LEVERAGING Q&A PLATFORMS TO IMPROVE ISSUE MANAGEMENT IN SOFTWARE PROJECTS

by

AADITYA BHATIA

A thesis submitted to the

School of Computing

in conformity with the requirements for

the degree of Master of Science

Queen's University

Kingston, Ontario, Canada

September 2019

Abstract

TRADITIONAL issue management systems like Bugzilla are widely used in open source and commercial projects. Stack Exchange uses its online question and answer (Q&A) platform to manage bugs, which brings several new unique features that are not offered in traditional bug management systems. In this thesis, we divide our study into the Q&A features that are being used in the Stack Exchange issue management system, as well as study the differences between bug reports and feature requests that have been refined due to community collaborations. For the former part, we study the unique features of Stack Exchange Q&A platform that allows users to directly edit a bug report (i.e., the in-place editing feature) instead of commenting about the bug report; use different communication channels (i.e., the answering and commenting features) to discuss reported bugs, and vote on those bug reports, answers, and their associated comments (i.e., the voting feature). We study how these unique features are used to manage bugs, and provide insights to the designers of traditional

bug management systems who are considering introducing such features in their bug management system.

The later part of this thesis performs a study of the differences between bug reports and feature requests in the Stack Exchange issue management system. We perform this study because: 1) Stack Exchange contains a large number of issues that have been carefully tagged as bug reports versus feature requests 2) the issue management is carried out through a Q&A platform, which provides us with a richer perspective on the differences between the management of bug reports and feature requests. We found that bug reports and feature requests differ significantly from each other along many dimensions such as the amount of community participation, the content of the issues, and the characteristics of the participating users. We are able to automatically identify bug reports from feature requests with a median AUC of 0.90. The developers of issue management systems, software practitioners and researchers can leverage such understanding to improve issue management processes in large software projects.

Acknowledgments

I would like to show my sincere gratitude towards my supervisor, Prof. Ahmed E. Hassan, who has provided me with his immensely valuable guidance and support. My thesis would not have been possible without the motivation, direction, and feedback provided by Prof. Hassan. I would like to sincerely thank him for providing this opportunity and environment where I could learn and grow.

I would also like to express my deepest gratitude to thank the committee members — for finding the time in their busy schedules, and help me provide their valuable feedback, comments, and insights.

I truly thank my collaborators, Dr. Shaowei Wang, Dr. Muhammad Assaduzamad, and Gopi Krishnan Rajbahadur for their guidance throughout my research. I pay my deepest gratitude to Dr. Shaowei Wang, for teaching and guiding me throughout my thesis

and Dr. Muhammad Assaduzaman for providing valuable comments that greatly improved my research. I would like to acknowledge my special thanks and deepest gratitude to Gopi Kirshnan Rajbahadur, a mentor, and a true friend.

I am very fortunate to work with my lab-mates, Daniel Lee, Mohamed Ibrahim Hasan, Abdullah Ahmad Zarir, Kundi Yao and Shahnaz Shariff who are some of the most hard-working people that I have ever known. I would like to thank the postdoctoral researchers (Dr. Gustavo Ansaldi Oliva, Dr. Safwat Mohamed Ibrahim Hassan, and Dr. Mohammed Sayagh) and PhDs (Muhammad Ahasanuzzaman, Jiayuan Zhou, and Filipe Roseiro Cogo) in our lab, for providing an amazing work environment. I would like to dedicate this work to my father (Manish Bhatia) and mother (Vinita Bhatia) and would like to thank my maternal grandmother (Krishna Malik) who showered me with her love and provided motivation at the darkest times. I would also like to thank my uncles and aunts for their help and support. Special thanks to my aunt, Mangala Malik for helping me throughout my masters. This work would not have been possible without my friends and family, who always helped me refine myself, and strive to become a better person in life.

Table of Contents

List of Tables

List of Figures

CHAPTER 1

Introduction

ISSUE management systems are a crucial tool in software projects. These systems contain a wealth of information about the various aspects of a project and their progress. Hence, the management of issue reports can impact the quality of software products. In the past, rich bug datasets have been used to develop methods to avoid the occurrence of future bugs (D'Ambros et al., 2010; Bhattacharya and Neamtiu, 2011; Kamei et al., 2010), to identify the most suitable developer to work on a new issue (Anvik et al., 2006; Mani et al., 2019; Xuan et al., 2012; Nguyen et al., 2014), and to determine the needed time to address an issue (Weiss et al., 2007; Giger et al., 2010; Marks et al., 2011).

Several challenges are associated with bug management (Just et al., 2008). For instance, in traditional bug management systems, such as Bugzilla, any changes to the bug report (e.g., adding new information, making clarifications, and editing incorrect information) are made via the process of sequential commenting, which can lead to very long comment threads. Finding useful information in such a long comment thread is a challenging task (Arya et al., 2019). Moreover, when reporting a bug, users are required to include the erroneous program behavior, and steps to reproduce the bug. However, Bettenburg et al. (2007) observed that reporters do not provide adequate information in their bug reports, which has proven to be a challenge for software developers.

Nowadays, we observe a trend in using online Question and Answer (Q&A) websites to collect and manage such bug reports. For instance, Stack Exchange is a network which is composed of 173 question and answering (Q&A) websites. It is designed to maximize community contributions and collaboration by applying gamification concepts using its question and answer platforms (Mamykina et al., 2011). Unlike traditional bug management systems, the bug management process of Stack Exchange is performed on its Q&A platform, namely, Meta Stack Exchange[1]. In this thesis, we study the Q&A features that are being used in the Stack Exchange issue management system, as well as study the differences between bug reports and feature requests that have been refined through community collaborations on Q&A platforms.

[1] https://meta.stackexchange.com/

1.1 Thesis Statement

The Stack Exchange issue management system uses a Q&A platform to manage issues (i.e., bugs and features), offering a set of unique features that are not available in traditional issue management systems. Understanding such features can help the designers of traditional issue management systems to decide whether they should integrate such features. Moreover, the Stack Exchange issue management system contains a rich set of manually tagged bug reports and feature-requests offering us a unique opportunity to understand the differences between the management of bugs and features at a large scale over prior studies which had to manually tag such data using non-experts.

1.2 Thesis Overview

In this section, we provide an outline of our thesis.

1.2.1 Chapter 2: Related Work

In this chapter, we perform a survey of the related work in issue reporting and management. We survey the related works along four directions; namely; studying and improving the issue-reporting process, leveraging crowdsourcing to support issue management activities; studying Technical Q&A websites, and automatic classification of bug reports and feature requests.

1.2.2 Chapter 3: Background

In this chapter we discuss the background of Stack Exchange question and answering processes in issue management.

1.2.3 Chapter 4: A Study of Bug Management Using the Stack Exchange Question and Answering Platform

Stack Exchange uses its online Q&A platform to collect and manage bugs, which brings several new unique features that are not offered in traditional bug management systems. Users can directly edit a bug report (i.e., the in-place editing feature) instead of commenting about the bug report; users can also use different communication channels (i.e., the answering and commenting features) to discuss the reported bugs. Moreover, users can vote on bug reports, answers, and their associated comments (i.e., the voting feature).

In this chapter, we study how these three unique Stack Exchange features can provide insights to the designers of the traditional bug management systems, like Bugzilla, who might wish to add these features into their systems. In particular, we study (1) how users leverage the in-place editing feature in the bug management system of Stack Exchange, (2) how users leverage the answering and commenting features in the bug management process of Stack Exchange, and (3) how users leverage the voting feature in the bug management system of Stack Exchange. We observe that the in-place editing feature is used steadily on bug reports over the course of years, and 57% of the in-place edits were performed to improve the quality of bug reports. Then, we observe that commenting is used on 76% of the bug reports, which is more frequent than answering (60%). This shows that the usages of commenting and answering differ from

each other. Commenting provides a channel for discussing bug related information, whereas answering provides a channel for including causes of bugs and bug-fix information. Lastly, we observe that upvoting is utilized more frequently on bug reports and answers than comments over the course of years. Most downvotes in bug reports were due to the disagreement of the reported "bug" being either a real bug or being of low quality.

1.2.4 Chapter 5: An Exploratory Study of the Differences Between the Management of Bugs and Features Using the Stack Exchange Question and Answering Platform

In a bug management system, intuitively, one would expect that feature requests and bug reports (and their management) would differ. Yet, no prior study has ever explored such differences systematically since there exist no large enough datasets where issues are tagged correctly as bug reports versus feature requests. In this chapter, we leverage the Stack Exchange Q&A platform to study the differences between the reported bugs and feature requests. Our study offers a rich empirical understanding of the differences between bug reports and feature requests (and their management) and demonstrates the the usefulness adopting Q&A aspects to the traditional issue management systems. In particular, we study the differences between bug reports and feature requests that are refined through community collaborations by understanding how the management of bug reports differ from the management of feature requests and if we can automatically identify bug reports from feature requests. We observe that:

1. Feature requests are longer than bug reports, and receive longer comments and answers. Feature requests also receive more contributions through comments, answers, and votes in contrast to bug reports. Feature request comments exhibit more positive sentiment than those for bug reports. In addition, users who report and contribute to bug reports have higher reputation than those for feature requests.

2. The Q&A specific features (e.g., editing and answering versus simple commenting) by themselves—without using the text of the issue reports—identify bug reports from feature requests with a median AUC of 78%. Furthermore, Q&A factors along with the sentiment and the text of the issue report enable an automated classification of bug reports and features requests with a median AUC of 89.8%. The expressed sentiment in the issue report and the Q&A characteristics of the issue reports are the most influential factors in identifying bug reports from feature requests.

1.2.5 Thesis Contributions

This thesis is focused towards helping the users and designers of issue management systems by providing insights to Q&A based issue management. We study the Stack Exchange issue management system, where a reward-based Q&A mechanism is in place to encourage heavy community participation. The two key points of this thesis are:

This thesis is the first work to study the use of Q&A platforms for issue management. Our study will help designers and users of traditional issue management system.

This thesis is the first work to explain the differences between managing bug reports and feature requests at a large scale using data that has been curated by domain experts.

CHAPTER 2

Related Work

In this chapter, we discuss prior studies that are related to our study.

2.1 Studying and Improving the Issue Reporting Process

Several studies have been conducted to understand the challenges associated with managing issue reports. Just et al. (2008) performed a study to understand the reported issues in open source projects. (e.g., missing crucial issue information) and provided some suggestions (e.g., providing tool support for users to collect and prepare the information that developers need) to fix issues for next-generation issue management systems. Bettenburg et al. (2007) interviewed Eclipse developers and noted that the

steps to reproduce an issue was highly required by developers and that inaccurate information hinders the resolution of issues. The authors also revealed a mismatch in the information provided by reporters and information required by developers to fix issues.

Furthermore, a considerable number of prior studies have been done to study and improve the quality of issue reports. Dal Sasso et al. (2016) performed a large-scale study on 650,000 issue reports to understand which components of an issue report have important impacts on the resolution time of the issue report and they found that some core elements (e.g., screenshot and the stack trace) of an issue report do impact the resolution time. Hooimeijer and Weimer (2007) modeled the quality of the issue report by analyzing their features. Surprisingly, the self-reported severity of the issue report is not a reliable indicator of the importance of the issue. To improve issue descriptions quality by alerting reporters about the missing information when reporting an issue, Chaparro et al. (2017) designed an automated approach to detect the absence (or presence) of observed behavior and the steps to reproduce an issue report. Rejaul (2019) proposed an approach to predict the missed key features (e.g., expected behavior) in an issue report. Tian et al. (2016) studied the severity of issue reports and proposed an approach to mitigate unreliable factors that lead to false assessment of issue severity.

While prior studies focus on improving issues by studying which types of information are important and how to detect those information automatically, this thesis investigates the impact of new features (e.g., in-place editing) in issue management processes.

2.2 Leveraging Crowdsourcing to Support Issue Management Activities

Crowdsourcing could be leveraged to improve the issue management process, such as issue reporting, issue triaging, and issue fixing. Breu et al. (2010) studied 600 issue reports, finding that interacting with developers provides help in fixing issues in term of shortening the resolution time. Zhou and Mockus (2015) found that users involved in development activities, like issue reporting and participating in the community, are more likely to become long-time contributors. Badashian et al. (2016) proposed issue triaging and assignment approaches by leveraging developers' contribution to Q&A websites (i.e., Stack Overflow) to estimate their expertise in particular domain. Liu and Zhong (2018) developed an approach to leverage the crowdsourced knowledge of Stack Overflow to repair issues automatically. Jiang et al. (2019) constructed crowdsourcing elicited attributes for bug reports. The authors validate the effectiveness of the crowdsourced attributes using logistic regression. Different from prior studies which leverage crowdsourcing to support issue management activities, we investigate the use of three unique features (e.g., in-place editing, answering and commenting, and voting) in Q&A style issue management systems and provide insights for traditional issue management systems.

2.3 Studying Technical Q&A Websites

Technical Q&A websites like Stack Overflow have accumulated a large volumes of knowledge leading to huge impacts for developers and software engineering researchers.

Various studies have studied how to ensure the quality of knowledge of such Q&A websites. A considerable number of studies have been done to study and improve the quality of content on such Q&A websites. Anderson et al. (2012) performed an empirical analysis of the properties of communities participating in various activities on Stack Overflow Q&A platform. The authors collected several Q&A factors of Stack Overflow questions, and used these factors to predict the long-term impact of a question and whether a question has been adequately answered. Zhang et al. (2019) analyzed obsolete answers on Stack Overflow and found that more answers are getting obsolete, and obsolete answers are poorly maintained. To combat this, they provide actionable suggestions for Stack Overflow. Wang et al. (2018) performed a large-scale study on the revisions of answers on Stack Exchange and reveals several flaws of the current incentive system (e.g., users are gaming the system get more badges). Fischer et al. (2017); Ragkhitwetsagul et al. (2019); Meng et al. (2018) have shown that the code snippets are not insecure and users should pay additional caution when reusing them In addition, researchers conducted studies on such Q&A websites to extract software engineering related knowledge, e.g., code search and code reuse was studied by Sirres et al. (2018); Rahman and Roy (2018); Wu et al. (2018), comment generation was studied by Wong et al. (2013) and developer discussions were studied by Barua et al. (2014). Huna et al. (2016) demonstrate that different users provide different quality of the content at Stack Exchange and not all contributors ensure good quality collaborations. Asaduzzaman et al. (2013) provide insights on the reasons for open questions at Stack Overflow. Zhang et al. (2019) perform a study of information content provided in Stack Overflow answers, that becomes obsolete and becomes in misleading. On a similar ground, Chua and Banerjee (2015) perform a study of unanswered questions,

and attribute the unanswered questions to asker popularity, participation, and asking time of the questions. Novielli et al. (2014) perform a psychometric analysis of the role of emotions in community collaboration in Stack Overflow.

Different from existing studies which focus on studying the quality of technical Q&A websites and leveraging the knowledge from technical Q&A websites, this thesis provides insights into how to leverage the features of technical Q&A platforms to improve traditional issue management system.

2.3.1 Automatic classification of bug reports and feature requests

Antoniol et al. (2008a) used text based features to identify bug reports from feature requests. To do so, the authors compared logistic regression and naive Bayes classifiers against decision trees. Whereas, Maalej and Nabil (2015) classified reviews on application stores as a bug report, a feature request, a praise, user experience or a rating. The authors used sentiment analysis on text, along with the metadata of the rating. **?** automated the classification of bug reports and feature requests using a multi-stage approach that combined factors that were obtained from text mining and other issue report factors. Finally, Pingclasai et al. (2013) developed a technique to classify bug reports based on topic modeling. Although, prior researches extract features from textual analysis, this thesis differs from these prior studies on two important aspects. First, none of the prior studies used sentiment factors nor Q&A factors to identify bug reports from feature requests. Second, all the prior studies, use small datasets that are tagged by researchers. However, in this thesis, we use a dataset of 9,048 issue reports which have been curated by the domain experts.

CHAPTER 3

Background

STACK Exchange provides a Q&A platform for its users to share knowledge across various domains (e.g., programming, statistics, and mathematics). One unique feature of Stack Exchange is that it uses one of its Q&A platform, namely, Meta Stack Exchange[1] to manage its issues. In this section, we introduce the issue management on Stack Exchange along the following two aspects: issue reporting at Stack Exchange, and community contributions to those issue reports. In the latter part, we describe how community users edit, answer, comment, and vote on issue reports.

[1] https://meta.stackexchange.com/

3.1 Issue Reporting at Stack Exchange

Users are encouraged to report issues about Stack Exchange on the Meta Stack Exchange[2]. Issue reports are treated as a regular question in the Meta Stack Exchange, and are managed through the same Q&A platform. All bug reports and feature-requests are labeled with the "bug" and "feature-request" tags respectively. Thus, we consider all questions with the "bug" and "feature-request" tags as issue reports for our study. From here onwards, we refer a Meta Stack Exchange question with the tag "bug" as a bug report and "feature-request" as feature-request if not otherwise mentioned.

Stack Exchange expects its users to provide a title, body, and tags for each reported issue. There is a limit of 150 and 30,000 characters for the title and the body of an issue report, respectively. The title briefly describes the reported issue and the body provides a detailed description of the issue. Besides the tags "bug" and "feature-request", users are also encouraged to label an issue report with other tags to better organize the issues. All issue reports are open to the whole community to view, edit, vote on, leave comments, and provide answers. Figure 3.1 shows an example of a bug report[3]. The bug is about a visual design issue (i.e., UI). Besides being tagged with "bug", the bug report is also tagged with "stackexchange.com", indicating that all sites in the network are affected by the bug. The bug report also includes the reporter information, enabling any member of the community to visit the profile of the reporter.

On Stack Exchange, moderators are users with special privileges. Such moderators maintain the website content and guide other users in the various community activities. Moderators have the unique ability to change the status of an issue report to

[2] https://meta.stackexchange.com/
[3] https://meta.stackexchange.com/questions/299290

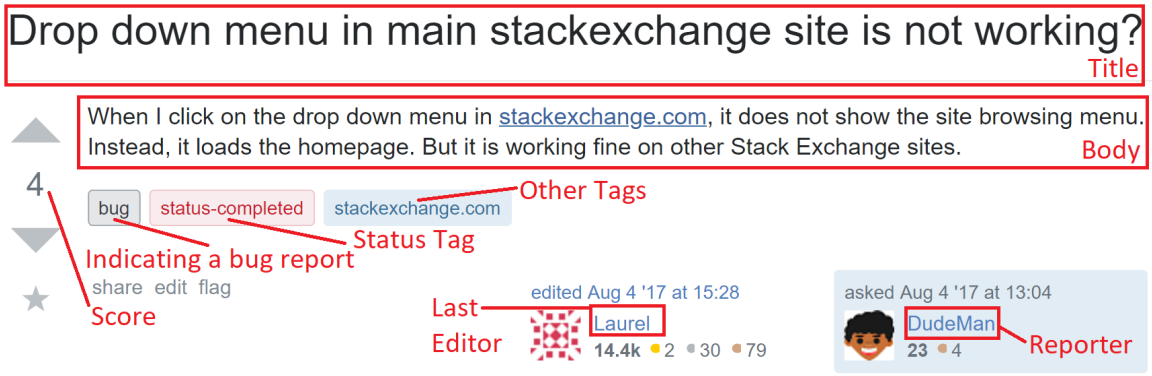

Figure 3.1: An example of a reported bug on the Meta Stack Exchange website.

