How Are Discussions Associated with Bug Reworking? An Empirical Study on Open Source Projects

Yu Zhao[1], Feng Zhang[2], Emad Shihab[3], Ying Zou[1] and Ahmed E. Hassan[2]
[1]Department of Electrical and Computer Engineering, Queen's University, Canada
[2]School of Computing, Queen's University, Canada
[3]Department of Computer Science and Software Engineering, Concordia University, Canada
[1]{yu.zhao, ying.zou}@queensu.ca, [2]{feng, ahmed}@cs.queensu.ca, [3]eshihab@cse.concordia.ca

ABSTRACT

Background: Bug fixing is one major activity in software maintenance to solve unexpected errors or crashes of software systems. However, a bug fix can also be incomplete and even introduce new bugs. In such cases, extra effort is needed to rework the bug fix. The reworking requires to inspect the problem again, and perform the code change and verification when necessary. Discussions throughout the bug fixing process are important to clarify the reported problem and reach a solution. *Aims:* In this paper, we explore how discussions during the initial bug fix period (*i.e.*, before the bug reworking occurs) associate with future bug reworking. We focus on two types of "reworked bug fixes": 1) the initial bug fix made in a re-opened bug report; and 2) the initially submitted patch if multiple patches are submitted for a single bug report. *Method:* We perform a case study using five open source projects (*i.e.*, Linux, Firefox, PDE, Ant and HTTP). The discussions are studied from six perspectives (*i.e.*, duration, number of comments, dispersion, frequency, number of developers and experience of developers). Furthermore, we extract topics of discussions using Latent Dirichlet Allocation (LDA). *Results:* We find that the occurrence of bug reworking is associated with various perspectives of discussions. Moreover, discussions on some topics (*e.g.*, code inspection and code testing) can decrease the frequency of bug reworking. *Conclusions:* The discussions during the initial bug fix period may serve as an early indicator of what bug fixes are more likely to be reworked.

Keywords

bug reworking; re-patch; re-open; discussions; Latent Dirichlet Allocation

1. INTRODUCTION

It is estimated that 80% of software development effort is spent on software maintenance [27]. One major activity in software maintenance is to fix bugs. In general, bugs are reported, investigated, fixed and verified. However, a bug fix may partially solve the reported problems, or even introduce new bugs [26]. In such cases, bug fixes need to be reworked. Reworking a bug fix requires re-analysis of the root cause, re-implementation of the bug fix, and re-verification on the changes. As a result, the overall development process can be delayed [24, 26]. We focus on two types of reworked bugs:

1) re-opened bugs, whose bug fix passes the verification, then the bug report is marked as "*Resolved*" initially but later is re-opened due to problems in the bug fix;

2) re-patched bugs that have multiple patches submitted for resolving a single bug report.

Previous studies have shown that a software project can have up to 35% of re-opened bugs [24], and 60% of re-patched bugs [28].

After problems are reported to an issue tracking system (*e.g.*, Bugzilla[1]), the process of bug fixing starts. Shihab et al. [24] and Zimmermann et al. [32] find that the key reasons for bug re-opening include the misinterpretation of bug descriptions and the insufficient information provided in bug reports. Tao et al. [26] report that the problematic implementation and the lack of communication can lead to bug re-patching. Therefore, the discussions among developers may impact the occurrences of bug reworking.

In this paper, we perform an in-depth analysis to enrich the view on the association between discussions and the occurrences of bug reworking. We analyze the discussions occurring only in the initial bug fix period, *i.e.*, from the creation of the bug report to the time when the bug is initially marked as "*Resolved*" (for re-opened bugs) or when the first patch is submitted (for re-patched bugs). As such, all analyzed discussions happened before the bug reworking activity (*e.g.*, bug re-opening and bug re-patching). The discussions are analyzed from the following six perspectives:

- **Duration** captures the time period over which the discussion is performed.
- **Number of comments** measures the size of the discussion in terms of the number of comments.
- **Dispersion** quantifies how the discussion is spread over time in terms of the amount of variation in the interval of comments.
- **Frequency** describes how frequently a comment is posted.
- **Number of Developers** shows how many developers are involved in the discussion.
- **Experience of Developers** represents the average number of bugs fixed by the involved developers.

Furthermore, to understand the topics appearing in discussions, we apply Latent Dirichlet Allocation (LDA) [16] to extract topics of discussions, and identify the topics that are associated with bug reworking.

We perform case studies using five open source projects: Linux kernel[2], Firefox[3], Eclipse Plug-in Development Envi-

[1]http://www.bugzilla.org
[2]https://www.kernel.org
[3]http://www.firefox.com

ronment (PDE)[4], Ant[5], and Apache HTTP Server[6]. These projects are the popular projects of four influential organizations (*i.e.*, Linux, Mozilla, Eclipse and Apache), have varied size (*i.e.*, from 136K to 17.6M lines of code) and domains (*e.g.*, operating system and web server). The diverse development cultures and conventions offer a large variability in the developers' behaviours and discussions. In this study, we examine the following two research questions.

RQ1: How do the initial-fix discussions impact the likelihood of experiencing bug reworking? We find that the initial-fix discussions have a significant association with bug reworking. If the discussions experiencing a shorter duration, a higher frequency, a larger number of comments, a larger number of developers or the less experienced developers, respectively, the initial bug fixes are more likely to be reworked (for re-opened bugs). Regarding re-patched bugs, the initial fixes have a higher chance to be reworked, if the discussions last longer or there are more comments.

RQ2: Do initial-fix discussions raise different topics in the reworked bug fixes as compared to the bug fixes without reworking? We observe that several topics have significantly different distributions between the reworked bugs and bugs without reworking. For instance, during the initial bug fix period, code inspection is generally less discussed in re-opened bugs, and code testing is mentioned less in the discussions of re-patched bugs.

In summary, it is worth for developers examining the discussions (with our studied metrics and topics) when they consider a bug fix is complete and ready for submission. Developers should be more cautious if the discussion during the initial bug fixing period indicates a higher chance to rework the bug fix.

Paper organization. Section 2 shows typical scenarios of bug reworking. Section 3 presents the motivating examples and our metrics of discussions. Case study setup and results are discussed in Sections 4 and 5, respectively. We summarize related work in Section 6. We discuss threats to validity of our work in Section 7, and conclude in Section 8.

2. BACKGROUND

In this section, we describe the process of bug fixing and a typical scenario of bug reworking, as shown in Figure 1.

The common bug fixing process is briefly described as follows: 1) a bug is reported (status: *New*); 2) a developer is assigned to fix the bug (status: *Assigned*); 3) the developer fixes the bug (status: *Resolved*); 4) the bug fix passes verification (status: *Verified*); 5) the bug report is closed (status: *Closed*); and 6) the bug fix is re-opened if problems still exist (status : *Reopen*).

During the bug fixing process, a developer may discuss with others on bug descriptions, solutions to problems or the structure of source code. Once a bug is considered to be resolved, the bug status is changed to *Resolved* in the issue tracking system, with a particular type of resolution, such as CODE_FIX, FIXED, WORKFORME, INVALID, WONTFIX and DUPLICATE. If errors are found in a bug fix after the bug is marked as *Resolved*, the corresponding bug report needs to be re-opened and the status is changed to *Reopen*.

