Open Source Systems Bug Reports: Meta-Analysis

Wajdi Aljedaani
Dept. of Computer Technology
Al-Kharj College of Technology
Al-Kharj, Saudi
waljedaani@tvtc.gov.sa

Yasir Javed
Dept. of Computer Science
Prince Sultan University
Riyadh,Saudi Arabia
yjaved@psu.edu.sa

Mamdouh Alenezi
Dept. of Computer Science
Prince Sultan University
Riyadh,Saudi Arabia
malenezi@psu.edu.sa

ABSTRACT
Bug Tracking System (BTS) is a wealthy source of software development information. They contain many insights about the health status of the software project. Making sense of this information is a big challenge to software development communities. In this work, we perform an investigation on fixing time, components, and platforms related to five open source systems hosted by Bugzilla. The motive is to identify what are the most error-prone components and to allocate the right developer to fix the bug. The results are indicators for how data in BTS should be utilized for decision-making processes. The results reveal a strong relationship between bugs and committers, where it is seen that committer is usually related to fixing a single domain of bugs that also shows their expertise. This study can also help in the automated classification of bug allocation to the right kind of committers instead of manual allocation that will result in a reduction of fixing time and interloping between different committers.

CCS Concepts
• **Software and its engineering→Software testing and debugging** • **Software and its engineering→ Empirical software validation.**

Keywords
bug repository, bug reports; fixing-time; empirical studies.

1. INTRODUCTION
Bug Tracking System (BTS) is a rich source of software development information. They contain an ample amount of knowledge that can help in improving both process and product. BTS manages bug records and their status and tracks them. It enables users and developers to submit flaws, recommend improvements, and comments on bug reports. These bug reports are the primary source of software maintenance advice for producing systems that are more robust.

BTS effective use can enhance the development process in various ways [1], enabling non-co-developers. Track the evolution of the project and improve the project's quality.

In this work, we perform an investigation on fixing time, components, and platforms related to five open source systems hosted by Bugzilla. Our motivation is to identify what are the most error-prone components and allocating to write a committer to solve the bug in the least amount of time. Secondly, it will help future developers to a wide these kinds of bugs.

Bugzilla and GitHub were choices for this study. As desktop systems, they have responsibility for the majority of tracking issues for large and well-known companies to improve their systems. More precisely, they provide extensive detailed data (summary, OS, reported date, components, etc.), and freely available bugs and commits repositories. These repositories provide a diverse cluster of supporting information on the kinds and possible solutions to bugs identified by end customers and application developers; these are hard to locate, for instance, comprehensive multi-platform bug datasets.

Specifically, we examined four research questions in our study:

RQ1: *What are the most common components solved by committers?*

RQ2: *Is there a relation between components and time to resolve the bugs?*

RQ3: *How are bugs distributed over different platforms and what is the fixing time for platform?*

RQ4: *How is priority of bugs related to fixing time?*

The remainder of the paper is organized as follows. Section 2 provides information on the repositories and the selected systems. Section 3 presents our research approach and collection and processing of the data. Section 4 illustrates the study results of our four research questions. Section 5 shows the discussion and observations of our study. In section 6 presents related work, and we discuss the threats to validity in Section 7. Finally, we conclude the paper in Section 8.

2. BACKGROUND
In this section, we discuss background information about repository descriptions as well as about selected systems of this study.

2.1. Repository Descriptions
In this case study, the focus is on two of the larger open-source repositories that contain bug reports and commits sourced across a variety of systems that are Bugzilla and GitHub repositories. The characteristics of these repositories are as follows.

Bugzilla is an open-source system that used to tracking bug/issue in order to allow developers to manage tracking the issues of their products. It has been used by well-known projects, including Firefox and Bugzilla itself. In this research, Bugzilla is used because it contains distribution in the large corpus of bugs on multiple systems and platforms as Desktop, mobile, etc.

ICBDE '20, April 1–3, 2020, London, United Kingdom

ACM ISBN 978-1-4503-7498-9/20/04…$15.00
DOI: https://doi.org/10.1145/3396452.3396459

all bug reports are used that have been submitted for the entire repository in the selected systems.

GitHub is a web-based hosting service that uses Git to monitor variants. It is usually used for software code. It provides all of Git's distributed version control and source code management (SCM) characteristics as well as its characteristics. It gives access to project controls and offers different characteristics of cooperation, for example, task management, bug tracking, and feature and enhancement requests for the project. This study selected GitHub because most of the open-source systems utilized to record their commits to improve or update their systems. Commits are used in order to find out the fixing time of the bugs. The bug reports repository contains the reported date and time, but it missed the time when the bugs have been fixed.

