AN ABSTRACT OF THE THESIS OF

Nitin Mohan for the degree of Master of Science in Computer Science presented on March 9, 2012.

Title: Managing Bug Reports in Free/Open Source Software (FOSS) Communities.

Abstract approved:

Carlos Jensen

Free/Open Source Software (FOSS) communities often use open bug reporting to allow users to participate by reporting bugs. This practice can lead to more duplicate reports, as inexperienced users can be less rigorous about researching existing bug reports. The purpose of this research is to determine the extent of this problem, and how FOSS projects deal with duplicate bug reports. We examined 12 FOSS projects: 4 small, 4 medium and 4 large, where size was determined by number of code contributors. First, we found that contrary to what has been reported from studies of individual large projects like Mozilla and Eclipse, duplicate bug reports are a problem for FOSS projects, especially medium-sized projects. These medium sized projects struggle with a large number of submissions and duplicates without the resources large projects use for dealing with these. Second, we found that the focus of a project does not affect the number of duplicate bug reports. Our findings point to a need for additional scaffolding and training for bug reporters of all types.

Finally, we examine the impact that automatic crash reporting has on these bug repositories. These systems are quickly gaining in popularity and aim to help end-users submit vital bug information to the developers. These tools generate stack traces and memory dumps from software crashes and package these up so end-users can submit them to the project with a single mouse-click. We examined Mozilla's automatic crash reporting systems, Breakpad and Socorro, to determine how these

integrate with the open bug reporting process, and whether they add to the confusion of duplicate bug reports. We found that though initial adoption exhibited teething troubles, these systems add significant value and knowledge, though the signal to noise ratio is high and the number of bugs identified per thousand reports is low.

Managing Bug Reports in Free/Open Source Software (FOSS) Communities

by
Nitin Mohan

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Presented March 9, 2012
Commencement June 2012

Master of Science thesis of Nitin Mohan presented on March 9, 2012

APPROVED:

Major Professor, representing Computer Science

Director of the School of Electrical Engineering and Computer Science

Dean of the Graduate School

I understand that my thesis will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my thesis to any reader upon request.

Nitin Mohan, Author

ACKNOWLEDGEMENTS

First and foremost, I thank the almighty for His blessings by surrounding me with such wonderful people and providing me valuable experiences and memories to cherish throughout my life.

I am really grateful to my advisor, Dr. Carlos Jensen, for his constant support and patience throughout my time at Oregon State University. I appreciate his efforts in advising me on my research by offering helpful suggestions and feedback.

I would like to thank my committee members Dr. Ronald Metoyer, Dr. Alex Groce and Dr. Arijit Sinha for their valuable time in attending my defense and examining my credentials. I wish to thank everyone from the FOSS communities for providing me the data and help to finish this thesis. I would also like to thank the faculty and staff from the EECS department and the Academic Programs, Assessment and Accreditation (APAA) office for their help I received directly or indirectly throughout my time at this university.

I thank my colleague Jennifer Davidson for her help in my research. I would also like to thank my colleagues at Human Computer Interaction (HCI) group at Oregon State for their valuable suggestions and advice and helping me draft this thesis.

There are no adjectives to portray the greatness of the family I belong to and the support I receive from every one of them in my daily life. My parents, Mohan and Jayalakshmi Mohan, my brothers and their families, Naren, Kavitha and Nivedha (my little niece) and Navin and Kamini have offered me immeasurable love and affection and been with me at every facet of my life. My special friend Nithya, who has been my moral support, and her family, deserve mention for their constant encouragement and motivation to realize my dreams. I pray for their well being and happiness throughout their lives.

CONTRIBUTION OF AUTHORS

Jennifer Davidson is credited for drafting most of the contents in the first manuscript. She wrote Perl scripts to download revision histories from Free/Open Source (FOSS) projects' bug database. She assisted in pre-processing the bug reports in the XML format. She assisted in verifying statistical results.

I am the primary author for the second manuscript. I also assisted in drafting and making edits in the first manuscript. I wrote Bash scripts to download bug repositories from FOSS projects and pre-process bug reports in XML format. I did most of the statistical analysis of data presented in the Results section in the first manuscript.