[4]http://www.eclipse.org/pde/
[5]http://ant.apache.org
[6]http://httpd.apache.org

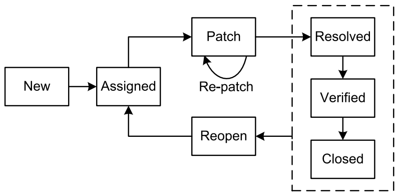

Figure 1: A typical scenario of bug reworking.

Sometimes, developers may submit a patch while resolving a bug, so that other developers can review the code changes and testers can verify the code changes. If the patch has errors, developers need to fix the bug again and resubmit a patch. We refer to such activity as re-patching the bug fix.

3. DISCUSSION METRICS

In this section, we present motivating examples and six metrics of discussions.

3.1 Motivating Examples

To motivate our study, we randomly sample several reworked bug fixes and inspect their comments to understand the reasons for the reworking. For example,

- (#1053434) After a bug is re-opened, a developer from the Firefox project says: *"Re-reading all the comments a third time."* This example shows that the discussion in the bug report is ambiguous for the developer. The ambiguity might be a reason for bug re-opening.
- (#114161) Developers from the PDE project discuss with each other and get a solution to the problem. However, after the close of the bug, a developer says: *"Reopening. Unfortunately, the current implementation isn't quite right"*. Then the developer explains the correct way to solve the bug. This example indicates that the earlier discussed solution is incorrect, which is another possible reason for bug re-opening.
- (#222945) A developer from the PDE project submits the initial patch, and later realizes that: *"the steps in comment #11 did not work for me"*. The developer describes the failure reason that he *"can't install things while self hosting"* and submits another patch. In this case, the wrong information provided by another developer leads to the re-patching.
- (#9487) A developer from the Linux project realizes that a previous submitted patch is incomplete. The developer creates a new patch later and says *"This patch provides a more complete fix to the problem."* The incompleteness of a patch can cause re-patching.

Inspired by the aforementioned examples and the earlier work (e.g., [24, 32]), we conjecture that the actual initial-fix discussions may play an important role in avoiding future bug reworkings. However, it is unclear how such discussions influence the likelihood of a bug being reworked. Therefore, we perform an in-depth analysis on how initial-fix discussions are associated with the occurrences of future bug reworking.

3.2 Discussion Metrics

To measure various metrics of initial-fix discussions, we apply the Goal/Question/Metric (QGM) measurement process [4]. As shown in Table 1, we first define the goal of our empirical study and then propose six questions towards our goal. Each question aims to capture a unique perspective of

Table 1: Goal, questions, and metrics (GQM).

Goal: Study the association between initial-fix discussions and the occurrences of future bug reworking (re-opening and re-patching).

Questions	Metrics
Q1 - How long do developers discuss during an initial bug fix?	Duration
Q2 - How many interactions between developers occur during an initial bug fix?	#Comments
Q3 - How are the posted comments distributed during an initial bug fix?	Dispersion
Q4 - How often do developers post comments during an initial bug fix?	Frequency
Q5 - How many developers are involved in the discussion during an initial bug fix?	#Developers
Q6 - How much experience of fixing bugs do involved developers have?	Experience

the initial bug-fix discussions with a metric. The detailed description of each metric is described as follows.

1) Duration. The duration of discussions is defined as the period of the initial bug fix. The initial bug fix period starts from the creation of the bug report until the time when the bug is initially fixed (*i.e.*, before the bug reworking).

2) Number of comments. The number of comments represents the number of comments posted during the initial bug fix period. A larger number of comments indicate that developers discuss more during bug fixing.

Shihab *et al.* [24] show that the duration and the number of comments are two important predictors of bug re-opening likelihood. We are interested to study if the duration and the number of comments are associated with bug reworking.

3) Dispersion. The dispersion denotes the amount of variation of the interval of the comments posted during the initial bug fix period. A dispersion close to 0 means the intervals between each two consecutive comments are close to each other, and a large dispersion value indicates the discrepancy among the intervals is high. There can be a large dispersion among comments for a bug report. For instance, the resolution of the bug #5948 (never re-opened) from the Linux project takes around 1 year and 6 months. There are in total 23 comments during the initial-fix discusison. The first 12 comments occur within 11 days. However, the next comment comes around 1 year later. The calculated dispersion value for the bug #5948 is 80.1 days. When fixing the re-opened bug #8791 in the Linux project, comments are equally distributed across 2 days before the initial fix. The dispersion value for the bug #8791 is 0.2 days.

4) Frequency. The frequency represents the average frequency for posting the comments. We sum the posting frequency of each comment and compute the average. In a re-opened bug report of the Apache HTTP project (#21523), a developer says that: *"Looks like I spoke too soon. I'm attaching the complete log file of my most recent loop test"* . This example indicates that the high frequency of discussions may negatively influence the quality of bug fixes.

5) Number of Developers. We count the commenter in a bug report as a developer. A higher number of developers indicate that more developers are involved in the discussion.

6) Experience. The experience of a developer is defined as the number of bug reports that the developer is involved in the history of the project. A higher experience value possibly suggests that the developer fixed more bugs. A bug fix made by more experienced developers may have a lower chance to get reworked. For each initial bug fix, we calculate the average experience of involved developers.

7) Other Metrics. In addition to the aforementioned six

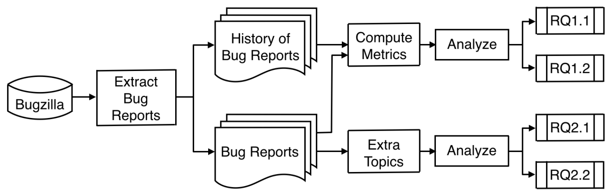

Figure 2: Overview of our approach

metrics, we are also interested in the interval of discussions and the extent of inequality of the distribution of comments. The interval denotes the average interval of two consecutive comments during the period of the initial bug fix. A lower interval reveals that the initial bug fix is discussed more intensively. We use the Gini coefficient [1] to measure the inequality. A Gini coefficient value of 0 means that the comments are spread equally over the discussion period. A larger Gini coefficient value indicates the comments are scattered unevenly among the discussion period.

Correlation Analysis. We compute Spearman's rank correlation to determine which discussion metrics are highly correlated with each other. A correlation higher than 0.8 (*i.e.*, $|\rho| \geq 0.8$) is considered as a high correlation [25]. If the correlation value of two metrics is higher than 0.8, only one metric of discussions is chosen. The interval and the Gini coefficient are highly correlated with the duration and the number of comments, respectively, in both studies for re-opening and re-patching. We choose to examine the duration and the number of comments, since they are much simpler and straightforward metrics.

4. CASE STUDY SETUP

In this section, we show our case study setup, such as the subject projects, the extraction of reworked bug fixes, the computation of our discussion metrics and the LDA topics.