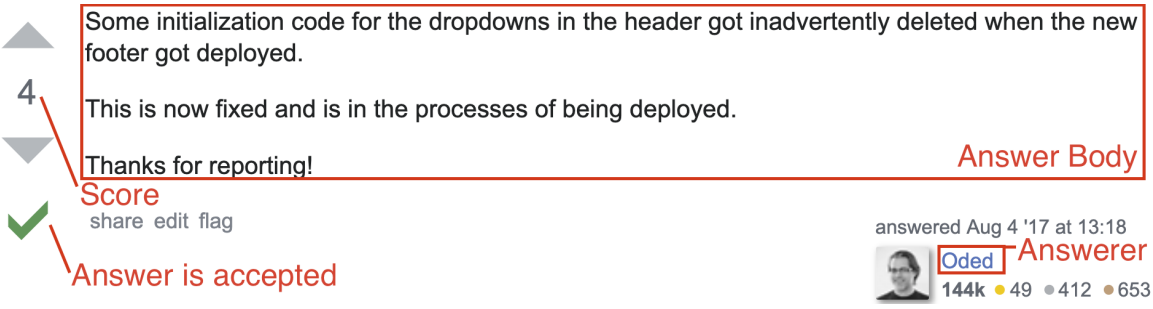

Figure 3.2: An example of a developer's response in the form of an answer to the bug report shown in Figure 3.1. The developer explained the cause of the bug and indicated that the bug had been fixed.

Status Tag	Description
status-completed	The bug has been fixed and deployed.
status-declined	The bug will not be fixed.
status-bydesign	The bug reports refers to a feature that is misunderstood to be a bug.
status-norepo	The site developers were not able to replicate the reported bug.
status-deferred	The bug is intended to be fixed, but not in the near future.
status-planned	The bug is intended to be implemented, ideally in the near future.
status-review	The report contains merit to consider, but requires further investigation. A decision on its decline or approval requires additional investigation.
status-reproduced	Indicates that the site developers were able to replicate the buggy behavior, but are not yet addressing the cause at this time.

Table 3.1: Different categories of status tags and their description.

indicate its fixing status. There are seven different "status" tags that can be applied on an issue report. Table 3.1 presents a detailed description for each of these "status" tags. For example, the status of the bug report, as shown in Figure 3.1, is "status-completed", indicating that the bug has been fixed and the fix has been deployed.

3.2 Community Contributions to Issue Reports

The Stack Exchange community is encouraged to make contributions to an issue report through the posting of answers and comments on issue reports as well as their associated answers. Users can perform the following actions to participate in the management of issue on Stack Exchange:

In-place editing: Users can edit an issue report to fix grammar and spelling mistakes; clarify the description of the issue (without changing its original meaning); add

additional information (i.e., related links)[4]. We refer to the feature which enables users to directly edit an issue report as the *in-place editing feature*. In-place editing can be performed by all users of the community. However, edits from users with less than 2,000 reputation points must go through a review process before being applied. For example, Figure 3.3 shows the edit history of the issue that was previously shown in Figure 3.1. A developer edited the associated tags of the bug report, and performed grammatical changes in the third edit.

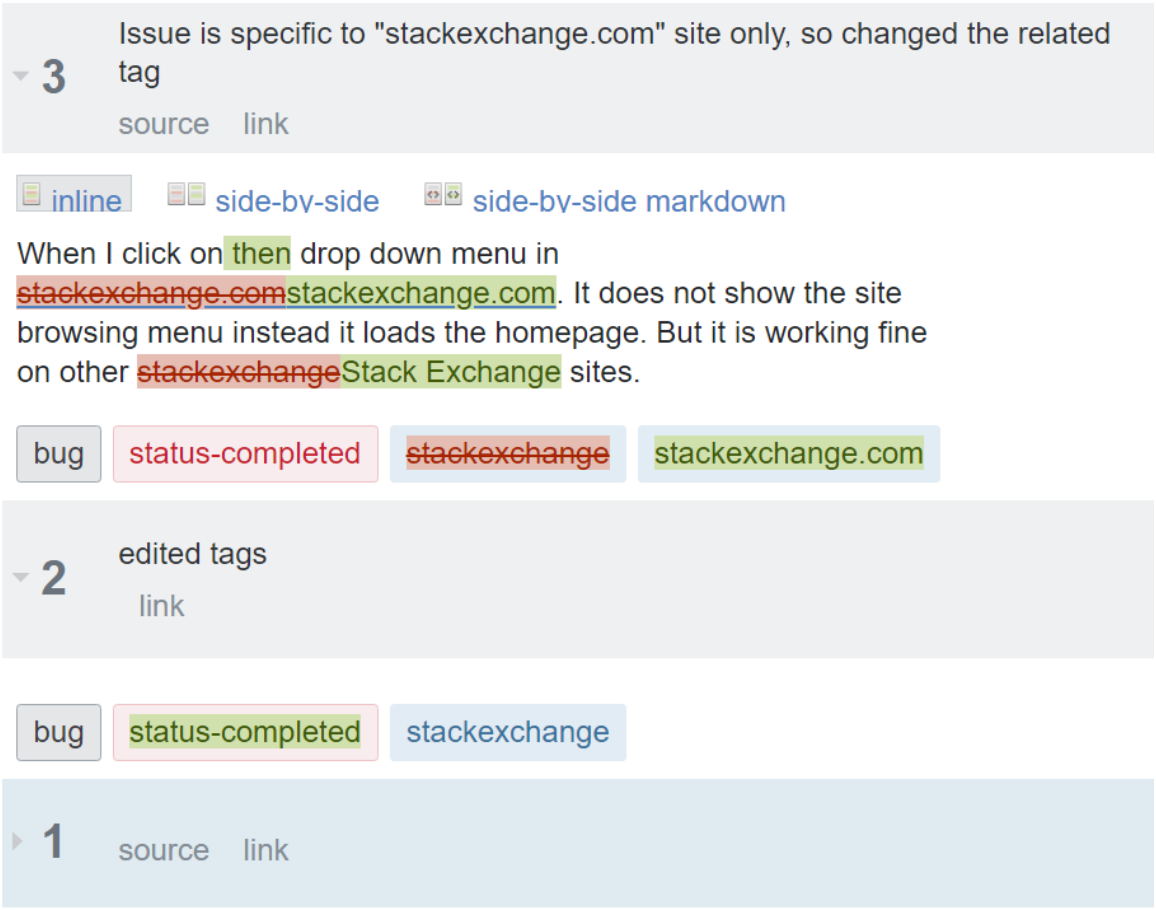

Figure 3.3: The edit history of a bug report. Note that the first version is the original bug report.

[4] https://meta.stackexchange.com/help/editing

Note that some edits can be automatically performed by the *community user*, which is a background process that helps keep the site clean[5]. In this study we only focus on the edits that are performed by humans, hence we remove all the edits of the *community user*.

In a traditional issue management system, the original issue report posted by a reporter cannot be edited. Any changes (e.g., adding new information and correcting wrong information) to the issue report is done through commenting, which can lead to an issue report with a long list of comments, making it difficult for developers to retrieve useful information from such a long thread of comments Arya et al. (2019). In-place editing could be a way to address this problem since **in-place editing enables the content to be located and polished in one place (i.e., within an issue report).** Therefore, in Section 5.3.1, we investigate how the in-place editing feature is used in the issue management system of Stack Exchange and provide insights to traditional issue management systems who might be considering integrating such a feature into their systems.

Answering and Commenting: All users are allowed to post answers to issue reports. Figure 3.2 shows an example of an answer to a bug report. If the reporter is satisfied with an answer that solves her/his question, then she/he can accept the answer by clicking the check mark beside the answer. As shown in Figure 3.2, the green tick indicates that the answer is accepted by the reporter. Commenting is available for both issue reports (i.e., referred to as a *issue-comment*) and answers (i.e., referred to as an *answer-comment*). Unlike answers, comments are only available to users who have more than 50 reputation points. However, issue reporters and answerers can comment

[5] https://meta.stackexchange.com/questions/19738/who-is-the-community-user

on their own posts without such a restriction. Users can only provide upvotes on comments (in contrast to both upvotes and downvotes on issue reports and their answers). The maximum limit of a comment is 600 characters[6]. We refer to an issue report and all its associated answers and their all associated comments as an *issue report thread*.

Unlike traditional issue management systems, which only allow sequential commenting, the Q&A platform enables its users to contribute to the issue management process through the "answering" of an issue report or through the "commenting" on an issue report or its associated answers. This offers two subtle differences: 1) users have different channels to contribute to discussions about issue (commenting or answering), and the appropriate channel is not clear from the Q&A oriented user interface (UI); 2) The ability to comment on an answer (*answer-comments*) enables threaded comments. These features are missing in current popular issue management systems, such as Bugzilla. In Section 5.3.2, we investigate how users leverage such commenting and answering features.

Voting: Stack Exchange allows users to vote on issue reports (i.e., *issue-vote*), answers (i.e., *answer-vote*), issue-comments (i.e., *issue-comment-vote*), and answer-comments (i.e., *answer-comment-vote*). Stack Overflow official documentation defines votes as follows: "votes reflect the perceived usefulness: well-written, well-reasoned, well-researched posts tend to get more attention and more upvotes."[7] An upvote leads to a gain of reputation points for the owner of the post (i.e., questions or answer), whereas a downvote leads to a loss of reputation points for both the owner of the post and the voter. The score of a post (an issue report or an answer) is calculated as follows: number of upvotes - number of downvotes, Unlike questions and answers, comments can only be

[6] https://meta.stackexchange.com/questions/71283
[7] https://stackoverflow.com/help/whats-meta

upvoted such votes do not lead to a gain or loss of reputation points. In Section 5.3.3, we investigate how voting is used on issue reports, their answers and comments.

CHAPTER 4

A Study of Bug Management Using the Stack Exchange Question and Answering Platform

TRADITIONAL bug management systems like Bugzilla are widely used in open source and commercial projects. Stack Exchange uses its online question and answer (Q&A) platform to collect and manage bugs, which brings several new unique features that are not offered in traditional bug management systems. For example, users can directly edit a bug report (i.e., the in-place editing feature) instead of commenting about the bug report; users can also use different communication channels (i.e., the answering and commenting features) to discuss reported bugs. Moreover, users can vote on those bug reports, answers, and their associated comments (i.e., the voting feature). Understanding how these unique features are used to

manage bugs can provide insights to the designers of traditional bug management systems, such as whether a feature should be introduced, and how would users leverage such a feature.

In this thesis, we performed a large-scale analysis of the bug management system of Stack Exchange by studying 191,51 bug reports, including their associated answers, comments, and edit histories. We studied three unique features, namely, the in-place editing feature, the answering and commenting features, and the voting feature. We find that: 1) All these three features are actively used by all stakeholders (i.e., reporters and other users). 2) 57% of the bug report edits were performed to improve the quality of bug reports, such as adding or correcting essential bug-related information (e.g., observed behavior, and environment information). 3) The answering and commenting features are used differently. While commenting on a bug report provides a channel for discussing bug related information, answering offers a channel for explain the causes of a bug and for providing bug-fix information. 4) The majority of the downvotes are due to the disagreement on whether the reported bug is a real bug or not, and the low quality of a bug report due to incomplete/incorrect/duplicated bug information. Based on our findings, we provide several suggestions for traditional bug management systems. For instance, traditional bug management systems should consider introducing the answering and commenting features to encourage the community to express different content through different channels (i.e., comments and answers).

4.1 Introduction

Collecting and managing bug reports is a crucial part of software development due to the prevalence of bugs. As a result, a number of traditional bug management systems,

like Bugzilla[1], are commonly used in practice. For instance, Bugzilla, which is a popular online bug tracking system, is used by many open source projects like Apache[2], Linux[3], Open Office[4], as well as a significant number of private organizations, such as IBM and NASA.

In traditional bug management systems, bugs are reported, discussed, and assigned to developers for fixing. However, there are several challenges associated with bug management (Just et al., 2008). For instance, in traditional bug management systems, like Bugzilla, any changes to the bug report (e.g., adding new information, making clarifications, and editing incorrect information) are made through sequential commenting, which can lead to very long comment threads. Finding useful information in such a long comment thread is a challenging task (Arya et al., 2019).

Unlike traditional bug management systems, the bug management process of Stack Exchange is performed on its Q&A platform, namely, Meta Stack Exchange[5]. The use of its platform for bug management results in several unique aspects in the bug management process. For example, Stack Exchange allows users to edit a bug report instead of performing sequential commenting (i.e., in-place editing feature). It also allows users to discuss bugs through two different communication channels, namely, comments and answers (i.e., answering and commenting features). Therefore, it is important to study how such unique features are used in Stack Exchange in the form of Q&A style. Understanding the use of such features can provide insights to the designers of traditional bug management systems, like Bugzilla, who might wish to integrate such features into their systems.

[1] https://www.bugzilla.org/
[2] https://bz.apache.org/bugzilla/
[3] https://bugzilla.kernel.org/
[4] https://bugs.documentfoundation.org/
[5] https://meta.stackexchange.com/

In this study, we performed a large-scale analysis of the bug management system of Stack Exchange by studying 19,151 bug reports, including 15,904 answers, 42,050 edits, 68,207 comments on bug reports, and 28,785 comments on answers. We studied three unique features, namely, the in-place editing feature, the answering and commenting features, and the voting feature. We structure our study along the following three research questions:

- **RQ1: How do users leverage the in-place editing feature of the Stack Exchange bug management system?**

  The in-place editing feature is used steadily over years. 57% of the in-place edits improved the quality of bug reports, such as adding or correcting essential bug-related information (e.g., observed behavior and environment information). The ability to rollback edits provides a mechanism to resolve conflicts or mistakes that arise during the in-place editing, something that is not possible in the sequential commenting thread of traditional bug management systems.

- **RQ2: How do users leverage the answering and commenting features of the Stack Exchange bug management system?**

  In general, commenting is used on 76% of the bug reports, while answering is used on 60% of the bug reports. The usages of commenting and answering differ from each other. Commenting provides a channel for discussing bug related information, whereas answering provides a channel for including the causes of a bug and bug-fix information.

- **RQ3: How do users leverage the voting feature of the Stack Exchange bug management system?**

  Upvoting is used more frequently on bug reports and answers than comments.

The use of downvoting has increased gradually over the years, with more downvotes on bug reports than on their associated answers. Most of the downvotes on bug reports are made due to disagreement about whether the reported "bug" is a real bug and the low quality of bug reports (i.e., reported bugs are incorrect, insignificant, incomplete and non-reproducible).

Based on our findings, we provide insights to the designers of the traditional bug management systems, like Bugzilla, who might wish to add these features into their systems. For instance, we suggest that traditional bug management systems should consider introducing the in-place editing feature. Traditional bug management systems also should consider introducing the answering and commenting features to encourage its users to better structure their contribution. Separating content into different channels can help finding the target information more easily.

Chapter Organization: The remainder of the chapter is organized as follows. Section 4.2 discusses our data collection process, Section 4.3.1 presents the motivation, approach, results of our three research questions. Section 4.4 discusses the implications of our findings and 4.5 describes threats to the validity of our observations. Finally, Section 4.6 concludes this chapter.

4.2 Data Collection

This section discusses how we collect the dataset that we used to answer our research questions.

For this study, we download a publicly available data dump of Meta Stack Exchange from archive.org[6]. The data dump contains all the site activities between June 28, 2019,

[6] https://archive.org/download/stackexchange

and March 3, 2019. The data dump consists of a collection of XML files containing information about questions, associated answers, post histories, post links, comments, and votes.

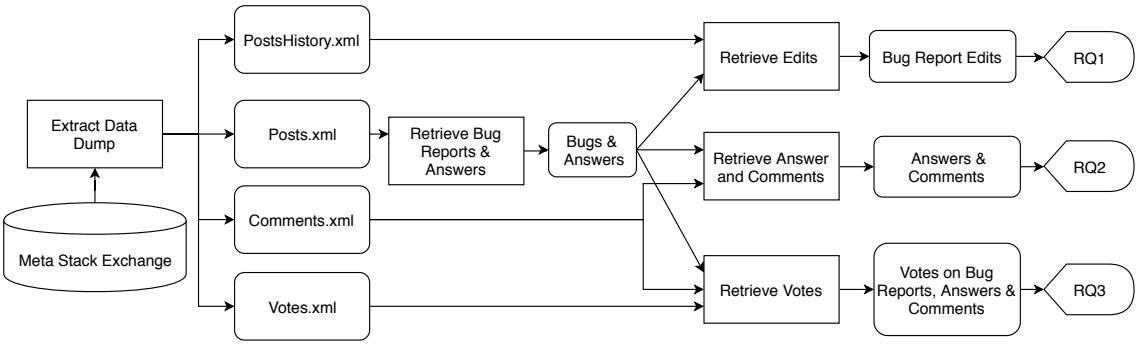

Figure 4.1: An overview of the data collection process.

A brief description of the data collection process is summarized in Figure 4.1. To collect the bug reports, we collected all questions with the tag "bug". We then collected the answers that are associated with those bug reports. The comments that are attached to those questions (i.e., bug-comments) and answers (i.e., answer-comments) are also collected using the Comments.xml file. Likewise, we collected all the edits that were performed on those bug reports by leveraging the PostHistory.xml file. We focus only on those edits that were performed on the title, body, and tags of bug reports, including those edits that were rolled back. In addition, we collected all the votes that are associated with those bug reports (i.e., bug-votes), answers (i.e., answer-votes), and comments (i.e., bug-comment-votes and answer-comment-votes). In total, we collected 19,151 bug reports with 15,904 associated answers for our analysis. Table 5.1 gives an overview of our studied dataset.

Data	Count
Bug reports	19,151
Answers	15,904
Bug-comments	68,207
Answer-comments	28,785
Bug-votes	137,673
Answer-votes	77,734
Edits(Title, Body and Tag)	42,050
Bug-comment votes	59,605
Answer-comment votes	27,090

Table 4.1: Overview of the studied dataset.

4.3 Research Questions and Results

4.3.1 RQ1: How do users leverage the in-place editing feature of the Stack Exchange bug management system?

Motivation: In traditional bug management systems, like Bugzilla[7], the original bug report cannot be edited. Any changes to the bug report in terms of adding new information, removing irrelevant information, making clarifications, or editing incorrect information are made through sequential commenting, leading to a long thread of comments for each report. Finding useful information from such a long thread is time consuming (Arya et al., 2019). Unlike traditional bug management systems, Stack Exchange incorporates the in-place editing feature that can address the above-mentioned difficulties. In this RQ, we study how users leverage the in-place editing feature (see Section 3) to support the bug management process of Stack Exchange. For instance, how often do users edit bug reports? What is the rationale for such edits (e.g.,

[7]https://www.bugzilla.org/

adding bug related information or fixing grammatical mistakes) and whether such edits can improve the quality of bug reports? Answering this research question can provide insights about the potential benefit of having an in-place editing feature in traditional bug management systems.

We first conduct a quantitative analysis to understand how often the in-place editing feature is used for bug reports and who perform those edits. Then, we conduct a qualitative analysis to understand the rationale for editing bug reports. The following subsections describe the approach and findings for our quantitative and qualitative analysis.

Quantitative Analysis

Approach: We calculate the total number of bugs that were edited each year to understand how the in-place editing feature has been used over time and visualize the results in a plot. We also check the role of users who are involved in the in-place editing process. We categorize the stakeholders of bug management system as reporters and non-reporters (i.e., any users other than the reporters.)

Findings: The in-place editing feature has been used steadily over the years. As shown in Figure 4.2, the proportion of the reported bugs that are edited over the years is consistently above 80%. In total, 83% of the bug reports were edited through the in-place editing feature.