TABLE OF CONTENTS

Page

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF TABLES

Managing Bug Reports in Free/Open Source Software (FOSS) Communities

1. Introduction

Free/Open Source Software (FOSS) is an approach to software development where the users are granted freedom to study, modify or redistribute the source code of any project licensed under free software licenses. This open approach to software development has gained in popularity and has been adopted by many corporations. Software widely used in categories such as web browsers, operating systems, servers, and databases are primarily open source (Ghosh et al., 2006).

Typical FOSS project are made u of like-minded people from different organizations, cultures, geographic locations or ethnicities working together towards a common goal. Participation in contributing to projects is usually voluntary and is primarily motivated by altruism (“the gift culture”) (Raymond, 1998). He proposes that developers gift their source code expecting reciprocation from other contributors. Wu et al. (2007) surveyed 148 FOSS participants and found that the participants are continually motivated by satisfaction through participation, enhancing personal skills and capabilities, and career opportunities.

The software development model of open source projects is very different compared to traditional closed source approach. Generally, open source projects tend to be highly distributed while closed source developments are more often geographically contained. FOSS developers collaborate using text-based channels including forums, mailing lists, IRCs and bug repositories (Chung et al. 2010). Unlike closed source projects, roles in FOSS projects are defined by the interests and skills of individual contributors and their level of participation (by code contribution, communication between other members, etc.) in the community (Jensen and Scacchi, 2007). A developer can assume multiple roles within the community. According to Ye and Kishida (2003), every FOSS community can be visualized as a layered structure in which the amount of participation increases towards the center. This structure is referred to as “the onion model” and shown in figure 1.

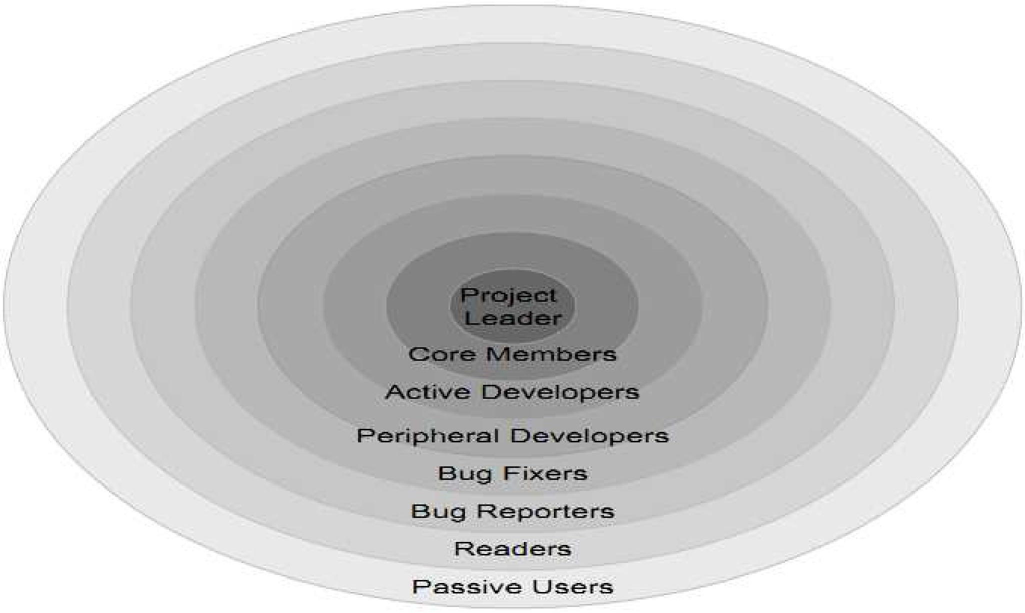

Figure 1. General Structure of an FOSS Community (Ye and Kishida, 2003)

Ye and Kishida identified eight major roles:

- Passive User: Only uses the software and does not participate in software development
- Reader: Reads the source code to understand the software
- Bug Reporter: Report bugs/feature enhancements to the developers of the project
- Bug Fixer: Fixes bugs submitted by reporters
- Peripheral Developer: Contributes feature to the software
- Active Developer: Actively involves in contributing features and fixing bugs
- Core Member: Coordinates the direction of design for the software
- Project Leader: Owns the project and involves in project direction

Any newcomer, referred to as “newbies”, who wishes to contribute, goes through “the joining process”. The newbies are required to have a good understanding of the technical and social nature of the project before starting to contribute. They generally “lurk” in the mailing lists, forums and IRCs to learn about the community and understand the technical jargons used by the members in the community to discuss about the issues. The lack of in-person meetings and the steep learning curve required during the initial stages makes the joining process more intimidating and drives away many people from contributing to open source.