4.1 Overview of Our Approach

Figure 2 presents an overview of our approach. First, we collect bug reports and their change history from the issue tracking system (*i.e.*, Bugzilla). Second, we identify re-opened bugs using the change history of bug reports, and locate re-patched bugs based on the patches recorded in bug reports. Then, we compute our metrics from comments of bug reports to characterize the different perspectives of bug discussions. To understand the content of discussions, we further extract topics of discussions and examine if bug reworking is associated with particular topics. To extract topics, we apply the Latent Dirichlet Allocation (LDA) algorithm [16]. The discussion metrics and topics are extracted only from the discussions that happen during the initial bug fix period (*i.e.*, before the bug reworking). In the following subsections, we describe the details of each step.

4.2 Subject Projects

We select five open source projects of varying sizes, *i.e.*, Linux kernel, Mozilla Firefox, Eclipse PDE, Ant and Apache HTTP server. The five subject projects are from various application domains: 1) Linux kernel is an operating system (OS) kernel that handles interactions between hardware and software; 2) Mozilla Firefox is a popular cross-platform web browser; 3) PDE is an Eclipse plug-in development environment; 4) Ant is a build system for applications; and 5) Apache HTTP server is a widely adopted web server. The five projects are developed in different programming languages (*i.e.*, Java and C/C++).

Table 2: The descriptive statistics on the number of bug reports, developers and comments in our subject projects.

Project	LOC	# *Fixed*	# Re-opened	# Re-patched	# Dev.	# Comments
Linux	17.6M	8,410	382	848	7,411	86.1K
Firefox	7.0M	27,759	1,844	6,890	10,537	464.1K
PDE	472K	8,510	524	417	1,284	46.4K
Ant	136K	2,704	137	38	2,073	10.5K
HTTP	510K	2,333	177	76	2,498	12.6K

Table 2 describes the statistics of the five subject projects. The lines of code written in the main programming languages are calculated on the code snapshot taken on September 1, 2015. The third column reports the number of *fixed* bug reports (*i.e.*, resolution type as FIXED or CODE_FIX). We count the number of developers for a project by recording the total number of commenters. In our study, the first comment is excluded, since it is the description of a bug report in Bugzilla. The number of comments for the *fixed* bug reports is shown in the last column.

4.3 Extracting Reworked Bugs

All the five subject projects use Bugzilla as their issue tracking system. For each project, we download bug reports and their change history from Bugzilla. We select all *fixed* bug reports to the date of February 23, 2016. The *fixed* bug reports are marked as $Resolved$, $Verified$ or $Closed$ with resolution type as FIXED or CODE_FIX [21]. Moreover, we filter out bug reports with less than two comments, since any meaningful discussion can only be formed with at least two comments. In total, we filter out 14.5% of $fixed$ bug reports (7,231 out of 49,716). Our discussion metrics and topics are computed from the initial bug fix period, as shown in Figure 3. The detailed steps to identify the period are described as follows.

1) We extract the time when a bug report is created (T_{opened}). T_{opened} is always the start date of the initial fix. We choose T_{opened} as the start date because developers may discuss anytime after the creation of bug reports.

2) We extract the time when a bug is initially resolved ($T_{initialResolve}$). The end date of the initial fix (*i.e.*, $T_{initialFix}$) is $T_{initialResolve}$ in the study of bug re-opening. $T_{reworked}$ is the time when the bug report is re-opened when studying re-opened bugs.

3) We extract the time when the patch is initially submitted ($T_{firstPatch}$). $T_{initialFix}$ is $T_{firstPatch}$ in the study of bug re-patching. $T_{reworked}$ is the time when a patch is resubmitted when studying re-patched bugs.

4) We identify re-opened and re-patched bug fixes. Table 2 shows the statistics of re-opened bug reports and re-patched bug reports. The detailed steps are described as follows:

Re-opened bugs. For each bug report, we extract all statuses appearing from the creation time to the final resolution time. We consider a bug report as a re-opened bug report, if one of its historical statuses is *Reopen*. We use $T_{initialResolve}$ to represent the time when the status is initially changed to *Resolved* or *Closed*. The interval from the time when a bug report is opened (T_{opened}) to the time when the bug is initially resolved ($T_{initialResolve}$) is the period of the initial bug fixing in the study of bug re-opening. We further mine the comments and their timestamps from bug reports before the initial resolution.

Re-patched bugs. To characterize different perspectives of discussions, we extract the comments and their times-

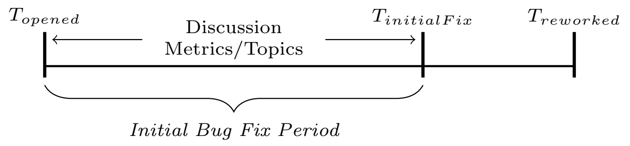

Figure 3: Illustration on how to determine the initial bug fix period for computing discussion metrics and topics.

tamps before the initial patch is submitted. We count the total number of patches for each bug report to determine if the bug fixing is re-patched or not. Patches containing only test cases are filtered out, since test cases are not for bug fixes but are used for verifying bug fixes. If the patch name matches "test", or names of all files match "test", we remove it from our case study. To ensure that all the studied bug reports are patched at least once, we filter out bug reports that do not have patches.

4.4 Computing Discussion Metrics

We compute six metrics to measure the varied perspectives of discussions for the study of bug re-opening and re-patching, separately. For each bug report, we mine the comments and compute the intervals between two consecutive comments during the initial bug fix period. We detail our computation method for each metric as follows.

1) Duration. The duration is defined as $T_{initialResolve} - T_{opened}$ when studying re-opened bugs. When studying re-patched bugs, the duration is defined as $T_{firstPatch} - T_{opened}$.

2) Number of comments. We simply count the number of comments during the period of the initial bug fix.

3) Dispersion. We extract the timestamp of each comment and compute the interval of any two consecutive comments. T_i is used to represent the timestamp of the ith comment. Then the interval between the ith comment and the $(i+1)th$ comment is defined as $I_i = T_{i+1} - T_i$ $(i \geq 1)$. We calculate the standard deviation of I_i to measure the dispersion of discussions.

4) Frequency. For each comment, the frequency F_i is computed as the inverse of the interval I_i (*i.e.*, $F_i = 1/I_i$). Then the average frequency is computed as $\frac{\sum_{i=1}^{m} F_i}{m}$, where m is the total number of intervals.

5) Number of developers. The number of developers is defined as the total number of unique commenters involved in the discussion before the bug is initially fixed.

6) Experience. We define E_i as the total number of bug reports that a developer involved throughout the entire development of the project. The developer's experience for a particular bug report is $\frac{\sum_{i=1}^{n} E_i}{n}$, where n is the number of the involved developers during the initial bug fix period.

4.5 Extracting Discussion Topics

To understand what developers discuss during the initial bug fix period, we extract topics from the initial-fix discussions using Latent Dirichlet Allocation (LDA) [16]. LDA is a popular generative topic model that can discover relationships between words and unstructured documents. It assumes that documents consist of different probabilistic combinations of topics. In LDA, each textual document is modelled as a distribution of latent topics, and each topic is considered as a probabilistic distribution over words.

