An Empirical Study of Bug Fixing Rate

Weiqin Zou*, Xin Xia†, Weiqiang Zhang‡, Zhenyu Chen‡, and David Lo§
*Department of Information Engineering, Jiangxi University of Science and Technology, China
†College of Computer Science and Technology, Zhejiang University, Hangzhou, China
‡Software Institute, Nanjing University, Nanjing, China
§School of Information Systems, Singapore Management University, Singapore
weiqinzou315@gmail.com, xxia@zju.edu.cn, {zwq08, zychen}@software.nju.edu.cn, davidlo@smu.edu.sg

***Abstract*—Bug fixing is one of the most important activities in software development and maintenance. A software project often employs an issue tracking system such as Bugzilla to store and manage their bugs. In the issue tracking system, many bugs are invalid but take unnecessary efforts to identify them. In this paper, we mainly focus on bug fixing rate, i.e., the proportion of the fixed bugs in the reported closed bugs. In particular, we study the characteristics of bug fixing rate and investigate the impact of a reporter's different contribution behaviors to the bug fixing rate. We perform an empirical study on all reported bugs of two large open source software communities Eclipse and Mozilla. We find (1) the bug fixing rates of both projects are not high; (2) there exhibits a negative correlation between a reporter's bug fixing rate and the average time cost to close the bugs he/she reports; (3) the amount of bugs a reporter ever fixed has a strong positive impact on his/her bug fixing rate; (4) reporters' bug fixing rates have no big difference, whether their contribution behaviors concentrate on a few products or across many products; (5) reporters' bug fixing rates tend to increase as time goes on, i.e., developers become more experienced at reporting bugs.**

Keywords*—*Bug Fixing Rate, Empirical Study, Statistical Analysis, Bug Reports

I. Introduction

Bug fixing is an essential activity to ensure software quality. Software projects often use issue tracking systems to store and manage bug reports, especially for large projects. Bugzilla[1] is a popular issue tracking system used by many software projects such as Eclipse[2] and Mozilla[3]. A typical bug report contains a number of important fields, such as the summary, description, reporter, fixer, severity, and priority fields. These fields play an important role in bug fixing.

There have been a number of studies on bug report management and bug report quality management [1]–[5], such as bug triage [1], [6], duplicate bug detection [7], [8], bug localization [2], [9], severity and priority prediction [10], and bug fixing time prediction [11]. For example, Anvik et al. propose a machine learning approach to triage a bug report to a suitable fixer [1]. Bettenburg et al. make use of social interactions to better predict whether a piece of code is buggy or not [12]. Zhou et al. propose an information-retrieval based method to better locate where the bug is [2]. Weiss et al. studied how long it will take to fix a bug [11]. Wang et al. propose a method using natural language and execution information to detect duplicate bugs [8]. Marcelo et al. [5] build a social network with reporters' interactions to predict whether a bug is valid or not. They find that the reporter's social role is a strong indicator of a bug's validity.

Developers need to handle a large number of bug reports in the issue tracking systems. However, many bugs are reported and closed but not fixed, i.e., non-fixed bugs. Developers have to spend unnecessary time to identify these non-fixed bugs, which would cause a waste of precious developers resources and results in many valid bugs not to be fixed in time. We collect all bugs reported before August 2014 in two large open source software communities Eclipse (440,024 bugs) and Mozilla (847,741 bugs). By performing a statistical analysis, we find that non-fixed bugs take a great proportion in both projects, for specific, 34.3% in Eclipse and 55.8% in Mozilla. It is of great significance to gain some insights into these non-fixed bugs.

In this paper, we mainly focus on bug fixing rate, i.e., the ratio of fixed bugs to the reported bugs. We try to gain some knowledge about bug fixing rate from two aspects. From the aspect of community, we try to get a brief view of the bug fixing rate in the whole project. From the aspect of reporters, we try to find the factors correlated with reporters' bug fixing rates. We also study the evolution of reporters' bug fixing rates. All bugs of two popular software projects, i.e., Eclipse (bugs reported before August 8, 2014 are collected) and Mozilla (bugs reported before August 1, 2014 are collected), are used to conduct our experiments. We use longitudinal analysis to study the bug fixing rate. We try to answer the following research questions:

RQ1: What is the overall situation of a project's bug fixing rate?

There are many non-fixed bugs in both projects, especially for Mozilla. And both projects' fixing rates tend to increase along with the evolution of projects.

RQ2: Will the amount of bugs a reporter ever reported, fixed or commented impact his bug fixing rate?

Yes, the fixing contribution is a strong indicator for a reporter's bug fixing rate. The reporting contribution has weak correlation with the bug fixing rate.

RQ3: Is it likely that the higher a reporter's bug fixing rate is, the less time it takes to close the bugs he reports?

Yes, there is a negative correlation between the time needed to close a reporter's bugs and his bug fixing rate.

[1]http://bugzilla.org
[2]https://bugs.eclipse.org/bugs
[3]https://bugzilla.mozilla.org

RQ4: Will a reporter's degree of concentration on products impact his bug fixing rate?

No, there is no strong relationship between a reporter's reporting, commenting or fixing concentration on products and his bug fixing rate.

RQ5: How will reporters' bug fixing rates change along with the software evolution?

Reporters in Eclipse and Mozilla tend to be more experienced at reporting bugs if they stably contributes to a project.

The main contributions of this paper are:

1) To our best knowledge, it is the first time bug fixing rate is proposed and studied. We study bug fixing rate from two aspects, the aspect of community and the aspect of reporters.
2) We perform a large-scale empirical study on bug fixing rate in two open source communities Eclipse and Mozilla. We investigate the correlations between a reporter's bug fixing rate with multiple factors, such as the contribution behaviors, the degree of concentration on products.

In the remainder of this paper, we first describe preliminary materials in Section II. The description of our experimental data and case study setup are separately presented in Section III and Section IV. Section V presents the results. Section VI gives threats to validity of our conclusions. Section VII describes the related work and Section VIII presents our conclusions.

II. Preliminaries

A. Fixed Bugs and Non-fixed Bugs

During the development of software projects, issue tracking systems such as Bugzilla are often employed to record and track bugs. A software bug is stored as a bug report in an issue tracking system. Fig. 1 presents a bug report in Bugzilla of Eclipse. When a bug is newly reported (the **Reported** field records the reporter and the bug's reported time) and confirmed by developers, its status is NEW. After the triager assigns it to a suitable fixer, its status becomes ASSIGNED. When the fixer resolves this bug, its status is changed to RESOLVED, CLOSED, or VERIFIED with a resolution item attached. There are five resolution labels for closed bugs, i.e., FIXED, INVALID, WONTFIX, DUPLICATE and WORKSFORME. The FIXED resolution means the bug is a real bug and has been fixed. INVALID means the problem described is not a bug. WONTFIX means the problem described is a bug but will be not fixed for now. DUPLICATE means the problem is a duplicate of an existing bug. WORKSFORME means the bug cannot be reproduced. If more information is provided later, a bug can be reopened. Participants can comment a bug and each comment memorizes its commentor and commented time. All historical operations on a bug will be recorded in the **history** field. In this paper, we refer the bugs closed with FIXED resolution as fixed bugs and other closed bugs as non-fixed bugs.