Table 1. Statistics of Datasets Used in Our Study

Project	#Bug Reports	Start Date	End Date
Ant	1,621	16-03-2001	03-11-2017
SWT	10,833	11-10-2001	10-04-2017
Birt	9,266	23-02-2005	01-11-2016
AspectJ	3,171	30-12-2002	15-12-2016
UI	7,734	14-06-2005	06-04-2017
Total	**32,625**		

2.2. Selected Systems

Empirical study is applied to five open source projects that are written in Java. In choosing the projects to be examined, we utilized several criteria. First, all the projects that have been studied hosted in the Bugzilla tracking system. We have chosen Bugzilla because it is well- known and widely used for most of the popular open-source projects. Second, a large number of bug reports, since we are interesting long- term software projects so that we can have a large number of bug reports to be examined. Third, the commits for each project should be publicly available online to have the ability to link between the bug reports and their commits. All test projects are used GitHub for their Version Control System.

Table 1 presents high-level data on selected projects. The second column shows the number of bug reports in each system. The third and fourth columns give the time span we considered from each project, where we aimed to analyze the entire time for each project from the first bug record in the tracking system till the time we collected the data. We now provide an overview of each project.

Ant known widely as Apache Ant, which is a library made in Java used for developing Java applications. Ant provides the most required tasks such as assembling of Java program, compilation, test, and execute. It can also be used to build non-Java applications such as C++ or C applications. Apache is flexible, no coding conventions are imposed, and most importantly, users can make their Ant libraries using existing Ant Libraries [2].

SWT Standard Widget Toolkit (SWT) is toolkit made in Java and is open source. It is used for designing the user-interfaces on any operating system (OS) as it uses the native graphical libraries of OS. SWT and Swing have similar goals but with different implementation style as SWT eases the use of graphical libraries [3].

UI is written in Java and provides an efficient way of implementing the Rich Internet Application and also can be used in making a desktop application. The application developed in UI can run on any device such as a tablet, mobile, or desktop. It is based on Swing and Advance Windowing Toolkit [4].

AspectJ is written in Java and is based on cross-cutting modularity principals defined in aspect-oriented programming. It allows the implementation of essential aspects that were not considered in the traditional implementation of JAVA, such as the implementation of security as a modular point. It can be executed like JAVA wherever there is Java Virtual Machine (JVM), making is most widely used [5].

Birt is a Java library used for creating data visualizations and reports that can be implemented in rich web applications. Birt has a component for doing graphical design of data and reports. It can print reports in many formats like pdf or web and can access many data sources such as XML, SQL or Pojo etc. [6].

3. STUDY DESIGN

In order to address the four research questions, this research focused on five open source projects by collecting the data and then processing the data to find the answers to identified research questions.

3.1. Study Approach

This section presents the design of an adapted study approach on the bug reports and commits repositories of all five examined projects. Figure 1 shows an overview of our approach. First, we check the bug repository and commits for all projects. Then, for each project, we extracted the entire data from the bug repository. Table 2 highlights the list of selected features extracted along with descriptions and possible contents. After we have all the data crawled, we merge the bug reports data with the commits data by the bug id(s). The table shows already cleaned data to be more organized, such as discarding the unnecessary data such as summary, duplicated, etc. as they are not related to our study. Using this data, we compute numbers of metrics such as average fixing time, frequency, and fixing time. Lastly, a statistical comparison was made on these metrics between projects based on each research question.

3.2. Data Collection

In this study, we collected all the bug reports of five selected projects, Ant[1], SWT[2], Birt[3], AspectJ[4], and UI[5] for the period from March 16, 2001, to November 03, 2017, in a total of 32,625 bug reports. All five projects make their bug monitoring systems available to the public. Bugzilla [6] bug tracker repository was used by the projects. We used Scrap[7] as an open-source web-scraping tool to obtain and crawl bug reports in the bug tracking system. In addition, we collected the data of commits for each of the bug reports from Github[8] repository. We used a script written in **R** that uses GitHub API in order to crawl commits data from the GitHub repository. Note that we refer to a bug report as an issue without distinguishing whether Bug or Features.