For each type of bug reworking (*i.e.*, re-opening and re-patching), we formulate the corpus for LDA using the comments posted during the initial fix, respectively. We treat the comments of a bug report during the initial fix period as a document for LDA. Before applying LDA, we normalize words in documents. Non-english characters in a word such as punctuation and numbers are removed. We use an English dictionary to remove non-English words. Moreover, we stem words (*e.g.*, “fixes” to “fix”) and remove all stop words (*e.g.*, “a” and “the”) [3]. We set the same configurations of hyper-parameters (*i.e.*, $\alpha = 0.1$, β=0.1) and the same number of iterations (*i.e.*, 1,000) as the work by Hindle et al. [10] who apply LDA on SE data. LDA generates a specified number of topics for each document, and assigns a score of each topic in each document. The sum of the scores is 1. As we only consider the major topics in each bug report, we choose the topics with the scores higher than the average (*i.e.*, $\frac{1}{\text{number of topics}}$) as the topics of the bug report.

5. CASE STUDY RESULTS

In this section, we present our approach to investigate the association between discussions and the occurrences of bug reworking, and discuss our findings.

5.1 Discussion Metrics

Bug reworking requires additional human effort, thus is expensive. To help developers locate the bug fixes that are likely to reworked, we aim to understand how the initial-fix discussions impact bug reworking. Previous studies find that insufficient discussions can increase the likelihood of bug reopening [32] or re-patching [26]. However, discussions have only been investigated from a very limited perspective.

In this paper, we study the association between bug reworking and discussions along with six metrics (*i.e.*, duration, number of comments, dispersion, frequency, number of developers and developer experience). To investigate the two types of bug reworking, we study the two subquestions:

RQ1.1 How do the initial-fix discussions impact the likelihood of experiencing bug re-opening?

RQ1.2 How do the initial-fix discussions impact the likelihood of experiencing bug re-patching?

5.1.1 Approach

To address each question, we separate bug reports into two groups along a single metric. For each metric, 1) one group is the control group that contains bug reports with smaller values (*i.e.*, less than a threshold) of the corresponding measurement; and 2) the other group is the experimental group that contains the remaining bug reports. We choose the median value of each metric of discussions as the threshold.

To answer RQ1.1 and RQ1.2, we test the following null hypotheses for each discussion metric, respectively:

$H0_{11}$*: the proportion of re-opened bug fixes in the experimental group and the control group has no difference.*

$H0_{12}$*: the proportion of re-patched bug fixes in the experimental group and the control group has no difference.*

Hypothesis $H0_{11}$ and $H0_{12}$ are evaluated using the Fisher's exact test [23] with 95% confidence level (*i.e.*, p-value < 0.05). The Fisher's exact test examines if there exists non-random association between the occurrences of bug reworking and the measurement of each discussion metric. Since we investigate five projects for each metric of discussions, we apply Bonferroni correction to adjust the p-value by dividing the number of tests (*i.e.*, 5 tests). If there is statistical significance (*i.e.*, p value is less than 0.05/5=0.01), we reject the null hypothesis.

We further compute the odds ratio [23] that measures the likelihood of experiencing the reworking of bug fixes in the experimental group. We calculate the odds p and q of reworked bug fixes in the experimental and control groups, respectively. Then the odds ratio is computed as $OR = \frac{p/(1-p)}{q/(1-q)}$. If $OR = 1$, reworked bug fixes and non-reworked bug fixes are equally distributed in the two groups. Otherwise, the chances to experience reworked bug fixes are different in the two groups. If $OR > 1$, bug fixes belonging to the experimental group are more likely to be reworked, and vice versa.

5.1.2 Findings

The association between the discussions and the bug reworking is significant in various projects. In the following paragraphs, we report the detailed findings in the bug reopening and re-patching studies.

RQ1.1 (Bug re-opening). Discussions with a shorter duration, more comments, a higher frequency, more developers or less experienced developers increase the chance of re-opening resolved bugs, respectively. Table 3 presents the thresholds of the measurement and the detailed odds ratios for each metric in the study of bug reopening. We describe the association between each metric of discussions with bug re-opening as follows.

1) Duration. Shorter duration of discussions before the initial fix is more likely to increase the likelihood of bug re-opening overall (except for the Firefox project). The p-values of the Fisher's exact test in all the five projects are lower than the corrected threshold p-value (*i.e.*, 0.01). In such cases, we reject the null hypothesis $H0_{11}$. The rejection shows that re-opened bug fixes have significant different distributions in bug reports with short period of initial-fix discussions compared to the remaining bug reports. In addition, the odds ratios of the four projects (*i.e.*, Linux, PDE, Ant and HTTP) are all less than one, indicating that the longer duration of the initial-fix discussions reduces the likelihood to experience bug re-opening. However, the Firefox project shows a different trend with an odds ratio larger than one; and this is discussed in Section 5.1.3.

2) Number of comments. Bug reports with a larger number of comments are more likely to be re-opened. We observe a significant association between number of comments and bug re-opening in the Linux, Firefox and PDE projects. In particular, discussions with a larger number of comments before the initial fix increase the likelihood of experiencing bug re-opening by 2.49, 1.35 and 1.31 times for the Linux, Firefox and PDE projects, respectively.

3) Dispersion. The dispersion overall does not show a significant association with bug re-opening. We only observe the significant p-value in the Firefox project. Thus regarding dispersion, we can reject the null hypothesis $H0_{11}$ only for the Firefox project.

4) Frequency. Bug fixes with a higher frequency of comment postings tend to increase the likelihood of re-opening. Among all five projects, the frequency has a significant association in four projects (*i.e.*, except for the PDE project). Bug reports with discussions of high frequency are 2.16, 1.24, 2.33 and 1.69 times more likely to be re-opened in the Linux, Firefox, Ant and HTTP projects, respectively.

5) Number of developers. The number of developers has a significant association with bug re-opening in the Linux, Firefox and PDE projects. The odds ratios of the

Table 3: The result of Fisher's exact test in the study of bug re-opening. (τ denotes the threshold value of the measurement. OR is the odds ratio and "-" represents not statistically significant, the same below)

Project		Linux	Firefox	PDE	Ant	HTTP
Duration	τ	48.1 days	16.2 days	6.9 days	34.6 days	149.6 days
	OR	**0.69**	1.15	**0.78**	**0.60**	**0.60**
# Comments	τ	6	8	4	3	4
	OR	**2.49**	**1.35**	**1.31**	-	-
Dispersion	τ	3.8 days	1.5 days	0.3 days	0.7 days	8.1 days
	OR	-	1.15	-	-	-
Frequency	τ	5.6e-04	8.9e-04	8.5e-04	9.9e-05	7.4e-05
	OR	**2.16**	**1.24**	-	**2.33**	**1.69**
# Developer	τ	3	4	2	2	2
	OR	**2.03**	**1.41**	**1.33**	-	-
Experience	τ	129.5	1222.3	891.0	316.0	130.0
	OR	1.32	**0.48**	**0.77**	**0.52**	-

three projects are larger than one, showing that larger number of developers involved in the initial-fix discussion increase the likelihood of bug re-opening generally.