B. Community's Bug Fixing Rate

According to whether to compute bug fixing rate by year, we have two ways to calculating the community's bug fixing rate:

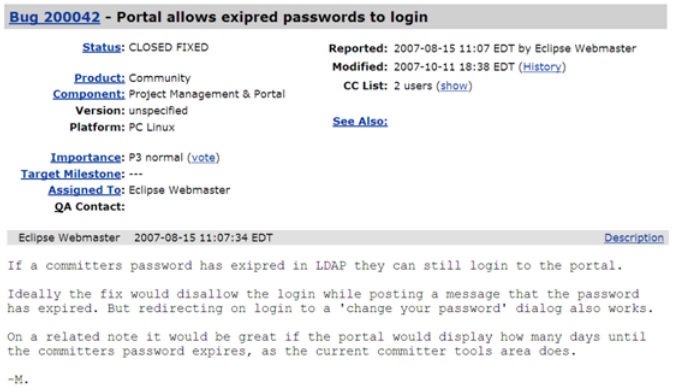

Fig. 1: A bug report of Eclipse with bugID 200042

1) Fixing rate on the whole: after counting all fixed bugs and all closed bugs of the whole project, the community's bug fixing rate is defined as the ratio of the fixed bugs to the closed bugs.

2) Fixing rate by year: first count fixed bugs and closed bugs of the whole project by year. For each year, we define the community's bug fixing rate as the ratio of the fixed bugs to the closed bugs in that year.

We give an example to help explain these two calculation methods. Suppose a project evolved from 2011 to 2012. 10 bugs are reported in 2011, of which 8 bugs are closed. Among these 8 closed bugs, 6 are fixed bugs. 20 bugs are reported in 2012, of which 15 bugs are closed. Among these 15 closed bugs, 10 bugs are fixed bugs. Then the community's bug fixing rate on the whole is (6+10)/(8+15), while the community's bug fixing rates by year are 6/8 in 2011 and 10/15 in 2012.

C. Reporter's Bug Fixing Rate

We calculate a reporter's bug fixing rate by year. The calculation method is as follows. If a reporter reported m bugs in the year Y, of which n bugs are closed. Among the n closed bugs, the amount of fixed bugs is k, then the reporter's bug fixing rate of year Y is k/n.

III. Case Study Data

We conduct our case studies on two popular software projects, i.e., Eclipse and Mozilla. Both projects contain many products. Eclipse has 249 products such as platform, core, etc. Mozilla has 106 products such as firefox, thunderbird, etc. We collect all the bugs of Eclipse up to August 8, 2014 and all the bugs of Mozilla up to August 1, 2014. Table I presents the statistics of the collected data.

The next step is to extract the contributors and the corresponding contribution time for each bug report. It is easy to get the reporter and commenters for a bug as well as the contribution time since they are recorded in the corresponding fields in a bug report. We use the **history** field with the help of heuristics rules[4] proposed by Anvik et al. [1] to determine a

[4]Heuristics For Labeling Bug Reports, http://www.cs.ubc.ca/labs/spl/projects/bugTriage/assignment/heuristics.html

TABLE I: Statistics of the collected data.

Project	Eclipse	Mozilla
time	2001.10-2014.08	1994.09-2014.08
num.products	249	106
num.reporters	41256	137488
num.bugs	440024	847741
num.fixed bugs	241881	323035
num.non-fixed bugs	126523	407772
num.comments	1461430	7427459

bug's fixer. We also use the history items to determine a bug's fixed time. For Mozilla, we first use the approved time-stamp of the fixer's last code patch as a bug's fixed time. If it does not exist, we use the time when the bug is marked as FIXED. If neither of the information exists in the history items, we use the last modification time of the bug. For Eclipse, we use the time when the bug is marked as FIXED as the fixed time for all fixed bugs. We removed the bugs whose fixers still cannot be determined with the help of the heuristics.

To avoid the bias influenced by inactive reporters whose data may mislead us, we filter out the records of inactive reporters who report less than 10 bugs in a year. We also remove all the bugs which have not been closed, since we cannot determine their final resolutions.

IV. Case Study Setup

In this section, we first present the research questions about the bug fixing rate. Then we present the longitudinal analysis method we use and the corresponding evaluation criterion.

A. Research Questions

The goal of our study is to better understand the bug fixing rate, so that developers in practice can spend more time on valid bugs and be more experienced at reporting bugs. In particular, we would like to study the bug fixing rate from two aspects. From the aspect of community, we investigate the whole bug fixing rate of a project, separately in all the time and by year. From the aspect of reporters, we investigate the characteristics of a reporter's bug fixing rate and explore how much a reporter's contribution behaviors affect his bug fixing rate. We conduct our study using the following five research questions:

RQ1: What is the overall situation of a project's bug fixing rate?

We use all fixed bugs and all closed bugs to calculate the whole project's bug fixing rate. We present the bug fixing rate in two forms according to whether to count bugs by year.

RQ2: Will the amount of bugs a reporter ever reported, fixed or commented impact his bug fixing rate?

We first filter out inactive reporters. Then, we calculate each reporter's bug fixing rate and the amount of bugs the reporter reports, fixes or comments before. Last, we calculate the Spearman correlation [13] between them. We find that the amount of bugs one reporter ever fixed is much more correlated with his fixing rate than the amount of bugs that he has reported or commented before.

RQ3: Is it likely that the higher a reporter's bug fixing rate is, the less time it takes to close the bugs he reports?

We calculate each reporter's bug fixing rate and the average time cost to close his reported bugs by year. Then we compute the Spearman correlation coefficient between these two variables. We find that there actually exists a negative correlation between them.

RQ4: Will a reporter's degree of concentration on products impact his bug fixing rate?

We first calculate each reporter's product entropy of the bugs he reported, fixed and commented before, then calculate the Spearman correlation between the fixing rate and these three kinds of product entropies. We find that a reporter's bug fixing rate has no big difference, no matter whether a reporter's contributions focus on a few products or across many products.

RQ5: How will reporters' bug fixing rates change along with the software evolution?

We first analyze the bug fixing rate evolution of all the reporters who work constantly for every year of the recent 10 years. Then we choose top 4 reporters in Eclipse and Mozilla separately, and plot each reporter's bug fixing rate over the whole project. From the results, we find that reporters' bug fixing rates tend to increase with the evolution of software projects.

B. Longitudinal Analysis

This work intends to answer the aforementioned research questions. Since both Eclipse and Mozilla have been evolving for many years, some contributors may be just active in some years. Furthermore, note that we define the bug fixing rate as the ratio of the fixed bugs to all the closed bugs. To make our experiments more rational and sound, we adopt the longitudinal analysis method to study the bug fixing rate, i.e., we calculate bug fixing rates and reporters' contributions by year.

More specifically, we first collect all bugs one reporter ever reported, fixed and commented by year. Based on the collected data, we then calculate every reporter's contributions, the degree of concentration on products, average time cost to close his bugs, and his bug fixing rate by year. Last we measure how much correlated a reporter's contribution behaviors are with his bug fixing rate. As to the evolution of reporters' bug fixing rate, we first calculate each reporter's bug fixing rate by year, then filter out inactive reporters, and finally plot the evolution track for each reporter.

C. Evaluation Criteria

We use Spearman correlation coefficient to measure how much correlated two variables are. The Spearman correlation coefficient ranges from -1 to 1. -1 and 1 separately mean a perfect negative and positive monotonic correlation. 0 means the variables are independent of each other. Table II describes the meanings of various correlation coefficient values and the corresponding correlation levels [14].

V. Case Study

In this section, we answer the research questions posed before.