All stakeholders (i.e., reporters and non-reporters) are editing bug reports. 43% and 90% of the bug reports were edited by reporters and non-reporters. We also observed a sizable number of cases where reporters and non-reporters edited bug reports

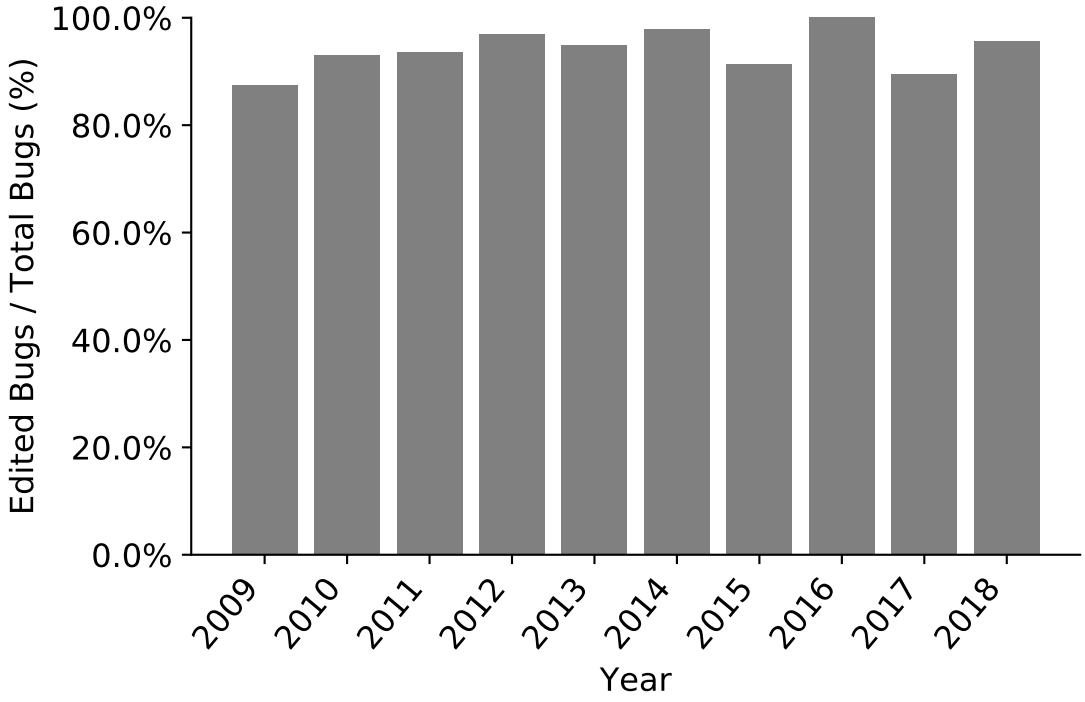

Figure 4.2: The proportion of the reported bugs that are edited over the years.

together – 33% of the bug reports were edited by both reporters and non-reporters. In total, 39% and 61% of the edits were performed by reporters and non-reporters, respectively.

Qualitative Analysis

Approach: We perform a qualitative analysis to understand the rationale for the performed edits on the collected bug reports. To achieve a confidence level of 95% with a confidence interval of 5%, we randomly sampled 380 edits from all the edits of the studied bug reports (i.e., 22,105 edits in total) and identified the rationale for such edits. Bettenburg et al. investigated the types of bug related information that are desired by

developers when resolving bugs. If a user edits a bug report to add such information, we considered that such an edit helps improve the quality of the bug report. Table 4.2 (a) provides a brief description of rationales for editing bug reports that help improving the quality of such reports. To examine if the edits improve the quality of bug reports, we reuse the types of information that were defined by Bettenburg et al. to label our studied edits. We also observed several types of Stack Exchange-specific edit rationales that are not defined in Table 4.2 (a), such as fixing grammar issues and formatting the text. We note that such types of edits are only possible due to the in-place editing feature of Stack Exchange. We performed the following process to derive a list of rationales for edits and labeled the randomly sampled edits. This process involves three phases and is performed by the first two authors (i.e., A1 & A2) of this study:

- Phase I: A1 started with the rationales for edits defined by Bettenburg et al. Bettenburg et al. (2008) and derived a draft list of Stack Exchange-specific rationales for edits based on 50 randomly sampled edits. Then, A1 and A2 used the draft list to label the edits collaboratively. During this phase, the Stack Exchange-specific rationales for edits were revised and refined (the rationales are shown in Table 4.2 (b)).

- Phase II: A1 and A2 independently applied the resulting rationales from Phase I to label all 380 edits. A1 & A2 took notes regarding the deficiency or ambiguity of the labeling for the rationale of edits. During this phase, no new rationales were introduced.

- Phase III: A1, A2 discussed the coding results that were obtained in Phase II to resolve any disagreements until a consensus was reached. The inter-rater agreement of this coding process has a Cohen's Kappa of 88.4% (measured before the

start of Phase III), which indicates that the agreement level is substantial.

Findings: 57% of the studied edits improved the quality of their associated bug reports by adding/correcting essential bug-related information (e.g., observed behavior and environment information). In general, 59% of such edits enhanced the content of the bug report (i.e., content enhancement edits). The remaining non-content enhancement edits were concerned with grammatical and formatting changes, accounting for 37% and 9% of the edits respectively. Note that the sum of rationale percentages exceeds 100%, since in some cases, one edit could have more than one rationales.

We further focus on the edits that are related to content enhancement. A brief description of all content enhancement edits and their percentages are provided in Table 4.2. We observe that 81% of the content enhancement edits added essential information that is crucial to understand, reproduce, and fix a bug as defined by Bettenburg et al. (2008) (i.e., bug-related information). Bettenburg et al. identified incomplete information as the most severe problem for bug reports Bettenburg et al. (2008). Results from our manual analysis suggest that **in-place editing improves the quality of bug reports by adding bug-related information.** We also observe other rationales for edits, which were not observed by Bettenburg et al. (2008), to improve the quality of bug reports. These new rationales include making corrections (8%), and providing clarifications (11%). All content enhancement edits, except for *"Ad-hoc solution"* (2%) and *"Adding bug-fix information"* (2%), improved the quality of the bug reports. **In total, 57% (96% out of the 59% content enhancement edits) of the edits improved the quality of bug reports.**

In-place edits are sometimes rolled back to resolve conflicts or errors that arise

(a) Rationale for edits as documented by Bettenburg et al. (2008)		
Edit Rationale	**Description (D) - Example (E)**	**Percentage**
Observed behavior	(**D**) Adding a more detailed observation of the bug. (**E** #60189) regarding buggy behavior in tags, an edit added: "Edit: The tag has disappeared from the tags page."	41.3%
Screenshot	(**D**) Addition of a screenshot to visually demonstrate the occurrence of the reported bug. Editors add additional text to demonstrate the screenshot. (**E** #259358) "Please see in this figure a screenshot of the error: .. http://i.stack.imgur.com/PBcTD.png" was added.	26.9%
Version	(**D**) Adding information about the software version on which the reported bug was observed. (**E** #273515) "UI items displayed twice bug in android: Stack Exchange Android App Version 1.0.77" was added.	13.5%
OS	(**D**)Adding information about the operating system on which the reported bug was observed. This also includes browser information. (**E** #273724) "On iOS 9.2.1 Mobile Safari." was added.	9.9%
Severity	(**D**) Stressing on the importance of the reported bug. (**E** #279117) regarding text, "Seriously, this is slightly irritating, I am not just posting this to point out how annoying.." was added.	4.9%
Expected behavior	(**D**) Adding a description of what is expected as a normal/optimal behavior to emphasize that the abnormality caused by reported bug. (**E** #311991) the editor added "But the warning as stated is misleading and could possibly be replaced by a different warning about a poorly phrased title or something similar."	4.5%
Steps to reproduce	(**D**) Adding additional information steps for replicating the bug. (**E** #142696) "Steps followed: *Go to chat site. *Found that I was logged in as account 1 * Where is logout? (bug 1) ... " was added.	4.0%
Stack traces	(**D**) Adding stack traces related to the reported bug. (**E** #263612) on keyboard shortcuts, the original content "the user presses 'Command + L' the 'Insert Hyperlink'" was edited to "the user presses >kbd<Command>/kbd< + >kbd<L>/kbd< the 'Insert Hyperlink'".	2.7%

Hardware	(**D**) Adding information related to hardware in the bug report. (**E** #75805) regarding lag in website: "I just noticed that the bug seems absent on a fourth PC, my old 1.5 GHz laptop with Win XP SP 3 and 500 MB RAM. I have only tried .." was added	1.3%
Summary	(**D**) Adding a summary of the bug report to emphasise the important content. (**E** #296240) editor (who is also the reporter) added "To recap, the bug is: the title on the page doesn't..."	0.9%
(b) Stack Exchange specific in-place edit rationales		
Clarification	(**D**) Adding non-bug related information to further clarify the reported bug. (**E** #299048) editor clarified his original post by replacing the text "My reputation remains unchanged (6)" with "My reputation on stack overflow was 6, at the time of asking the above question ..."	10.8%
Correction	(**D**) Correctting bug related information, such as the OS information. (**E** #217448) "This happens for me on Ubuntu 13.04 (64-bit)" was edited to "This happens for me on Ubuntu 13.10 (64-bit)".	7.6%
Ad-hoc solution	(**D**) Editing the original bug report to add an ad-hoc solution or workaround.(**E** #194556) with incorrect coloring, editor added "EDIT4: Can be fixed simply by running the JavaScript $('rect').css('stroke-width', '0') or a userstyle rect { stroke: none }. I may make a user[style|script] to fix this."	2.2%
Adding bug-fix information	(**D**) Information about the bug fix or the deployment of a bug fix. (**E** #229833) "I'm fairly sure that this has happened before but it was fixed and I can't find the bug report." was added.	1.8%

Table 4.2: The identified rationales for editing bug reports along with an example for each rationale. Edit rationales that provide bug related information were previously documented by Bettenburg et al. (2008) (however, the description and examples of these rationales are provided by us to highlight them in the context of our study) (a) The Stack Exchange-specific rationales (b) are derived from our manual study. The complete URL for each of the above-mentioned examples is `https://meta.stackexchange.com/posts/Bug-ID`, where Bug-ID is mentioned in each example in the table. The types are ordered by their percentage.

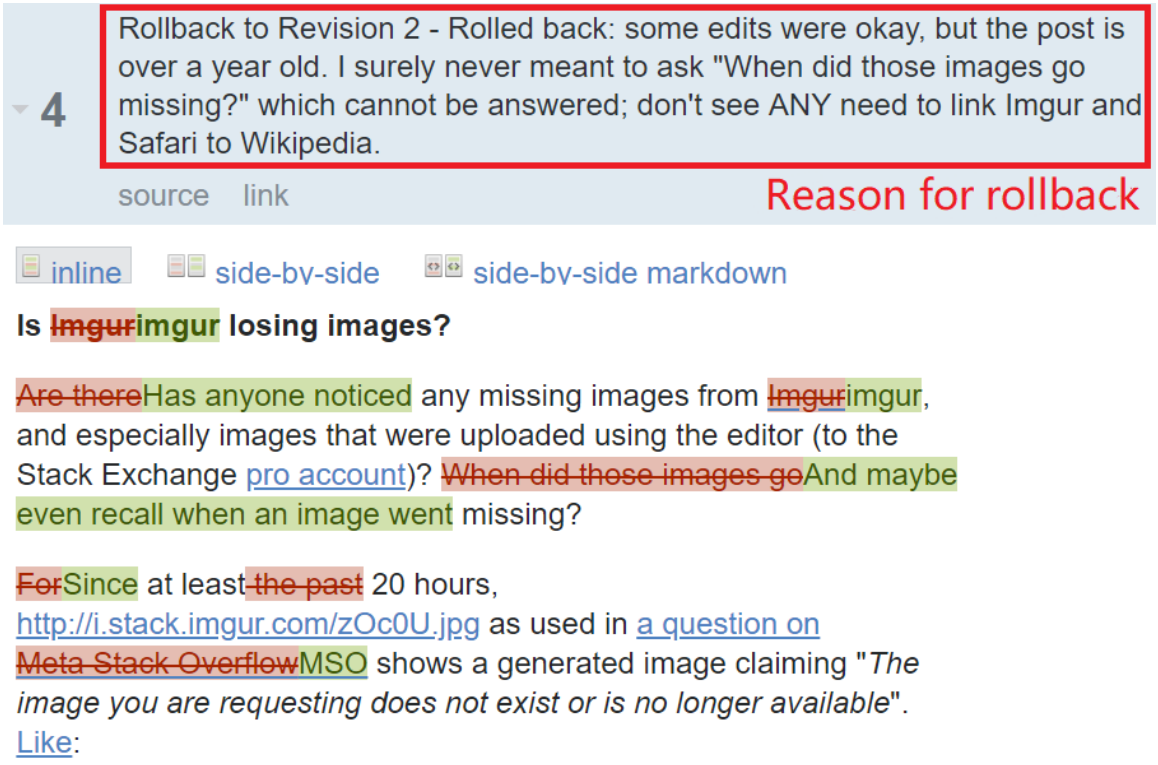

Figure 4.3: An example of a rollback on a bug report.

when multiple users perform in-place editing. To understand the rationale behind such rollbacks, we randomly examined a sample of 50 rollbacks out of the 335 rollbacks in our dataset. From our manual analysis, we identified the following three main rationales for such rollbacks: 1) to correct bug related information (48%), 2) to undo incorrect grammatical changes (34%), and 3) to undo unnecessary formatting changes (18%), indicating that rollbacks provide a mechanism to resolve conflicts or errors that arise due to in-place editing. An example[8] of a rollback is shown in Figure 4.3. The prior edit changed the original meaning of the bug report and the author of the bug report then rolled back the edit.

[8] https://meta.stackexchange.com/posts/75105/revisions

Summary of RQ1

In-place editing feature commonly is used. 57% of the in-place edits improved the quality of bug reports, e.g., adding/correcting/clarifying essential bug-related information (e.g., observed behavior and environment information). In-place edits are sometimes rolled back to resolve conflicts or errors that arise due to in-place editing.

4.3.2 RQ2: How do users leverage the answering and commenting features of the Stack Exchange bug management system?

Motivation: Unlike traditional bug management systems where sequential commenting is the only medium for discussion, Stack Exchange allows the community to discuss and contribute to a bug report through two different channels, namely, commenting and answering. In this RQ, we study how users of the Stack Exchange bug management system leverage the *answering* and *commenting* features to manage bug reports. For instance, what do users discuss when answering versus commenting on a bug report? Do users raise different issues in answers versus comments? This research question helps designers of traditional bug management systems to make informed decisions on whether to integrate such a feature into their systems.

We first conduct a quantitative analysis to understand how frequently answering and commenting features are used in Stack Exchange bug reports. We then conduct a qualitative analysis to understand what types of issues are discussed in bug-comments and answers.

Quantitative Analysis

Approach: We investigate the use of answering and commenting. Hence, we calculate the number of comments and answers that were posted on each bug report. Users are allowed to post comments on bug reports (bug-comments) and answers (answer-comments). Therefore, we also compare the proportion of the comments that were posted on a bug report and those that were posted on its answers after a bug report received a status tag, to investigate if users' interests shift or not.

Findings: In general, commenting is used more frequently than answering. 76% of the bug reports have at least one bug-comment, whereas 60% of the bug reports have at least one answer. A bug report has a median of two bug-comments and one answer. Compared with answers, bug-comments are used more frequently. In addition, we observe that after receiving a status tag, the number of bug-comments decreases, whereas the number of answer-comments increases. In 56.8% of the bug report threads, only answers received comments (answer-comment), while bug reports stop receiving any comment after receiving a status tag. In other words, once a status tag was applied to a bug report, users shift their contributions from commenting on the bug report to commenting on its answers.

Qualitative Analysis

Approach: We perform a qualitative analysis of the discussed issues in bug-comments and answers to understand if users discuss different issues through these two channels. We randomly sampled a statistically representative sample of 380 bug-comments from 68,207 bug-comments and 380 answers from 15,094 answers of bug reports to achieve a 95% confidence level and a confidence interval of 5% in our analysis. We employed the

same process that we performed in RQ1 on these 380 bug-comments and 380 answers. The identified types of the discussed issues in bug-comments and answers are shown in Tables 4.3 and 4.4, respectively. Cohen's Kappa values are 80.0% and 89.1% for our tagging of the answers and bug-comments, respectively.

Findings: **Users leverage bug-comments to ask for or provide more bug-related information (e.g., observed buggy behavior, replication confirmation) as well as to clarify the reported bug, whereas answers are commonly used to provide the solution and explain the cause of a bug.** We observe that more than 66% of the bug-comments were posted to report the observed behavior, replication confirmation, related links, or to ask more information from the reporter in an attempt to clarify the reported bug (see Table 4.3). Interestingly, we also observed in some cases, users integrated the information that was discussed in a bug-comment into the corresponding bug report via the in-place editing feature. For example, in a bug report[9], a user asked the reporter a clarification question and the reporter used the in-place editing feature to add the requested information to clarify the reported bug. In other words, bug-commenting provides a mechanism to ask and collect additional information for a bug report.

Answerers are more likely to report the deployment time for a bug-fix (60%) and explain the cause of a bug (35%) (see Table 4.4). For example, an answerer mentioned that *"We threw 10,230 errors here (network-wide) due to a web server exhausting memory (due to another, competing application pool being a bully). I posted some details.."*[10], which indicates the *cause of the bug*. In another example, the answerer mentioned that "The bug happened when submitting a comment while the text cursor was not at the

[9] https://meta.stackexchange.com/questions/233775
[10] https://meta.stackexchange.com/questions/296438

end of the comment..."[11]. The *cause of bug* provides a technical reason for which the bug was manifested, and *reasoning for no-fix* provides a justification for the reason behind *no-fix*. The rationale *reasoning for no-fix* (27%) includes those examples where features of a system were incorrectly reported as bugs. Such bugs receive "status-declined" or "status-bydesign" tags from the moderators respectively (see Section 3). Interestingly, we observed only a few cases (1%) where users provided screenshots of the bug in the answers.

We also observe some issues that are discussed in both bug-comments and answers, such as information about the bug-fix deployment, ad-hoc solutions, and bug image information. As to why such issues are discussed in answers and not in bug-comments? We conclude the following possible reasons: 1) Stack Exchange does not provide any guideline for using commenting and answering channels. Thus, it may cause confusion about where to discussed such information for users. 2) Comments have a maximum length of 600 characters, which is much shorter than answers. Thus, Stack Exchange users use bug-comments to provide bug-fix information or solution when the content is not too long to describe, otherwise, they use the answering channel. 3) Answers can help the owner make reputation points, but comments do not. This can help the users in posting answers instead of comments.

Furthermore, we observe that once a status tag was applied to a bug report, users shift their contributions from commenting on the bug report to commenting on its answers (as the platform permits users to comment on the bug reports or answers). We performed a qualitative analysis of the content of answer-comments by randomly sampling 50 answer-comments that were posted after the status tag was added to a

[11] https://meta.stackexchange.com/questions/222292

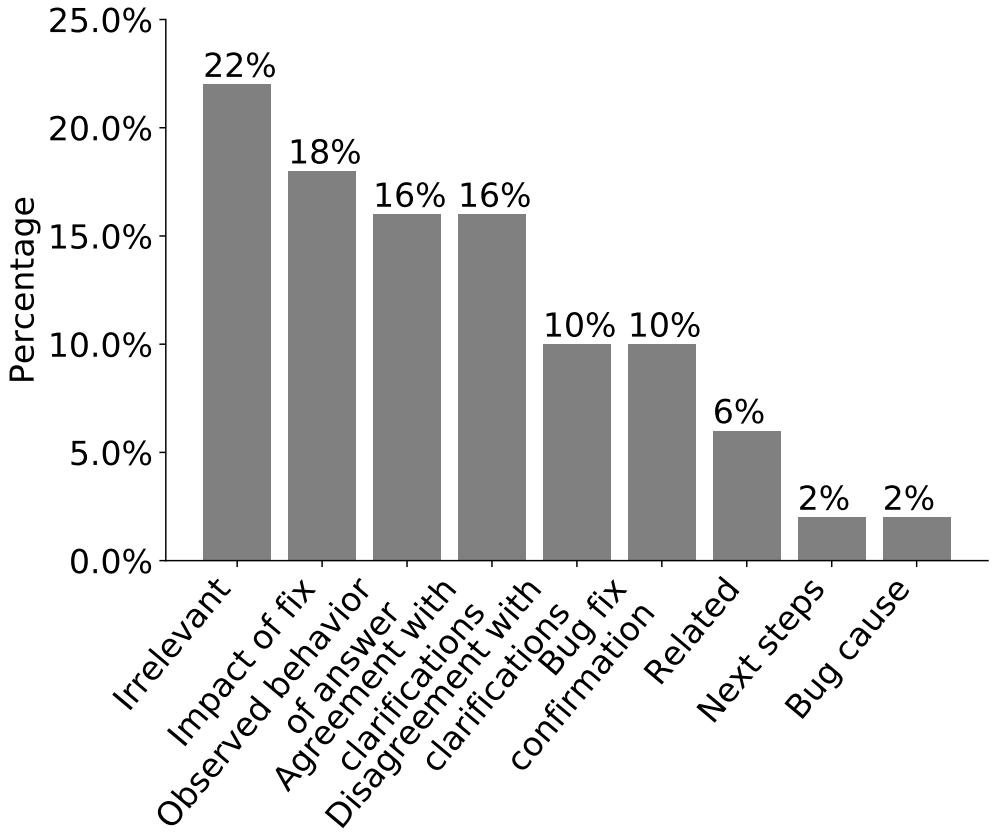

Figure 4.4: Qualitative analysis of answer-comments.

bug reports.