6) Experience. A bug report fixed with less experienced developers is more likely to be re-opened overall. Four projects (except for the HTTP project) exhibit a significant association between the developer experience and the bug reworking. Moreover, in the Firefox, PDE and Ant projects, the odds ratios are all lower than one, indicating that bug fixes with more experienced developers have lower chances to be re-opened. However, different from the three projects, we observe that in the Linux project, the odds ratio is larger than one and the threshold is 129.5. The threshold value means that on average, developers in 50% of bug reports are involved in less than 2% of fixed bug reports. Such a low number (*i.e.*, 2% of fixed bug reports) reflects the characteristics of the Linux project that developers may have expertise in the specific subsystem, such as file systems and networking. The number (*i.e.*, 2%) is not sufficient to represent the experience. When setting the threshold value as 841 (*i.e.*, 10% of fixed bug reports) for the Linux project, the odds ratio is consistent as other projects (*i.e.*, lower than one).

RQ1.2 (Bug re-patching). Discussions with a longer duration, more comments, a larger dispersion and more developers are more likely to get the initial bug fix re-patched. Table 4 shows the Fisher's exact test result and the thresholds for the measurements of each metric in the study of bug re-patching. Similar to bug re-opening, a bug fix with more comments and more developers before submitting an initial patch has higher chances of re-patching. To be more specific, three projects (*i.e.*, Linux, Firefox and PDE) show a significant association (*i.e.*, p-value < 0.01) between the number of comments and bug re-patching. Discussions with more comments are 1.60, 1.31 and 1.81 times more likely to get the initial fix re-patched in the Linux, Firefox and PDE projects. We observe that the metric of the number of developers has a significant association with bug re-patching in the Linux and Firefox projects. Moreover, both of the odds ratios are larger than one, indicating that a larger number of developers involved in the discussion increase the likelihood of re-patching.

Different from bug re-opening, the longer duration of discussions before submitting an initial patch has a higher chance

Table 4: The result of Fisher's exact test in the study of bug re-patching.

Project		Linux	Firefox	PDE	Ant	HTTP
Duration	τ	9.7 days	11.4 days	11.1 days	18.2 days	72.0 days
	OR	-	**1.75**	**1.76**	-	**2.21**
# Comments	τ	7	5	4	3	4
	OR	**1.60**	**1.31**	**1.81**	-	-
Dispersion	τ	0.8 days	1.0 days	0.30 days	0.2 days	2.0 days
	OR	-	**1.55**	**2.02**	-	-
Frequency	τ	1.3e-03	5.4e-04	7.2e-04	2.2e-04	2.6e-04
	OR	1.59	0.81	-	-	-
# Developer	τ	3	3	2	2	3
	OR	**1.51**	**1.24**	-	-	-
Experience	τ	87.5	696.0	1007.5	158.5	103.3
	OR	-	0.73	-	-	-

to re-patch a bug fix. The duration of the initial-fix discussions in the Firefox, PDE and HTTP projects has a significant association with bug re-patching. In particular, a longer duration of discussions increases the likelihood to re-patch a bug fix by 1.75, 1.76, 2.21 times in the Firefox, PDE and HTTP projects, respectively. The dispersion has a significant association with bug re-patching in two projects (*i.e.*, Firefox and PDE). The odds ratios of the two projects are larger than one. In particular, discussions with a larger dispersion are 1.55 and 2.02 times more likely to get the initial bug fix re-patched in the Firefox and PDE projects, respectively.

Discussions performed before the initial fix have a significant association with bug reworking from various perspectives. Indeed, initial-fix discussions with fewer comments, fewer developers are less likely to experience bug re-opening and re-patching.

5.1.3 Discussions

To better understand the possible causes of bug reworking (*i.e.*, bug re-opening and bug re-patching), we randomly sampled 341 reworked bug reports with 95% confidence level and 5% confidence interval[7]. The first author manually analyzes the sampled bug reports, and the detailed findings are presented as follows.

1) Duration. A shorter duration of discussions has two major reasons for re-opening bugs: a) Defective fixes. The approaches to fix the bug may not be fully discussed so that there are still errors which remain, such as Linux #8791, #9475 and HTTP #21371; and b) Incorrect categorization of duplicated bugs. From the bug description, developers may think of it as a duplicate bug and mark the status as resolved. Further analysis reveals that such reports are misclassified, thus the bug reports are re-opened. Example cases are Linux #8709, PDE #463822 and HTTP #13991.

We have an opposite observation in the Firefox project that a longer duration of discussions increases the chance to get bugs re-opened. Firefox is a large project designed for a web browser. Developers may have a long discussion about complex issues, such as customization settings (no unique solution can be easily reached due to varying preferences

[7]http://www.surveysystem.com/sscalc.htm

of different users) and URL autocomplete (implementation of complex machine learning algorithms may be involved). When resolving a bug, the fix can conflict with the existing code or incur unexpected issues, such as Firefox #179666 and #400061.

A longer duration of discussions before submitting an initial patch has a higher chance to get the initial patch rejected. There can be multiple alternative solutions to fix a bug, thus developers need spend efforts to find the best one. However, our manual analysis reveals that a patch creator may be unaware of the correct way to fix the bug when creating a patch. As a result, incomplete or unnecessary changes can occur. Example cases can be found in the bug reports PDE #434303, #462288 and HTTP #44736.

2) Number of comments and developers. More comments and more developers involved in discussions before resolving a bug are more likely to get the bug fix re-opened. One possible reason is that developers do not reach a consensus about the solution. After closing the bug, a better approach is found or the discussed solution needs to be improved. Example cases are bug reports Linux #5832, Firefox #396816 and PDE #115484.

3) Frequency. The higher frequency of discussions is associated with bug re-opening. We find that the high average frequency of discussions can be classified into two types, *i.e.*, a) one or more burst of comment postings; and b) high density of comment postings throughout the initial bug fixing period. Example bug reports with a burst of comment postings are Linux #2884, Ant #5907 and HTTP #16137. We conjecture that the burst of comment postings may address a particular problem but does not necessarily result in a thorough solution. Example bug reports with comment postings of high density are Linux #5889, Firefox #1078539 and Ant #32300. One possible reason of reworking these bugs is that their comment postings mainly focus on finding a solution other than code review and testing.

5.2 Discussion Topics

Manually inspecting topics of bug reports can help us understand specific reasons behind the reworking of bug fixes. However, there are in total 3,064 re-opened bugs in our subject projects (see Table 2). Even for one project, it is tedious to manually inspect all bug reports. Hence, it is necessary to automatically abstract detailed comments to a higher level so that the content of discussions is more understandable. In this section, we aim to investigate if it is possible to identify specific topics that are associated with bug reworking.

To study the association between discussion topics and the two types of bug reworking, we propose the following two questions:

RQ2.1 Do initial-fix discussions raise different topics in the re-opened bug fixes as compared to the bug fixes without re-opening?

RQ2.2 Do initial-fix discussions raise different topics in the re-patched bug fixes as compared to the bug fixes without re-patching?

5.2.1 Approach

To address the two research questions, we first extract topics of the initial-fix discussions from bug reports and then investigate the association between each individual topic with bug reworking.