TABLE II: Spearman's correlation and correlation level

correlation coefficient	correlation level
0.0 - 0.1	None
0.1 - 0.3	Small
0.3 - 0.5	Moderate
0.5 - 0.7	High
0.7 - 0.9	Very High
0.9 - 1.0	Perfect

RQ1: What is the overall situation of a project's bug fixing rate?

Motivation: Knowing the bug fixing rate of the whole project can help developers better understand the project from a higher view. The whole bug fixing rate can assist developers measure how many bugs totally are fixed and can help them better make decisions to some extent.

Approach: We calculate two kinds of community's bug fixing rates, i.e., fixing rate on the whole and fixing rate by year. The detailed calculation method is described in Section II.

Results - fixing rate on the whole: Table III presents the results. We can find that the bug fixing rate of the whole project is not that high, and is much lower for Mozilla. This means that there exist many non-fixed bugs in these two projects.

TABLE III: Whole bug fixing rates on Eclipse and Mozilla

Project	num. closed bugs	num. fixed bugs	bug fixing rate
Eclipse	368404	241881	0.66
Mozilla	730807	323035	0.44

Results - fixing rate by year: Table IV and V present the results. The percentage of non-fixed bugs ranges from 23% to 51% in Eclipse, and from 28% to 77% in Mozilla. More bugs tend to be invalid on Mozilla than that on Eclipse. Besides, we can find that both projects' fixing rates tend to increase as time goes on.

TABLE IV: Bug fixing rate by year in Eclipse

year	No. closed bugs	No. fixed bugs	bug fixing rate
2001	5995	2954	0.49
2002	21447	11371	0.53
2003	19684	9987	0.51
2004	30655	15952	0.52
2005	36933	21713	0.59
2006	42055	26273	0.62
2007	38400	26320	0.69
2008	37820	27053	0.71
2009	29926	21669	0.72
2010	27690	20164	0.73
2011	26950	19833	0.74
2012	22004	16434	0.75
2013	19120	14638	0.77
2014	9725	7520	0.77

In summary, there are many non-fixed bugs in both projects, especially for Mozilla. And both projects' fixing rates tend to increase along with the evolution of projects.

RQ2: Will the amount of bugs a reporter ever reported, fixed or commented impact his bug fixing rate?

Motivation: In an issue tracking system such as Bugzilla, contributors not only can report bugs, but also fix bugs and comment on any bugs. Such contribution behaviors differ from people to people. Knowing how much the three contributions affect a reporter's bug fixing rate not only can help people better identify valid bugs, but also give advice on how to achieve better bug fixing rate.

TABLE V: Bug fixing rate by year in Mozilla

year	No. closed bugs	No. fixed bugs	bug fixing rate
1994	2	1	0.50
1995	1	0	0.00
1996	5	2	0.40
1997	32	10	0.31
1998	2010	1112	0.55
1999	20176	10492	0.52
2000	39118	14950	0.38
2001	50587	16131	0.32
2002	61965	14561	0.23
2003	39929	9146	0.23
2004	44035	10498	0.24
2005	42163	11774	0.28
2006	38261	14321	0.37
2007	37659	15945	0.42
2008	50134	21604	0.43
2009	53347	26794	0.50
2010	44257	23513	0.53
2011	65123	37683	0.58
2012	31748	19741	0.62
2013	65206	42138	0.65
2014	45049	32619	0.72

Approach: Our experiment includes two parts. The first part is to calculate a reporter's bug fixing rate and his three kinds of contributions, i.e., the amount of bugs the reporter reported, fixed and commented. For each reporter, we count all the closed bugs which he reports in one specific year as his reporting contribution. For example, if a reporter reports n bugs in a certain year, among which m bugs are closed. Then his reporting contribution for that year is m. In a similar way, if a reporter fixes n bugs in a certain year, then his fixing contribution for that year is n. Since a reporter can submit multiple comments on a single bug, we use the number of comments rather than the number of bugs he ever comments on as his commenting contribution. Then if a reporter commented j times on bugs in a certain year, the reporter's commenting contribution for that year is j.

The second part is to calculate the correlation between the bug fixing rate and the contributions. We compute Spearman correlation coefficient between each kind of contributions and the fixing rate. The contribution values are normalized before calculation.

Results: The detailed statistics of the correlations are shown in Table VI and VII. In the two tables, the column **year** means the year the bug is reported. The **rptCnt_FR** means the correlation between a reporter's reporting contribution and his fixing rate. Similarly, **fixCnt_FR** and **cmtCnt_FR** are correlation between a reporter's fixing and commenting contributions and his fixing rate. For every kind of contribution, the Spearman correlation coefficient, the corresponding correlation level, and the significance of the correlation are presented.

From the statistics, we observe that most of the Spearman's correlation coefficients are positive, indicating that there is a positive relationship between contributions and bug fixing rate. In other words, the more contributions a reporter makes, the more likely the bugs he reports will be fixed. The correlation is much more obvious in Mozilla than in Eclipse.

TABLE VI: Correlation between reporters' contributions and bug fixing rate in Eclipse

	rptCnt_FR			fixCnt_FR			cmtCnt_FR		
year	corr	corr_level	p-value	corr	corr_level	p-value	corr	corr_level	p-value
2001	-0.076	None	0.557	0.238	Moderate	0.063	-0.039	None	0.761
2002	0.289	Small	$1.37e^{-05}$	0.332	Moderate	$4.54e^{-07}$	0.294	Small	$9.00e^{-06}$
2003	0.404	Moderate	$3.76e^{-11}$	0.528	Strong	$3.07e^{-19}$	0.399	Moderate	$7.09e^{-11}$
2004	0.313	Moderate	$2.91e^{-10}$	0.508	Strong	$8.56e^{-27}$	0.289	Small	$6.40e^{-09}$
2005	0.303	Moderate	$2.88e^{-12}$	0.496	Moderate	$7.79e^{-33}$	0.271	Small	$5.03e^{-10}$
2006	0.224	Small	$8.54e^{-09}$	0.495	Moderate	$3.40e^{-41}$	0.196	Small	$5.20e^{-07}$
2007	0.121	Small	0.003	0.388	Moderate	$3.51e^{-23}$	0.144	Small	0.000
2008	0.208	Small	$3.58e^{-07}$	0.397	Moderate	$1.31e^{-23}$	0.238	Small	$5.35e^{-09}$
2009	0.138	Small	0.003	0.367	Moderate	$1.25e^{-16}$	0.135	Small	0.003
2010	0.201	Small	$1.08e^{-05}$	0.329	Moderate	$2.57e^{-13}$	0.192	Small	$2.79e^{-05}$
2011	0.147	Small	0.002	0.329	Moderate	$1.78e^{-12}$	0.175	Small	0.000
2012	0.016	None	0.755	0.226	Small	$8.68e^{-06}$	-0.002	None	0.968
2013	0.115	Small	0.038	0.337	Moderate	$3.70e^{-10}$	0.090	None	0.101
2014	0.190	Small	0.007	0.364	Moderate	$9.89e^{-08}$	0.074	None	0.298

