An Empirical Analysis of Reopened Bugs Based on Open Source Projects

Qing Mi
Department of Computer Science
City University of Hong Kong
Kowloon, Hong Kong
Qing.Mi@my.cityu.edu.hk

Jacky Keung
Department of Computer Science
City University of Hong Kong
Kowloon, Hong Kong
Jacky.Keung@cityu.edu.hk

ABSTRACT

Background: Bug fixing is a long-term and time-consuming activity. A software bug experiences a typical life cycle from newly reported to finally closed by developers, but it could be reopened afterwards for further actions due to reasons such as unclear description given by the bug reporter and developer negligence. Bug reopening is neither desirable nor could be completely avoided in practice, and it is more likely to bring unnecessary workloads to already-busy developers. ***Aims:*** To the best of our knowledge, there has been a little previous work on software bug reopening. In order to further study in this area, we perform an empirical analysis to provide a comprehensive understanding of this special area. ***Method:*** Based on four open source projects from Eclipse product family, they are CDT, JDT, PDE and Platform, we first quantitatively analyze reopened bugs from perspectives of proportion, impacts and time distribution. After initial exploration on their characteristics, we then qualitatively summarize root causes for bug reopening, this is carried out by investigating developer discussions recorded in Eclipse Bugzilla. ***Results:*** Results show that 6%-10% of total bugs will lead to reopening eventually. Over 93% of reopened bugs place serious influence on the normal operation of the system being developed. Several key reasons for bug reopening have been identified in our empirical study. ***Conclusions:*** Although reopened bugs have significant impacts on both end users and developers, it is quite possible to reduce bug reopening rate through the adoption of appropriate methods, such as promoting effective and efficient communication among bug reporters and developers, which is supported by empirical evidence in this study.

CCS Concepts

•**Software and its engineering** → **Empirical software validation;** *Risk management;* Software testing and debugging;

Keywords

reopened bugs, bug reports, bug tracking system, open source projects, empirical software engineering

EASE '16, June 01-03, 2016, Limerick, Ireland
 ISBN 978-1-4503-3691-8/16/06…$15.00
DOI: http://dx.doi.org/10.1145/2915970.2915986

1. INTRODUCTION

During software development and maintenance, bug fixing is one of the most significant and indispensable processes, of which requires a considerable amount of labor-intensive resources. Although developers have been striving to their best to ensure a software being produced is of high quality, software bugs are still inevitable in practice. Normally, a software bug would experience a life cycle from newly reported to finally closed. However, there exists a certain type of bugs for which might be reopened for further processing once or more times during the entire life period. We consider this type of bugs as *reopened bugs*.

A reopened bug is a disappointment to end users in the open source community, assuming that a particular bug was announced to be fixed, but it is actually otherwise and needs to be reopened for further actions. In addition, a high bug reopening rate may indicate instability of development process as given in Zimmermann's study [23]. Research on reopened bugs is important as it provides capabilities needed to:

- Measure the quality of debugging process.
- Estimate software maintenance effort taking account of reopening rate.
- Identify procedures and tools which need to be optimized.
- Raise developer's awareness of reopening-prone bugs.

A number of related work have focused in the area of software defects, such as predicting the amount of defects in a system [9, 13, 21], or measuring the time it takes to fix a defect [3, 11, 17], whereas all bugs were treated equally in those work. To the best of our knowledge, only a few previous studies considered reopened bugs issue separately. For instance, Shihab et al. [15] built decision trees to predict whether or not a bug will be reopened. Zimmermann et al. [23] assessed the impact of various factors on bug reopening using Windows bug reports. The purpose of this paper is to complement the findings of previous studies and provide a complete horizon on its future research directions. Thus, we perform a comprehensive analysis of reopened bugs and present conclusions from four different perspectives: proportion, impacts, time distribution and root causes. The main results of this paper are as follows:

- **Measure impacts of reopened bugs.** There are 6%-10% of total bugs leading to reopening, among which 98% are reopened once or twice. Reopened bugs negatively affect both end users and developers. Over 93% of reopened bugs put serious influence on the normal operation of the system. In addition, more than 20% workloads are brought to already-busy developers due to bug reopening.
- **Locate time-consuming steps.** The average lifetime for a reopened bug is 3-6 months. Half of the time is spent in the hidden period. Once a bug gets reopened, it takes approximately a month's time to fix it after being assigned to the right developer.
- **Identify factors affecting bug reopening.** Bugs which are not resolved by a check-in into the code repository are considered reopening-prone. There are various reasons for bug reopening, in which incomplete fix is the most common one.

We believe that our results are able to provide developers and researchers with better understanding of patterns and factors behind reopened bugs, this is to minimize bug reopening rate as the ultimate goal.

This paper is organized as follows. Section 2 introduces background information about Bug Tracking Systems. Section 3 first describes three research questions and target projects used in this study. Then an overview of data analysis methods is presented. Section 4 shows results of our study. Section 5 presents root causes for bug reopening summarized from empirical investigation. In Section 6, we discuss threats to validity. Section 7 provides related work and Section 8 concludes this paper.

2. BACKGROUND

2.1 Bug Tracking System

A Bug Tracking System (BTS) is a software application, which enables end users to file bugs directly. BTS aims to improve debugging process in a way of keeping track of various bug-related information with a well structure, for instance, the reporter's identity, the operating system where the bug is found, the description of erroneous behavior, the developer who is responsible for the bug, etc. Generally, BTSs fall broadly into three categories[1]: client-server (e.g., Jira[2]), distributed (e.g., Fossil[3]) and hosted (e.g., YouTrack[4]).

Bugzilla[5] is one of mature client-server BTSs. Currently, quite a number of open source projects adopt Bugzilla as their own BTSs in an attempt to manage and keep track of all reported bugs. A great many bug reports containing abundant information are stored in Bugzilla, which is of great significance for further research.

2.2 Bug Life Cycle

Different Bug Tracking Systems may involve different bug status and follow different bug life cycles correspondingly. In Bugzilla, a bug may experience all of or part of the following status during the entire lifetime[6]:

- **UNCONFIRMED**: a bug should be labeled as UNCONFIRMED by default when a user submit it at the very beginning.
- **NEW**: a developer with proper permissions examines the bug to be valid, and then changes its status to NEW.
- **ASSIGNED**: a bug has been assigned to a developer yet not resolved should get ASSIGNED.
- **RESOLVED**: a bug should be marked as RESOLVED after a resolution has been performed.
- **VERIFIED**: a bug has been successfully verified, which indicates the resolution is appropriate.
- **CLOSED**: a bug has been fixed or it will not be worked on any more.
- **REOPENED**: a bug has been claimed to be solved by developers, nevertheless, someone reproduces the issue on a version that the bug should not appear.