Contributions to FOSS projects can be of any kind namely, giving feedback to developers through bug reports and feature requests, writing technical documentation, submitting bug fixes and code patches, managing the community, etc. FOSS communities often rely on users to participate in Quality Assurance (QA) by submitting bug reports and bug fixes and encourage future participation in the project. This “more eyes, more shallow bugs” approach (Raymond, 2000) is hugely beneficial to the projects since it provides a valuable mechanism for feedback from the users. Bug tracking systems like Bugzilla, Trac, Jira, etc. are deployed by the projects to manage the bug reporting process. These systems have in-built capabilities for managing bug reports like assigning bug IDs, time stamping, identifying severity, status and resolution of bugs and “triaging” the reports. Bug triaging is an important phase in bug management and refers to the process from when a bug is reported to when it is resolved, and the steps taken to manage that process. There has been quite a lot of research on predicting suitable developers to get assigned to bugs. Anvik et al. (2006) provided a semi-automated and a text classifier approach to build recommender systems for assigning bugs to developers. Matter et al. (2009) provided a vocabulary based approach to match bugs based on developer skills.

However, open bug reporting has its share of pitfalls. Allowing users to participate in QA can potentially lead to high bug traffic and higher probability of bug duplication (same bug reported by multiple users). Also, the reports from the users may be faulty, incomplete or low in quality and correctness since the users cannot be expected to have the knowledge and technical terms to describe bugs. Some FOSS projects deploy automatic crash reporting tools to help users in the bug reporting process. The purpose of these tools is to automatically gather important information from a software crash and prompt the user to report the crash to the developers.

This thesis examines how FOSS communities manage the bug duplication problem and how automatic crash reporting systems fit into this overall picture. To do this we have the following research questions.

RQ1. How significant of a problem are duplicate bug reports for FOSS projects?
RQ2. How does project size and focus affect the number and impact of duplicate bugs?
RQ3. What impact does automatic crash reporting systems have on FOSS projects?
RQ4. What overhead do automatic crash reporting systems add to the bug triaging process?
RQ5. Do crash reporting systems discourage user participation in the bug reporting process?

This thesis manuscript consists of two research papers. The first paper deals with the bug duplication problem. We analyzed the bug repositories of 12 FOSS projects: 4 small, 4 medium and 4 large where size was determined by the number of code contributors. We found that duplicate bug reports are a problem for FOSS projects, especially medium-sized, which struggle with a large number of submissions without enough resources of large projects. We also found that focus of the projects does not affect the number of duplicate bug reports. Our findings indicate a need for additional scaffolding for training bug reporters. The second paper deals with automatic crash reporting in Mozilla project and how it affects their bug management process. We performed quantitative analysis on the crash report dumps available publicly in Mozilla website. We also interviewed 5 developers and QA members to gather feedback about this system.

2 First manuscript: Coping with Duplicate Bug Reports in Free/Open Source Software Projects

Jennifer Davidson, Nitin Mohan, Carlos Jensen

School of EECS
Oregon State University
Corvallis, Oregon, 97331, USA
{davidsje, mohanni, cjensen} @ eecs.oregonstate.edu

Proceedings of the 2011 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), Pittsburgh, PA, USA
SESSION: Debugging and Program Understanding

2.1 Abstract

Free/Open Source Software (FOSS) communities often use open bug reporting to allow users to participate by reporting bugs. This practice can lead to more duplicate reports, as users can be less rigorous about researching existing bug reports. This paper examines how FOSS projects deal with duplicate bug reports. We examined 12 FOSS projects: 4 small, 4 medium and 4 large, where size was determined by number of code contributors. First, we found that contrary to what has been reported from studies of individual large projects like Mozilla and Eclipse, duplicate bug reports are a problem for FOSS projects, especially medium-sized, which struggle with a large number of submissions without the resources of large projects. Second, we found that the focus of a project does not affect the number of duplicate bug reports. Our findings indicate a need for additional scaffolding and training for bug reporters.

2.2 Introduction

Free/Open Source Software (FOSS) projects have to deal with different challenges, and consequently adopt different development practices than “traditional” closed-source software groups. In the FOSS community, the majority of contributors are volunteers, roles are less strictly defined, and most contributors assume multiple roles within projects. The volunteer labor force is both the strength and the Achilles heel of many FOSS projects. On one hand, volunteers allow projects to grow more rapidly, and involve users more directly. On the other hand, FOSS projects often have to deal with increased turnover, and occasional lack of training and coordination of contributors and resources.