Users are more likely to discuss the impact of the bug fixing (or no-fix) in answer-comments compared to bug-comments. We observe that a number of answer-comments (60%) discuss the impact of the bug fixing or no-fix (18%), observed behavior of the bug-fix (16%), and discuss agreement (16%)/disagreement (10%) regarding a no bug-fix decision. The types of discussions identified in answer-comments are provided in Figure 4.4. For example, in a bug report[12] regarding syntax highlighting in Pascal, the answer explained how the bug-fix integrated the required libraries. A user concerned about the impact of the bug-fix: "Could this cause code-comments to break in Delphi?". Commentary about the solution also involved confirmation for the working of bug-fix (10%) and additional information related to the answer (6%). We also observe few cases where subsequent actions after the bug-fix(i.e., "Next steps") and the cause (i.e., "Bug cause") of the bug were reported. Bug-comments and answers-comments serve different purposes for discussions regarding the bug and its fix or no-fix decision.

Summary of RQ2

In general, commenting is used more frequently than answering and they are used for different purposes. Bug-comments provide a channel for asking and providing more bug-related information, whereas answering provides a channel for including the causes of a bug and bug-fix information.

[12] https://meta.stackexchange.com/questions/171666

4.3.3 RQ3: How do users leverage the voting feature of the Stack Exchange bug management system?

Motivation: Traditional bug management systems provide only limited support for voting. For instance, in Bugzilla, votes can be given on bug reports, indicating that users want those bugs to be fixed. This is analogous to upvotes in the voting system of Stack Exchange. However, there is no support for downvotes in Bugzilla[13]. Moreover, the gamification of the Stack Exchange platform ensures that such votes are integrated more widely across the system (e.g., not just the report but also on the other user contributions like answers and comments). Furthermore, such Stack Exchange votes have some intrinsic value. For example, one has limited number of downvotes to offer (since the user will lose reputation points due to downvotes), ensuring that one would not downvote bug reports arbitrarily and would put some deeper thought in their voting. In this RQ, we aim to provide the designers of traditional bug management systems an empirical understanding of how the voting feature is used in the bug management system of Stack Exchange, enabling them to decide if they should consider integrating such a feature into their systems. For example, we study how the voting feature, typically downvoting (not offered in traditional bug management systems), is used in a bug report and its answers.

Quantitative Analysis

Approach: We perform a quantitative analysis to understand how the use of voting differs across different components of a bug report. We consider votes on bug reports, answers, bug-comments, and answer-comments as bug-votes, answer-votes,

[13] https://www.bugzilla.org/docs/4.4/en/html/voting.html

bug-comment-votes, and answer-comment-votes, respectively.

First, we calculate the proportion of each component that has at least one upvote or downvote out of the total number of that particular components over the years. To show the differences between upvoting and downvoting, we divide our analysis into the use of the upvoting and downvoting, respectively.

Second, we calculate the ratio of votes that were received by different components with respect to the total number of votes received across all the components of a bug report. For example, the ratio of answer-votes is measured as the total number of received votes on all answers of a bug report divided by the total number of votes received in all components of a bug report.

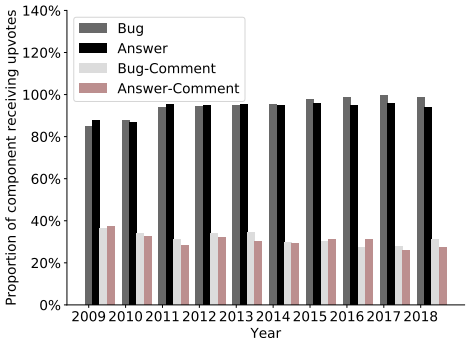

Figure 4.5: The use of upvoting

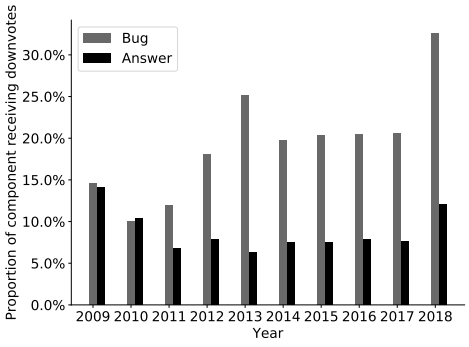

Figure 4.6: The use of downvoting

Figure 4.7: The use of upvoting and downvoting across different components of a bug report, namely: the bug report, answers, bug-comments, and answer-comments. Note that downvoting is not allowed on comments.

Findings: More than 80% of the bug reports and answers receive upvotes consistently over the years. The use of upvoting on bug-comments and answer-comments is less frequent than bug reports and answers. Figure 4.7 shows the use of the upvoting and downvoting on the different components of a bug report (i.e., bug reports,

answers, bug-comments, and answer-comments) over the years in terms of the proportion of received votes across all the components. As mentioned in Section 3, both upvotes and downvotes can be given on bug reports and answers, however, comments (i.e., bug-comments and answer-comments) can only receive upvotes. Compared with upvotes on bug reports and answers (more than 80%), a smaller proportion of bug-comments and answer-comments (less than 40%) received upvotes.

In addition, if we look at the proportion of votes that were received by each component of a bug report, the bug report received the largest proportion of votes (median proportion is 57%), while bug-comments (median proportion is 5%) and answer-comments received the least amount of votes (median proportion is 0%) (see Figure 4.8).

The use of the downvotes has increased gradually over the years, with more downvotes on bug reports than on answers (see Figure 4.7 (b)). In recent years, the significant increase of downvotes is surprising because users lose reputation points when they downvote. One possible explanation of this phenomenon is that over the years, users have earned more reputation points to spare. Another possible reason is that users have become more critical about bug reports, thus are more likely to be more vocal and to express their sentiments through downvoting.

Qualitative Analysis

Approach: We observed that some users left the rationale for downvoting in their bug-comment. Therefore, we perform a qualitative analysis to understand the rationale for the downvoting of bug reports and answers. First, we find all the bug reports that satisfy the following criteria:

1. Reports having one of the following keywords “downvote”, “down-vote” or “down

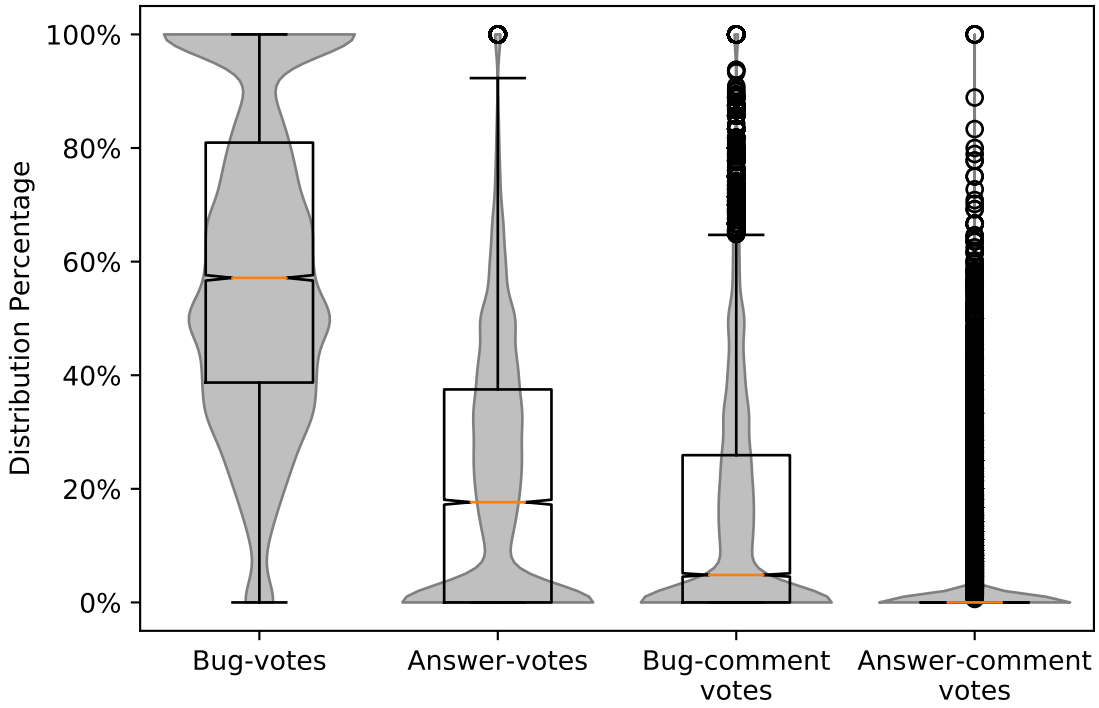

Figure 4.8: The ratio of bug-votes, answer-votes, bug-comment-votes, and answer-comment-votes.

vote" in their associated comments. Please note that users are not required to leave a reason for their downvotes. However, we observed several cases where users mentioned the reasons for downvotes in the comments of those bug reports.

2. Reports having a negative score. As mentioned in Section 3, the score is the sum of upvotes and downvotes. A negative score indicates that a bug report receives more downvotes than upvotes.

We added the second criterion since in some cases, a comment with the listed keywords (i.e.,"downvote", "down-vote" or "down vote") does not necessarily mean the

commenter share the reason for downvoting (e.g., users discussed the downvote itself in comments). We ended up with 98 bug reports that satisfy the above criteria. We performed a lightweight open coding like process as we did in RQ1 to understand the rationale behind those downvotes. We also performed the same process to investigate the rationale for downvotes on answers. We ended up with 44 answers. Our Cohen's Kappa for these two manual studies are 0.72 and 0.92, respectively, which indicate a sufficient agreement.

Findings: Most of the downvotes on bug reports were made due to the disagreement about whether the reported "bug" is a real bug or the low quality of the bug report. A brief overview of the rationale that we identified from our qualitative analysis results is shown in Table 4.5. 43% of the studied downvotes were given to express disagreement with the reported "bug" being a real bug. Furthermore, the proportion of bug reports that were tagged with "status-bydesign" and have a negative score (37%) is much higher than those with a positive score (10%), a likely indicator that users are using voting to express their displeasure with developer decision (something that is not possible in the traditional bug management systems, where users are not able to communicate their opinions about the developer decisions). 4%, 7%, and 1% of the bug reports were downvoted due to the incomplete, incorrect, and duplicate bug information, respectively. For example, one commenter mentioned "i downvoted because the bug report is incomplete and not helpful. the first step to debugging is almost always to reproduce the problem, and by not providing the browser the op is making it exceedingly hard to verify this bug report"[14]. In other words, 12% of the downvotes were made due to the low quality of bug reports. 13% of the bug reports received downvotes due to the insignificant or low impact of the bug.

[14] https://meta.stackexchange.com/questions/151844

Summary of RQ3

Upvoting is used more frequently on bug reports and answers than comments over the years. The use of the downvoting has increased gradually over the years, with more downvotes on bug reports than on their associated answers. Most of the downvotes on bug reports were made due to disagreement about whether the reported "bug" is an actual bug and the low quality of bug reports (i.e., reported bugs are incorrect, insignificant, incomplete and non-reproducible).

4.4 The Implications of our Findings

This section discusses the implications of our findings.

Traditional bug management systems should consider introducing the in-place editing feature. Finding information from a long sequential comment thread of a bug report is a challenging task (Arya et al., 2019). The in-place editing feature enables users to edit a bug report directly instead of leaving long thread of comments to change/add information about the bug report. As we observed in RQ1, the in-place editing feature is steadily used over the years in the Stack Exchange bug management system. 57% of the studied edits improved the quality by adding/correcting/clarifying bug-related information (e.g., observed behavior and environment information). **The rollback feature is used to resolve any conflict due to in-place editing.**

Traditional bug tracking systems should consider introducing the answering and commenting features to encourage the community to better organize and structure

their contribution to bug reports. As we observed in RQ2, users discuss different issues in bug-comments and answers, i.e., Users tend to clarify bug-related information in bug-comments; provide deployment time of a bug-fix and cause of the bug in answers; while discussing the expected impact of the bug-fix in answer-comments. We also observed some issues that are discussed in both bug-comments and answers, such as information about the bug-fix deployment, which may indicate that users are confused about where to discuss such information. **Clear guideline about where (e.g., answers or comments) to discuss a particular issue may need to provide to users.**

Traditional bug reporting systems should consider support for downvoting. In RQ3, we observe that 60% of the studied downvotes were made on bug reports due to the disagreement that a reported "bug" is not a real bug. Traditional bug management systems can include the downvoting feature to eliminate bug reports that are not real bug, rather a feature of the system. Bettenburg et al. (2008) reported that bug reporters do not always provide the content required by developers to resolve the bug. Voting on bugs and answers may provide a mechanism for traditional issue reporting systems to ingrain desirable behavior in reporting, and better maintain bug reports. For example, in RQ3, we observe that users downvote a bug report if the bug report does not contain complete or adequate information about the bug. Meanwhile, **the rationales for downvoting should be explained when performing downvoting. (something that is not implemented by the Stack Exchange platform)** Such explanations can help in improving the bug report and can provide feedback and help novice users to learn how to write a good bug report.

4.5 Threats to Validity

External validity: Threats to external validity are related to the generalizability of our findings. In this study, we performed a large-scale analysis of the Stack Exchange bug management system. Our results may not generalize to other Q&A platforms. However, we would like to point to the fact that leveraging a Q&A website for the purpose of bug management is not very common due to the prevalence of traditional bug management systems. Thus, our studied dataset provides an rare opportunity for the designers of traditional bug management system to learn about the unique features of managing bugs through the Q&A platform.

Internal validity: Threats to internal validity relate to experimenter bias and errors. Our study involved several qualitative analyses in each of our RQs. To reduce the bias from human factors, two authors independently labeled the studied data for each analysis and discrepancies were discussed until a consensus was reached. We do note that the levels of inter-rater agreement of all our qualitative studies are high. When conducting the qualitative analysis on edits, answers, and comments in our RQs, it is practically impossible to manually analyze all of them due to their large size and needed effort to perform such in-depth tagging. To minimize the bias in our findings, we studied statistically representative random samples of our studied datasets to ensure a 95% confidence level and 5% confidence interval commonly done by prior studies by Wang et al. (2018); Chen et al. (2014).

4.6 Conclusion

In this section, we study how three unique features, namely, the in-place editing feature, the answering and commenting features, and the voting feature are used in the Stack Exchange bug management system. We find that: 1) 57% of the edits improved the quality of bug reports, such as adding/correcting/clarfying essential bug-related information (e,g., observed behavior and environment information). 2) The commenting and answering features are used differently, i.e., commenting on a bug report provides an avenue for discussing bug related information, while answering offers an avenue for providing the solution and explaining the causes of those bugs. 3) Most of the downvotes on bug reports were made due to disagreement about whether the reported "bug" is an actual bug and the low quality of bug reports.

Based on our analysis, we offer the following suggestions: 1) Traditional bug management systems should consider incorporating the in-place editing and the roll-back features to avoid long hard-to-follow threads that are associated with bug reports. 2) Bug management systems should consider incorporating the answering and commenting features that can help better structure discussions about bugs and their resolution, however clear guidelines regarding where to contribute (e.g., answers or comments) needs to be provided. 3) Downvoting provides a corrective feedback to bug reporters. While Stack Exchange does not employ a mechanism to explain the reasons for downvotes, we suggest downvotes must be accompanied with an explanation. Such an explanation can help in improving the bug report, provide feedback and can help novice users to learn how to write a good bug report.

Type	Description(D) - Example (E)	Percentage
Observed buggy behavior	(**D**) Providing more information about the buggy behavior. (**E** #254351) the commenter added the following information: “Note it seems to randomly affect Question edits now. Question posting does not seem affected but ..."	23.2%
Replication confirmation	(**D**) Confirming the reproducability of a reported bug. (**E** #66417) a commenter provided replication confirmation by including the following comment: “yup it is reproducible.”	20.0%
Irrelevant	(**D**) Providing any comments that is not related to bug-fix or bug related information. (**E** #26740) a commenter mentioned: “This is the right place to report bugs and support questions. ’	12.1%
Asking more information about bug	(**D**) Asking more information about the buggy behavior. (**E** #196930) the following question was asked in a comment: “Do you have a link to the question?”	11.6%
Links related to the bug	(**D**) Adding possible duplicates of the bug report or related links relevant to the reported bug.(**E** #112096) a link related to the bug report was added in a comment: “Sort of related: meta.stackexchange.com/questions/1050..”	10.5%
Expected behavior of system	(**D**) Providing what is expected (non-buggy) behavior of the system. (**E** #141867) a commenter mentioned “This doesn’t look like it’s behaving as it should, ... the suggested edit should be recorded as improved, not rejected. ”	6.8%
Clarification of system design	(**D**) Providing clarifications about how the system works. (**E** #209229) a commenter clarified the problem by adding the following comment: “This isn’t really a bug. This is typical, as double clicking something will highlight it. In this case, it’s just highlighted oddly due to the positioning of the element.”	5.8%
Bug-fix deployment information	(**D**) Indicating when a bug-fix will be deployed. (**E** #83661) the expected time to fix the bug was reported in the comment: ‘fixed to be deployed some time tomorrow”	5.3%
Ad-hoc solution	(**D**) Providing an unofficial fix or ad-hoc solution to the reported bug. (**E** #19622) an unofficial bug-fix was provided by the following comment: “temporary solution: meta.stackexchange.com/questions/27702/.."	4.2%

Bug image link	(**D**) Providing a URL for an image to illustrate the reported bug. (**E** #163475) a URL of an image was added in the comment: "imgur.com/aL6zn it's white."	0.8%

Table 4.3: The types of discussions identified in bug-comments. The types are ordered by their percentage.

Type	**Description (D)- Example (E)**	**Percentage**
Bug-fix deployment information	(**D**) Same as Bug-fix deployment information in Table 4.3. (**E** #226249) the following answer was provided: "You should see this changed with our next build today. Thanks for catching it."	59.7%
Cause of bug	(**D**) Explaining why a bug occurred. (**E** #250507) an answerer replied: "... The problem was that the web has a few permutations of MathJax: Most sites."	34.7%
Reasoning for no fix	(**D**) Clarifying about how the system should work, and why the bug will not be fixed/a fix is not required. (**E** #175684) an answerer replied: "We will not fix this. The issue is in a specific combination of OS and browser, and ..."	27.4%
Ad-hoc solution	(**D**) Same as Ad-hoc solution in Table 4.3. (**E** #27494) an answerer suggested a possible solution: "As others have mentioned, try in a different browser to see if that helps as well."	4.5%
Replication confirmation	(**D**) Same as Replication confirmation in Table 4.3. (**E** #101453) an answerer confirmed the replication of the bug: "I was able to reproduce it in Safari on Snow Leopard."	3.4%
Bug image	(**D**) Providing images to demonstrate the bug (and not the answer). (**E** #267214) a screenshot was added to explain the bug: "Here's a screenshot of the one I currently can't pass ... https://i.stack.imgur.com/EY8nE.png"	1.3%

Table 4.4: The types of discussions identified in answers. The types are ordered by their percentage.