Figure 1 depicts the simplified life cycle (workflow) of a bug in Bugzilla. When a bug is newly reported by a common user, it begins life with the status UNCONFIRMED. The status will be updated to NEW after further confirmation. It should be noticed that a bug could be labeled as NEW directly if its reporter is a person with appropriate permissions (e.g., "can confirm" privilege). By default, the bug is first assigned to the component owner who has the right to re-assign bugs to relevant developers. Note that the status of a bug would be converted back to NEW if it goes through reassigning. Once the bug is accepted by the appointed developer for further processing, its status would be updated to ASSIGNED. After a particular resolution (e.g., FIXED, DUPLICATE, WONT-FIX) is offered, the bug will be labeled as RESOLVED. Ideally, all reported bugs should be verified by test engineers. Once the bug is confirmed to be fully resolved, it is then marked as VERIFIED and CLOSED successively. However, the bug may be REOPENED for further actions due to diversified reasons.

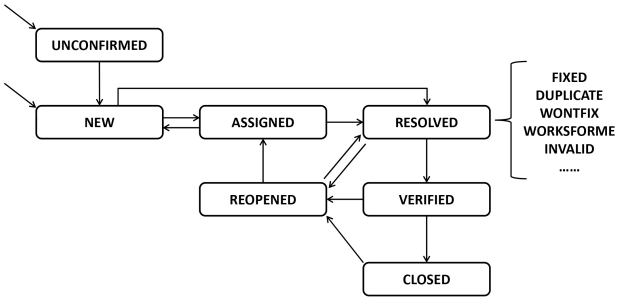

Figure 1: Bug Fixing Process (Simplified Life Cycle of a Bugzilla Bug)

Particularly, the status UNCONFIRMED is not included in our analysis, for such a reason that bug reporters might file an invalid bug which is actually an intended behavior. Therefore, we consider NEW as the initial status for a real bug. Furthermore, only resolved bugs, which have gone through the complete life cycle, are involved in our study.

[1]https://en.wikipedia.org/wiki/Bug_tracking_system
[2]https://www.atlassian.com/software/jira
[3]http://www.fossil-scm.org
[4]http://www.jetbrains.com/youtrack
[5]https://www.bugzilla.org
[6]https://www.bugzilla.org/docs/3.6/en/html/lifecycle.html

3. RESEARCH DESIGN

In this section, we first describe the research questions and corresponding motivations in detail. Then, we briefly introduce the target projects and dataset adopted in our study. After that, the research methodologies are presented for each research question.

3.1 Research Questions

In order to gain insight into reopened bugs, we conduct an empirical investigation in this study. More specifically, we will address the following research questions:

RQ1. What is the proportion of reopened bugs in bug repository? What are the impacts to end users and developers?

The proportion of reopened bugs indicates the seriousness of the problem and the significance of our research, which should be identified first.

Also, it is important to know whether reopened bugs would have negative influences on functionality of the system, and moreover, would affect user experience seriously. If the answer is no, it is unnecessary to conduct further study. Additionally, measuring impact from developer's perspective is significant as well.

RQ2. How long does it take to fully fix a reopened bug? Where is most of the time spent?

The answer to this question is important, for the reason that understanding the time distribution of reopened bugs is helpful to both engineers and scientists. They can work on reducing the time-consuming during the overall debugging process based on our results.

RQ3. What kinds of bugs are reopening-prone? What are the underlying reasons for bug reopening?

Comprehending the root causes for bug reopening can make developers to be careful if they want to close a bug. Also, these information is able to help researchers conduct further studies more methodically.

3.2 Target Projects and Dataset

Eclipse[7] is one of the open source projects which introduce Bugzilla to implement bug tracking and test management. It is a powerful and cross-platform (i.e., Linux, Mac OS X, Solaris, Windows) IDE (Integrated Development Environment), which provides tools for coding, building, running and debugging applications. As a widely used software, Eclipse has a long history as well as sufficient bug reports to support in-depth analysis. Research results obtained from Eclipse's bug repository are meaningful and persuasive.

Instead of including all products of the Eclipse project, a subset of the four most active products which come from different domains are selected as our research objects, namely CDT[8], JDT[9], PDE[10] and Platform[11]. CDT (C/C++ Development Tooling) is an industrial-strength C/C++ IDE based on the Eclipse platform. JDT (Java Development Tools) provides a set of plug-ins implementing a Java IDE to the Eclipse platform. PDE (Plug-in Development Environment) supplies comprehensive tools to support development of Eclipse plug-ins. Platform is a group of frameworks and services, which introduces a batch of developing toolkit to support the usage of Eclipse.

According to our requirements, Lamkanfi et al's [8] defect tracking dataset, which is filtered to contain only genuine bugs and designed to cover the whole bug-triage life cycle, is adopted in this study to conduct our investigation. The dataset provides all the updates of the most relevant bug report attributes (e.g., severity, priority, resolution) in the format of a bundle of elementary XML files, which is easy to parse, import, transform, etc.

3.3 Methodology

We first describe data analysis methods used for measuring the impact of reopened bugs on end users and developers respectively to address RQ1. After that, we present the definitions and descriptions of metrics used to resolve RQ2.

3.3.1 The Impact of Reopened Bugs on End Users

There are two attributes in Bugzilla can be used to measure the impact of reopened bugs on end users, namely *severity* and *priority*. Severity is provided by bug reporters and should accurately reflect the level of impact the bug has upon the reporter. Although this attribute could be modified by developers afterwards if it is obviously incorrect, it is still the most representative of the reporter's perspective. On the contrary, priority is set by the component owner or the developer who the bug is assigned to, which indicates the bug's relative importance as compared to others. It is usually utilized to schedule tasks. A bug with high priority may have less severity, and vice versa.

In summary, we prefer severity, which is more precise and meaningful to reflect the impact of reopened bugs on end users, over priority to address the first half of RQ1. According to the documentation of Eclipse Bugzilla, the severity attribute can be of following types[12], which is arranged from highest to lowest level.

- **blocker**: no development and/or testing work can be implemented due to the bug.
- **critical**: the bug affects critical functionality and/or critical data, such as crashes and severe memory leak.
- **major**: major loss of function in an important area because of the bug.
- **normal**: the functionality as a whole is not impacted. However, under specific circumstances, some loss of important feature is caused by the bug.
- **minor**: the bug affects minor functionality and/or non-critical data, where easy workaround is presented.
- **trivial**: the bug does not affect functionality or data. It is merely an inconvenience, such as misspelled words.