Users provide a major resource for Quality Assurance (QA) in FOSS projects by submitting bug reports and code fixes. This is a role promoted by most FOSS projects, which rely on users to help evolve the software and encourage future participation in the project. This practice is at the heart of leveraging what has become known as Linus’ Law; “Given a large enough beta-tester and co-developer base, almost every problem will be characterized quickly and the fix will be obvious to someone”

(Raymond, 2000). Consequently, FOSS projects are heavy users of open bug reporting, which enables anyone from the community to submit bug reports.

While open bug reporting is beneficial, allowing users to participate in QA potentially means faulty, incomplete or duplicate bug reports. The quality and correctness of reports is a major issue when this process is opened to those with minimal QA experience. While the quality and correctness of reports is based on the experience of the users, and whatever scaffolding or training the project makes available, one expects duplicate bugs to be an increasing problem as project size and participation grows. While it is easy to spot duplicates in a small bug database, this gets harder as that database grows, meaning that both the rate of duplication and the cost of detecting bugs increases. We expect the user population to affect the rate of duplicate reports as well; if a project has a technical user base, there should be fewer duplicate reports. This idea was touched upon by Calvacanti et al. (2010), where they hypothesize that certain projects have fewer bug reports because they have fewer end-users.

FOSS communities do not always see duplicate bug reports as a bad thing. Ko and Chilana (2010) studied open bug reporting in the context of Mozilla. While duplicates were not the main focus of their paper, they found that they were astonishingly common, and in some cases developers find duplicate bug reports helpful. Duplicates, when identified, often provide additional information useful to narrowing down the source of a bug (especially if reports are incomplete). They also found that duplication can be used to identify the most critical bugs. Out of 100 duplicate reports, they found that 82 of them were identified as such the day they were reported, which means that Mozilla has very effective strategies for identifying and dealing with duplicate reports. This may, in part, explain why duplicates were not perceived as a major problem for the project.

Similarly, Bettenburg et al. (2006) studied duplicate reports in the Eclipse project. From a survey, they found that most developers did not consider duplicate bug reports to be a serious problem. They ran an experiment to show how merging the duplicate bug report with the “master” report improves the accuracy of a machine learning algorithm that triages bugs. They did not study the time spent on duplicate detection, but they did propose better search tools.

Mozilla and Eclipse are very atypical FOSS projects (Green et al., 2009 and Krishnamurthy, 2002). We therefore chose to look at how a more representative set of projects dealt with duplicate reports, whether these are considered helpful or problematic, and what factors influence the impact and perception that duplicate bug reports have on FOSS projects. To that end, our research questions were as follows:

RQ1. How significant of a problem are duplicate bug reports for FOSS projects?

RQ2. How does project size and focus affect the number and impact of duplicate bugs?

While previous research has provided a solid basis for addressing problems of bug management, most, if not all, focused on very large FOSS projects. Our contribution is analyzing a larger, more varied dataset for bug reporting practices. As most FOSS projects are small (Green et al., 2009 and Krishnamurthy, 2002), with large projects being extreme outliers, it is important to span the size gamut – from small to large projects. It is also important to examine technical- and consumer-oriented projects, as these groups likely approach bug reporting differently. Our study represents the diverse nature of the FOSS community by examining 12 projects that have from three to thousands of contributors to determine how they deal with bug triaging and duplicate bug reports.

The rest of the paper is organized as follows: Section 2 describes related work in FOSS project management, current research related to bug triage and duplicate bug reporting. Section 3 describes our methodology, including our project selection process and classification methods. Section 4 describes the results of our analysis of bug repositories. Section 5 continues with a discussion of our findings and possible shortcomings, and Section 6 concludes the paper with insights into minimizing the negative impacts of duplicate bugs.

2.3 Related Work

2.3.1 FOSS Workflow and Project Management

FOSS projects are volunteer-driven where people join to address common needs, for skill development, and to gain experience (Hars et al., 2001 and Yunwen, 2008). Most communication and collaboration is done using text-based channels, including IRC, wikis, forums, mailing lists and bug repositories (Chung et al., 2010 and Green et al., 2009). People who join projects often “lurk” on mailing lists before contributing code, learning the culture, conventions, and how they can contribute through observation and review of archived communication (von Krogh et al., 2003). Because of the volunteer nature of FOSS communities, there are few incentives for the community to engage in mentorship and training of newcomers until they prove themselves, as they may disappear from one day to the next. This unstructured and unsupervised joining process can be difficult to navigate.