Rationale for downvotes	**Description (D) - Example (E)**	**Percentage**
Not a bug	(**D**) The reported "bug" is not a real bug. (**E** #308111), commenters responded "No, that is by design, not a bug. That is how Markdown works."	59.7%
Insignificant	(**D**) The reported bug has a low impact or does not create a significant difference to the software. (**E** #96300), commenters mentioned: "I don't really care where it's from, if it's helpful and could be helpful to others...", indicating that the bug isn't helpful and is insignificant.	16.9%
Incorrect	(**D**) Reporting a bug without a proper analysis of the buggy behavior, e.g., reporting mistaken bug information. (**E** #145250) regarding reputation loss by 2 points, a commenter mentioned: "It looks fine to me. Are you sure it decreased twice?"	9.1%
Non-reproducible	(**D**) Bug reports that were not reproducible. (**E** #159376), a commenter responded: "Nope, don't see that kind of behaviour here at all"	9.1%
Incomplete	(**D**) The bug report does not contain complete or adequate information about the bug. (**E** #112819) regarding reputation decrease, a commenter mentioned: "what went down by 2? the question/answer score? the user's rep? **your** rep? care to provide a link?"	3.9%
Duplicate	(**D**) Reporting a duplicate bug report. (**E** #99429), "it seems to be a duplicate, yes. thanks guys. you can vote to close."	1.3%
Offensive	(**D**) Using offensive language to express anger while reporting a bug. (**E** #292490), "I took the liberty of changing the title to your post. calling things "bogus" attracts downvotes. hopefully this more neutral title will make the question better received."	1.3%

Table 4.5: The identified rationales for downvotes on bug reports. The rationales are sorted from the most observed to the least observed rationale. The complete URL for each of the above-mentioned examples is https://meta.stackexchange.com/posts/Bug-ID, where Bug-ID is mentioned in each example (**E**) in the table.

CHAPTER 5

An Exploratory Study on the Differences Between Bugs and Features Using the Stack Exchange Question and Answering Platform

Intuitively, one would expect that feature requests and bug reports (and their management) would differ. Yet, no prior study has ever explored such differences in a systematic manner since there exists no large enough datasets where issues are tagged correctly as bug reports versus feature requests.

This chapter reports on a study of the differences between bug reports and feature requests in the Stack Exchange issue management system. We focus our study on the Stack Exchange system since: 1) it contains a large number of issues that have been carefully tagged as bug reports versus feature requests by domain experts, rather than

researchers (ensuring the robustness of our findings), 2) the issue management is carried out through a Q&A platform, which provides us with a richer perspectives on the differences between the management of bug reports and feature requests.

Our study finds that bug reports and feature requests differ significantly from each other along many dimensions such as the amount of community contributions, the content of the issues, and the characteristics of the participating users. Feature requests are generally longer than bug reports, receive more contributions through comments, answers and votes compared to bug reports. Also higher reputation users tend to contribute more to bug reports than to feature requests. We are also able to automatically identify bug reports from feature request with a median AUC of 89.8%. Our study offers a rich empirical understanding of the differences between bug reports and feature requests (and their management). The developers of issue management systems, software practitioners and researchers can leverage such understanding to improve issue management processes in large software projects.

5.1 Introduction

Issue management systems are an integral part of software development practices. Such systems contain a wealth of information about the various aspects of a project and its progress. Such systems track reported bugs, and manage their resolution while also capturing feature requests, and tracking their progress. Such rich data has been used to develop methods to avoid the occurrence of future bugs (D'Ambros et al., 2010; Bhattacharya and Neamtiu, 2011; Kamei et al., 2010), to identify the most suitable developer to work on a new issue (Anvik et al., 2006; Mani et al., 2019; Xuan et al., 2012;

Nguyen et al., 2014), and to the determine the needed time to address an issue (Weiss et al., 2007; Giger et al., 2010; Marks et al., 2011).

Intuitively, one would expect that the management of feature requests and bug reports would differ. A good understanding of such differences would help better leverage this rich data. Yet, no study has ever explored such differences in a systematic manner. A key reason for the lack of such a study is that there does not exist large enough datasets where issues are tagged correctly as bug reports versus feature requests. Much of the prior studies Antoniol et al. (2008b); Herzig et al. (2013); **?** tagged a small number of issues and such tagging was performed by researchers instead of domain experts. The small number of tagged data limits the generalisation and robustness of any derived observations.

Therefore, in this chapter, we study the differences between bug reports and feature requests using the issue reports in the Stack Exchange issue management system. We choose to study the Stack Exchange issues for the following reasons: 1) Stack Exchange's issue management system contains a large number of issue reports that have been carefully tagged as bug reports versus feature requests by Stack Exchange developers and community members (i.e., domain experts), rather than researchers (ensuring the robustness of our findings), 2) the issue management process is carried out through a Q&A platform, which provides us with a richer perspective on the differences between the management of bug reports and feature requests.

In contrast to traditional issue management systems where contributions to an issue report are performed through the commenting on an issue, Q&A platforms enable users to edit issue reports in-place, answer or comment on the reports, or even comment on the answers. Hence, we can examine whether the management of bugs vs

features differ along the perspectives of the unique Q&A features. In particular, we structure this chapter along these two research questions:

1. **How does the management of bug reports differ from the management of feature requests?**

   Feature requests (290 words in median) are generally longer than bug reports (190 words in median), meanwhile feature requests receive longer comments and answers. Feature requests receive more contributions through comments, answers, and votes in contrast to bug reports. The received comments for feature reports exhibit more positive sentiment than those for bug reports. In addition, users who report and contribute to bug reports have higher reputation than those for feature requests.

2. **Can we automatically identify bug reports from feature requests?**

   The Q&A specific features (e.g., editing and answering versus simple commenting) by themselves—without using the textual content of the issue reports can identify bug reports from feature requests with a median AUC of 78%. Furthermore, Q&A factors along with the sentiment and the text of the issue report enable an automated classification of bug reports and features requests with a median AUC of 0.90%. The expressed sentiment in the issue report and the Q&A characteristics of the issue reports are the most influential factors in identifying bug reports from feature requests.

Our study shows that bug reports and feature requests are significantly different across various dimensions and they are managed differently. Furthermore, we can automatically and effective classify bug reports and feature requests by considering the textual content of an issue report along the Q&A platform specific features. Our

findings provide insights for further research efforts on how to automatically identify bug reports and feature requests and demonstrate the usefulness of Q&A platforms for managing issues. The gained empirical understanding from our study can help developers of issue management systems, software practitioners and researchers improve issue management processes in large software projects.

The remainder of this chapter is organized as follows. Section 5.2 introduces our studied dataset. The study procedures and the results of our above mentioned research questions are described in Sections 5.3, 5.4 respectively. Section 5.5 presents threats to the validity of the observations of our study. Finally, Section 5.6 concludes our study.

5.2 Data Collection

This section discusses how we collect the dataset that we used to answer our research questions.

For this study, we download a publicly available data dump of Meta Stack Exchange from archive.org[1]. The data dump contains all the activities around posted questions between June 28, 2019, and March 3, 2019. The data dump consists of a collection of XML files containing information about questions, associated answers, post histories, post links, comments, and votes. We use the following files from this set:

1. **Posts.xml:** Contains all questions and answers. To collect bug and feature requests, we collected all questions with the tag “bug” and “feature” respectively. We then collected the answers that are associated with those bug reports and feature requests.

[1] https://archive.org/download/stackexchange

2. **Comments.xml**: Contains all comments that are associated with questions and answers. We collected the comments that are associated with bug reports (bug-comments) and their answers (bug-answer-comments). Similarly, we also collect the comments that are associated with feature requests (feature-comments) and their answers (feature-answer-comments).

3. **PostHistory.xml**: Contains all edits that affect the content of any questions or answers. We only collected those edits that affect the title, body and tags of bug reports and feature requests.

4. **Votes.xml**: Contains all the votes that are associated with questions or answers. We collected all the votes that are associated with bug reports (i.e., bug-votes), answers (i.e., bug-answer-votes), and comments (bug-comment-votes and bug-answer-comment-votes). Similarly we collected all the votes that are associated with the different parts of a feature request (i.e., feature-votes, feature-comment-votes, feature-answer, and feature-answer-comment-votes).

Since, we work on the issues that have been developed, such issues do not contain any noise due to the presence of duplicate issues. We found some cases where reporters failed to correctly identify bugs (219 in total) or feature requests (143 in total). We identified such misclassifications by tracking any edits the replace the “bug” tag with the “feature-request” tag or vice-versa. After removing those cases we ended up with 6,586 and 2,524 bugs and features.

Category	Bug Reports	Feature-requests
Issues	6,546	2,502
Answers	6,057	2354
Issue Comments	4,606	1,817
Answer Comments	3,435	1,772
Edits	16,681	7,105
Votes	57,694	60,103

Table 5.1: Overview of the studied dataset.

5.3 RQ1: How does the management of bug reports differ from the management of feature requests?

Intuitively, the management of bug reports and feature requests would differ. For example, closing a bug report might be much faster than a feature request; developers are more likely to have more extensive discussions about feature requests compared to bug reports. In this RQ, we investigate whether bug reports and feature requests differ from each other along several dimensions (i.e., their content, community contributions, and contributing users). Our empirical investigation offers us a deeper understanding of the differences between bug reports and feature requests, not only from the perspective of their content, but also based on how the community contributes to them and user contribution.

To understand whether bug reports and feature requests differ from each other, we investigate them along three dimensions to answer the following questions

- **Do bug reports and feature requests differ in terms of community contributions?** Q&A based contributions to the issue reports happen through four avenues: a) answering b) commenting c) editing, and d) voting.

- **Do bug reports and feature requests differ in terms of their content characteristics?** We consider the followings aspects of textual content: a) length in characters b) readability score c) expressed sentiment.

- **Do bug reports and feature requests differ in terms of participating users?** We check whether user contribution varies across bug reports and feature requests.

Below we describe in detail the approaches that we follow to perform our quantitative analysis along the three abovementioned dimensions.

5.3.1 Do bug reports and feature requests differ in terms of community contributions?

Approach: Stack Exchange issue management system offers different avenues for community contributions. We consider contributions through answering, commenting, editing, and voting on comments and answers. It should be noted that Stack Exchange does not allow downvotes on comments (i.e., issue-comment and answer-comment).

In traditional issue management systems, all contributions to an issue appear as a long thread of comments. Thus, we merge all these contributions together to understand whether bug reports and feature requests differ when the issue management systems does not offer different avenues for contributions like the Stack Exchange Q&A platform. Furthermore, we consider those contributions across different time periods to understand whether community contributions vary across time. We not only calculate contributions that occurred within one, three, and seven days from the reporting of an issue report, but also calculate the total number of contributions throughout the lifetime of an issue (i.e., from issue reporting to receiving a status-completed tag.

We perform a Mann-Whitney U test and a Cliff's d test to determine whether or not any observed differences between bug reports and feature requests (along the above-mentioned dimensions) are statistically significant and to measure the magnitude of the differences. We use the Cliff thresholds as used by Kelley and Preacher (2012) to determine the effect size, where $|d| < 0.147$ indicates that the effect size is negligible, $|d| < 0.33$ indicates that the effect size is small, $|d| < 0.474$ indicates that the effect size is medium, otherwise the effect size is large.

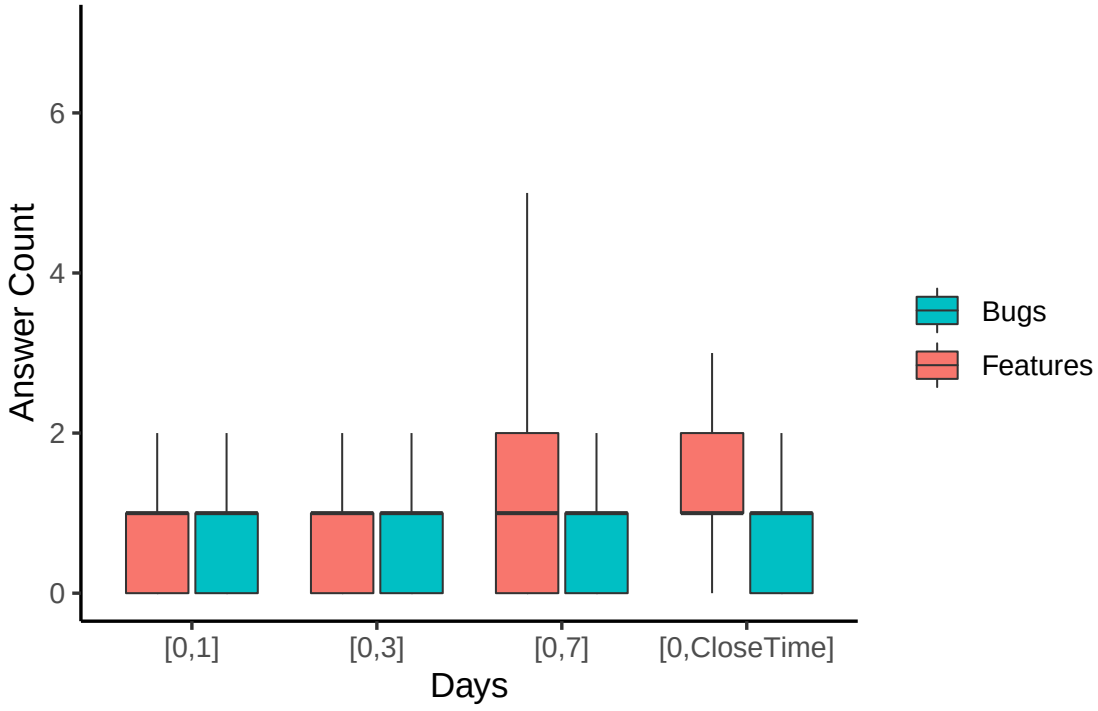

Figure 5.1: Received answers within one, three, and seven days of the reporting of a bug report or a feature request. We also consider all the received answers in the lifetime of an issue report

Findings: Feature requests receive more answers than bug reports. Figure 5.1 compares the number of received answers between bug reports and feature requests. The Wilcoxon rank sum shows that their difference is statistically significant (p-value is

≤ 0.05) with a large effect size (Cohen's D estimate is -0.66 (i.e., medium)). While bug reports receive a median of one answer within seven days (minimum is 0 and maximum is 11) of their reporting, feature requests receive a median of two answers (minimum is 0 and maximum is 15) for that same time period. The differences become more pronounced when we count all the answers throughout the lifetime of a bug report and a feature request.

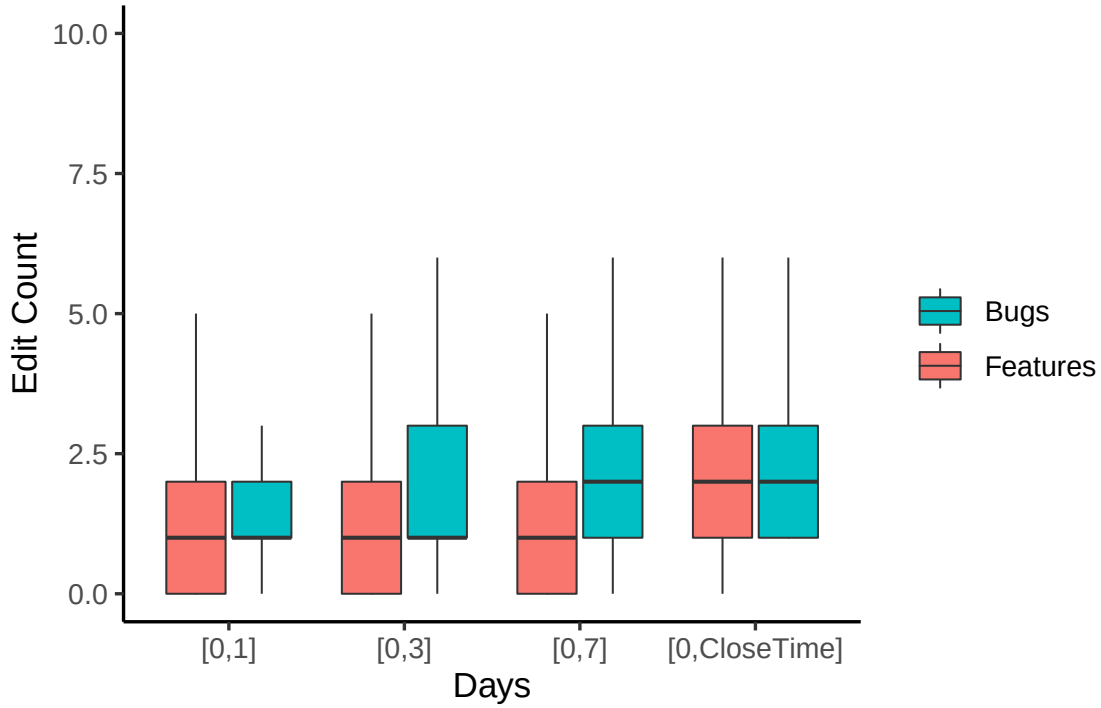

Figure 5.2: Received edits within one, three, and seven days of the reporting of a bug report or a feature request. We also consider all the received edits in the lifetime of an issue report

Bug reports and feature requests receive a median of two edits. However, a bug report receives its second edit within three days (median), whereas a feature request receives its second edit after seven days of its reporting – highlighting that bug reports receive edits sooner than feature requests.

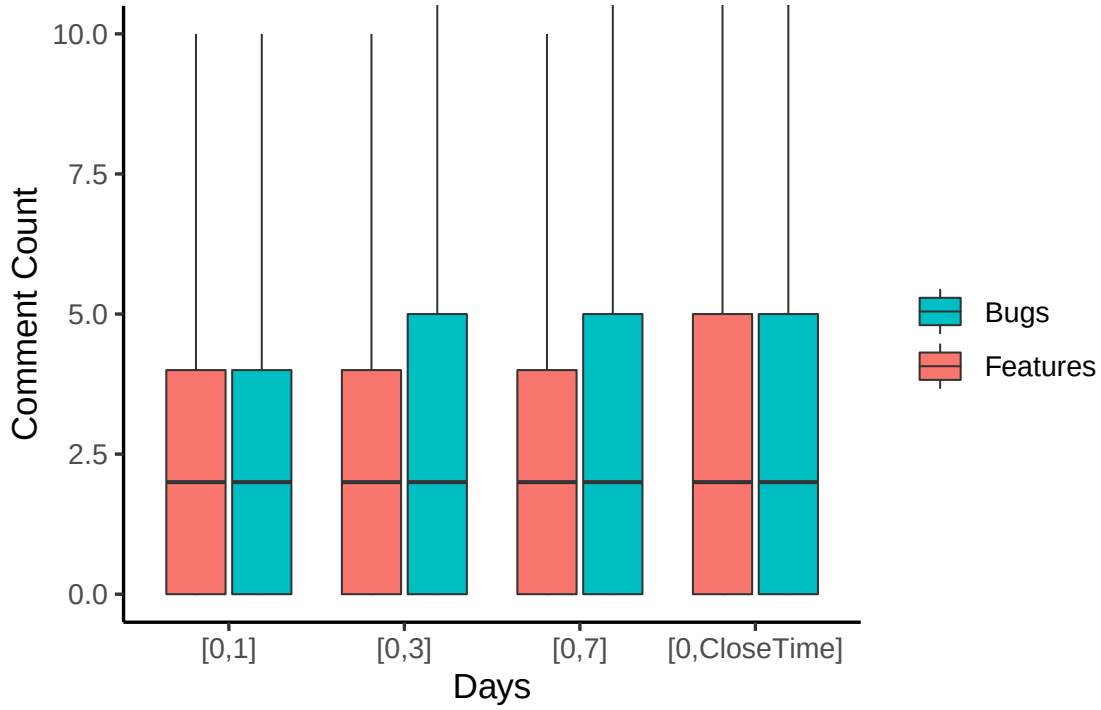

Figure 5.3: Received comments within one, three, and seven days of the reporting of a bug report or a feature request. We also consider all the received comments in the lifetime of an issue report

Both bug reports and feature requests attract comments (issue-comments), however, feature requests are more likely to continue receiving comments even 7 days after their initial reporting. Figure 5.3 shows that feature requests receive a median of four comments throughout their lifetime, when compared to bug reports which receive a median of three comments. Unlike bug reports, community contributions to feature requests continue on for a longer period of time.