The degree of a bug's severity may be changed during its full lifetime as a result of re-evaluating by relevant staff. Taking account of this situation, we adopt the last status, which we believe is the considered opinion, as a bug's real severity level.

Based on the above analysis, bugs with severity of normal or above could well prevent the system from working properly. Therefore, we calculate the proportion of reopened bugs with severity level of normal or above to measure their impacts on end users.

[7]https://www.eclipse.org
[8]https://eclipse.org/cdt
[9]https://eclipse.org/jdt
[10]https://eclipse.org/pde
[11]https://eclipse.org/platform
[12]https://wiki.eclipse.org/Eclipse/Bug_Tracking

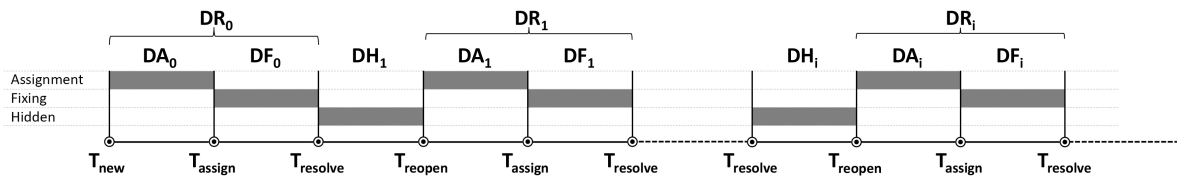

Figure 2: Typical Timeline of a Reopened Bug

3.3.2 The Impact of Reopened Bugs on Developers

Reopened bugs also place considerable influence on developers. To quantify this impact, we count the number of developers involved in the entire debugging process based on the *assigned_to* attribute in Bugzilla, which is used to demonstrate the person who is responsible for the bug in current stage. Although there may be other reasons for bug re-assigning (e.g., rescheduling tasks), this attribute is still a strong support in terms of measuring the extra workloads for developers.

As described in Section 2.2, generally, a reported bug is first assigned to the inbox of the selected component (e.g., cdt-debug-inbox@eclipse.org). If the bug is valid, it is then re-assigned to a proper developer for further processing. Therefore, a convention bug usually involves two developers, whereas reopened bugs may well require more. To rationalize our statistics, we introduce a weighted score of 0.5 for the assigned developer who was responsible for that bug before, since he/she has had experiences on analyzing, locating and fixing such bug, which is a clear advantage that needed to be taken into account. Besides, it should be noticed that the assigned_to attribute may be empty if the bug is determined not to require any further action at the very beginning. For instance, the bug is a duplicate of an existing bug, which has already been resolved. Thus, only one developer involved in this case.

In conclusion, to answer the second half of RQ1, we first divide bug repository into two groups according to whether they have been reopened, and then count the number of involved developers respectively. Afterwards, Wilcoxon-Mann-Whitney test [10] (using 5% level) is conducted to examine whether the means of two groups are statistically equal. To reduce the chances of obtaining false-positive results (type I errors), Bonferroni correction is applied to the significance level of the Wilcoxon-Mann-Whitney test.

3.3.3 The Time Distribution of Reopened Bugs

In this section, we define terms and metrics used for measuring time distribution of reopened bugs.

Slightly different from previous description in Section 2.2, a reopened bug experiences a typical life cycle as: 1) newly filed by a reporter; 2) assigned to a developer; 3) resolved by the assigned developer; 4) reopened due to some reason; 5) assigned to a developer (probably the same one as in stage 2); 6) resolved by the assigned developer, in which stage 4-6 may appear several times sequentially.

Based on these knowledge, we first define several key time points as shown in Table 1, and then split the entire lifetime of reopened bugs into several intervals correspondingly. Figure 2 serves as a visual demonstration for the timeline of a reopened bug.

It should be noticed that a bug may be resolved directly without any assignment. In this case, we consider the assignment being implemented immediately and all of the time is spent in resolution stage. Besides, a bug may be re-assigned several times before it gets to the right developer who fixes the bug eventually. Thus, we choose the time-stamp of last assignment as T_{assign} in our analysis.

Table 1: Definition of Time Points Extracted from the Lifetime of Reopened Bugs

Time Point	Description
$\mathbf{T_{new}}$	The time-stamp* when a bug is confirmed as valid and starts its life cycle.
$\mathbf{T_{assign}}$	The time-stamp when a bug is assigned to the right developer.
$\mathbf{T_{resolve}}$	The time-stamp when a bug is labeled as resolved by the assigned developer.
$\mathbf{T_{reopen}}$	The time-stamp when a bug is reopened due to diversified reasons.

*UNIX time notation with an accuracy of second is used as time-stamp in the dataset.

Following the aforementioned definitions in Table 1, the metrics we use to measure the time distribution of reopened bugs are listed as follows:

- **Duration of Survival by Days (DS):**

  The survival time (lifetime) of a reopened bug, which is depicted by Equation 1. The specific meaning of parameters involved in Equation 1 are detailed below.

  It should be noticed that i is used to indicate the i-th reopening. When i equals to 0, it stands for normal debugging procedure.

$$DS = DR_0 + \sum_{i=1}^{n} (DH_i + DA_i + DF_i)\,,\ n = 1, 2, 3, \cdots \quad (1)$$

- **Duration of Assignment by Days ($\mathbf{DA}_i$):**

  The interval between a bug is reopened (T_{reopen}) and it is assigned later (T_{assign}) during the i-th reopening.

- **Duration of Fixing by Days ($\mathbf{DF}_i$):**

  The interval between a bug is assigned (T_{assign}) and it is resolved later ($T_{resolve}$) during the i-th reopening.

- **Duration of Resolution by Days ($\mathbf{DR}_i$):**

  The interval between a bug is reopened (T_{reopen}) and it is resolved later ($T_{resolve}$) during the i-th reopening. Mathematically, $DR_i = DA_i + DF_i$.

- **Duration of Hidden by Days (DH_i):**

  The interval between a bug is resolved ($T_{resolve}$) and it is reopened later (T_{reopen}) the i-th time, which is used to measure the hidden time of unresolved bugs.

Generally, Bug Tracking Systems conserve the whole history of each bug's status changes. Making use of these information, DA_i, DF_i, DR_i and DH_i for all reopened bugs can be figured out accordingly.

4. RESULTS

In this section, we present the results of our empirical study with respect to the three research questions.