TABLE VII: Correlation between reporters' contributions and bug fixing rate on Mozilla

	rptCnt_FR			fixCnt_FR			cmtCnt_FR		
year	corr	corr_level	p-value	corr	corr_level	p-value	corr	corr_level	p-value
1998	0.088	None	0.573	0.143	Small	0.361	-0.016	None	0.921
1999	0.247	Small	$3.62e^{-05}$	0.606	Strong	$9.02e^{-29}$	0.443	Moderate	$1.41e^{-14}$
2000	0.308	Moderate	$1.25e^{-14}$	0.594	Strong	$1.20e^{-58}$	0.560	Strong	$6.92e^{-51}$
2001	0.337	Moderate	$5.59e^{-24}$	0.662	Strong	$1.34e^{-108}$	0.623	Strong	$1.01e^{-92}$
2002	0.257	Small	$3.41e^{-16}$	0.669	Strong	$5.68e^{-128}$	0.579	Strong	$1.28e^{-88}$
2003	0.317	Moderate	$9.00e^{-14}$	0.737	Very Strong	$4.79e^{-91}$	0.654	Strong	$1.74e^{-65}$
2004	0.248	Small	$7.21e^{-09}$	0.736	Very Strong	$3.96e^{-91}$	0.646	Strong	$1.24e^{-63}$
2005	0.387	Moderate	$3.14e^{-18}$	0.752	Very Strong	$1.03e^{-86}$	0.612	Strong	$1.16e^{-49}$
2006	0.369	Moderate	$2.55e^{-15}$	0.719	Very Strong	$9.39e^{-70}$	0.557	Strong	$1.60e^{-36}$
2007	0.400	Moderate	$4.37e^{-18}$	0.684	Strong	$5.90e^{-61}$	0.537	Strong	$9.61e^{-34}$
2008	0.307	Moderate	$1.63e^{-13}$	0.678	Strong	$5.76e^{-76}$	0.466	Moderate	$3.41e^{-31}$
2009	0.314	Moderate	$3.30e^{-14}$	0.626	Strong	$6.12e^{-62}$	0.411	Moderate	$4.28e^{-24}$
2010	0.272	Small	$2.51e^{-10}$	0.572	Strong	$8.98e^{-47}$	0.371	Moderate	$1.58e^{-18}$
2011	0.275	Small	$6.84e^{-14}$	0.629	Strong	$2.44e^{-80}$	0.354	Moderate	$1.49e^{-22}$
2012	0.205	Small	$2.51e^{-06}$	0.572	Strong	$1.07e^{-46}$	0.292	Small	$1.03e^{-11}$
2013	0.139	Small	$2.26e^{-05}$	0.518	Strong	$3.65e^{-64}$	0.204	Small	$4.75e^{-10}$
2014	0.098	None	0.007	0.534	Strong	$1.52e^{-56}$	0.216	Small	$2.37e^{-09}$

In Eclipse, the commenting and reporting contributions mostly have small impact on a reporter's bug fixing rate. But the fixing contribution has a relatively stronger positive impact on a reporter's bug fixing rate. The correlation coefficients mostly range from 0.3 to 0.5, indicating a moderate correlation between fixing contribution and bug fixing rate.

In Mozilla, the reporting contribution has also little direct impact on a reporter's bug fixing rate. The commenting contribution has a much stronger positive correlation with a reporter's bug fixing rate. Especially for the years of 2000-2007, all the correlation coefficients are more than 0.5. Compared to the other two kinds of contribution, the fixing contribution has a very high positive correlation with the bug fixing rate. The correlation coefficients fall in the range of 0.5 to 0.8, which indicate strong positive correlations. In other words, we can conclude that the more bugs a reporter ever fixed, the higher his bug fixing rate is. This is helpful while to estimate whether a reporter's newly reported bug needs to be fixed.

In summary, in both projects, the fixing contribution is a strong indicator for a reporter's bug fixing rate. The reporting contribution has weak correlation with the bug fixing rate. The commenting contribution is more referable in Mozilla while considering a reporter's bug fixing rate. These findings are helpful when developers perform software tasks such as bug triage.

RQ3: Is it likely that the higher a reporter's bug fixing rate is, the less time it takes to close the bugs he reports?

Motivation: Different reporters hold different bug fixing rates. Some reporters may be more experienced at reporting bugs. If the answer to RQ3 is positive, developers can focus more on bugs reported by reporters with higher bug fixing rate. More real bugs can be solved in time, which will naturally improve software quality.

Approach: Our experiments include two parts. The first part is to test whether the time cost to close a fixed bug differs from that to close a non-fixed bug. We conduct a Wilcoxon test [15] between fixed bugs and non-fixed bugs. The second part is to investigate whether there exists a negative correlation between the average time cost to close a reporter's bugs and his bug fixing rate. We calculate the two variables by year. We filter out the inactive reporters who report less than 10 bugs in a year. Then we use Spearman correlation to measure their correlation.

Results: The results of Wilcoxon test on both projects show that the differences in time cost to close a fixed and a non-fixed bug are significant. The p-value for Eclipse is $4.65e^{-10}$ and $2.2e^{-16}$ for Mozilla. This means that it takes more time to close a non-fixed bug than a fixed bug.

Table VIII and Table IX present our statistic results. On the whole there exists a negative correlation between a reporter's

TABLE VIII: Correlation between reporter's ave. time cost to close a bug and his bug fixing rate in Eclipse

Eclipse			
year	corr	p-value	corr_level
2001	-0.447	0.000	Moderate
2002	-0.376	$8.14e^{-09}$	Moderate
2003	-0.173	0.006	Small
2004	0.076	0.132	None
2005	-0.127	0.004	Small
2006	-0.196	$5.36e^{-07}$	Small
2007	-0.235	$4.99e^{-09}$	Small
2008	-0.291	$6.11e^{-13}$	Small
2009	-0.211	$3.29e^{-06}$	Small
2010	-0.308	$8.22e^{-12}$	Moderate
2011	-0.250	$1.15e^{-07}$	Small
2012	-0.211	$3.22e^{-05}$	Small
2013	-0.187	0.0006	Small
2014	-0.180	0.0103	Small

TABLE IX: Correlation between reporter's ave. time cost to close a bug and his fixing rate in Mozilla

Mozilla			
year	corr	p-value	corr_level
1998	0.067	0.671	None
1999	0.158	0.008	Small
2000	0.249	$6.22e^{-10}$	Small
2001	0.241	$1.12e^{-12}$	Small
2002	0.263	$5.86e^{-17}$	Small
2003	0.104	0.017	Small
2004	0.004	0.930	None
2005	-0.212	$3.50e^{-06}$	Small
2006	-0.454	$2.97e^{-23}$	Moderate
2007	-0.377	$4.33e^{-16}$	Moderate
2008	-0.469	$1.34e^{-31}$	Moderate
2009	-0.471	$3.36e^{-32}$	Moderate
2010	-0.357	$3.07e^{-17}$	Moderate
2011	-0.404	$1.45e^{-29}$	Moderate
2012	-0.423	$5.55e^{-24}$	Moderate
2013	-0.348	$1.51e^{-27}$	Moderate
2014	-0.111	0.002	Small

fixing rate and the time cost to close a bug he reported. The correlation is much stronger in Mozilla than in Eclipse. In most cases especially in recent years, the correlation coefficient in Mozilla ranges from -0.3 to -0.5, indicating a moderate negative correlation. Though the correlation in Eclipse is not that strong, it is still a small to moderate negative correlation. From the above results, we can conclude that the lower a reporter's bug fixing rate is, the more time it needs to close a bug reported by him.

In summary, there is a negative correlation between the time needed to close a reporter's bugs and his bug fixing rate. In other words, the higher a reporter's bug fixing rate is, the lesser time it needs to close his reported bugs.

RQ4: Will a reporter's degree of concentration on products impact his bug fixing rate?