3.3. Data Processing

Mining Bug Reports Repository. Importance is usually reported as P1 to P5, where P1 refers to low severity, while P5 is the highest. There are a number of types of severity that may be reported, such as major or critical refereeing to the bug that needs a quick fix as it is halting the core compilation. The severity of bugs refers to the impact level of the bug-like critical means bug will cause the failure of the whole

program, and low may refer to non-standard or maybe only aesthetic feel. The reporting of bug always comes in a combination of priority and severity. We modified the severity level from 15 to 10, where least priority bugs were classified as a single number, and later in terms of p5, we took three levels (8, 9, and 10). Where 0 is considered to lowest priority and lowest severity and ten corresponds to the highest priority and highest severity. We selected a critical attribute that is the platform, where we used all for representing the three platforms that are Windows, Macintosh, and Linux. We also picked individual categories for Windows, Macintosh, and Linux, but in our dataset, we merged different versions of these systems to one as a single platform as mostly the developers can solve the bugs for all of the different versions of operating systems unless specified. We also used a category ML to represent the bugs that have been represented in two versions of different operating systems, and also, if it is for other vendors, we classified them as Multilevel can help the automated system

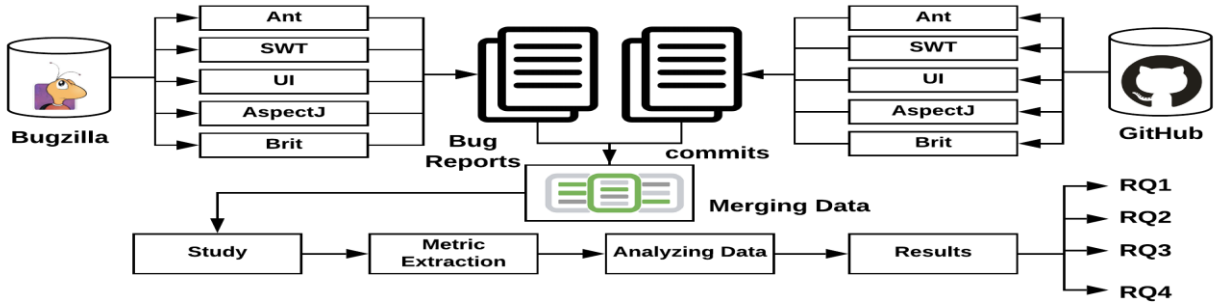

Figure 1: Overview Approach of Our Study

Table 2: Information Included in Our Dataset

Items	Description	Items Content
Platform	Operating system of hardware that is being used for the selected project	Windows, Macintosh, Linux, Mutiple OSs
CCList	How many persons are being informed for the reported bug	Number of people
Product	The name of the project for which bug is reported and solved	Ant, AspectJ, Brit, UI, SWT
Importance	The severity of reported bug	10 to 0 where 10 is the highest number
Component	The component of project for which the bug is reported	Refer to Table 2
Status	The current status of reported bug	Refer to Table 3
Committer	The person who committed after solving the bug	Name of the committer
ReporterName	The person who identified and reported the bug	Name of the reporter
Days	The number of days to solve the bug	Number of days

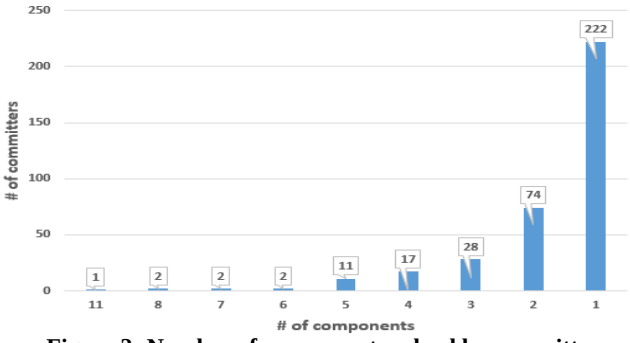

Figure 2: Number of components solved by committers

4. STUDY RESULTS

RQ1: *What are the most common components solved by committers?*
Motivation. The reason behind this research question is to find-out the priority of the committer to solve any component(s). This to assign the right kind of committer to its expertise that can result in a reduction of bug-fixing time as well as will reduce ticket hopping.

Approach. [7], [8], [9], [10] shows in their study that mostly the bugs are allocated to committer manually and may result in tossing or wrong classification. We compared the committers to their component resolving efforts, and we considered all the commits. For this, we considered all the committers and their committed efforts in solving all the components.