Roles in FOSS projects are defined by the interests and skills of the individual contributor, the needs of the project, and the amount of code a person contributes as well as their level of participation in the community (Jensen and Scacchi, 2007). One of the main transition points from observer to developer is the submission of a patch. Submitting a patch means interacting with developers via a mailing list or a bug repository – oftentimes posting questions to mailing lists or forums (Ducheneaut, 2005). In most FOSS projects, the bulk of the code is contributed by a small percentage of the contributors (Ghosh et al., 2000 and Mockus et al., 2002).

2.3.2 Bug Triaging in FOSS

Bug triage is a commonly used term to refer to the process from when a bug is reported to when it is resolved, and the steps taken to manage that process. The main steps in bug triaging include determining whether reports are unique or duplicates, determining the reproducibility of bugs, the priority of bugs, deciding which developer should be assigned to a bug, and determining whether the issue has been resolved before closing any tickets. Most FOSS projects rely on bug tracking systems to track bugs and to manage the efforts to address the issues listed above.

Many FOSS projects engage in open bug reporting where anyone is able to submit a bug report regardless of experience or prior participation. Project websites usually provide guidelines to try and ensure bug reports meet minimum requirements in terms of information and completeness. Often these guidelines include instructions for searching to see if the bug already exists in the bug repository. This is done to reduce the number of duplicate bug reports. Despite these efforts, there are documented issues with duplicate reports and the quality of them.

Bettenburg et al. (2008) found a disconnect between the information users provide in bug reports and the information developers found useful. They advocate tackling poor quality bug reports through a scaffolding/mentoring process, in this case a plug-in for Bugzilla named CUEZILLA, which provides feedback to the user as he/she submits a bug report on how to provide more and better information about their issue. Bettenburg et al. (2007) also proposed a version called quZilla that would provide immediate feedback to the user about his/her bug report in the context of Eclipse (Bettenburg et al., 2007).

Calvacanti et al. (2010) ran two statistical studies on eight projects to investigate the duplicate bug reporting issue. The study combined private projects and FOSS projects, which is interesting considering that these projects would operate differently. All FOSS projects in their study fell into the “medium” or “large” category, leaving out smaller projects. They rated how various factors affected the number of duplicate bug reports and concluded:

- The number of Lines Of Code (LOC) is a weak/ moderate factor
- The size of the repository does not seem to be factor
- The project life-time does not seem to be a factor
- The amount of staff seems to be a moderate factor
- The amount of submitters does not seem to be a factor

While they base “size” on Lines of Code and number of bug reports, we instead based size on the number of code contributors. Furthermore, they looked at profiles of individual users to determine expertise level, while we looked at the project focus in a holistic manner.

Ko and Chilana (2010) focused on the Mozilla bug repository, examining the value of user-submitted bug reports. They also found the quality of bug reports lacking. However, this was offset in the eyes of developers by the fact that bug reporting served as a path for users to become more engaged, and possibly transition into contributors. Anvik et al. studied duplicate bugs and bug triaging in the Firefox and Eclipse projects. They found that detecting duplicate bugs was an issue: “It’s essential that duplicates be marked without developers having to look at them, there are just so many”. They concluded that there is a need for tools to help projects deal with duplicates and bug triaging.

To address some of these problems, work has been done using machine learning to automate duplicate detection (Wang et al., 2008) and automatic bug assignment (Anvik et al., 2005 and 2006). Jalbert and Weimer provided a classifier to detect duplicate bug reports as they were being reported (Jalbert and Weimer, 2008). While automatic duplicate detection is a useful approach to dealing with duplicate bugs, Bettenburg et al. (2008) point out that detection is not the last step in triaging duplicates. When a duplicate bug is marked as such the bug’s information is discarded. A study was conducted using the Eclipse bug repository that showed how duplicate bug reports included additional information useful in tracking down the source of a problem. Therefore, duplicates should not automatically be discarded, but rather new information should be merged into other reports. They also suggested improvements to bug tracking systems, including but not limited to better search tools for users, encouraging users to update existing bug reports, and allowing users to renew old bug reports. All of these suggestions might decrease the frequency of duplicates.