Motivation: Both Eclipse and Mozilla are composed of many products. In an issue tracking system such as Bugzilla, participants can contribute to many products or a few products. Knowing how much different contribution behaviors affect a reporter's bug fixing rate can help reporters better contribute to a project.

Approach: Based on the contribution data we get in RQ2, we further calculate the entropy of products of bug reports that a reporter reports, fixes or comments as his degree of concentration on the products by year. The product entropy is calculated as follows:

$$prodEn = \sum_{i \in products} -p_i * log(p_i) \quad (1)$$

p_i is the proportion of a given product's bugs to the whole closed bugs in one year. For example, consider that one reporter has 10 bugs closed in a certain year. Among the 10 bugs, 2 bugs belong to product p1, 3 bugs belong to p2 and 5 bugs belong to p3. Then his product entropy of reported bugs is: -(2/10*log(2/10)+3/10*log(3/10)+5/10*log(5/10))=1.486.

The product entropy of fixed bugs or commented bugs is calculated in a similar way. The smaller the entropy is, the higher degree of concentration on products the reporter has. Then we perform Spearman correlation tests between the bug fixing rate and the three kinds of product entropies.

Results: The results are shown in Table X and Table XI. Column **rptProdEn_FR** contains the correlation between a reporter's reporting product entropy and his fixing rate. Columns **fixProdEn_FR** and **cmtProdEn_FR** contain the correlation between his fixing rate and his fixing and commenting product entropies respectively.

In Eclipse, all three kinds of product entropies have negative correlations with the reporter's fixing rate. But the correlation is not that strong: the correlation coefficients for reporting and commenting product entropies mostly range from -0.2 to -0.4, which indicate a moderate negative correlation. The correlation for fixing product entropy is much weaker. The values range only from -0.0 to -0.2, which indicate None to Small correlation. These results indicate that reporters in Eclipse that report and comment bugs on fewer products tend to have at least slightly higher bug fixing rate. However, it does not make a big difference whether a developer fixed bugs on many products or not.

In Mozilla, all the correlation levels for three kinds of products entropies are None to Small. These results indicate that whether a reporter contributes on a few products or across many products makes no big difference to his fixing rate.

In summary, in both projects, there is no strong relationship between a reporter's reporting, commenting or fixing concentration on products and his bug fixing rate. In other words, a reporter's degree of concentration on products does not have much impact on whether bugs he reported will be fixed.

RQ5: How will reporters' bug fixing rates change along with the software evolution?

Motivation: During the evolution of software projects, some participants contribute to the projects constantly. Knowing the evolution characteristics of such reporters' bug fixing rates can not only deepen the understanding of the project, but also help better solve relevant software tasks such as bug triage and identifying valid bugs, etc.

Approach: To answer this question, we conduct two experiments:

1) Observe the bug fixing rate trend on the whole. Here we only consider the bug fixing rate of all reporters who reported bugs

TABLE X: Correlation between reporting, fixing and commenting product entropy and fixing rate in Eclipse

	rptProdEn_FR			fixProdEn_FR			cmtProdEn_FR		
year	corr	corr_leve	p-value	corr	corr_leve	p-value	corr	corr_leve	p-value
2001	0.181	Small	0.159	0.354	Moderate	0.051	0.167	Small	0.199
2002	-0.020	Small	0.763	-0.001	None	0.995	-0.030	None	0.654
2003	-0.151	Small	0.017	0.030	None	0.745	-0.161	Small	0.011
2004	-0.214	Small	$2.21e^{-05}$	-0.005	None	0.944	-0.207	Small	$4.05e^{-05}$
2005	-0.222	Small	$4.30e^{-07}$	-0.048	None	0.430	-0.223	Small	$3.94e^{-07}$
2006	-0.370	Moderate	$2.08e^{-22}$	-0.082	None	0.112	-0.371	Moderate	$1.76e^{-22}$
2007	-0.365	Moderate	$1.78e^{-20}$	-0.154	Small	0.002	-0.368	Moderate	$8.58e^{-21}$
2008	-0.403	Moderate	$2.36e^{-24}$	-0.101	Small	0.047	-0.424	Moderate	$6.20e^{-27}$
2009	-0.430	Moderate	$6.82e^{-23}$	-0.196	Small	0.000	-0.423	Moderate	$4.84e^{-22}$
2010	-0.360	Moderate	$7.13e^{-16}$	-0.087	None	0.108	-0.352	Moderate	$3.61e^{-15}$
2011	-0.369	Moderate	$1.56e^{-15}$	-0.079	None	0.155	-0.370	Moderate	$1.33e^{-15}$
2012	-0.315	Moderate	$3.72e^{-10}$	-0.101	Small	0.085	-0.304	Moderate	$1.58e^{-09}$
2013	-0.265	Small	$1.07e^{-06}$	-0.068	None	0.279	-0.284	Small	$1.59e^{-07}$
2014	-0.185	Small	0.008	-0.185	Small	0.017	-0.164	Small	0.020

TABLE XI: Correlation between reporting, fixing and commenting product entropy and fixing rate in Mozilla

	rptProdEn_FR			fixProdEn_FR			cmtProdEn_FR		
year	corr	corr_leve	p-value	corr	corr_leve	p-value	corr	corr_leve	p-value
1998	-0.263	Small	0.088	0.128	Small	0.541	-0.222	Small	0.152
1999	-0.139	Small	0.021	0.137	Small	0.062	-0.114	Small	0.060
2000	-0.084	Small	0.041	0.272	Small	$1.43e^{-07}$	-0.120	Small	0.003
2001	-0.004	Small	0.899	0.301	Small	$8.52e^{-11}$	-0.147	Small	$1.63e^{-05}$
2002	-0.025	Small	0.428	0.279	Small	$1.94e^{-10}$	-0.102	Small	0.001
2003	-0.075	Small	0.085	0.273	Small	$2.04e^{-06}$	-0.238	Small	$3.28e^{-08}$
2004	-0.057	Small	0.191	0.260	Small	$1.19e^{-05}$	-0.118	Small	0.007
2005	-0.114	Small	0.013	0.199	Small	0.000	-0.231	Small	$-4.20e^{-07}$
2006	-0.114	Small	0.017	0.165	Small	0.006	-0.272	Small	$-9.60e^{-09}$
2007	-0.019	Small	0.687	0.232	Small	$3.38e^{-05}$	-0.081	None	0.093
2008	-0.108	Small	0.011	0.140	Small	0.004	-0.233	Small	$2.79e^{-08}$
2009	-0.149	Small	0.000	-0.046	None	0.338	-0.209	Small	$6.14e^{-07}$
2010	-0.176	Small	$5.04e^{-05}$	0.059	None	0.220	-0.233	Small	$7.16e^{-08}$
2011	-0.231	Small	$3.69e^{-10}$	0.023	None	0.578	-0.262	Small	$9.64e^{-13}$
2012	-0.233	Small	$7.90e^{-08}$	-0.106	None	0.023	-0.243	Small	$1.96e^{-08}$
2013	-0.087	Small	0.008	0.059	None	-0.098	-0.094	None	0.004
2014	-0.130	Small	0.000	0.075	None	0.057	-0.112	Small	0.002

every year of the recent 10 years, i.e., 2004-2013. Note that we filter out the data of 2014 since our collected reports are only before August, 2014. There are 85 reporters selected in Eclipse, and 220 reporters in Mozilla. We calculate the whole bug fixing rate by year. Then we use the boxplot graph to describe the whole bug fixing rate evolution trend of them.