Results and Discussion. As shown in figure 2, we have observed that 82.45 percent of committers are dedicated to solving only one or two kinds of bugs, while less than 18 percent of the committers are solving the issues focused on more than two components. It reveals that mostly the specialty of people is around single components or two in case if they are related. This study can also reveal that bugs can give the right kind of committers who can solve it quickly. This study highlights that most of the committers are specialized in solving a single or two components, and thus, we can easily automate the assignment of bugs to relevant committers.

RQ2: *Is there a relation between components and time to resolve the bugs?*

Motivation. The reason behind this research question is to identify the time to solve each bug. It will help in many research directions, such as creating rules for maximum time to solve a specific bug in a component. It can also help in the prediction of the resolution of bug time.

Approach. [11], [12], [13] in their research classified the resolution of bugs according to the open-source system and their prediction time according to those systems. We studied in this research about the completion time of each component to help in predicting the real approximate time for each system. We calculated the average for each component, fixing time from the selected project. The average time is calculated based on the total time to fix the bugs of the specific component over the total number of bugs of the specific components.

Table 3. Distribution of components with relation to their average fixing time and reported number of bugs

Project	Component	#Component Frequency	#Avg. To Solve
Ant	Antidote	1	33
	AntUnit	1	9
	buildP	30	33.021
	Documentation	148	131.165
	Imp	1,025	234.371
	Lib	3	71.913
	opt	388	219.628
	Wrapperscripts	25	98.84
SWT	SWT	10,833	713.405
UI	UI	7,734	123.029
AspectJ	AJBrowser	24	51.541
	AJDoc	47	119.829
	Ant	26	84.923
	buildP	95	1
	Compiler	2,499	95.328
	Docs	63	61.174
	IDE	79	64.227
	Lib	20	1
	LTWeaving	284	99.232
	Runtime	33	389.060
	Testing	1	3
Birt	buildP	572	1
	Chart	1,772	46.838
	Converters	2	51.5
	DataAccess	1,366	44.144
	DataAccessAccess	250	61.608
	Documentation	99	1
	Report	689	30.179
	ReportDesigner	2,785	40.664
	ReportEngine	1471	54.834
	ReportViewer	222	51.333
	TestSuite	38	10.552

Results and Discussion. As shown in Table 3, we have observed that the SWT component takes more time to solve that is around 700 days, while bugs in testing tend to take the least time that is three days. It is also observed generally that documentation and testing types of components take less time that is, on average less than 100 days while run time error may take around a year or less to resolve.

RQ3: *How are bugs distributed over different platforms and what is the* fixing *time for platform?*

Motivation. There is a number of bugs associated with the system, but in most cases, the bugs belong to a single platform or operating system. The research question is posed to find out whether bugs are found according to some operating system and whether the resolution time between different platforms vary.

Table 4. Distribution of bugs over different platforms and their fixing time

Platform	# of Bugs	Avg Fixing Time
Windows	17,839	222.56
All	6,584	612.82
Macintosh	2718	109.94
Linux	5,350	256.79
ML	134	253.26

Approach. [11], [14],[13] in their research, just observed the platforms as one of the parameters in their research for bug evaluation, but there is a lack of study for how the bugs are distributed over different platforms. In order to evaluate this, we classified the bugs into five different platforms from multiple levels of reported platforms as we considered that different versions of Windows to be a single platform as well as for other operating systems.

Results and Discussion. As shown in Table 4, the average fixing time is calculated in days. It is observed that most bugs are related to Windows, while second in the list are bugs related to all platforms. This is related to the fact that most of the development for the open-source system is done over Linux, and their movement to other platforms might raise compatibility issues. It is also seen that in order to fix single platform related issues takes less time than making the bugs fixed for all platforms. It is clear to understand that cross-platform issues require more effort and testing than on a single platform. The last time to fix the issues is in Macintosh that is around 110 days.

RQ4: *How is priority of bugs related to fixing time?*

Motivation. Priority of bugs classifies the severity of bugs. It is a general trend that the highest priority bugs are solved first. This research focuses on finding the same holds for open source systems or not.

Approach. This research question is vital in finding out how the priority of bugs helps in fixing the time of bugs. It can tell about the average fixing time of bugs according to each priority and can help the automated system in predicting the time for each bug [12].