RQ1. Proportion & Impacts.

Proportion. As shown in Table 2, reopened bugs account for approximately 7.5% of the entire bug repository. The proportion ranges from 6% to 10% in each project. We believe that this is a huge number, which is indicative of the significance for further study.

Table 2: Proportion of Reopened Bugs

Projects	# of Bug Reports	Reopened Bugs		# of Reopened Times					
		#	%	1	2	3	4	5	8
CDT	5640	394	6.986%	362	28	4	-	-	-
JDT	10814	1055	9.756%	915	123	11	5	-	1
PDE	5655	348	6.154%	316	30	2	-	-	-
Platform	24775	1711	6.906%	1533	146	23	7	2	-
Total	46884	3508	7.482%	3126	327	40	12	2	1

A deeper analysis of Table 2 reveals that, within the scope of reopened bugs, bugs which have been reopened once or twice make up the two biggest items. Together they comprise over 98% of the total. Only a minority of bugs (approximately 2%) have been reopened thrice or more. Considering that these bugs are reopened repeatedly, we prefer to analyze their reopening reasons first in the following study.

Impact on end users. Table 3 demonstrates the severity distribution of reopened bugs. Our results show that reopened bugs have severity level of normal or above account for 93.3% of the total. The proportion varies from 90.6% to 96.0% in each project. Based on the assumption as given in Section 3.3.1, we obtain the conclusion that 93.3% of reopened bugs place serious influence on end users.

Table 3: Severity Distribution of Reopened Bugs

Projects	Blocker	Critical	Major	Normal	Minor	Trivial
CDT	9	20	38	307	13	7
JDT	12	38	150	756	69	30
PDE	2	3	47	282	10	4
Platform	42	82	240	1245	71	31
Total	65	143	475	2590	163	72

Note that only 4.1% of reopened bugs are critical, while the percentage for major bugs is 13.5%. Nonetheless, the absolute numbers are 143 and 475 respectively. Taking account of the serious effects (e.g., crashes, major loss of functions), these numbers are high. Thus, we give our priority to study the reopening reasons of these severe bugs in the following sections.

Impact on developers. Table 4 presents the statistical results: the number of developers involved in debugging process of reopened bugs is 1.196-1.311 times as much as those of non-reopened bugs. The P-value of Wilcoxon-Mann-Whitney test, which is less than 2.2e-16 in all four projects, also provides powerful support to our results. From another perspective, we have observed that around 11.0% ($\frac{378}{3508}$) of reopened bugs are re-assigned 5 times or more, which also indicates the complexity and severe influence of bug reopening.

Table 4: Number of Developers Involved

Projects	Mean		Mean Ratio	P-value
	Reopened	Non-reopened		
CDT	2.028	1.695	1.196	< 2.2e-16
JDT	2.285	1.907	1.198	< 2.2e-16
PDE	2.223	1.696	1.311	< 2.2e-16
Platform	2.308	1.874	1.232	< 2.2e-16

RQ2. Survival Time & Time Distribution.

Survival time. To quantify and measure the survival time (lifetime) of reopened bugs, we use the same grouping and testing method as in RQ1. In Table 5, we can observe that it takes 56.48-99.71 days on average to bring a non-reopened bug to the end, while the duration for a reopened bug runs up to 1.64-2.14 times than normal. The P-value of Wilcoxon-Mann-Whitney test, which is less than 2.2e-16, also demonstrates the lifetime of reopened bugs is significantly longer than that of non-reopened bugs from statistical angle.

Time distribution. Table 6 exhibits the average and maximum DH_i, DA_i and DF_i in all four projects. Due to the low data volume, it is difficult to obtain meaningful and reliable statistical result on the basis of information from bugs reopened thrice or more. Thus, we stop our iterative calculation at the twice reopening.

Apart from DA_0 and DF_0, we consider each DH_i, DA_i and DF_i as a group representing the i-th reopening. It can be observed that DH_i is approximately equal to DR_i ($DR_i = DA_i + DF_i$) in each group. As the reopening times increases, both DH_i and DR_i decrease significantly. We suppose DA_i and DF_i would decline as well, for the reason that a bug which has been reopened several times should have come into notice of developers. To our surprise, DA_i and DF_i present no obvious pattern as time goes by. As shown in DF_i, it usually takes one month to fix a bug. Moreover, DF_i is much larger than DA_i in general, the ratio between DF_i and DA_i even reaches 5 to 6.

Besides, it should be noticed that the maximum DH_i for some of the bugs go up to 4 years. Considering that Eclipse Foundation performs annual release since 2006, these bugs actually remain undiscovered for several versions. On the other hand, the maximum DA_i and DF_i illustrate that, it takes more than 4.5 years to assign a bug and another 4.5 years to fix it, which is also a surprising phenomenon.

RQ3. Reopening-proneness & Root Causes.

In Eclipse Bugzilla, when a bug is resolved by developers, a particular attribute *resolution* is offered to describe what happened to the bug. The classification of resolution is listed as follows[13]:

- **FIXED**: the bug has been fixed by a modification to source code.
- **DUPLICATE**: the bug has already been reported, which means it is a duplicate of an existing bug.
- **INVALID**: the reported issue is not a bug, perhaps an intended behavior.

[13]https://www.bugzilla.org/docs/3.6/en/html/lifecycle.html

Table 5: Descriptive Statistics of DS* and Results of Wilcoxon-Mann-Whitney Test

Projects	Groups	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	Mean Ratio	P-value
CDT	non-reopened	0.0002	0.4204	4.6780	87.3900	44.1700	1875.0000	1.64	< 2.2e-16
	reopened	0.0366	7.8150	36.3700	143.0000	172.5000	1481.0000		
JDT	non-reopened	0.0002	0.9121	8.8640	74.1300	47.1900	1860.0000	2.14	< 2.2e-16
	reopened	0.0068	9.5870	52.8700	159.0000	188.8000	1791.0000		
PDE	non-reopened	0.0002	0.3225	4.3400	56.4800	33.9100	1552.0000	1.64	< 2.2e-16
	reopened	0.0149	5.5180	26.1200	92.3500	77.5200	1338.0000		
Platform	non-reopened	0.0002	0.9089	10.3100	99.7100	69.2600	1935.0000	1.82	< 2.2e-16
	reopened	0.0026	9.7220	47.8500	181.9000	209.0000	1908.0000		