2) Observe the bug fixing rate trend of the top reporters. We choose top 4 reporters to perform our experiments. We only consider those people reporting more than 100 bugs in every year of 2004-2013. If there are not enough reporters meeting the requirement, we then loose our limitation, i.e., a reporter can report less than 100 bugs in a year. After choosing top 4 reporters for Eclipse and Mozilla, we then plot each reporter's bug fixing rate to observe their bug fixing rate evolution trend.

Results - bug fixing rate trend on the whole: Fig.2 and 3 show the results. The whole bug fixing rates of both projects tend to increase as time goes on. This indicates that reporters will become more experienced over time.

Results - bug fixing rate trend of top reporters: In Fig. 4 and 5, each reporter exhibits an obvious increasing trend of his bug fixing rates. This means that a reporter will become more experienced at reporting bugs when he stably contributes to a certain project. In other words, the longer an active reporter works in a project, the more likely it is that the bugs he reports will be fixed.

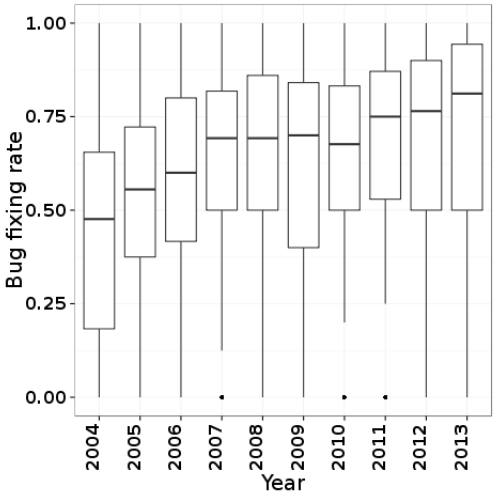

Fig. 2: Fixing rate trend on the whole in Eclipse

In summary, reporters in both projects, especially the top reporters tend to be more experienced at reporting bugs if they stably contributes to a project.

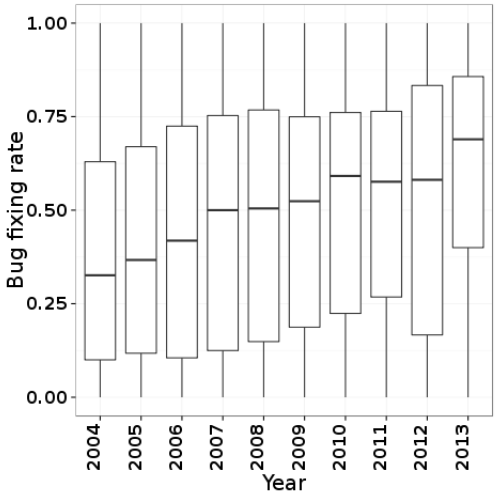

Fig. 3: Fixing rate trend on the whole in Mozilla

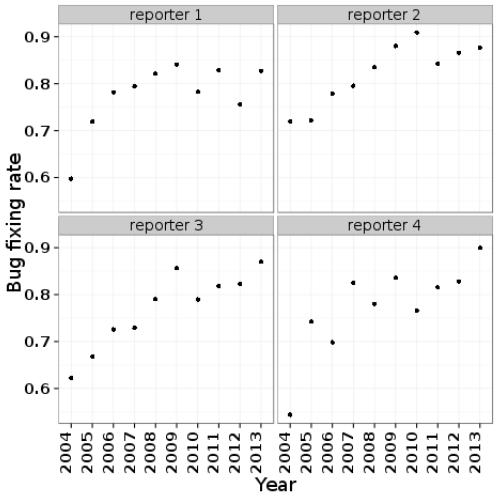

Fig. 4: Fixing rate trend of top 4 reporters on Eclipse

VI. Threats to validity

There are several threats that may potentially impact the validity of our work. First, the generalization of our conclusions is limited. Since we only perform our experiments on just two open-source projects, the conclusions may not be applicable for other open-source projects and non-open-source projects. We need more projects to validate our findings and this is also our future work. Whereas, since both Eclipse and Mozilla are large scale, typical and popular projects, the results can still offer some advices to participants' behaviors while contributing to a project. Second, while we investigate the correlation of a reporter's reporting/fixing/commenting contribution with his bug fixing rate, we cannot yet explain the exact causal relationships behind these phenomena. For example, we still do not know why someone's reported or commented bug amount seems to have no direct correlation with his bug fixing rate, while his fixed bug amount does have the correlation. Our next plan is to try to find some rational supportive evidence to help us gain more understanding of what we have concluded.

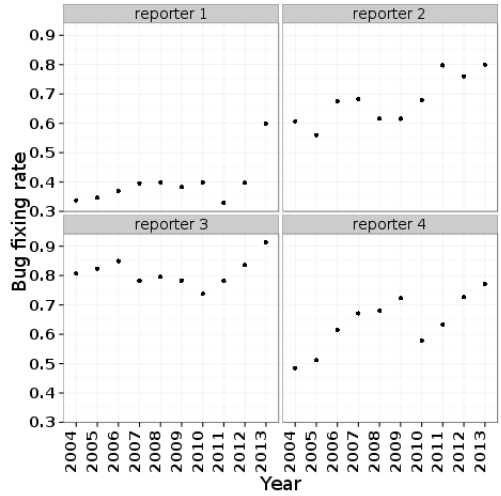

Fig. 5: Fixing rate trend of top 4 reporters on Mozilla

Third, when we analyze the correlation, we only use the Spearman correlation coefficient to measure it. This may limit our findings. More correlation metrics may be suitable to help us gain more insights into the factors affecting the bug fixing rate.

VII. Related work

The areas related to our work include bug report quality management and bug report management.

A. Bug Report Quality Management

In recent years, researchers are taking more efforts on the study of bug quality. In [16], Zimmermann et al. perform a study on the quality of bug reports in bug repositories. They find that there exist bugs with bad quality reported by inexperienced reporters and conduct a survey on what makes a good bug report. Bird et al. [17] find that many bug reports in bug repositories are unable to be linked with the corresponding source code. The bias would affect the effectiveness of bug prediction models. Xia et al. [18] conduct an investigation on bug field reassignment. They find that many fields get reassigned and the bugs with more reassigned fields need more time to be closed.

There have been some studies on reopened bug prediction [19]–[21]. Shihab et al. [19] make an investigation on the factors that impact the reopen probability of a bug. Zimmermann et al. [20] conduct a study on the causes of reopened bugs and build a model to predict which bugs get reopened. Xia et al. [21] propose a *ReopenPredictor* to automatically predict reopened bugs with high accuracy.

There are also several studies that predict the severity of bug reports. Menzies and Marcus [22] propose Severis to perform multi-class classification to predict the five severity labels of bug reports in NASA. Lamkanfi et al. [23] extend their work by predicting two severity labels, i.e., severe and not severe. They also investigate the effectiveness of a number of classification algorithms to predict the severity of bug reports

[24]. Tian et at. [25] use information retrieval based nearest neighbor classification to conduct the bug severity prediction.

B. Bug Report Management

Researchers in this area mainly focus on using bug reports to help address traditional software maintenance tasks. These tasks can be grouped into five categories, i.e., bug triage, duplicate bug detection, bug prediction, bug localization and bug categorization.