[18] focused on looking at the priority of bugs by looking at the impact of severity and priority. The priorities of bugs have five levels, and severity in seven-plus categories were critical, and significant are topmost while Enhancement is lowest. We look into the priority and severity together, where we made eleven categories. 10 is the highest severity and priority that is Priority is P5, and severity is critical and significant. We also look for the number of days for fixing each bug and take the average.

Results and Discussion. As shown in Table 5, we calculated the fixing time in terms of no of days. It is shown that the highest priority bugs are fixed earlier where they are fixed in less than 75 days, while one of the outliers is level 8 bugs that take maximum time around 3200 days, but it is because of the compiler as a component and compiler tasks are mostly time taking. Generally, the least priority bugs take above 1800 days

as they are mostly cosmetic changes or adding the notes in the documentation. The number of days to fix level 3 priority bugs is the lowest around 22 days. That is because the component to be done is a report designer, and there are three bugs like that.

Table 5. Bug solving time with respect to bug priority and severity

Priority Level	# of Bugs	Average Fixing Time
10	200	74.92
9	167	38.29
8	636	3198.47
7	796	89.65
6	1,742	411.35
5	4,020	97.38
4	23,925	197.31
3	3	22
2	973	1535.23
1	61	1739.80
0	102	1,879.10

5. DISCUSSION AND OBSERVATIONS

We have studied five open-source projects and focused on four research questions defined in research objectives. It is observed that there is a strong relation between bugs and components that are mostly committers tend to solve only single or two types of components that define the expertise of committers and developers. This study can also help in the auto-classification of bug allocation to the right kind of committers instead of manual allocation. Our research study also gives an idea that how long a bug of a specific type may stay pending and how long a committer will take to solve the bug. Usually, the technical bugs take more time to solve, and so are the experienced committers who tend to take more time to solve and bug and do commit. Platforms also tend to play an important role, and our study reveals that Macintosh bugs are quickly solved while most number of bugs appears in windows and least in Linux. Component also decides that how long a bug is going to take to solve like a cosmetic and documentation change will require less time, and core changes or compilation changes will require the most amount of time to resolve the bugs.

This research provides a guideline to design an efficient bug allocation and fixing time prediction based on several factors. It will allow an automated allocation to the right committer and will inform the reporter about the potential time taken to solve the system. It will also help in the allocation of priority levels according to component type, automatically removing the steps for manual allocation.

6. RELATED WORK

In this section, we introduce some related literature on bug repositories, qualitative analysis, bug management, and empirical studies.

6.1. Bug Repositories and Qualitative Analysis

Bertram et al. [15] has qualitatively researched the use of bug tracking systems as correspondence and coordinated medium effort among customers, project managers, quality assurance employees, and software engineers. From surveys and interviews, they observed bug trackers to be a consistently developing learning store where every partner contributes information and data by means of individual bugs and features.

Zibran et al. [16] also qualitatively studied bug reports from multiple open-source projects to identify and rank usability bugs detected with structuring and developing software APIs. Guo et al. qualitatively investigated Microsoft Windows Vista OS project bug reports to find the vital reasons behind [17] bug reassignment. They used a qualitative bug reporting study to support their Microsoft employee survey results. Andrew et al. [18] conducted a qualitative study of 100 bug reports from three unique open-source projects to find out how distributed developer teams discuss bug reports and reach an agreement.

Bettenburg et al. [19], [20] conducted a questionnaire on engineers, and reporters link with bug repositories to discover the characteristics of a decent bug report. The engineers and reporters acknowledged and placed unique data types in bug reports, i.e., steps to reproduce, test cases, and stack traces in their analysis. We attempted to use this understanding in our research to know about bug results and bug fixing process.

6.2. Bug Management and Empirical studies

Bug management systems gather substantial knowledge about software projects [21], including historical bug resolutions, buggy components, experienced developers, historical software bugs, etc. For example, one of these bug management systems, the Eclipse Bugzilla, has collected, For instance, over 485,000 bug reports [22] have been gathered by one of these bug management schemes, the Eclipse Bugzilla. Bug reports are the primary drivers for maintaining and sustaining a [23] system's long-term running. For example, developers' usual method of fixing freshly reported bugs is to look for comparable alternatives in historical bug reports [23].

Bug reports have been essential in bug fixing, because of their nature as a source of lots of details such as description, reproducible steps, stack traces that will ease the developer life in locating and fixing the bug [24].