*DS: Duration of Survival by Days

Table 6: Time Distribution of Reopened Bugs*

Projects	DA_0		DF_0		DH_1		DA_1		DF_1		DH_2		DA_2		DF_2	
	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.
CDT	24.29	640.80	18.23	1464.00	74.05	854.10	7.91	713.00	12.05	757.10	40.47	588.20	4.09	48.90	9.08	85.61
JDT	19.03	1673.00	30.04	1357.00	45.88	1164.00	9.43	1170.00	48.04	1701.00	51.44	527.10	5.13	319.30	43.60	1418.00
PDE	17.85	875.20	14.93	428.90	18.46	731.10	6.29	388.00	29.58	1252.00	9.99	95.68	21.16	254.40	33.18	392.70
Platform	17.28	1501.00	29.79	1292.00	57.03	1319.00	15.42	1265.00	45.55	1468.00	49.21	979.70	30.07	1100.00	41.71	992.70
Total	18.65	1673.00	27.09	1464.00	51.76	1319.00	11.87	1265.00	40.95	1701.00	46.07	979.70	17.95	1100.00	39.07	1418.00

*DA: Duration of Assignment by Days; DF: Duration of Fixing by Days; DH: Duration of Hidden by Days.

- **WORKSFORME**: the bug cannot be reproduced.
- **WONTFIX**: a decision is made not to fix the bug.
- **NOT_ECLIPSE**: the bug is caused by other software or circumstance.
- **LATER**: the bug will not be fixed in the current release. LATER is no longer part of the default list of resolutions since Bugzilla 3.0[14].
- **REMIND**: further information is needed from the reporter or someone else. REMIND is no longer part of the default list of resolutions since Bugzilla 3.0[14].

In order to address the first half of RQ3, we first divide the whole bug repository into two groups (i.e., fixed and non-fixed) based on whether they are resolved via modification to source code, and then calculate and compare the reopening rate in both groups.

The first part of Table 7 illustrates the statistical results according to our grouping. In each field, the value before slash is of fixed group, while the value after slash is of non-fixed group. It can be observed that the fixed group accounts for 58.63% ($\frac{27487}{46884}$) in the whole bug repository, in which around 6.54% bugs result in reopening in the end. While the number for non-fixed group is 8.81%, which indicates reopening-proneness of non-fixed bugs.

To verify our hypothesis, we take all reopened bugs as population to conduct the same analysis one more time. The result is presented in the second part of Table 7, which is 9.63% versus 13.33% for group of fixed and non-fixed respectively. To our surprise, statistics show that reopening rate goes up in both groups as compared to the first analysis, which signifies reopened bugs are more likely to be reopened again to some extent.

5. ROOT CAUSES FOR BUG REOPENING

Following the methodology used by Saha et al. [14], we continue in-depth study as part of RQ3 in order to find out the main reasons behind bug reopening.

[14]https://www.bugzilla.org/releases/3.0/new-features.html

5.1 Bug Case Selection Criteria

Within the range of reopened bugs, we select 3% proportionally from the four target projects. The absolute number is 105, in which CDT, JDT, PDE and Platform account for 12, 32, 10 and 51 respectively.

In each project, we give preference to the bugs reopened three or more times, which are typical cases involving rich information for exploring reopening reasons. Besides, for bugs reopened once or twice, those with severity level of major or above, which indicates serious impact on user experience, are preferred. We present the selection results for our investigation in Table 8.

Table 8: The Results of Selection for Investigation

Projects	Bugs Reopened Thrice or Above	Severity			Total
		Blocker	Critical	Major	
CDT	4	0	5	3	12
JDT	17	2	5	8	32
PDE	2	2	3	3	10
Platform	32	8	5	6	51

5.2 Root Causes

Generally, each reported bug in Eclipse Bugzilla has a description and several comments from the bug reporter and developers. A normal bug usually involves a few comments, whereas a reopened bug, which has longer life cycle and more confusing causation, tends to raise intense discussion and large numbers of comments, in which contain rich information from technical solutions to sentiments.

As a result of analyzing developer comments recorded in Eclipse Bugzilla, we summarize eight main reasons (see below) for bug reopening in this section. Several key comments are quoted in each category for better understanding.

Table 9 offers an overview of our investigation. Normally, a bug ID consists of 6 numeric digits. The suffix *-i* denotes the *i*-th reopening of the bug.

1) **Bug reporter's unclear description**

Table 7: Fixed VS. Non-fixed

Projects	Reported Bugs			Reopened Bugs		
	Total #	Reopened #	Reopening Rate	Total #	Reopened #	Reopening Rate
CDT	4203 / 1437	295 / 99	7.02% / 6.89%	336 / 58	28 / 4	8.33% / 6.90%
JDT	5724 / 5090	423 / 632	7.39% / 12.42%	595 / 460	66 / 74	11.09% / 16.09%
PDE	3932 / 1723	238 / 110	6.05% / 6.38%	276 / 72	22 / 10	7.97% / 13.89%
Platform	13628 / 11147	843 / 868	6.19% / 7.79%	1108 / 603	107 / 71	9.66% / 11.77%
Total	27487 / 19397	1799 / 1709	6.54% / 8.81%	2315 / 1193	223 / 159	9.63% / 13.33%

Description of the bug offered by the reporter may be incomplete, ambiguous, even inaccurate. Misguided developers can not reproduce or locate the bug. Based on such unclear information, they directly mark the bug, which is actually a real issue need further action, as non-fixed (i.e., DUPLICATE, NOT_ECLIPSE, INVALID, WORKSFORME, WONTFIX, LATER or REMIND).

Another situation is that developers attempt to get more detailed description from the reporter but failed. The emergence of valuable information afterwards enables the possibility of bug reproduction or bug location, which leads to reopening.

Basically, this kind of reopening could be classified as bug reporter's fault. Bug #144957 in project Platform can be served as a typical example for this reason:

A new bug was filed by the reporter:

- "As of version 3.2, Eclipse allows the creation of nested projects. However, the CVS plugin does not handle this correctly. It reports updates for the parent project and the projects in the subfolders."

However, the assigned developer could not understand it and then required for further information:

- "Marking as REMIND since I don't really understand mean when you say "reports updates for the parent project and the projects in the subfolders". Perhaps you could describe the scenario in which you use nested projects and how you think CVS should work in that scenario (and please reopen the bug when you do so)."

The reporter realized his/her own problem and reopened the bug with detailed description:

- "Sorry for being unclear. My source structure is e.g. as follows and lays outside of my eclipse workspace."