1) Bug Triage. Mockus et al. [26] study the impact of triage on resolving bugs. To save triagers' time, Murphy et al. [1], [27] propose a semi-automatic method based on text categorization to recommend proper developers to fix a bug. There are also many works to enhance the triage accuracy. Xuan et al. [6] use a semi-supervised method to augment the number of labeled bugs to better triage a bug. They also propose a data set reduction method to facilitate the bug triage process [28]. Xia et al. propose DevRec which combine developer-based and developer-based components to recommend developers to resolve bugs [29], [30]. Encouraged by the bug's reassignment phenomenon, Jeong et al. [31] employ a tossing graph to improve the accuracy of bug triage. Kim et al. [32] propose a cost-ware triage algorithm to achieve a good balance between triage accuracy and cost time, etc.

2) Duplicate Bug Detection. Runeson et al. [7] build a duplicate bug detection model with natural language processing to process unstructured text. Wang et al. [8] not only use natural language information but also execution information to enhance the accuracy of duplicate bug detection. Sureka et al. [33] use character n-gram-based features to detect duplicate bugs. Sun et al. [34] use discriminative models for information retrieval to detect duplicate bug reports more accurately. Nguyen et al. combine information retrieval technology and topic modeling to perform duplicate bug detection [35] etc.

3) Bug Prediction. With future fault predictors and a linear regression model, Ostrand et al. identify top 20% of problematic files in a project [36]. Kim et al. propose a bug cache algorithm to predict future faults based on previous fault localities [37]. Many researchers propose to incorporate developers' behaviors to help better predict bugs. Meneely et al., Bird et al., and Pinzger et al. use churn information and dependency relationships to build DSNs to do defect prediction [3], [38], [39], while [12], [40] use quantitative analysis to build developer networks to predict whether a piece of code is buggy or not. Shivaji et al. [41] use reduced features to perform bug prediction. To avoid the bug prediction bias resulted by non-corrective bug reports, Zhou et al. [42] propose a strategy that combines the text mining and data mining techniques to identify the corrective bugs, etc.

4) Bug Localization. To point out which piece of code a bug locates in, Abreu et al. [43] propose a spectrum-based fault localization method. Zeller et al. [44] use delta debugging to help developer locate bugs. Artzi et al. [9] use dynamic test and model checking to locate bugs. Zhou et al. [2] propose an information retrieval based method to perform bug localization. Wang et al. [45] use compositional vector space models to improve bug localization. Zou et al. [46] make use of correlations among crash reports to improve bug localization, etc. Considering the language of bug reports and source code are different, Xia et al. propose CrosLocator which is based on language translation [47].

5) Bug Categorization. Garcia and Shihab study the blocking bugs in open source projects and propose the usage of random forest to predict blocking bugs [48]. In a later work, Xia et al. propose ELBlocker which is an ensemble learning approach to further improve the performance of blocking bug prediction [49]. Xia et al. propose a fuzzy set based feature selection approach which selects important terms from the natural language description of bug reports to categorize bugs based on their fault triggering conditions [50]. Thung et al. classify defects into IBM's Orthogonal Defect Classification (ODC)'s defect types by learning a discriminative model based on textual features extracted from bug reports and code features extracted from bug fixing changes [51]. Huang et al. propose AutoODC which leverages a text classification technique to categorize defects according to their impact [52]. Xia et al. propose the usage of text mining techniques to identify configuration bugs [53].

VIII. Conclusion

In this paper, we perform an in-depth investigation on bug fixing rate in Eclipse and Mozilla. We study bug fixing rate from two aspects. From the community aspect, we find that there are many non-fixed bugs in both projects and the whole fixing rates of both projects tend to increase along with projects' evolution.

From the reporter aspect, our experiments include two parts. The first part is to study the characteristics of a reporter's bug fixing rate. We find that there is a negative correlation between a reporter's bug fixing rate and the average time cost to close the bugs he reports. We also find that reporters tend to be more experienced at reporting bugs if they stably contribute to a project. The second part is to investigate how much a reporter's contribution behaviors (reporting, fixing and commenting bugs) affect his bug fixing rate. From the results, we find that the amount of bugs one reporter ever fixed has a strong correlation with his bug fixing rate, i.e., the more bugs a reporter ever fixed, the more likely it is that the bugs he reports will be fixed. On the contrary, it seems that the number of times that someone reports or comments bugs has little direct or obvious relationship with his bug fixing rate. We also find that there is no big difference to a reporter's bug fixing rate whether he contributes on a few products or across many products. All these findings in this paper help us acquire more understanding of bug reporters' behaviors during software development.

Acknowledgment

The research is supported by Jiangxi University of Science and Technology Nanchang Campus self-fund under Grant NSFJ2014-G36, the National Natural Science Foundation of Jiangxi Province under Grant No.20132BAB201044, and the Fundamental Research Funds for the Central Universities.

References

[1] J. Anvik, L. Hiew, and G. C. Murphy, "Who should fix this bug?" in *ICSE*. ACM, 2006, pp. 361–370.

[2] J. Zhou, H. Zhang, and D. Lo, "Where should the bugs be fixed? more accurate information retrieval-based bug localization based on bug reports," in *ICSE*. IEEE, 2012, pp. 14–24.

[3] A. Meneely, L. Williams, W. Snipes, and J. Osborne, "Predicting failures with developer networks and social network analysis," in *FSE*. ACM, 2008, pp. 13–23.

[4] A. Kumar and A. Gupta, "Evolution of developer social network and its impact on bug fixing process," in *ISEC*. ACM, 2013, pp. 63–72.

[5] M. S. Zanetti, I. Scholtes, C. J. Tessone, and F. Schweitzer, "Categorizing bugs with social networks: A case study on four open source software communities," in *ICSE*. IEEE Press, 2013, pp. 1032–1041.

[6] J. Xuan, H. Jiang, Z. Ren, J. Yan, and Z. Luo, "Automatic bug triage using semi-supervised text classification." in *SEKE*, 2010, pp. 209–214.

[7] P. Runeson, M. Alexandersson, and O. Nyholm, "Detection of duplicate defect reports using natural language processing," in *ICSE*. IEEE, 2007, pp. 499–510.

[8] X. Wang, L. Zhang, T. Xie, J. Anvik, and J. Sun, "An approach to detecting duplicate bug reports using natural language and execution information," in *ICSE*. ACM, 2008, pp. 461–470.

[9] S. Artzi, A. Kiezun, J. Dolby, F. Tip, D. Dig, A. Paradkar, and M. D. Ernst, "Finding bugs in web applications using dynamic test generation and explicit-state model checking," *Software Engineering, IEEE Transactions on*, pp. 474–494, 2010.

[10] J. Xuan, H. Jiang, Z. Ren, and W. Zou, "Developer prioritization in bug repositories," in *ICSE*. IEEE, 2012, pp. 25–35.

[11] C. Weiss, R. Premraj, T. Zimmermann, and A. Zeller, "How long will it take to fix this bug?" in *MSR*. IEEE, 2007, p. 1.

[12] N. Bettenburg and A. E. Hassan, "Studying the impact of social interactions on software quality," *Empirical Software Engineering*, pp. 375–431, 2013.

[13] G. U. Yule, *An introduction to the theory of statistics*. C. Griffin, limited, 1919.

[14] W. G. Hopkins, *A new view of statistics*. Will G. Hopkins, 1997.

[15] H. B. Mann and D. R. Whitney, "On a test of whether one of two random variables is stochastically larger than the other," *The annals of mathematical statistics*, pp. 50–60, 1947.