In our corpus, there are in total 42,203 and 14,273 documents for the bug re-opening and re-patching studies, respectively. We set the number of topics for LDA to be 50, and the number of keywords for each topic to be 10. To better understand each topic, we manually assign labels to each topic based on the 10 keywords produced by LDA. Hindle *et al.* [10] report that the topics generalized from LDA match the perception of developers and managers. To verify if the topics extracted by LDA are representative, the first author conducted a qualitative study to manually extract topics from the previously sampled bug reports without any knowledge of the LDA results.

To investigate the association between specific topics and the occurrences of bug reworking, we separate bug reports into two groups. One group contains bug reports that are never re-opened (respectively never re-patched), and the other group contains bug reports that experiences re-opening (respectively re-patching). For each topic, we investigate if the distribution of the topic across the two groups is similar by testing the following null hypothesis:
$H0_2$*: There is no difference in the distribution of topics between bugs that are never reworked and ones that are reworked.*

We perform Wilcoxon rank-sum test [23] to examine the null hypothesis $H0_2$, using the 95% confidence level (*i.e.*, p-value < 0.05). Wilcoxon rank-sum test is a non-parametric statistically test to assess whether the population of two sampled groups are the same against the null hypothesis. The Wilcoxon test makes no assumption about the distribution of samples. Since we perform five tests on each topic, we apply the Bonferroni correction to control for family-wise errors. We reject the null hypothesis if there is a statistically significant difference (*i.e.*, p-value < 0.01).

5.2.2 Findings

Some topics of discussions experience statistically significantly different distributions between reworked and not reworked bug reports. For instance, the topic A4 (*i.e.*, *code inspection*) occurs less frequently in re-opened bugs than the remaining bug reports in two projects (*i.e.*, Linux and HTTP). The topic B4 (*i.e.*, *code testing*) has lower percentage in re-patched bugs than the remaining bug reports in two projects (*i.e.*, Linux and Firefox).

Furthermore, our manual verification shows that the topics produced by LDA with our configuration (*e.g.*, 50 topics) are similar to the manually extracted topics.

RQ2.1 (Re-opened bugs) Seven topics exhibit statistically significant different distributions (*i.e.*, p-value < 0.01) in at least two projects. We reject the null hypothesis $H0_2$ for these topics in the corresponding projects. Table 5 summarizes the topics that have significantly different distributions in at least two projects in the study of re-opened bugs. Table 5 (a) presents the percentage of bug reports that contain the topic in each of the two groups (*i.e.*, bug reports with and without re-opening). Table 5 (b) shows the keywords of the significant topics. Overall, no topics are consistently associated with the chance of bug re-opening in all five projects. However, in particular projects, some topics do have an association with the likelihood of bug re-opening.

Among the 7 topics, we have consistent observations. The topics A1 (*i.e.*, *commit verification*), A4 (*i.e.*, *code inspection*) and A5 (*i.e.*, *patch creation*) tend to appear more often

Table 5: Topics which have significant p-values in at least two projects in our study of bug re-opening

(a) The percentage of the topics in bug reports that are re-opened (Yes) and are never re-opened (No).

Topics	**Linux**		**Firefox**		**PDE**		**Ant**		**HTTP**	
	Yes	No	Yes	No	Yes	No	Yes	No	Yes	No
A1	53%	**60%**	-	-	-	-	-	-	32%	**41%**
A2	-	-	**33%**	27%	-	-	**51%**	40%	-	-
A3	-	-	**29%**	25%	-	-	-	-	**35%**	24%
A4	12%	**20%**	-	-	-	-	-	-	31%	**43%**
A5	-	-	38%	**46%**	29%	**38%**	-	-	37%	**48%**
A6	**28%**	19%	-	-	**34%**	27%	-	-	**35%**	24%
A7	-	-	-	-	**26%**	19%	-	-	**31%**	20%

(b) Keywords of the topics

Topics	**Keywords**
A1	commit,fix,write,report,date,regress,bad,tree,bisect,git
A2	build,version,install,update,release,extension,mac,os,night,run
A3	page, link, content, site, show, user, secure, bug, click, document
A4	check,fix,revise,attach,previous,patch,create,trunk,branch,comment
A5	patch,attach,create,apply,propose,fix,please,top,submit,include
A6	set,default,opinion,change,prefer,add,support,custom,override,ad
A7	size,width,screen,height,drag,position,font,move,animation,space

A1: commit verification, A2: build system, A3: web page design, A4: code inspection, A5: patch creation, A6: customization settings, A7: UI related.

Table 6: Topics which have significant p-values in at least two projects in our study of bug re-patching

(a) The percentage of the topics in bug reports that are re-patched (Yes) and are never re-patched (No).

Topics	**Linux**		**Firefox**		**PDE**		**Ant**		**HTTP**	
	Yes	No	Yes	No	Yes	No	Yes	No	Yes	No
B1	**21%**	15%	**40%**	36%	-	-	-	-	-	-
B2	-	-	**19%**	15%	-	-	-	-	**93%**	79%
B3	-	-	**37%**	29%	**39%**	31%	-	-	-	-
B4	9%	**14%**	9%	**11%**	-	-	-	-	-	-
B5	-	-	**38%**	28%	**44%**	32%	-	-	-	-

(b) Keywords of the topics

Topics	**Keywords**
B1	comment,reply,guess,understand,mention,case,does,maybe,why,affect
B2	request,connect,server,cache,send,client,sync,network,response,header
B3	work,code,bug,start,assign,fine,make,interest,broken,made
B4	test,slave,expect,warn,revise,debug,option,code,uncaught,return
B5	implement,code,make,service,provide,require,support,case,solution,part

B1: content misunderstanding, B2: client-server model, B3: bug assignment, B4: code testing, B5: solution implementation.

in bug reports without re-opening in at least two projects. The other four topics consistently appear more frequently in re-opened bug reports than bug reports without re-opening. The topic A5 (*i.e.*, *patch creation*) has a higher discussion rate in the bug reports that are never re-opened in the Firefox, PDE and HTTP projects. It shows that fixing bugs by revising codes can have a higher chance to get bugs not re-opened. When talking about the topic A6 (*i.e.*, *customization settings*), there are no universal solutions to the problem. It is possible that developers find a better approach to satisfy user's preferences after resolving a bug. Another interesting topic is A4 (*i.e.*, *code inspection*). The Linux and HTTP projects have a significant lower percentage on discussing code inspection in re-opened bugs. Code inspection can improve code readability and discover software defects. Although its nature (*i.e.*, time-consuming and complex) restrains its wide adoption in practice [2], developers should increase the discussion of code review. Otherwise, as Shihab et al. [24] show, the chance of bug re-opening increases, and possibly leads to lengthening the bug fixing time.

RQ2.2 (Re-patched bugs) Initial-fix discussions have five topics experiencing significant different distributions in re-patched bug reports and bug reports without re-patching in two projects. Table 6 concludes the topics with a significant p-value (*i.e.*, p-value < 0.01) for the Wilcoxon rank sum test in at least two projects in our study of bug re-patching. We do not observe topics with consistent associations with bug re-patching in the five projects. However, in the Firefox project, all the five topics in Table 6 have significant different distributions in re-patched bug reports and bug reports without re-patching.