2) **Developer's negligence**

Under some circumstances, bug reporters do provide exhaustive information even with several test cases in the bug report. Nonetheless, developers do not treat the bug seriously. Although the reporter emphasizes negative impact of the issue repeatedly, the reported bug is labeled as non-fixed without analysis and reproduction. Therefore, the bug has to be reopened later as a result of developer's negligence. The following comments regarding to bug #181655 in project Platform illustrates this scenario:

The bug was labeled as WORKSFORME by the assigned developer at the very beginning:

- "Unless I am misunderstanding something, it sounds like this is working as designed (i.e. you actually didn't want to use the same proxy for all proxy types). If you feel there is still an issue, please reopen."

The reporter reopened the bug since the issue still exists:

- "Yes, there is still an issue. The cvs code is not paying attention to the "No Proxy for" field."

Realizing his/her own problem, the developer fixed the bug eventually:

- "Oops, sorry, I missed that."
- "I understand the issue now. I was confused because I was assuming we were talking about extssh but now I see that we are talking about pserver."

3) **Introduction of other problems**

The original bug has been completely fixed, but some unexpected problems, such as instability, even crash, are brought into the system. This kind of issue can be extremely harmful to the whole system. We blame it upon ability deficiency of involved developers. For instance, the reopening of bug #123861 in project Platform:

An unexpected exception was discovered after the bug was resolved:

- "I got this exception after choosing to filter closed projects: java.lang.IllegalArgumentException: Index out of bounds."

The assigned developer reopened the bug to fix the problem:

- "Oops, forgot to check upper bounds of current child index, fixed in HEAD."

4) **A better solution**

The bug has been fully resolved by the assigned developer. Yet a better solution is figured out afterwards (usually by the same person). Considering the advantages, such as the higher efficiency in run-time performance, or the better prevention of potential problems, the bug is reopened in an attempt to merge into the new modification. A developer's comment upon bug #134172 in project JDT represents such a scene:

- "Actually, there is still room for improvement. We can cache the jar files during reconcile."

5) **Incomplete fix**

The bug is thought to be fixed once, but after a period time, someone reproduces the issue on a version that the bug should not appear. The assigned developer makes an attempt to fix the problem, but without complete inspection and verification, the issue reappears eventually. We believe that a good sense of responsibility as well as an overall testing would significantly improve this situation. The reopening reason of bug #306903 in project PDE is displayed in following comment as an example:

Table 9: Investigation of Reopening Reasons

Reasons	Projects	Bug IDs	#	%
1)	CDT	—	27	11.20%
	JDT	134975-1, 139798-1, 139798-2, 142087-1, 144976-1, 154574-2, 155450-1, 195183-1, 276938-1, 178819-3, 178819-4, 178819-6		
	PDE	—		
	Platform	162763-1, 162763-2, 179753-1, 195225-1, 195225-2, 144957-1, 144957-2, 153809-1, 209834-1, 209921-1, 209921-2, 276559-2, 276559-3, 295059-1, 149008-1		
2)	CDT	190730-1, 131824-1, 132702-1, 160359-1	28	11.62%
	JDT	134684-1, 139798-3, 154574-1, 183211-1, 184581-1, 186858-2, 197999-1, 330842-1, 178819-7		
	PDE	233715-1, 206721-1		
	Platform	124263-1, 124263-2, 124263-3, 128545-3, 128545-4, 130440-1, 209921-3, 275589-1, 275589-2, 276559-1, 204953-1, 181655-1, 137190-1		
3)	CDT	293634-2	10	4.15%
	JDT	183211-4, 286601-3		
	PDE	313922-1, 211386-1		
	Platform	123861-3, 209834-3, 162833-1, 184433-2, 305961-1		
4)	CDT	—	19	7.88%
	JDT	125673-1, 125673-3, 128605-1, 128605-2, 134975-2, 183211-2, 277643-1, 332030-1		
	PDE	282803-1, 282803-2, 282803-3		
	Platform	128545-2, 231081-1, 123861-1, 144957-3, 153809-2, 205331-3, 294722-3, 302702-2		
5)	CDT	131824-1, 284690-1, 284690-2, 284690-3, 293634-1, 293634-3, 139797-1, 144095-1, 147999-1, 304588-1	97	40.25%
	JDT	134975-3, 145784-1, 154574-3, 183211-3, 186858-3, 197999-2, 197999-3, 207093-3, 277643-2, 286601-1, 286601-2, 325408-1, 325408-2, 325408-3, 332030-2, 332030-3, 332030-4, 178819-1, 178819-2, 178819-8		
	PDE	233715-2, 230990-1, 274149-1, 173393-1, 227949-1, 306903-1, 306903-2, 306903-3		
	Platform	195225-3, 195225-4, 231081-2, 231081-3, 231081-4, 285243-1, 285243-3, 285243-4, 297310-1, 297310-2, 297310-3, 297310-4, 124257-1, 124257-2, 130440-2, 130440-3, 137091-1, 137091-2, 137091-3, 140834-2, 140834-3, 143936-1, 143936-2, 143936-3, 146880-1, 146880-2, 153809-3, 162534-1, 162534-2, 162534-3, 165108-1, 205229-1, 205229-2, 205229-3, 205331-2, 209834-2, 224908-1, 224908-2, 224908-3, 262262-1, 262262-2, 275589-3, 277120-1, 277120-2, 277120-3, 294722-1, 294722-2, 295059-3, 302702-1, 157044-1, 184433-1, 231352-1, 324155-1, 126901-1, 139262-1, 331818-1, 125703-1, 247444-1, 305357-1		
6)	CDT	190730-2, 190730-3, 236798-1, 236798-2, 236798-3	26	10.79%
	JDT	205418-3, 205418-4, 207093-1, 207093-2, 284871-1, 284871-2, 284871-3, 284872-1, 284872-2, 284872-3, 178819-5		
	PDE	—		
	Platform	162763-3, 162763-4, 162763-5, 124263-4, 179753-2, 179753-3, 179753-4, 165108-3, 205331-1, 295059-2		
7)	CDT	—	24	9.96%
	JDT	125673-2, 130472-1, 130472-2, 130472-3, 132005-1, 135227-1, 135227-2, 135227-3, 136291-1, 186858-1, 205418-1, 205418-2, 236255-1, 259445-1, 286601-4		
	PDE	—		
	Platform	128545-1, 123861-2, 124257-3, 140834-1, 146880-3, 165108-2, 132856-1, 144430-1, 162617-1		
8)	CDT	—	10	4.15%
	JDT	139798-4, 276938-2		
	PDE	—		
	Platform	140861-1, 140861-2, 140861-3, 140861-4, 140861-5, 285243-2, 262262-3, 302702-3		

- "Still the same problem happens in I20100423-0906 build. I have a new finding. I added -Dfile.encoding option to eclipse.ini to determinate the causes."