The Wilcoxon Rank-Sum test p-value between the comment count of bug reports and feature requests is statistically significant though the Cohen's D is -0.13 (negligible).

Feature requests get more votes than bug reports. Figure 5.5 and Figure 5.6 show

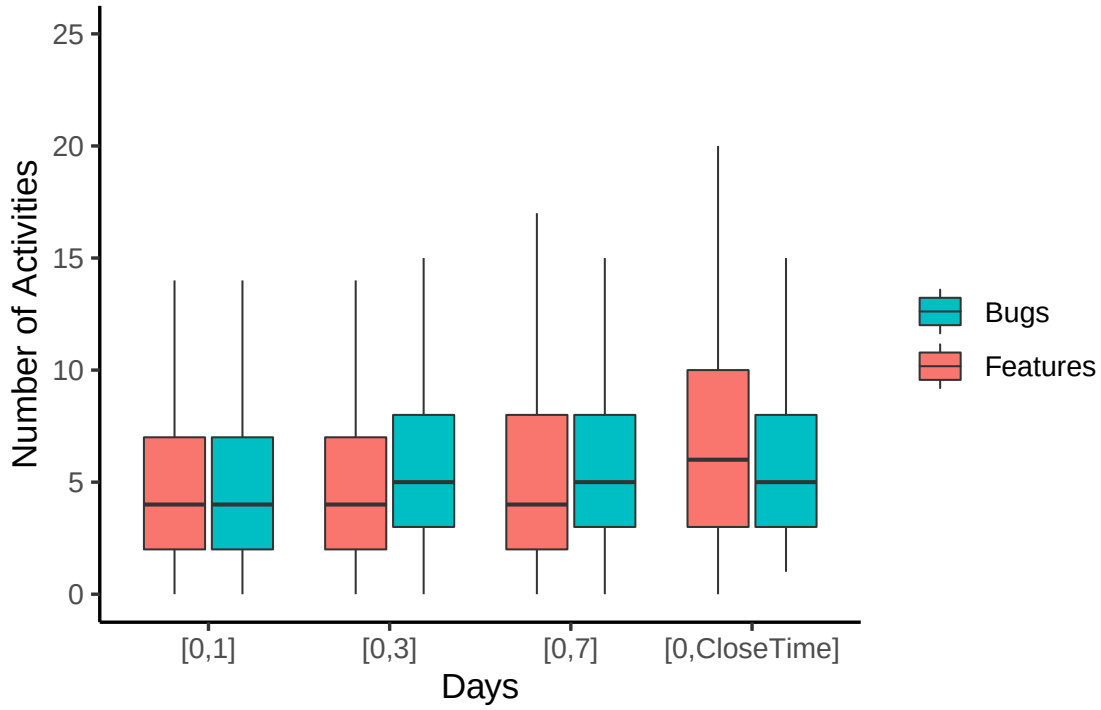

Figure 5.4: Total number of received contributions (i.e., answer count, comment count, edit count, and vote count) received within one, three, and seven days of reporting. We also consider all the contributions in the lifetime of an issue report

upvotes and downvotes contributions on the different components of an issue report, respectively. Upvoting for feature requests is higher than that for bug reports with a median of 6 and 11 upvotes on bug reports and feature requests, respectively.

Both bug reports and feature requests get a median of zero downvotes. 11.7% of the bug reports and 33.0% of the feature requests received downvotes. A closer investigation reveals that community members use voting to indicate whether they like the idea of a given bug report or feature request. Hence when community members do not like a requested feature, they would cast a downvote – explaining a large proportion of feature requests receive more downvotes than bug reports. While for bug reports, the downvoting is used to complain about the validity of a bug report. For example, in a

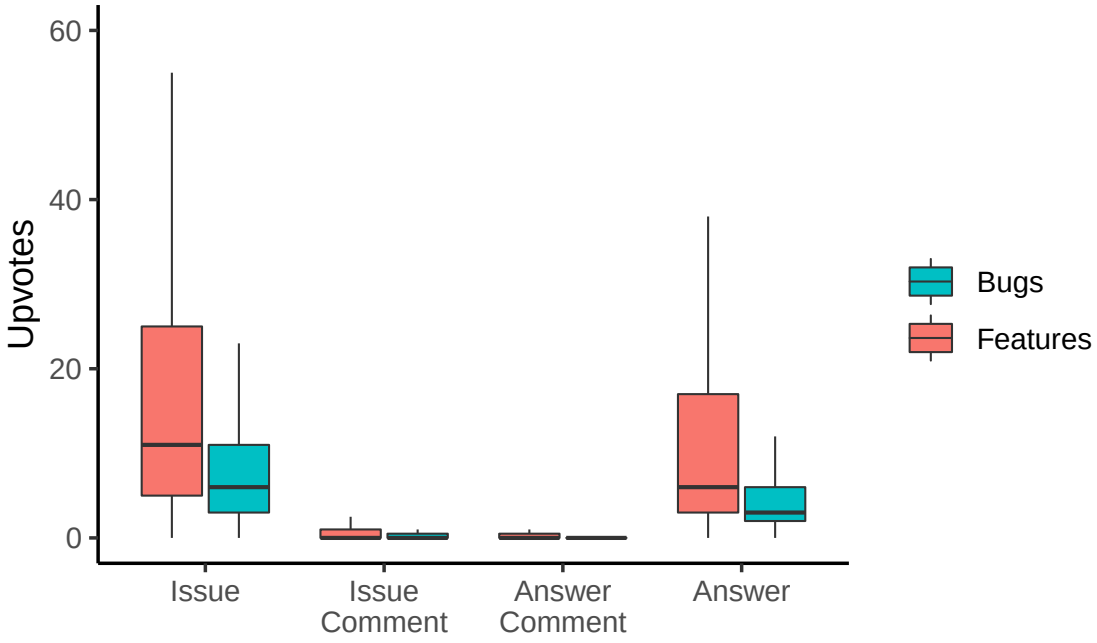

Figure 5.5: Upvotes on the different components of a bug report and feature request

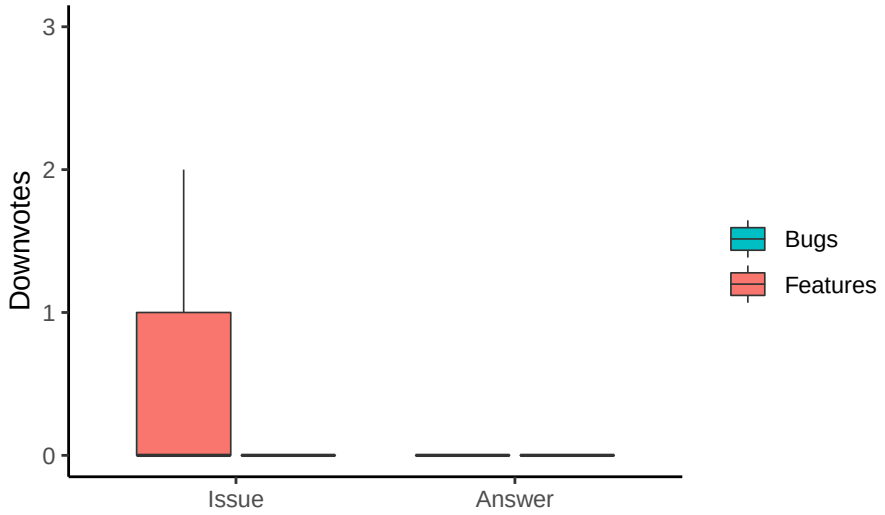

Figure 5.6: Downvotes on issue reports and their answers. Please note that comments cannot be downvoted

bug report[2], one of the bug-comments mentioned: "Nope, don't see that kind of behaviour here at all", demonstrating that the bug report was downvoted because of the reported bug was not reproducible.

5.3.2 Do bug reports and feature requests differ in terms of their textual content?

Approach: In this subsection, we examine the length, readability score and expressed sentiment this sentiment stuff is not consistent in its wording.. fix throughout of the textual content to understand whether these aspects vary between bug reports and feature requests. First, we calculate the length of an issue by considering the number of words in it. Second, we calculate the readability score of an issue using the Flesch Kincaid readability score(Buse and Weimer, 2008) that indicates the difficulty of understanding a passage in English. A higher readability score indicates that a given issue is easier to read. We use the Python package texstat[3] package to calculate the FleschâĂŞKincaid readability score. We calculate the readability for the body, as well as other components of an issue report.

To check if users express different sentiments in bug reports versus feature requests, we performed a sentiment analysis on the issue reports.

Sentiment analysis determines the polarity or the expressed emotion in a sentence or document (in our case, a bug report or a feature request). We use the python **Vader** package (Hutto and Gilbert, 2014) to perform the sentiment analysis. Vader is a lexicon and rule-based sentiment analysis tool that provides a compound score given any text.

[2] https://meta.stackexchange.com/questions/159376
[3] https://pypi.org/project/textstat/

The compound score provides a uni-dimensional measure of the sentiment for a given piece of text. We compare bug reports and feature requests using the polarity score that we obtained through Vader.

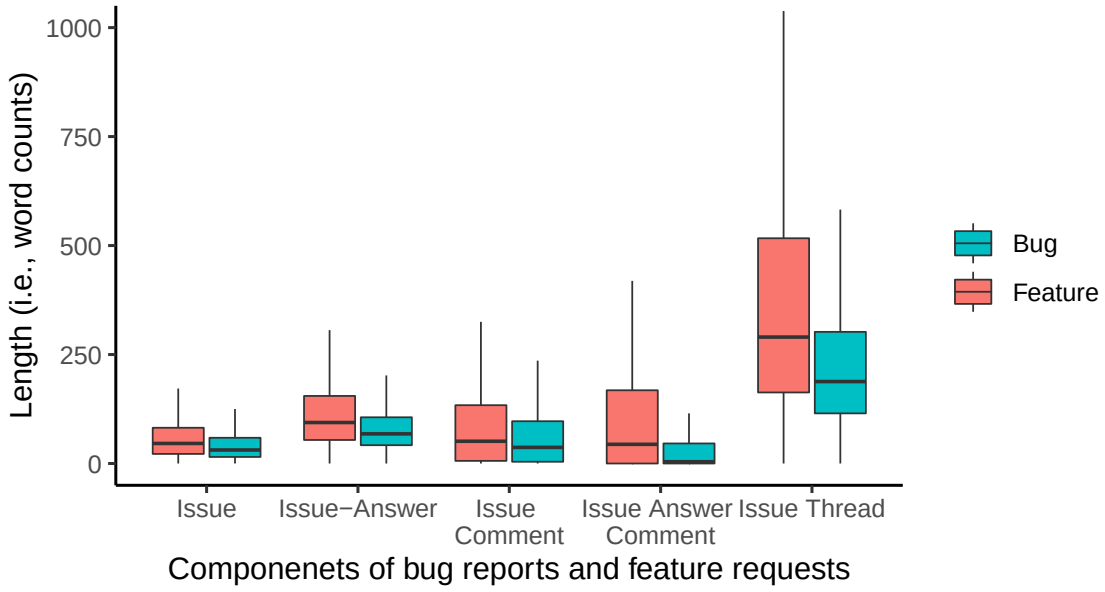

Figure 5.7: Length of different components of bug reports and feature requests. We use the term combined to refer to the length of a bug report or a feature request.

Results: Feature request threads are longer than bug report threads (word count wise), as shown in Figure 5.7. The Wilcoxon Rank-Sum p-value for the length difference between bug report and feature request threads is statistically significant ($p \leq 0.05$), with a medium effect size (Cohen's D is equal to -0.61 (medium)). We hypothesize that this is likely due to feature requests being more open for discussion compared to bug reports as well feature requests needing more explanations to justify the importance of the requested features.

While the median length of a bug report thread is 68, the median length increases to 98 for feature requests. We also observe that more words are used for the various

components of a bug report than those of a feature requests.

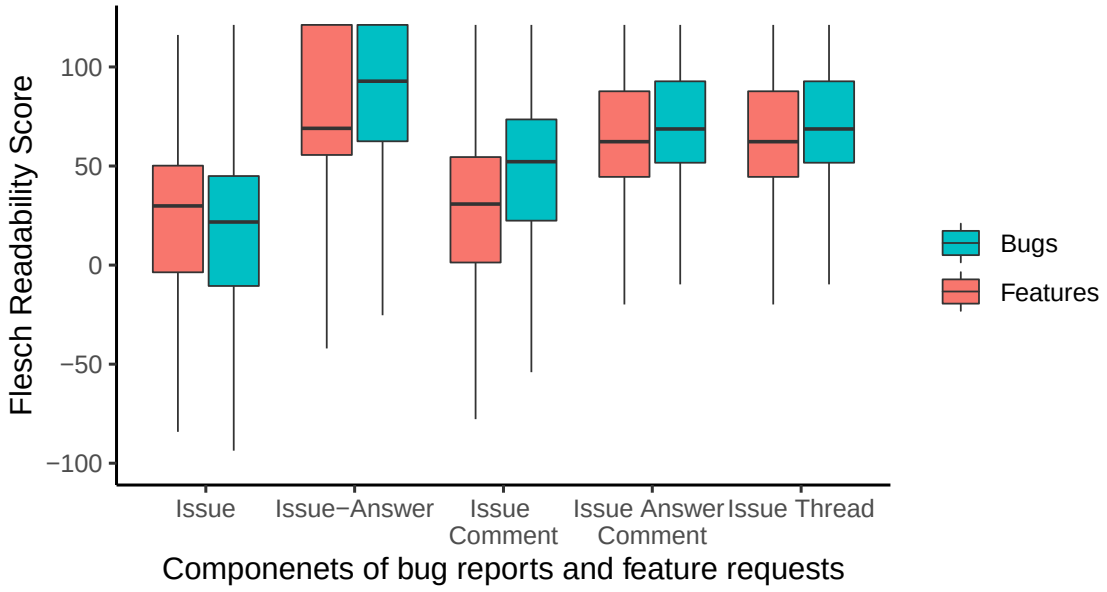

Figure 5.8: Readability scores of the different components of bug reports and feature requests. We use the term combined to refer to the readability score of a bug report thread or a feature request thread

Feature requests have higher readability scores than bug reports, as shown in Fig. 5.8, indicating that feature requests are more difficult to comprehend than bug reports. While the median readability score of bug reports is 21 (minimum is -1,434, maximum is 121), the score is 30 for feature requests is (minimum is -637, maximum is 116). The Wilcoxon Rank-Sum p-value is statistically significant ($p \leq 0.05$). However, we observe that for answers, answer-comments and issue-comments the median readability score is higher for bug reports than feature requests – this likely due to the simplicity of the followup comments/answers about a bug report versus a feature request where such followups are part of deeper discussions. We also observe similar results when we consider bug report and feature request threads, the median values are 31, and 28 respectively.

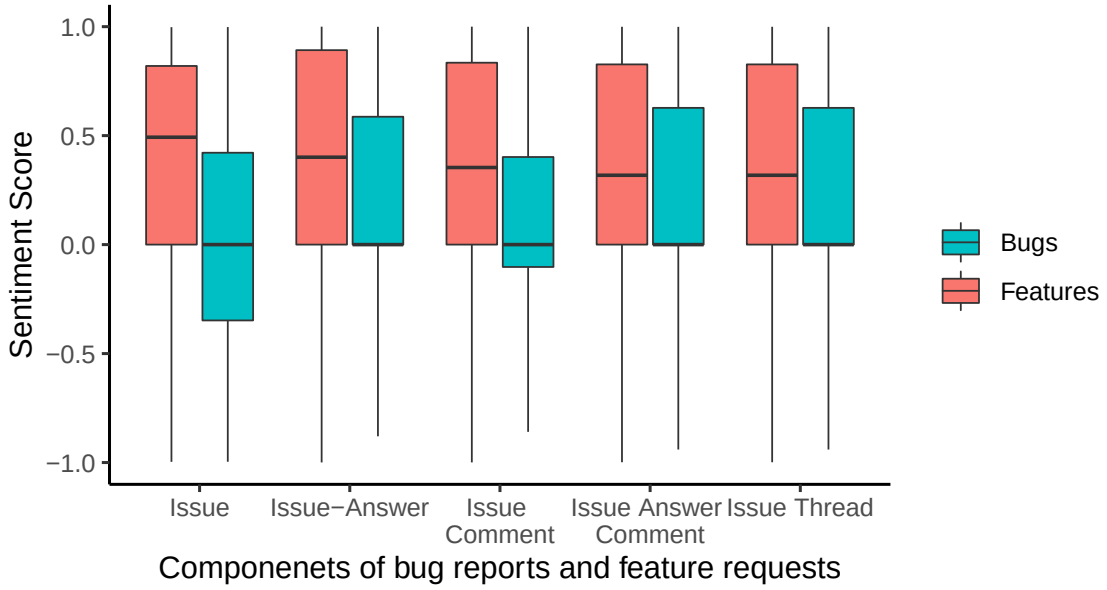

Figure 5.9: Sentiment score of different components of bug reports and feature requests. We use the term combined to refer to the sentiment score of a bug report or a feature request thread.

Feature request threads have higher sentiment scores than those of bug report threads, as shown in Fig 5.9. For example, bug report and feature request threads have a median of 0 and 0.32 compound sentiment scores respectively. Cohen's D for this analysis is -0.2500107 (small) and the results are statistically significant ($P \leq 40.05$). The sentiment score of many of the feature request components lies beyond the 0.05 threshold value compared to that of bug reports, indicating that more feature requests components express positive sentiments and many of the bug reports express a neutral sentiment.

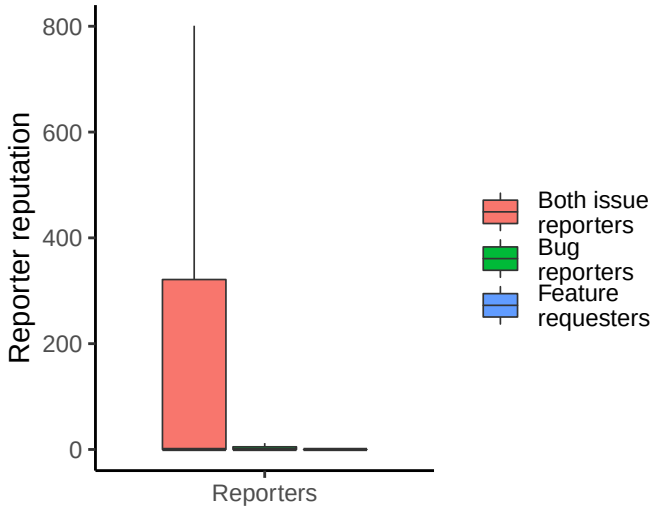

Figure 5.10: Reputation of issue reporters at the time that they reported the studied issues.