To be more specific, the topic B4 (*i.e.*, *code testing*) appears more often in the bug reports without re-patching compared to re-patched bug reports in the Linux and Firefox projects. All the other four topics have a higher percentage in the re-patched bug reports. The Linux and Firefox projects are long-lived (more than 10 years development history) and are developed by a large community of developers. The topics B1 (*i.e.*, *content misunderstanding*) and B4 (*i.e.*, *code testing*) have significant different distributions in the two projects. We looked into our sampled re-patched bug reports with the topic B1 (*i.e.*, *content misunderstanding*) and investigated possible reasons for re-patching. We find that developers are uncertain about the solutions or have misunderstandings before submitting a patch in our sampled bug reports. Examples are Firefox #815847, #1123309 and #950399. Code testing is an essential step to validate the code and expose the defects [9]. Before submitting a patch, a developer is encouraged to conduct testing. Otherwise, the patch could be rejected and requires reworking. As a result, it increases the workload of patch creators and reviewers [26].

Among all the 50 extracted topics, there are several topics showing a consistent association with bug re-opening and bug re-patching in at least two projects. In particular, re-opened bugs have less discussions on the topic code inspection. If developers are unclear about the solution and conduct less code testing, the chance of bug re-patching increases.

6. RELATED WORK

In this section, we present the related studies on bug reworking, and discussion analysis.

6.1 Bug Reworking

Re-opened and re-patched bugs are reworked bug fixes. Shihab et al. [24] are the first to examine re-opened bugs. They built models to predict re-opened bugs using four groups of metrics *i.e.*, work habits, bug report, bug fix and team, and find that comment text is the most important indicator of re-opened bugs in the Eclipse and OpenOffice projects.

Zimmermann et al. [32] investigate the reasons for bug re-opening and find that bugs identified by code analysis tools or code review processes are less likely to be re-opened. Caglayan et al. [6] report that developers' activities are important factors that cause bugs to be re-opened.

Tao et al. [26] systematically investigate the reasons for rejecting patches and find that problematic implementation and the lack of communication are two major reasons. Rigby and Storey [22] report that the reasons for re-patching not only include technical issues but also contain feature, scope and political issues. Nurolahzade et al. [19] find that ambiguous information in the discussions during code review may affect the newly submitted patches. Other studies about patches are related to the acceptance of patches and the prediction of review time using features that are extracted from patches and bug reports [12] [13].

Our study complements the previous studies since we focus on the association between the discussions and the bug reworking. We study two types of bug reworking, *i.e.*, bug re-opening and bug re-patching. The association is studied in terms of six unique perspectives and discussion topics.

6.2 Discussion Analysis

The impact of discussions on software quality has been investigated in many prior studies. For example, Ohira et al. [20] conclude that discussions before bug assignment can help prevent bug reassignment and therefore reduce bug fixing time. McIntosh et al. [17] suggest that discussions should be conducted during the code review process to reduce the post-release defects. Rigby and Storey [22] report that unproductive discussions such as the "bike shed" problems (*e.g.*, developers intensively express multiple opinions and response during bug fixing activities) slow down the decision process. Zhang et al. [30] find that discussions impact the bug fixing time. Discussions are also widely used to mine social networks among developers [5, 11, 14].

A number of researches use text mining techniques to understand discussions among developers. For instance, Zhou et al. [31] analyze text and code snippets extracted from API discussions to categorize the contents of discussions. Dit and Marcus [7] mine previous discussions and bug description to build a recommendation system for comments related to bugs. Topic modeling approaches such as Latent Semantic Indexing (LSI) and Latent Dirichlet Allocation (LDA) are also widely used in the discussion analysis [8, 15, 18].

Different from the previous studies, we analyze the impact of discussions on the likelihood of experiencing bug reworking. In addition, we apply LDA to find topics of discussions that are associated with the likelihood of bug reworking.

7. THREATS TO VALIDITY

In this section, we discuss the threats to the validity of our study, following Yin's guidelines for case study research [29].

Threats to conclusion validity are concerned with the relationship between the treatment and the outcome. We divide bug reports into two groups to study the association between each discussion metric and the bug reworking. We choose the median value of each metric of discussions as the threshold to equally separate bug reports into two groups. Although applying other thresholds may result in different values of odds ratio, the direction of odds ratios (*i.e.*, either $OR < 1$ or $OR > 1$) can be still the same.

Threats to internal validity are concerned with our selection of subject projects and analysis methods. In this study, our six metrics of discussions may not capture all aspects of discussions, although they are indeed associated with the likelihood of the occurrences of bug reworking. Exploring more aspects of discussions is encouraged. Our heuristic for filtering patches with only test cases is based on the keyword "test", possibly introducing false positives and false negatives. However, our manual analysis shows that the impact of our heuristic is trivial.

Threats to external validity are concerned with the generalizability of our results. To obtain a more general finding, we choose well-known subject projects in various domains and developed in widely used programming languages (*i.e.*, Java and C/C++). However, some findings might relate to specific projects. Therefore, we believe that each project which wishes to leverage our results should conduct our analysis on their historical data.

Threats to reliability validity are concerned with the possibility of replicating this study. Our subject projects are open source projects, and are publicly accessible. In addition, we attempt to provide all details in this paper.

8. CONCLUSION

Through discussions, developers can clarify bug descriptions and brainstorm solutions. Effective discussions can help developers avoid the reworking of bug fixes. In this paper, we investigate two types of bug reworking (*i.e.*, re-opening and re-patching). We investigate the association between initial-fix discussions and the occurrences of future bug reworking from six perspectives (*i.e.*, the duration, the number of comments, the dispersion, the frequency, the number of developers and the developer experience). In addition, we apply LDA to extract topics of discussions and then examine the association between different topics and the occurrences of bug reworking.

We perform an empirical study using five open source projects (*i.e.*, Linux, Firefox, Eclispe PDE, Ant and HTTP), and find that various metrics of initial-fix discussions have different associations with the occurrence of bug reworking. We summarize our major findings as follows.

- **Re-opened bug fixes.** A shorter duration, more comments and a higher frequency of discussion before the initial fix increase the likelihood of bug re-opening. Bug fixes with a larger number of developers and less experienced ones have a higher likelihood to get re-opened. Different topics of discussions are associated with the likelihood of bug re-opening. For instance, if developers discuss less about code inspection, the chance of a bug report getting re-opened increases.
- **Re-patched bug fixes.** Discussions before submitting the initial patch with a longer duration, more comments, a higher dispersion and more developers are more likely to get bug fix re-patched. Re-patched bug fixes tend to discuss less about code testing before an initial patch is submitted compared to bug reports without re-patching.

As a summary, our observed effect of discussions on bug reworking can provide an early warning to developers about bug reworking. Practitioners can develop tools that monitor the discussions. When developers are about to resolve a bug or submit a patch, the discussion metrics and topics can indicate whether the bug is likely to be reworked. In the future, we plan to investigate the occurrences of bug reworking from more social aspects (*e.g.*, developers' activities).