Actually, a number of previous work [1, 7] have conducted in-depth studies regarding incomplete bug fixes.

6) **Misapprehension**

The bug has been successfully fixed. However, it is reopened later by the reporter, who mistakenly consider the issue still exists. After discussion, the reporter and developers reach an agreement to end the bug. For instance, the submitter reopened bug #284871 in project JDT many a time. Several comments were offered by the developer to explain the situation:

- "You need to get fixed translations from Babel."
- "This is fixed in JDT."
- "As said before, this needs to be fixed in the translation files, so please stop reopening the JDT bugs. If it's not fixed for you then please report against the translation owner."

7) **Inappropriate status**

Although the bug has been totally resolved, the accompanying attributes (e.g., assigned_to, resolution) might be incorrectly or improperly filled. The bug gets reopened in order to update these information. Another scenario which falls into this category is to remove obsoleted status (i.e., LATER and REMIND). A developer's comment upon bug #139427 in project JDT represents such a scene:

- "Opening...since we didn't fix anything, I'm going to change the resolution to "works for me"."

8) **Other reasons**

We place some rare causes into this category. For example, bug #276938 in JDT was deliberately reopened for further review.

5.3 Discussion

As shown in Table 10, we classify root causes summarized in Section 5.2 into four categories for the purpose of systematically analyzing the effective way of reducing bug reopening rate.

Table 10: Classification of Reopening Reasons

Classification	Reasons	Resolution
Poor Communication	1) Bug reporter's unclear description	Non-fixed
	2) Developer's negligence	Non-fixed
	6) Misapprehension	Fixed
Further Improvement	4) A better solution	Fixed
	7) Inappropriate status	—
Unsuccessful Fix	3) Introduction of other problems	Fixed
	5) Incomplete fix	Fixed
Other Reasons	8) Other reasons	—

Reason 1), 2) and 6) in Table 10, which are primarily caused by the problem of misunderstanding and miscommunication, are placed in the same group. We believe that the majority of reopening in this category can be avoided with the following recommendations:

- Clear and detailed bug information (e.g., repro steps and test cases) is provided by bug reporters.
- An overall examination of the reported issue is performed by developers.
- The way of effective and efficient communication is guaranteed among bug reporters and developers.

Reason 4) and 7) are grouped into one single category in Table 10. During our investigation, we notice that some of the bug reopening is due to further improvement (e.g., better solution and more appropriate information) rather than unresolved issue, which is actually not consistent with the definition of the status REOPENED in Bugzilla.

Taken together Table 9, it can be observed that bug reopening caused by unsuccessful fix, i.e., reason 3) and 5), only constitutes approximately a half of the total samples. Thus, a considerable part of bug reopening does not involve modification to the source code in fact, which are relatively easy to prevent by appropriate actions.

6. THREATS TO VALIDITY

We focus on four large and well established open source projects in this study, namely CDT, JDT, PDE and Platform. We believe that these projects supply sufficient bug reports (more than 46,000) to validate our results. However, all the projects involved in our study come from the same source, the Eclipse product family. It is well known that the initial codebase of Eclipse originated from IBM as a corporate project. Therefore, our findings might not suitable for other open source projects. Further study is required to verify the generalization of our results on other popular projects (both open source and industrial).

On the other hand, we only used bugs that have been reported from 2006 to March 2011 to conduct our study. The limited time-span may bring some bias on our findings. This threat to generalization could be mitigated by extending the dataset to perform comprehensive analysis in the future study.

In order to seek out the real reasons for bug reopening, we explored 105 reopened bugs (approximately 3% of the total) in investigation, which may not be considered enough to derive a more general conclusion. Besides, the selection strategy adopted in this study leans towards severe bugs, which may place some limitations on our corresponding conclusions. For future work, we intend to improve selection criteria for bug cases as well as sample size in the investigation to better generalize our results.

7. RELATED WORK

Bug reports. Finding and fixing bugs has always been a major part of software maintenance. A great many studies have been performed to improve debugging process based on bug reports, for instance, bug-proneness prediction [4], bug triage [2, 5], bug localization [22], bug-fix prediction [6] and debugging effort estimation [19]. Zimmermann et al. [24] conducted a survey among developers and users to find out what makes a good bug report. Zhang et al. [20] performed an empirical study of bug-fixing time for three CA Technologies projects. Saha et al. [14] analyzed long lived bugs from different perspectives. Nevertheless, the majority of previous studies simply treated reopened bugs as new bugs. To complement those work, we focus solely on the analysis of reopened bugs in this paper.

Reopened bugs. To our best knowledge, there are only a few studies concentrate on reopened bugs. Shihab et al. [15] performed a case study to predict reopened bugs from four dimensions: work habits, bug report, bug fix and team. Making use of decision trees and top node analysis, they found that the comment text and the last status are the most important factors to predict bug reopening. Xia et al. [18] evaluated the effectiveness of 10 state-of-the-art supervised learning algorithms to predict if a bug would get reopened. They also demonstrated that Bagging and Decision Table (IDTM) achieved the best performance as compared to other algorithms. Souza et al. [16] studied how the adoption of rapid release cycles impacts bug reopening rate. Pan et al. [12] explored 24 interaction factors from logs recorded by Mylyn to study their influence on bug reopening. One thing our study and [12, 15, 18] have in common is that we all performed analysis based on Eclipse, a popular subject project. However, the target of our study is not to evaluate metrics or predict the likelihood of reopening, but to empirically investigate the typical features and underlying reasons for reopened bugs.

The most related research to ours is by Zimmermann et al. [23] who surveyed 358 experienced developers on the primary reasons for reopens, and then assessed the reasons using Windows bug reports. They also built a statistical descriptive model to describe the impact of various factors on bug reopening. One of the key differences between our study and that by Zimmermann et al. [23] is the source we used to summarize the fundamental reasons for bug reopens. They identified causes based on survey results, whereas we made use of developer's comments recorded in Eclipse Bugzilla. As compared to online survey adopted by Zimmermann et al. [23], the data source we choose to use is relatively objective and practical, which is more likely to reflect the real problems during the entire reopen process. As a result, we do obtain some new conclusions in this study from a different perspective. We believe that our findings perfectly complement the previous work rather than challenge it.