In order to assist developers in managing bug reports, different research studies were done to recommend bug assignee [25], locate bugs [26], detect duplicate bugs [27], ranking the dependent bugs [28] and comparison of mobile platforms bugs. However, it is not obvious if these techniques are widely accepted and adopted in the industry.

7. THREATS AND VALIDITY

Selection bias. We only selected the open-source project to experiment. Thus we did not include closed-source project due to the data are not publicly available. All the bug reports of the five projects are hosted only on Bugzilla and GitHub for the commits; we did not choose projects that hosted on other bug trackers such as Jire or Trac because their commits are not shared publicly. All the five projects are developed on Java language; we did not contain projects that developed on other projects.

Internal Validity. In this study, the data was gathered from Bugzilla for the different projects. For the computation of fixing- time of the issue, where the issue marked as fixed or resolved at the time of gathering the data. In any case, this may incidentally be skewed later on if and when any of these issues might be changed the status to re-opened.

Data processing. Despite the fact that all the dataset parts were recovered depending on open-source settings, it is conceivable that during the data crawling process, we might miss a few bugs' reports. In addition, the coordination procedure relies

fundamentally on the commit(s) between bugs and their associated fixes. We mostly classified the bug reports manually where the error may occur due to mis-classification of bugs type or priority.

8. CONCLUSIONS

This research focuses on finding out the major software bugs that can happen in open-source systems. In order to test one of the most widely used, five open source systems were selected. Software bugs evolve over time, and it needs to be identified that how they evolve over time, what are the most common components in the software system that have bugs, and what is the average time to solve each bug. This research addresses all mentioned research issues and found out that in each project, different kinds of components have bugs. In contrast, mostly, the bugs are solved either by a single or two committers; thus, an automated system can be built for auto allocation of specific bug type to a specific committer to avoid ticket hopping, and it will decrease average time to solve any bug. Mostly the bugs appear in a single kind of operating system out of them windows bugs are most. This can also help in predicting the average time to solve bugs according to the operating system type. This research provides software developers and security analysts an idea about the components that need more consideration to avoid the same kind of issues in the future. It also gives an idea to bug reporters that how long it can take to solve the reported issues.

9. REFERENCES

[1] Abrahamsson, Pekka, Outi Salo, Jussi Ronkainen, and Juhani Warsta. Agile Software Development Methods: Review and Analysis. ArXiv Preprint ArXiv:1709.08439, 2017.

[2] Ant, A. (2004). Apache software foundation. URL: http://ant. apache. org/manual/index. html.

[3] Northover, Steve, and Mike Wilson. Swt: The Standard Widget Toolkit, Volume 1. Addison-Wesley Professional, 2004.

[4] Redko, A., & Fedortsova, I. (2014). Working with JavaFX UI Componets. Release 8. E47848-02, Oracle, Java Platform, Standard Edition (Java SE) 8.

[5] Kiczales, Gregor, Erik Hilsdale, Jim Hugunin, Mik Kersten, Jeffrey Palm, and William G. Griswold. Getting Started with ASPECTJ. Communications of the ACM 44, no. 10 (2001): 59-65.

[6] Bondur, Tom, and Jason Weathersby. "BIRT: Building Next Generation BI." The Open Source Business Resource, 2009, 32.

[7] Nizamani, Zeeshan Ahmed, Hui Liu, David Matthew Chen, and Zhendong Niu. "Automatic Approval Prediction for Software Enhancement Requests." Automated Software Engineering 25, no. 2 (2018): 347-381.

[8] Wu, Hongrun, Haiyang Liu, and Yutao Ma. Empirical Study on Developer Factors Affecting Tossing Path Length of Bug Reports. IET Software 12, no. 3 (2018): 258-270.

[9] Murphy, G., and D. Cubranic. Automatic Bug Triage Using Text Categorization. In Proceedings of the Sixteenth International Conference on Software Engineering & Knowledge Engineering. Citeseer, 2004.

[10] Anvik, John, Lyndon Hiew, and Gail C. Murphy. Who Should Fix This Bug? In Proceedings of the 28th International Conference on Software Engineering, 361-370. ACM, 2006.

[11] Aljedaani, Wajdi, and Yasir Javed. Bug Reports Evolution in Open Source Systems. In 5th International Symposium on Data Mining Applications, 63-73. Springer, 2018.