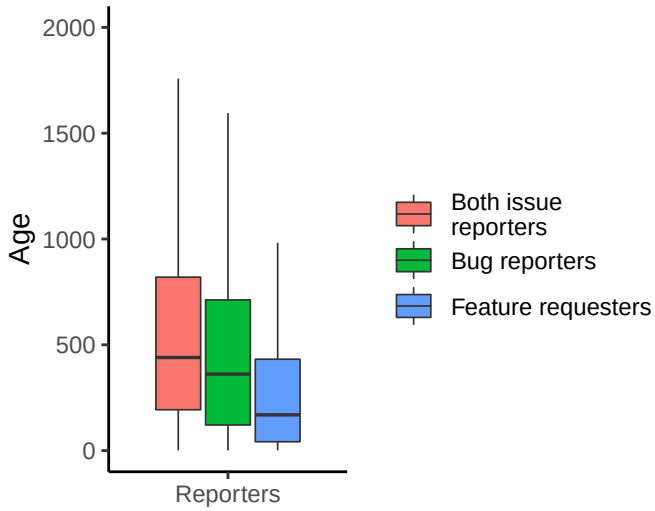

Figure 5.11: Age of issue reporters at issue reporting time (in number of days).

5.3.3 Do bugs and features differ in terms of participating users?

Approach: We analyze the characteristics of the users participating in bug reports versus feature requests along two different aspects: a) characteristics of users who report bugs or request features b) characteristics of users who participate in the discussion of bug reports or feature requests through answering, commenting or voting. We also compare the users participating in bug reports and feature requests by considering their reputation (users can earn reputation points through their contributions across the Stack Exchange websites) and age (i.e., the length of their membership, measured in days, in the Meta Stack Exchange site). Stack Exchange only provides the latest reputation score of any user. To mitigate this issue, we calculate the reputation score of a user (u) at any time (t) by considering the number of asked questions, the number of provided answers, and the number of received votes on those asked questions or provided answers prior to the time t. To do this, we follow the Stack Exchange's official guidelines for calculating reputation score[4].

Results: Users who report both bug reports and feature requests are high reputation users as compared to users who only report bug reports or feature requests. Figure 5.10 shows that the reputation of such users is high. To estimate the knowledge of a reporter in terms of issue reporting, we analyze the age and reputation of reporters. Figure 5.11 and Figure 5.10 shows that users who report both bugs reports and feature requests are more experienced than ones who only report bugs, or only request features. In general, bug reporters have been active for a much longer time than feature requesters, and hence they are higher reputation users.

More people contribute to feature requests than to bug reports. Figure 5.13 shows

[4] https://stackoverflow.com/help/whats-reputation

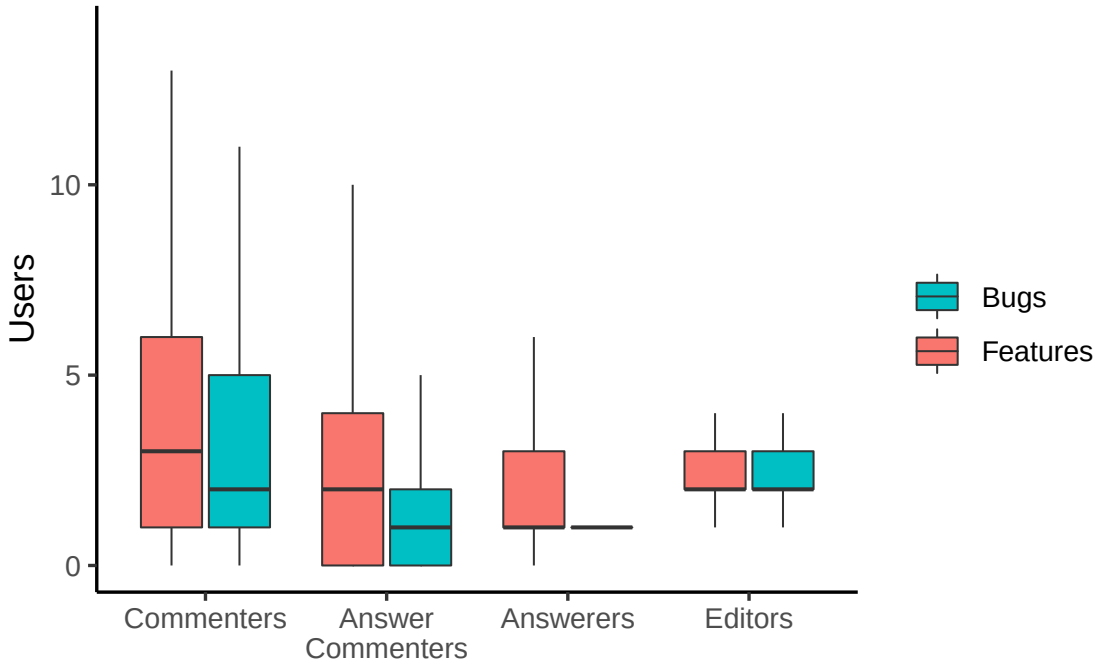

Figure 5.12: Number of people contributing to bug reports and feature requests of Stack Exchange

that feature requests have more contributing community members as compared to bug reports. We suspect that in a system where traffic is driven by community interest, bug reports have a limited amount of contributions (just reporting and fixing the bug). Whereas discussions around feature requests are often more elaborate and have much richer discussion as we observed as well through out analysis of the content of bug reports and feature requests (see Section 4.6). We also measure the number of unique users who are involved in contributing to different components. Figure 5.12 shows that in general, more users are involved in commenting on issue reports than on answering, or commenting existing answers. Since bugs reports get less answers and answer-comments as compared to feature requests, the number of answers and

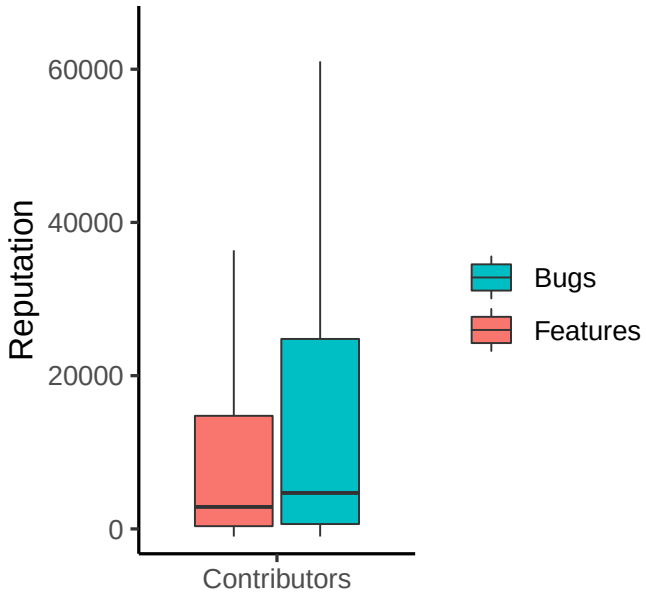

Figure 5.13: Reputation of people contributing to bug reports and feature requests of Stack Exchange

answer-comments is also lower for bug reports than for feature requests.

Summary of RQ1

Bug reports and feature requests differ in terms of readability and sentiment scores. Feature requests receive more answers than bug reports and a higher proportion of them are likely to be downvoted (33.0% of feature requests and 11.7% of bug reports receive downvotes). Feature requests are also longer than bug reports, possibly due to the richer discussions around features with more users contributing to feature requests than bug reports as well users expressing positive sentiment in feature requests. Users who report and contribute to bug reports have higher reputation than those who contribute to feature requests.

5.4 RQ2: Can we automatically identify bug reports from feature requests?

Motivation: From RQ1, we observed that bug reports and feature requests differ in terms of various dimensions (i.e., "community contributions", "content characteristics" and "participating users"). More specifically, in Section 5.3.1 and Section 5.3.3, we observe that the participating users in a Q&A platform interact and contribute to bug reports and feature requests differently. For instance, feature requests garner more answers than bug reports and typically higher reputation users contribute to both bug reports and feature requests. We hypothesize that such rich interactions that are enabled through the Q&A platform might help us discriminate between bug reports and feature requests without even considering the actual textual content of these issues. Hence, we examine the ability of a machine learning classifier that is build using the "community contribution" and "participating users" in discriminating between bug reports and feature requests.

Moreover from Section 5.3.2, we observe that the captured sentiment in the "textual content" dimension differs between bug reports and feature requests. For instance, feature requests have a higher compound sentiment score than bug reports. Therefore, we set out to examine if such differing sentiments (i.e., positive, negative, neutral and compound factors) in the "textual content" dimension along with the factors from the "community contribution" and "participating users" dimensions can improve the ability of our classifier in discriminating between bug reports and feature requests (again without any help from the actual textual content of these issues).

Finally, in this RQ, we add the actual textual content of these issues to our classifier

and examine whether the actual textual content can improve the ability of our classifier in discriminating between bug reports and feature requests.

Approach: To identify if the studied dimensions can enable the effective identification of bug reports from feature requests, we follow the steps given below:

(Step1) Data collection: we use a dataset of issue reports that have been tagged as bug reports or feature requests by Stack Exchange developers and community members.

(Step 2) Establishing dimensions: We identify the various factors in our dataset that make up each of the studied dimension. We do so, since some dimensions can be observed for both traditional and Q&A-backed issue management systems, whereas some dimensions can be observed only for Q&A-backed platforms. So we methodically segregate the dimensions as given below to form a clear segregation of factors. We also present all the studied dimensions, comprising of the set of factors that make up a dimension and their associated explanations in Table 5.2.

Traditional dimension (Traditional) comprises of the text in an issue report in a traditional issue management system. Equivalently, we use the text of the title, body, issue-comments, answers and answer comments (as they are available in both traditional and Q&A platforms) from our dataset as factors in this dimension. In order to use these factors in our classifiers, we convert them into tf-idf vectors. We preprocess the text by removing stop words, performing stemming, and lemmatization. We use the python package *nltk*[5] to remove the stop words, and *Spacy*[6] for stemming and lemmatization. Finally, we convert these words into tf-idf vectors using the R package, *superml*[7]. To avoid undue complexity, we only used the top most occurring 5,000 words in our tf-idf

[5] https://www.nltk.org/
[6] https://spacy.io/
[7] https://cran.r-project.org/web/packages/superml/superml.pdf

model.

Q&A Sentiment (QASentiment) Dimension is comprised of the sentiment factors about the text of the issue report that we could only extract from issue reports in a Q&A platform. For instance, the *textual_pos_ans* factor records the positive sentiment that is exhibited in the answer for a reported issue in a Q&A-backed platform. We only consider the sentiment factors in this RQ, as from Section 5.3.2, we observe that sentiment factors are the primary distinguishing factors between bug reports and feature requests in the "content characreteristics" dimension.

Traditional Sentiment (TradSentiment) comprises of sentiment factors that are derived from the issue report components, that are available in both traditional and Q&A-backed issue management systems. The sentiment across the body of the report is noted similar to that of the QASentiment dimension, however all the other text available in answers, comments and answer comments are considered together to generate a *ResponseThread* sentiment, as mentioned in Table 5.2. We do so, as traditional systems typically do not distinguish between comments and answers.

Q&A contribution (QAContributions) Dimension is made up of factors that record the contributions that are made to a given issue report in a Q&A-backed issue management systems in the form of activities like editing, commenting, answering and voting as giving in RQ1.

Q&A User (QAUser) Dimension is made up of factors pertaining to the participating users in a Q&A-backed issue management platform. Such factors could not be observed on traditional systems, though the traditional systems records the users who commented on a issue report similar to RQ1.

(Step 3) Classifier construction: We train classifiers on the specified dimensions in Table 5.3 to distinguish between bug reports and feature requests and note down the performance (in terms of AUC) and the feature importance scores similar to Rajbahadur et al. (2019); Ghotra et al. (2015); Lessmann et al. (2008). However, before building our classifiers, we remove the correlated and redundant variables using the R package *Hmisc*[8] as prior studies show that presence of correlated and redundant feature affects the interpretation of a classifier Jiarpakdee et al. (2019); Tantithamthavorn and Hassan (2018). As used by McIntosh et al. (2016); Rajbahadur et al. (2017), we used the threshold value of 0.7 to qualify highly correlated factors. To ensure the statistical robustness of our observations, we repeated the aforementioned process with 100-out-of-sample bootstrap similar to Rajbahadur et al. (2019, 2017); Tantithamthavorn et al. (2016). In each of the bootstrap iterations, the AUC and the feature importance scores are noted. For each iteration, a classifier is trained on data that was re-sampled with replacement, then classifier is tested using out-of-bag data points (i.e., data points that weren't used in the training of the classifier).

We built Random Forest classifiers because of their robust nature, and their ability to handle categorical and continuous explanatory factors Tian et al. (2015). We used the R package,"*ranger*"[9] and set the number of trees to be 500. Finally, we used the *Gini Importance technique* Menze et al. (2009) that is available in the "ranger" package to generate the feature importance scores.

(Step 4) Performance evaluation: We evaluate the following metrics for all our model: AUC, accuracy, precision, recall, and F1-Score. We used the Wilcoxon (1992) signed-rank test to test if the 100 AUC scores that are generated by the different classifier are

[8] https://cran.r-project.org/web/packages/Hmisc/Hmisc.pdf
[9] https://cran.r-project.org/web/packages/ranger/ranger.pdf

significantly different. Furthermore, we used *Cohen's d* Kelley and Preacher (2012) to measure the effect size of the difference.

(Step 5) Factor importance evaluation: Finally, to ascertain the factors that are influential in identifying bug reports from feature requests, we compute the feature importance ranks. We do so by taking the 100 feature importance scores that are generated by the **Gini importance method**. We then use the Tantithamthavorn et al. (2017) ScottKnottESD to generate the feature importance ranks similar to Rajbahadur et al. (2019).

Results: The QAContributions and QAUser dimensions by themselves—without any details about the textual content of an issue report—are capable of identifying bug reports from feature requests with a median AUC of 0.77% . Such a result highlights the value of the rich interactions around issue reports on a Q&A platform (an AUC of 0.5 indicates random guessing). Surprisingly, the Q&A specific factors that make up these dimensions do not have any textual content in them. To further investigate, we present the top five important factors generated by the ScottKnottESD test for the various classifiers that we trained in Table 5.4. We observe that the voting behavior (on both issues and answers), the answerer reputation and the editing behaviour are the most influential factors for our classifier. These results have significant implications. For instance, these results are encouraging for situations when the classification of bug reports vs feature requests need to be performed on sensitive issue repositories where exposing the actual content of an issue report to the classifier (or other practitioners) is not an option Peters et al. (2015). In cases such as those, these results indicate that we could effectively identify bug reports from feature requests by just looking at the

Dimension Name	**Factors**	**Description**
Traditional	Top 5,000 tf-idf words	Tf-idf words from all the words that are used across all the components of an issue report (Title, Body, Issue-comments, and answer-comments)
TradSentiment	trad_pos_Body, trad_neg_Body, trad_neu_Body, trad_comp_Body	Positive, negative, neutral and compound sentiments of the Body of an issue report.
	trad_pos_ResponseThread, trad_neg_ResponseThread, trad_neu_ResponseThread, trad_comp_ResponseThread	Positive, negative, neutral and compound sentiments of all issue-comment, answers and answer-comments.
QASentiment	ques_pos_body, ques_pos_comment, ques_pos_answer, ques_pos_answerComment	Positive sentiments expressed that is expressed in the issue report body, issue-comments, answers, and answer-comments
	ques_neg_body, ques_neg_comment, ques_neg_answer, ques_neg_answerComment	Negative sentiments expressed through issue report body, issue-comments, answers, and answer-comments
	ques_neu_body*, ques_neu_comment*, ques_neu_answer*, ques_neu_answerComment*	Neutral sentiments expressed through issue report body, issue-comments, answers, and answer-comments
	ques_comp_body, ques_comp_comment, ques_comp_answer, ques_comp_answerComment	Compound sentiments expressed through issue report body, issue-comments, answers, and answer-comments
	ques_FRE_body, ques_FRE_comment, ques_FRE_answer, ques_FRE_answerComment	Flesch Readability Ease of issue report body, issue-comments, answers, and answer-comments

QAContributions	contrib_edits1day*, contrib_edits3days*, contrib_edits7day, contrib_edits_lifetime	Edits performed within 1 day, 3 days, 7 days, and over the lifetime of the issue.
	contrib_answers1day*, contrib_answers3days*, contrib_answers7day, contrib_answers_lifetime	Answers received within 1day, 3 days, and over the lifetime of the issue
	contrib_comments1day*, contrib_comments3days*, contrib_comments7day, contrib_comments_lifetime	Comments received 1 day, 3 days, and over the lifetime of the issue
	contrib_favorites, contrib_downVotes, contrib_upVotes, contrib_viewcount	Favorites, downvotes, upvotes, viewcount, of the issue report.
	contrib_upvotes_ans, contrib_downvotes_ans, contrib_score_comments, contrib_score_ansComments.	Upvotes and downvotes of answers; score of issue-comments and answer comments.
QAUser	user_editors_body, user_answerers, user_commenters, user_answerCommenters	Number of editors, answerers, commenters, and answer-commenters.
	user_reputation_answerer, user_reputation_reporter	Reputation of answerers, and reporter.

Table 5.2: Description of the different factors across the studied dimension.

QAContributions and QAUser factors without ever needing to look at the actual textual content of these issues.

The sentiment factors (of the QASentiment dimension) significantly improve the discriminative ability of our initial QAContributions + QAUser classifier by 5.4%. We observe that the QAContributions + QAUser + QASentiment classifier significantly outperforms (Wilcox Sum-Rank is less than 0.05) the QAUser+QAContributions classifier with a large effect size of 6.3 . Furthermore, from Table 5.4 we find that three out of the five most influential factors for the QAContributions + QAUser + QASentiment classifier belong to the QASentiment dimension. More importantly, the composite sentiment of an issue body, answers and the positivity of the issue body are important for our classifier. This finding matches with our analysis results in RQ1 where we found in Section 5.3.1 that feature requests usually exhibit higher compound sentiment scores than bugs reports.

Furthermore, the sentiment of the components that are only available in traditional systems (TradSentiment dimension)—that lack the granularity offered by the Q&A platform—are equally informative in identifying bug reports from feature requests as factors from the QASentiment dimension. The QAContributions + QAUser + TradSentiment classifier has a median AUC of 0.83% which is similar to that of QAContributions + QAUser + QASentiment classifier (a non-significant difference, p-value=0.8). In short, although the sentiment in issue reports is important, the granularity in which they are tracked (i.e., irrespective of the the fact that if it is tracked per Q&A component, or overall as in traditional components) does not matter.

The combined classifier that uses factors from all the studied dimensions identifies bug reports and feature requests with a median AUC of 0.89. From Table 5.3 we

observe that the Traditional + QAContributions + QAUser + TradSentiment classifier (i.e., combined classifier) outperforms all the other constructed classifiers including the Traditional + TradSentiment classifier (i.e., the Traditional classifier–as it contains factors that are available in both Traditional and Q&A platforms) by 0.01. The performance of the combined classifier is larger than the traditional classifier with a statistical significance (Wilcox Rank-sum is less than 0.05) and a large effect size of 2.58. It is interesting to note that, the combined classifier performs extremely well, outperforming all the other constructed classifiers. Such a result could be attributed to the fact that the sentiment factors and textual factors are both informative for identifying bug reports from feature requests.

Two of the top five most important factors in the combined classifier and three of the top five factors in the traditional classifier are sentiment factors from TradSentiment. From Table 5.4, we observe that the same factors (i.e., trad_compound sentiment of the issue body (trad_compound_Body) and positive sentiment of the issue body (trad_pos_Body)) are important for both the baseline and the combined classifiers. Such a result indicates the usefulness and the discriminate power that the sentiment of the various component of the issue report contains. In addition, the word–"fix" also carries a discriminative capability for identify bug reports from feature requests.

Q&A factors also play an important role in helping the combined classifier. Two of the top five important factors of the combined classifier are Q&A specific factors pertaining to the number of received upvotes by an issue (contrib_upvotes) and the number of unique answerers (user_answerers). In summary, in addition to the textual content of an issue report,the sentiment factors and Q&A factors are important factors in identifying bug reports from feature requests.