[16] N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, and T. Zimmermann, "What makes a good bug report?" in *FSE*. ACM, 2008, pp. 308–318.

[17] C. Bird, A. Bachmann, E. Aune, J. Duffy, A. Bernstein, V. Filkov, and P. Devanbu, "Fair and balanced?: bias in bug-fix datasets," in *ESEC/FSE*. ACM, 2009, pp. 121–130.

[18] X. Xia, D. Lo, M. Wen, E. Shihab, and B. Zhou, "An empirical study of bug report field reassignment," in *CSMR-WCRE*. IEEE, 2014, pp. 174–183.

[19] E. Shihab, A. Ihara, Y. Kamei, W. M. Ibrahim, M. Ohira, B. Adams, A. E. Hassan, and K.-i. Matsumoto, "Predicting re-opened bugs: A case study on the eclipse project," in *WCRE*. IEEE, 2010, pp. 249–258.

[20] T. Zimmermann, N. Nagappan, P. J. Guo, and B. Murphy, "Characterizing and predicting which bugs get reopened," in *ICSE*. IEEE, 2012, pp. 1074–1083.

[21] X. Xia, D. Lo, E. Shihab, X. Wang, and B. Zhou, "Automatic, high accuracy prediction of reopened bugs," *Automated Software Engineering*, pp. 1–35, 2014.

[22] T. Menzies and A. Marcus, "Automated severity assessment of software defect reports," in *ICSM*. IEEE, 2008, pp. 346–355.

[23] A. Lamkanfi, S. Demeyer, E. Giger, and B. Goethals, "Predicting the severity of a reported bug," in *MSR*. IEEE, 2010, pp. 1–10.

[24] A. Lamkanfi, S. Demeyer, Q. D. Soetens, and T. Verdonck, "Comparing mining algorithms for predicting the severity of a reported bug," in *CSMR*. IEEE, 2011, pp. 249–258.

[25] Y. Tian, D. Lo, and C. Sun, "Information retrieval based nearest neighbor classification for fine-grained bug severity prediction," in *WCRE*. IEEE, 2012, pp. 215–224.

[26] J. Xie, M. Zhou, and A. Mockus, "Impact of triage: a study of mozilla and gnome," in *ESEM*. IEEE, 2013, pp. 247–250.

[27] D. Čubranić, "Automatic bug triage using text categorization," in *SEKE*. Citeseer, 2004.

[28] J. Xuan, H. Jiang, Y. Hu, Z. Ren, W. Zou, Z. Luo, and X. Wu, "Towards effective bug triage with software data reduction techniques," *IEEE Trans. Knowl. Data Eng.*, vol. 27, no. 1, pp. 264–280, 2015.

[29] X. Xia, D. Lo, X. Wang, and B. Zhou, "Accurate developer recommendation for bug resolution," in *WCRE*. IEEE, 2013, pp. 72–81.

[30] ——, "Dual analysis for recommending developers to resolve bugs," *Journal of Software: Evolution and Process*, pp. 195–220, 2015.

[31] G. Jeong, S. Kim, and T. Zimmermann, "Improving bug triage with bug tossing graphs," in *ESEC/FSE*. ACM, 2009, pp. 111–120.

[32] J.-w. Park, M.-W. Lee, J. Kim, S.-w. Hwang, and S. Kim, "Costriage: A cost-aware triage algorithm for bug reporting systems." in *AAAI*, 2011.

[33] A. Sureka and P. Jalote, "Detecting duplicate bug report using character n-gram-based features," in *APSEC*. IEEE, 2010, pp. 366–374.

[34] C. Sun, D. Lo, X. Wang, J. Jiang, and S.-C. Khoo, "A discriminative model approach for accurate duplicate bug report retrieval," in *ICSE*. ACM, 2010, pp. 45–54.

[35] A. T. Nguyen, T. T. Nguyen, T. N. Nguyen, D. Lo, and C. Sun, "Duplicate bug report detection with a combination of information retrieval and topic modeling," in *ASE*. IEEE, 2012, pp. 70–79.

[36] T. J. Ostrand, E. J. Weyuker, and R. M. Bell, "Predicting the location and number of faults in large software systems," *Software Engineering, IEEE Transactions on*, pp. 340–355, 2005.

[37] S. Kim, T. Zimmermann, E. J. Whitehead Jr, and A. Zeller, "Predicting faults from cached history," in *ICSE*. IEEE, 2007, pp. 489–498.

[38] C. Bird, N. Nagappan, H. Gall, B. Murphy, and P. Devanbu, "Putting it all together: Using socio-technical networks to predict failures," in *ISSRE*. IEEE, 2009, pp. 109–119.

[39] M. Pinzger, N. Nagappan, and B. Murphy, "Can developer-module networks predict failures?" in *FSE*. ACM, 2008, pp. 2–12.

[40] P. Bhattacharya, M. Iliofotou, I. Neamtiu, and M. Faloutsos, "Graph-based analysis and prediction for software evolution," in *ICSE*. IEEE Press, 2012, pp. 419–429.

[41] S. Shivaji, J. E. J. Whitehead, R. Akella, and S. Kim, "Reducing features to improve bug prediction," in *ASE*. IEEE Computer Society, 2009, pp. 600–604.

[42] Y. Zhou, Y. Tong, R. Gu, and H. Gall, "Combining text mining and data mining for bug report classification," in *ICSME*. IEEE, 2014, pp. 311–320.

[43] R. Abreu, P. Zoeteweij, R. Golsteijn, and A. J. Van Gemund, "A practical evaluation of spectrum-based fault localization," *Journal of Systems and Software*, pp. 1780–1792, 2009.

[44] A. Zeller and R. Hildebrandt, "Simplifying and isolating failure-inducing input," *Software Engineering, IEEE Transactions on*, pp. 183–200, 2002.

[45] S. Wang, D. Lo, and J. Lawall, "Compositional vector space models for improved bug localization," in *ICSME*. IEEE, 2014, pp. 171–180.

[46] S. Wang, I. T. o. Khomh, FouSoftware Engineering, and Y. Zou, "Improving bug localization using correlations in crash reports," in *MSR*. IEEE, 2013, pp. 247–256.

[47] X. Xia, D. Lo, X. Wang, C. Zhang, and X. Wang, "Cross-language bug localization," in *ICPC*. ACM, 2014, pp. 275–278.

[48] H. Valdivia Garcia and E. Shihab, "Characterizing and predicting blocking bugs in open source projects," in *MSR*. ACM, 2014, pp. 72–81.

[49] X. Xia, D. Lo, E. Shihab, X. Wang, and X. Yang, "Elblocker: Predicting blocking bugs with ensemble imbalance learning," *Information and Software Technology*, 2015.

[50] X. Xia, D. Lo, X. Wang, and B. Zhou, "Automatic defect categorization based on fault triggering conditions," in *ICECCS*. IEEE, 2014, pp. 39–48.

[51] F. Thung, D. Lo, and L. Jiang, "Automatic defect categorization," in *WCRE*. IEEE, 2012, pp. 205–214.

[52] L. Huang, V. Ng, I. Persing, R. Geng, X. Bai, and J. Tian, "Autoodc: Automated generation of orthogonal defect classifications," in *ASE*. IEEE, 2011, pp. 412–415.

[53] X. Xia, D. Lo, W. Qiu, X. Wang, and B. Zhou, "Automated configuration bug report prediction using text mining," in *COMPSAC*. IEEE, 2014, pp. 107–116.