[12] Javed, Yasir, and Mamdouh Alenezi. Defectiveness Evolution in Open Source Software Systems. Procedia Computer Science 82 (2016): 107-114.

[13] Bernardi, Mario Luca, Gerardo Canfora, Giuseppe A. Di Lucca, Massimiliano Di Penta, and Damiano Distante. The Relation between Developers Communication and Fix-Inducing Changes: An Empirical Study. Journal of Systems and Software 140 (2018): 111-125.

[14] Zheng, Wei, Chen Feng, Tingting Yu, Xibing Yang, and Xiaoxue Wu. Towards Understanding Bugs in An Open Source Cloud Management Stack: An Empirical Study of OpenStack Software Bugs. Journal of Systems and Software, 2019.

[15] D. Bertram, A. Voida, S. Greenberg, and R. Walker, "Communication, collaboration, and bugs: the social nature of issue tracking in small, collocated teams," in Proc. of the 2010 ACM conference on Computer supported cooperative work, ser. CSCW '10, 2010, pp. 291-300.

[16] M. Zibran, F. Eishita, and C. Roy, "Useful, but usable? factors affecting the usability of APIs," in Reverse Engineering (WCRE), 2011 18th Working Conference on, oct. 2011, pp. 151-155.

[17] P. J. Guo, T. Zimmermann, N. Nagappan, and B. Murphy, "Not my bug!âĂİ and other reasons for software bug report reassignments," in Proc. of the ACM 2011 conference on Computer supported cooperative work, ser. CSCW '11, 2011, pp. 395-404.

[18] A. J. Ko and P. K. Chilana, "Design, discussion, and dissent in open bug reports," in Proc. of the 2011 iConference, ser. iConference '11, 2011, pp. 106-113.

[19] N. Bettenburg, S. Just, A. Schroter, C. Weiss, R. Premraj, and T. Zimmermann,

[20] "What makes a good bug report?" in Proc. of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering, ser. SIGSOFT '08/FSE-16, 2008, pp. 308-318.

[21] N. Bettenburg, S. Just, A. Schroter, C. WeiÃ§, R. Premraj, and T. Zimmermann, "Quality of bug reports in Eclipse," in Proc. of the 2007 OOPSLA workshop on eclipse technology eXchange, ser. eclipse '07, 2007, pp. 21-25.

[22] Xia, Xin, David Lo, Emad Shihab, and Xinyu Wang. "Automated Bug Report Field Reassignment and Refinement Prediction." IEEE Transactions on Reliability 65, no. 3 (2016): 1094-1113.

[23] Li, Xiaochen, He Jiang, Dong Liu, Zhilei Ren, and Ge Li. Unsupervised Deep Bug Report Summarization. In Proceedings of the 26th Conference on Program Comprehension, 144-155. ACM, 2018.

[24] Xuan, Jifeng, He Jiang, Yan Hu, Zhilei Ren, Weiqin Zou, Zhongxuan Luo, and Xindong Wu. Towards Effective Bug Triage with Software Data Reduction Techniques. IEEE Transactions on Knowledge and Data Engineering 27, no. 1 (2015): 264-280.

[25] Zou, Weiqin, David Lo, Zhenyu Chen, Xin Xia, Yang Feng, and Baowen Xu. How Practitioners Perceive Automated Bug Report Management Techniques. IEEE Transactions on Software Engineering, 2018.

[26] Alenezi, Mamdouh, Kenneth Magel, and Shadi Banitaan. Efficient Bug Triaging Using Text Mining. Journal of Software 8, no. 9 (2013): 2185-2191.

[27] Le, Tien-Duy B., Richard J. Oentaryo, and David Lo. Information Retrieval and Spectrum Based Bug Localization: Better Together. In Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, 579-590. ACM, 2015.

[28] Safdari, Nasir, Hussein Alrubaye, Wajdi Aljedaani, Bladimir Baez Baez, An- drew DiStasi, and Mohamed Wiem Mkaouer. "Learning to Rank Faulty Source Files for Dependent Bug Reports." In Big Data: Learning, Analytics, and Appli- cations, 10989:109890B. International Society for Optics and Photonics, 2019. https://doi.org/10.1117/12.2519226.

[29] Aljedaani, Wajdi, Meiyappan Nagappan, Bram Adams, and Michael Godfrey. "A Comparison of Bugs Across the IOS and Android Platforms of Two Open Source Cross Platform Browser Apps." In 2019 IEEE/ACM 6th International.