8. CONCLUSIONS

Bug fixing is one of the essential activities in the software development and maintenance process. Even with the assistance of advanced methodologies and tools, it is still a challenge to address the issue once for all in the process, where there are closed bugs becoming reopened and this is unavoidable in actual practice. In order to obtain a comprehensive understanding of reopened bugs, we performed an empirical analysis using both quantitative and qualitative methods, the overall result shows:

- 6%-10% of total bugs will be reopened eventually. Over 93% of reopened bugs affect users' normal working experiences. 20% extra workloads are being brought to already-busy developers due to bug reopening (RQ1).
- Reopened bugs will survive 3-6 months on average, which is 1.6-2.1 times higher than the regular ones. Half of which a reopened bug's lifetime is spent in the hidden phase. Every time it gets reopened, it takes approximately a month's time to resolve this issue (RQ2).
- Several main reasons for bug reopening are summarized in our empirical investigation, which supports that a considerable part of reopening could be avoided by adopting appropriate actions such as promoting rig-

orous tests and encouraging effective communication among bug reporters and developers (RQ3).

We believe that these findings are beneficial for both researchers and practitioners to better understand the typical features and underlying reasons of reopened bugs, and take further actions to minimize reopening rate as the ultimate goal.

9. ACKNOWLEDGMENTS

This research is supported by the City University of Hong Kong research funds (Project No. 7200354, 7004222, 7004474).

10. REFERENCES

[1] An, L., Khomh, F., and Adams, B. Supplementary Bug Fixes vs. Re-opened Bugs. In *2014 IEEE 14th International Working Conference on Source Code Analysis and Manipulation* (sep 2014), IEEE, pp. 205–214.

[2] Anvik, J., and Murphy, G. C. Reducing the effort of bug report triage. *ACM Transactions on Software Engineering and Methodology 20*, 3 (2011), 1–35.

[3] Bhattacharya, P., and Neamtiu, I. Bug-fix time prediction models. *Proceeding of the 8th working conference on Mining software repositories - MSR '11* (2011), 207.

[4] Hata, H., Mizuno, O., and Kikuno, T. Bug prediction based on fine-grained module histories. In *2012 34th International Conference on Software Engineering (ICSE)* (jun 2012), IEEE, pp. 200–210.

[5] Hu, H., Zhang, H., Xuan, J., and Sun, W. Effective Bug Triage Based on Historical Bug-Fix Information. In *2014 IEEE 25th International Symposium on Software Reliability Engineering* (nov 2014), IEEE, pp. 122–132.

[6] Ihara, A., Kamei, Y., Monden, A., Ohira, M., Keung, J. W., Ubayashi, N., and Matsumoto, K.-I. An Investigation on Software Bug-Fix Prediction for Open Source Software Projects – A Case Study on the Eclipse Project. *2012 19th Asia-Pacific Software Engineering Conference 2* (2012), 112–119.

[7] Jihun Park, Miryung Kim, Ray, B., and Doo-Hwan Bae. An empirical study of supplementary bug fixes. In *2012 9th IEEE Working Conference on Mining Software Repositories (MSR)* (jun 2012), IEEE, pp. 40–49.

[8] Lamkanfi, A., Perez, J., and Demeyer, S. The Eclipse and Mozilla defect tracking dataset: A genuine dataset for mining bug information. In *2013 10th Working Conference on Mining Software Repositories (MSR)* (may 2013), IEEE, pp. 203–206.

[9] Li, B., Shen, B., Wang, J., Chen, Y., Zhang, T., and Wang, J. A Scenario-Based Approach to Predicting Software Defects Using Compressed C4.5 Model. *2014 IEEE 38th Annual Computer Software and Applications Conference* (2014), 406–415.

[10] Mann, H. B., and Whitney, D. R. On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other. *The Annals of Mathematical Statistics 18*, 1 (mar 1947), 50–60.

[11] Marks, L., Zou, Y., and Hassan, A. E. Studying the fix-time for bugs in large open source projects. *Proceedings of the 7th International Conference on Predictive Models in Software Engineering - Promise '11* (2011), 1–8.

[12] Pan, J., and Mao, X. An Empirical Study on Interaction Factors Influencing Bug Reopenings. In *2014 21st Asia-Pacific Software Engineering Conference* (dec 2014), vol. 2, IEEE, pp. 39–42.

[13] Premraj, R., and Herzig, K. Network Versus Code Metrics to Predict Defects: A Replication Study. *2011 International Symposium on Empirical Software Engineering and Measurement* (2011), 215–224.

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

[15] Shihab, E., Ihara, A., Kamei, Y., Ibrahim, W. M., Ohira, M., Adams, B., Hassan, A. E., and Matsumoto, K.-i. Studying re-opened bugs in open source software. *Empirical Software Engineering 18*, 5 (oct 2013), 1005–1042.

[16] Souza, R., Chavez, C., and Bittencourt, R. a. Do Rapid Releases Affect Bug Reopening? A Case Study of Firefox. *2014 Brazilian Symposium on Software Engineering* (2014), 31–40.

[17] Weiss, C., Premraj, R., Zimmermann, T., and Zeller, A. How Long Will It Take to Fix This Bug? In *Fourth International Workshop on Mining Software Repositories (MSR'07:ICSE Workshops 2007)* (may 2007), no. 2, IEEE, pp. 1–1.

[18] Xin Xia, Lo, D., Xinyu Wang, Xiaohu Yang, Shanping Li, and Jianling Sun. A Comparative Study of Supervised Learning Algorithms for Re-opened Bug Prediction. In *2013 17th European Conference on Software Maintenance and Reengineering* (mar 2013), IEEE, pp. 331–334.

[19] Zeng, H., and Rine, D. Estimation of software defects fix effort using neural networks. *Proceedings - International Computer Software and Applications Conference 2* (2004), 20–21.

[20] Zhang, H., Gong, L., and Versteeg, S. Predicting bug-fixing time: An empirical study of commercial software projects. *2013 35th International Conference on Software Engineering (ICSE)* (2013), 1042–1051.

[21] Zhang, Y., Shen, B., and Chen, Y. Mining Developer Mailing List to Predict Software Defects. In *2014 21st Asia-Pacific Software Engineering Conference* (dec 2014), IEEE, pp. 383–390.

[22] Zhou, J., Zhang, H., and Lo, D. Where Should the Bugs Be Fixed ? *Proceedings of the 34th International Conference on Software Engineering (ICSE'12)* (2012), 14–24.

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

[24] Zimmermann, T., Premraj, R., Bettenburg, N., Just, S., Schroter, A., and Weiss, C. What Makes a Good Bug Report? *IEEE Transactions on Software Engineering 36*, 5 (sep 2010), 618–643.