.

[30] L. D. Panjer, "Predicting Eclipse Bug Lifetimes," in Proc. of the 4th International Workshop on Mining Software Repositories, 2007, pp. 29–29.

[31] P. Bhattacharya and I. Neamtiu, "Bug-fix time prediction models: Can We Do Better?" in Proc. of the 8th Working Conference on Mining software repositories (MSR), 2011, pp. 207-210.

[32] C. S. S. Perry, E. Dewayne, "Software faults in evolving a large, real-time system: a case study," in Software Engineering—ESEC, 1993, pp. 48–67.

[33] I. Herraiz, D. M. German, J. M. Gonzalez-Barahona, and G. Robles, "Towards a simplification of the bug report form in eclipse," in Proc. of the International Workshop on Mining Software Repositories, 2008, pp. 145-148.

[34] J. Anvik, L. Hiew, and G. C. Murphy, "Who should fix this bug?" in Proceeding of the 28th International Conference on Software Engineering (ICSE), 2006, pp. 361-370.

[35] G. Bortis and A. van der Hoek, "PorchLight: A tag-based approach to bug triaging," in Proc. of the 35th International Conference on Software Eng. (ICSE), 2013, pp. 342–351.

[36] D. Kim, X. Wang, S. Member, S. Kim, A. Zeller, S. C. Cheung, S. Member, and S. Park, "Which Crashes Should I Fix First?: Predicting Top Crashes at an Early Stage to Prioritize Debugging Efforts," IEEE Transactions on Software Enineering, vol. 37, no. 3, pp. 430–447, 2011.

[37] S. Kim, T. Zimmermann, K. Pan, and E. Jr. Whitehead, "Automatic Identification of Bug-Introducing Changes," in Proc. of the 21st International Conference on Automated Software Engineering (ASE), 2006, pp. 81–90.

[38] S. Eldh, S. Punnekkat, H. Hansson, and P. Jönsson., "Component testing is not enough-a study of software faults in telecom middleware," in Testing of Software and Communicating Systems, 2007, vol. 4581, pp. 74–89.

[39] S. Eldh, "On Test Design," PhD Dissertation, Mälardalen University Press Dissertations, No. 105, 2011.

[40] M. Hamill and K. Goseva-Popstojanova, "Exploring fault types, detection activities, and failure severity in an evolving safety-critical software system," Springer Software Quality Journal, vol. 23, no. 2, pp. 229–265, 2014.