Dimension used	#Factors	AUC	Accuracy	Precision	Recall	F-Measure
Traditional	5,000	0.84	0.81	0.79	0.42	0.55
Traditional + TradSentiment	5,008	0.88	0.83	0.85	0.47	0.60
QAContrib + QAUser	19	0.78	0.79	0.72	0.41	0.52
QAContrib + QAUser + QASentiment	31	0.83	0.81	0.75	0.48	0.59
QAContrib + QAUser + TradSentiment	27	0.83	0.81	0.74	0.49	0.59
Traditional + QAContrib + QAUser + TradSentiment	5,027	0.90	0.84	0.88	0.50	0.63

Table 5.3: Median performance metrics that are obtained from bootstrapping 100 iterations.

Summary of RQ2

Q&A factors capture how users contribute to issues. Such factors can help us identify bug reports from feature requests with a median AUC of 0.78 —without evern considering the text of the issue report. Furthermore, considering the sentiment factors, the text available in the traditional issue reporting systems along with Q&A specific factors, we can identify bug reports from feature requests with a median AUC of 0.89% Furthermore, the sentiment factors and the Q&A specific factors are the most influential factors for our classifier. Finally, our results suggest that Q&A platforms are useful and needed as it provides the users with the required granularity to interact with bug reports and feature requests differently.

Classifier	**Rank 1**	**Rank 2**	**Rank 3**	**Rank 4**	**Rank 5**
Traditional	"fix"	"bug"	"implement"	"think"	"much"
Traditional + TradSentiment	"fix"	ques_comp_body	ques_pos_body	trad_comp_ResThread	"bug"
QAContributions + QAUser	contrib_upvotes	contrib_viewcount	contrib_upvotes_answer	rep_answerer	contrib_edits7days
QAContributions + QAUser + QASentiment	ques_comp_body	ques_pos_Body	ques_comp_Ans	contrib_upVotes	contrib_viewCount
QAContributions + QAUser + TradSentiment	trad_comp_body	contrib_upvotes	trad_pos_body	trad_comp_ResThread	user_answerers
Traditional + QAContributions + QAUser + TradSentiment	trad_comp_body	"fix"	trad_pos_body	trad_comp_ResThread	user_answerers

Table 5.4: Top 5 most important factors for each of the constructed classifiers

5.5 Threats to Validity

External validity: Threats to external validity are related to the generalizability of our findings. In this study, we performed a comparison between bug reports and feature requests. Our results may not generalize to other datasets. However, the leveraging of a a Q&A platform for bug management is quite rare to the prevalence of traditional bug management systems. Our studied dataset contains a large number of issue reports that have been carefully tagged as bug reports versus feature requests by Stack Exchange developers and community members, rather than researchers (ensuring the robustness of our findings). Thus, our studied dataset provides a rare opportunity to understand the differences between bug reports and feature requests and to evaluate the effectiveness of using Q&A specific factors in classifying bug reports and feature requests.

5.6 Chapter Summary

In this chapter, we study the issue reports of Stack Exchange to analyze the differences between bug reports and feature requests along various dimensions (i.e., community contributions, content characteristics and participating users). We find that bug reports and feature requests are different from each other. Feature requests not only get more answers than bug reports but also receive more downvotes. Moreover, feature requests are longer than bug reports. More users contribute to feature requests compared to that of bugs through comments and answers. Furthermore, higher reputation users contribute more to bug reports than feature requests.

In addition, we observe that Q&A factors by themeselves—without considering the

text of the issue reports—are capable of identifying the bug reports from feature requests with a median AUC of 0.78. In addition, we observe that factors derived from community contributions, the sentiment of the issue report content and the characteristics of the participating users, in addition to the available factors in traditional issue reporting systems, enable the automated identification of bug reports and feature requests with a median AUC of 0.90. Besides using the issue report's textual content, Q&A platform specific features help to separate bug reports and feature requests better. Furthermore, we observe that sentiment factors and the Q&A factors are the most influential factors that helps us identify bug reports from feature requests.

In essence, our study results show that bug reports and feature requests are not the same and they are managed differently in the Stack Exchange issue management system. Furthermore, we show that one can automatically and effectively identify bug reports from feature requests by considering the textual content of issue reports and Q&A platform specific features. Our study offers a rich empirical understanding of the differences between bug reports and feature requests (and their management). Finally, our results suggest that the additional features of Q&A platforms are leveraged well as they provide the additional granularity to interact with bug reports and feature requests differently. The developers of issue management systems, software practitioners and researchers can leverage such understanding to improve issue management processes in large software projects.

CHAPTER 6

Conclusions and Future Work

THIS chapter summarizes our work, and overview the major contributions of this thesis and potential avenues for future work.

In this thesis, we studied the issue management on a Q&A based platform to mitigate the challenges that are associated with traditional issue management platforms. We study the management of issues on the Q&A based platform of Stack Exchange and provide empirical evidence of the leveraging Q&A features in issue management. We encourage the designers of traditional issue management systems to introduce the "in-place-editing" feature, the division of community contribution into "answering and commenting", and enable the downvoting of issue reports. Providing reasons for such downvotes might help in *educating* issue reporters and mitigate the

challenges of inadequate/incorrect information in bug reports as noted by prior studies.

Also, prior studies notice that the gamified Q&A platform of Stack Exchange is effective in encapture community interest, and ensuring sustainable contributions, which can be leveraged in issue management. This also manifests in cleaner data for analysis of different factors of issue reports. In our exploratory study of bug reports and feature requests of Stack Exchange, we show that bug reports and feature requests are not the same and they are managed differently. Furthermore, we show that one can automatically and effectively classify bug reports and feature requests by considering the textual content of issue reports and Q&A platform-specific features. Our findings provide insights to further research on how to automatically classify bug reports and feature request and prevent the misclassifications of issue reports in practice.

6.0.1 Thesis Contributions

In this section, we discuss our major contributions: **The features in Q&A platforms can help improve bug report quality.** 57% of the edits were performed to improve the quality of bug reports, such as adding/correcting/clarifying essential bug-related information (e,g., observed behavior, and environment information). The commenting and answering features are used differently, i.e., commenting on a bug report provides an avenue for discussing bug related information, while answering offers an avenue for providing the solution and explaining the causes of those bugs.

Bug reports and feature requests can be automatically and effectively identified with the Q&A features playing an important role in such automated identification. We

observe that bug reports and feature requests can be automatically identified using the sentiment of issue reports, the text of the issue reports and Q&A factors.

6.0.2 Future Work

In this section, we discuss future research opportunities that we have not covered.

1. In this thesis, we only study the unique features of Stack Exchange Q&A based platform that allows users to directly edit a bug report (i.e., the in-place editing feature) instead of commenting about the bug report; use different contribution avenues (i.e., the answering and commenting features) to discuss reported bugs, and vote on those bug reports, answers, and their associated comments (i.e., the voting feature). However, researchers can leverage the issue report dataset in the Q&A platforms to perform automatic tasks in bug management systems, such as bug triaging.

2. The effectiveness of our approach to automatically identify bugs reports form feature requests should be tested on other available traditional issue management systems (e.g., Bugzilla) to ratify our findings. In particular the value of using the expressed sentiment in identifying a bug report from feature request in datasets for traditional issue management systems.

Bibliography

Anderson, A., Huttenlocher, D., Kleinberg, J., and Leskovec, J. (2012). Discovering value from community activity on focused question answering sites: a case study of stack overflow. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 850–858. ACM.

Antoniol, G., Ayari, K., Di Penta, M., Khomh, F., and Guéhéneuc, Y.-G. (2008a). Is it a bug or an enhancement?: a text-based approach to classify change requests. In *CASCON*, volume 8, pages 304–318.

Antoniol, G., Ayari, K., Di Penta, M., Khomh, F., and Guéhéneuc, Y.-G. (2008b). Is it a bug or an enhancement?: A text-based approach to classify change requests. In *Proceedings of the 2008 Conference of the Center for Advanced Studies on Collaborative Research: Meeting of Minds*, CASCON '08, pages 23:304–23:318, New York, NY, USA. ACM.

Anvik, J., Hiew, L., and Murphy, G. C. (2006). Who should fix this bug? In *Proceedings of the 28th International Conference on Software Engineering*, ICSE '06, pages 361–370, New York, NY, USA. ACM.

Arya, D., Wang, W., Guo, J. L. C., and Cheng, J. (2019). Analysis and detection of information types of open source software issue discussions. In *Proc. of the 41st IEEE/ACM International Conference on Software Engineering*, pages 454–464.

Asaduzzaman, M., Mashiyat, A. S., Roy, C. K., and Schneider, K. A. (2013). Answering questions about unanswered questions of stack overflow. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, pages 97–100. IEEE.

Badashian, A. S., Hindle, A., and Stroulia, E. (2016). Crowdsourced bug triaging: Leveraging q&a platforms for bug assignment. In *Proc. of the International Conference on Fundamental Approaches to Software Engineering*, pages 231–248.

Barua, A., Thomas, S. W., and Hassan, A. E. (2014). What are developers talking about? an analysis of topics and trends in stack overflow. *Empirical Software Engineering*, 19(3):619–654.

Bettenburg, N., Just, S., Schröter, A., Weiss, C., Premraj, R., and Zimmermann, T. (2007). Quality of bug reports in eclipse. In *Proc. of the OOPSLA Workshop on Eclipse Technology eXchange*, pages 21–25.

Bettenburg, N., Just, S., Schröter, A., Weiss, C., Premraj, R., and Zimmermann, T. (2008). What makes a good bug report? In *Proc. of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 308–318.

Bhattacharya, P. and Neamtiu, I. (2011). Bug-fix time prediction models: can we do better? In *Proceedings of the 8th Working Conference on Mining Software Repositories*, pages 207–210. ACM.

Breu, S., Premraj, R., Sillito, J., and Zimmermann, T. (2010). Information needs in bug reports: improving cooperation between developers and users. In *Proc. of the ACM Conference on Computer Supported Cooperative Work*, pages 301–310.

Buse, R. P. and Weimer, W. R. (2008). A metric for software readability. In *Proceedings of the 2008 international symposium on Software testing and analysis*, pages 121–130. ACM.

Chaparro, O., Lu, J., Zampetti, F., Moreno, L., Di Penta, M., Marcus, A., Bavota, G., and Ng, V. (2017). Detecting missing information in bug descriptions. In *Proc. of the 11th Joint Meeting on Foundations of Software Engineering*, pages 396–407.

Chen, T.-H., Nagappan, M., Shihab, E., and Hassan, A. E. (2014). An empirical study of dormant bugs. In *Proc. of the 11th Working Conference on Mining Software Repositories*, pages 82–91.

Chua, A. Y. and Banerjee, S. (2015). Answers or no answers: Studying question answerability in stack overflow. *Journal of Information Science*, 41(5):720–731.

Dal Sasso, T., Mocci, A., and Lanza, M. (2016). What makes a satisficing bug report? In *Proc. of the IEEE International Conference on Software Quality, Reliability and Security*, pages 164–174.

D'Ambros, M., Lanza, M., and Robbes, R. (2010). An extensive comparison of bug prediction approaches. In *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*, pages 31–41. IEEE.

Fischer, F., Böttinger, K., Xiao, H., Stransky, C., Acar, Y., Backes, M., and Fahl, S. (2017). Stack overflow considered harmful? the impact of copy&paste on android application security. In *Proc. of the IEEE Symposium on Security and Privacy*, pages 121–136.

Ghotra, B., McIntosh, S., and Hassan, A. E. (2015). Revisiting the impact of classification techniques on the performance of defect prediction models. In *Proceedings of the 37th International Conference on Software Engineering-Volume 1*, pages 789–800. IEEE Press.

Giger, E., Pinzger, M., and Gall, H. (2010). Predicting the fix time of bugs. In *Proceedings of the 2Nd International Workshop on Recommendation Systems for Software Engineering*, RSSE '10, pages 52–56, New York, NY, USA. ACM.

Herzig, K., Just, S., and Zeller, A. (2013). It's not a bug, it's a feature: How misclassification impacts bug prediction. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 392–401, Piscataway, NJ, USA. IEEE Press.

Hooimeijer, P. and Weimer, W. (2007). Modeling bug report quality. In *Proc. of the 22nd IEEE/ACM International Conference on Automated Software Engineering*, pages 34–43.

Huna, A., Srba, I., and Bielikova, M. (2016). Exploiting content quality and question difficulty in cqa reputation systems. In *International Conference and School on Network Science*, pages 68–81. Springer.

Hutto, C. J. and Gilbert, E. (2014). Vader: A parsimonious rule-based model for sentiment analysis of social media text. In *Eighth international AAAI conference on weblogs and social media*.

Jiang, H., Li, X., Ren, Z., Xuan, J., and Jin, Z. (2019). Toward better summarizing bug reports with crowdsourcing elicited attributes. *IEEE Transactions on Reliability*, 68(1):2–22.

Jiarpakdee, J., Tantithamthavorn, C., and Hassan, A. E. (2019). The impact of correlated metrics on the interpretation of defect models. *IEEE Transactions on Software Engineering*.

Just, S., Premraj, R., and Zimmermann, T. (2008). Towards the next generation of bug tracking systems. In *Proc. of the IEEE symposium on Visual languages and Human-Centric computing*, pages 82–85. IEEE.

Kamei, Y., Matsumoto, S., Monden, A., Matsumoto, K.-i., Adams, B., and Hassan, A. E. (2010). Revisiting common bug prediction findings using effort-aware models. In *2010 IEEE International Conference on Software Maintenance*, pages 1–10. IEEE.

Kelley, K. and Preacher, K. J. (2012). On effect size. *Psychological methods*, 17(2):137.

Lessmann, S., Baesens, B., Mues, C., and Pietsch, S. (2008). Benchmarking classification models for software defect prediction: A proposed framework and novel findings. *IEEE Transactions on Software Engineering*, 34(4):485–496.

Liu, X. and Zhong, H. (2018). Mining stackoverflow for program repair. In *Proc. of the 25th IEEE International Conference on Software Analysis, Evolution and Reengineering*, pages 118–129.

Maalej, W. and Nabil, H. (2015). Bug report, feature request, or simply praise? on automatically classifying app reviews. In *2015 IEEE 23rd international requirements engineering conference (RE)*, pages 116–125. IEEE.

Mamykina, L., Manoim, B., Mittal, M., Hripcsak, G., and Hartmann, B. (2011). Design lessons from the fastest q&a site in the west. In *Proc. of the SIGCHI conference on Human factors in computing systems*, pages 2857–2866.

Mani, S., Sankaran, A., and Aralikatte, R. (2019). Deeptriage: Exploring the effectiveness of deep learning for bug triaging. In *Proceedings of the ACM India Joint International Conference on Data Science and Management of Data*, CoDS-COMAD '19, pages 171–179, New York, NY, USA. ACM.

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

McIntosh, S., Kamei, Y., Adams, B., and Hassan, A. E. (2016). An empirical study of the impact of modern code review practices on software quality. *Empirical Software Engineering*, 21(5):2146–2189.

Meng, N., Nagy, S., Yao, D., Zhuang, W., and Arango-Argoty, G. (2018). Secure coding

practices in java: challenges and vulnerabilities. In *Proc. of the 40th IEEE/ACM International Conference on Software Engineering*, pages 372–383.

Menze, B. H., Kelm, B. M., Masuch, R., Himmelreich, U., Bachert, P., Petrich, W., and Hamprecht, F. A. (2009). A comparison of random forest and its gini importance with standard chemometric methods for the feature selection and classification of spectral data. *BMC bioinformatics*, 10(1):213.

Nguyen, T. T., Nguyen, A. T., and Nguyen, T. N. (2014). Topic-based, time-aware bug assignment. *SIGSOFT Softw. Eng. Notes*, 39(1):1–4.

Novielli, N., Calefato, F., and Lanubile, F. (2014). Towards discovering the role of emotions in stack overflow. In *Proceedings of the 6th international workshop on social software engineering*, pages 33–36. ACM.

Peters, F., Menzies, T., and Layman, L. (2015). Lace2: Better privacy-preserving data sharing for cross project defect prediction. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 1, pages 801–811. IEEE.

Pingclasai, N., Hata, H., and Matsumoto, K.-i. (2013). Classifying bug reports to bugs and other requests using topic modeling. In *2013 20th Asia-Pacific Software Engineering Conference (APSEC)*, volume 2, pages 13–18. IEEE.

Ragkhitwetsagul, C., Krinke, J., Paixao, M., Bianco, G., and Oliveto, R. (2019). Toxic code snippets on stack overflow. *IEEE Transactions on Software Engineering*, pages 1–1.

Rahman, M. M. and Roy, C. (2018). Effective reformulation of query for code search using crowdsourced knowledge and extra-large data analytics. In *Proc. of the IEEE International Conference on Software Maintenance and Evolution*, pages 473–484.

Rajbahadur, G. K., Wang, S., Kamei, Y., and Hassan, A. E. (2017). The impact of using regression models to build defect classifiers. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, pages 135–145. IEEE.

Rajbahadur, G. K., Wang, S., Kamei, Y., and Hassan, A. E. (2019). Impact of discretization noise of the dependent variable on machine learning classifiers in software engineering. *IEEE Transactions on Software Engineering*.

Rejaul, K. M. (2019). Key features recommendation to improve bug reporting. In *Proc. of the International Conference on Software and System Processes*, pages 1–4.

Sirres, R., Bissyandé, T. F., Kim, D., Lo, D., Klein, J., Kim, K., and Le Traon, Y. (2018). Augmenting and structuring user queries to support efficient free-form code search. *Empirical Software Engineering*, 23(5):2622–2654.

Tantithamthavorn, C. and Hassan, A. E. (2018). An experience report on defect modelling in practice: Pitfalls and challenges. In *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice*, pages 286–295. ACM.

Tantithamthavorn, C., McIntosh, S., Hassan, A. E., and Matsumoto, K. (2016). An empirical comparison of model validation techniques for defect prediction models. *IEEE Transactions on Software Engineering*, 43(1):1–18.

Tantithamthavorn, C., McIntosh, S., Hassan, A. E., and Matsumoto, K. (2017). An empirical comparison of model validation techniques for defect prediction models. (1).

Tian, Y., Ali, N., Lo, D., and Hassan, A. E. (2016). On the unreliability of bug severity data. *Empirical Software Engineering*, 21(6):2298–2323.

Tian, Y., Nagappan, M., Lo, D., and Hassan, A. E. (2015). What are the characteristics of high-rated apps? a case study on free android applications. In *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 301–310. IEEE.

Wang, S., Chen, T.-H. P., and Hassan, A. E. (2018). How do users revise answers on technical q&a websites? a case study on stack overflow. *IEEE Transactions on Software Engineering*, pages 1–1.

Weiss, C., Premraj, R., Zimmermann, T., and Zeller, A. (2007). How long will it take to fix this bug? In *Proceedings of the Fourth International Workshop on Mining Software Repositories*, MSR '07, pages 1–, Washington, DC, USA. IEEE Computer Society.

Wilcoxon, F. (1992). Individual comparisons by ranking methods. In *Breakthroughs in statistics*, pages 196–202. Springer.

Wong, E., Yang, J., and Tan, L. (2013). Autocomment: Mining question and answer sites for automatic comment generation. In *Proc. of the 28th IEEE/ACM International Conference on Automated Software Engineering*, pages 562–567.

Wu, Y., Wang, S., Bezemer, C.-P., and Inoue, K. (2018). How do developers utilize source code from stack overflow? *Empirical Software Engineering*, pages 1–37.

Xuan, J., Jiang, H., Ren, Z., and Zou, W. (2012). Developer prioritization in bug repositories. In *Proceedings of the 34th International Conference on Software Engineering*, ICSE '12, pages 25–35, Piscataway, NJ, USA. IEEE Press.

Zhang, H., Wang, S., Chen, T. P., Zou, Y., and Hassan, A. E. (2019). An empirical study

of obsolete answers on stack overflow. *IEEE Transactions on Software Engineering*, pages 1–1.

Zhou, M. and Mockus, A. (2015). Who will stay in the floss community? modeling participant's initial behavior. *IEEE Transactions on Software Engineering*, 41(1):82–99.