Acknowledgement

We would like to thank Dr. Iman Keivanloo for his feedback on the early version of this work.

References

[1] Anthony Barnes Atkinson and Francois Bourguignon. *Handbook of income distribution*, volume 1. Elsevier, 2000.

[2] Alberto Bacchelli and Christian Bird. Expectations, outcomes, and challenges of modern code review. In *ICSE*, pages 712–721. IEEE Press, 2013.

[3] Ricardo Baeza-Yates, Berthier Ribeiro-Neto, et al. *Modern information retrieval*, volume 463. ACM press New York, 1999.

[4] Victor R. Basili, Gianluigi Caldiera, and H. Dieter Rombach. The goal question metric approach. In *Encyclopedia of Software Engineering*. Wiley, 1994.

[5] Nicolas Bettenburg and Ahmed E Hassan. Studying the impact of social interactions on software quality. *Empirical Software Engineering*, 18(2):375–431, 2013.

[6] Bora Caglayan, Ayse Tosun Misirli, Andriy Miranskyy, Burak Turhan, and Ayse Bener. Factors reopened issues: a case study. In *Proceedings of the 8th International Conference on Predictive Models in Software Engineering*, pages 1–10. ACM, 2012.

[7] Bogdan Dit and Andrian Marcus. Improving the readability of defect reports. In *Proceedings of the 2008 international workshop on Recommendation systems for software engineering*, pages 47–49. ACM, 2008.

[8] Bogdan Dit, Denys Poshyvanyk, and Andrian Marcus. Measuring the semantic similarity of comments in bug reports. *Proc. of 1st STSM*, 8, 2008.

[9] Brent Hailpern and Padmanabhan Santhanam. Software debugging, testing, and verification. *IBM Systems Journal*, 41 (1):4–12, 2002.

[10] Abram Hindle, Christian Bird, Thomas Zimmermann, and Nachiappan Nagappan. Do topics make sense to managers and developers? *Empirical Software Engineering*, pages 1–37, 2014.

[11] Qiaona Hong, Sunghun Kim, SC Cheung, and Christian Bird. Understanding a developer social network and its evolution. In *Software Maintenance (ICSM), 2011 27th IEEE International Conference on*, pages 323–332. IEEE, 2011.

[12] Gaeul Jeong, Sunghun Kim, Thomas Zimmermann, and Kwangkeun Yi. Improving code review by predicting reviewers and acceptance of patches. *Research on Software Analysis for Error-free Computing Center Tech-Memo (ROSAEC MEMO 2009-006)*, 2009.

[13] Yujuan Jiang, Bram Adams, and Daniel M German. Will my patch make it? and how fast?: case study on the linux kernel. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, pages 101–110. IEEE Press, 2013.

[14] Amit Kumar and Avdhesh Gupta. Evolution of developer social network and its impact on bug fixing process. In *Proceedings of the 6th India Software Engineering Conference*, pages 63–72, 2013.

[15] Erik Linstead and Pierre Baldi. Mining the coherence of gnome bug reports with statistical topic models. In *Mining Software Repositories, 2009. MSR'09. 6th IEEE International Working Conference on*, pages 99–102. IEEE, 2009.

[16] David M.Blei, Andrw Y.Ng, and Michael I.Jordan. Latent dirichlet allocation. *Journal of Machine Learning Research*, 2003.

[17] Shane McIntosh, Yasutaka Kamei, Bram Adams, and Ahmed E Hassan. The impact of code review coverage and code review participation on software quality: A case study of the qt, vtk, and itk projects. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 192–201. ACM, 2014.

[18] Laura Moreno, Wathsala Bandara, Sonia Haiduc, and Andrian Marcus. On the relationship between the vocabulary of bug reports and source code. In *Software Maintenance (ICSM), 2013 29th IEEE International Conference on*, pages 452–455. IEEE, 2013.

[19] Mehrdad Nurolahzade, Seyed Mehdi Nasehi, Shahedul Huq Khandkar, and Shreya Rawal. The role of patch review in software evolution: an analysis of the mozilla firefox. In *Proceedings of the joint international and annual ERCIM workshops on Principles of software evolution (IWPSE) and software evolution (Evol) workshops*, pages 9–18. ACM, 2009.

[20] Masao Ohira, Ahmed E Hassan, Naoya Osawa, and Ken-ichi Matsumoto. The impact of bug management patterns on bug fixing: A case study of eclipse projects. In *Software Maintenance (ICSM), 2012 28th IEEE International Conference on*, pages 264–273, 2012.

[21] Lucas D. Panjer. Predicting eclipse bug lifetimes. In *Proceedings of the Fourth International Workshop on Mining Software Repositories*, MSR '07, pages 29–, Washington, DC, USA, 2007. IEEE Computer Society. ISBN 0-7695-2950-X. .

[22] Peter C Rigby and Margaret-Anne Storey. Understanding broadcast based peer review on open source software projects. In *Proceedings of the 33rd International Conference on Software Engineering*, pages 541–550. ACM, 2011.

[23] David J. Sheskin. *Handbook of Parametric and Nonparametric Statistical Procedures, Fourth Edition*. Chapman & Hall/CRC, January 2007. ISBN 1584888148.

[24] Emad Shihab, Akinori Ihara, Yasutaka Kamei, Walid M.Ibrahim, Masao Ohira, Bram Adams, Ahmed E.Hassan, and Ken ichi Matsumoto. Studying re-opened bugs in open source software. *Empirical Software Engineering*, pages 1–38, 2012.

[25] Giancarlo Succi, Witold Pedrycz, Snezana Djokic, Paolo Zuliani, and Barbara Russo. An empirical exploration of the distributions of the chidamber and kemerer object-oriented metrics suite. *Empirical Software Engineering*, 10(1):81–104, 2005.

[26] Yida Tao, DongGyun Han, and Sunghun Kim. Writing acceptable patches: an empirical study of open source project patches. In *Proceedings of 2014 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 271–280. IEEE, 2014.

[27] G. Tassey. The economic impacts of inadequate infrastructure for software testing. Technical Report Planning Report 02-3, National Institute of Standards and Technology, May 2002.

[28] Peter Weißgerber, Daniel Neu, and Stephan Diehl. Small patches get in! In *Proceedings of the 2008 international working conference on Mining software repositories*, pages 67–76. ACM, 2008.

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

[30] Feng Zhang, Foutse Khomh, Ying Zou, and Ahmed E. Hassan. An empirical study on factors impacting bug fixing time. In *Proceedings of the 19th Working Conference on Reverse Engineering*, WCRE '12, pages 225 – 234, oct. 2012.

[31] Bo Zhou, Xin Xia, David Lo, Cong Tian, and Xinyu Wang. Towards more accurate content categorization of api discussions. In *Proceedings of the 22nd International Conference on Program Comprehension*, pages 95–105. ACM, 2014.

[32] Thomas Zimmermann, Nachiappan Nagappan, Philip J. Guo, and Brendan Murphy. Characterizing and predicting which bugs get reopened. In *Software Engineering (ICSE), 2012*, pages 1074 –1083, june 2012. .