Another focus of study is bug triaging. Jeong et al (2009) created a visualization of “bug tossing” that showed how bug ownership gets “tossed” from developer to developer. The tool was created to shorten the time it takes for triagers to correctly assign a bug to a developer. Weiss et al. (2007) and Panjer (2007) studied how long it takes to fix bugs, or how long a bug stays open. They found that reducing the time it takes to fix a bug also limits the window for duplicates, which increases productivity.

2.4 Methodology

Our research goal was to build a deeper understanding of duplicate bugs in FOSS projects and the impact that these have on different types of FOSS projects. More specifically, we wanted to test the following three hypotheses:

H1. The more active the bug repository (the more bugs submitted per month), the more duplicates we see.

H2. Consumer-oriented projects will see a larger number of duplicates, as they have more inexperienced contributors.

H3. The more bugs, the longer it takes people to find the duplicate bugs (time needed to mark a bug as duplicate).

The method for sampling projects, as outlined in the Section 1, was based on project size (number of code contributors), focus (consumer vs. technical), and name recognition. As an example, some projects have an end-user focus, as with Mozilla, whereas other projects have a developer/admin user base, as with the Linux Kernel. We grouped these projects into one of two categories; “consumer” or “technical” based on a review of their community and website. We chose name recognition because it can be seen as a metric for projects that are mature in their development, and therefore have a good amount of information in their bug repositories. This diversity of projects allows for some generalizability of our results.

For our research we chose to focus on projects using the Bugzilla system because a) it is widely used by FOSS projects, b) bug information is easily downloadable for analysis, and c) it is the system that has been most widely studied in the past, which provided us the opportunity to readily compare our results to those of others. However, as discussed later, this may have skewed our selection of small projects because they may not need something as complicated as Bugzilla to manage their project.

Table 1. Project Selection. Data from Ohloh (ohloh.net) August 2010. Except Sudo, Open Watcom, and Eclipse. LOC for Fedora is articially low because Ohloh only counts RPM Spec files and patches.

	Contributors	**LOC**	**Focus**
Small			
Sudo	5	70,929	Technical
ClamAV	10	818,077	Consumer
Open Watcom	30	2,443,522	Technical
Nouveau	70	87,144	Consumer
Medium			
Apache httpd	102	686,316	Technical
Mandriva Linux	162	401,436	Consumer
Gcc	429	5,534,205	Technical
Fedora	677	66,963	Consumer
Large			
Mozilla Core	1,010	11,719,679	Consumer
Wine	1,181	2,028,254	Consumer
Linux Kernel 2.6	6,758	8,935,959	Technical
Eclipse	1,336	12,484,977	Technical

Table 1 gives an overview of the projects selected, their relative sizes, and their focus. With some exceptions detailed below, LOC and number of code contributors (over the entire lifetime of the project) were gathered from Ohloh (www.ohloh.net). For Sudo, we gathered the number of contributors from their webpage detailing “authors” of the project. We used the information on Open Watcom’s webpage listing “contributors”. Eclipse is a combination of many small projects. Ohloh separates each of these projects, so the number of contributors is artificially low. Therefore, we used the “total committers” column from the data table found on their website as the number of contributors. Contributor data was gathered August 2010. Note that in Table 1, we refer to contributor to mean code contributor. This metric was only used to classify the size of the project.

We chose thresholds for Small, Medium and Large projects based primarily on the number of code contributors. Small projects were defined as having less than 100 code contributors. Medium projects have less than 1,000 code contributors. Large projects have over 1,000 code contributors.

2.4.1 Analysis

For our analysis, XML files containing bug descriptions as well as HTML files containing bug revision histories were examined. Information from XML files was extracted using a script provided by Ko and Chilana (2010). To examine HTML files, we created perl scripts. To run statistical analyses on these two datasets, we used R. Most bug reports were publicly accessible. Some bug reports could not be examined because of insufficient permissions, internal database errors in the repository, or malformed content. Overall, these accounted for less than 5% of bugs in the repositories. We use the terms developer and reporter in this paper. These differ from the term code contributors. In this paper, developers have at least one bug assigned to them in the repository, while reporters have only ever reported bugs. We used ANOVA for all statistical inferences unless stated otherwise.

2.5 Results

2.5.1 Descriptive Project Statistics

The first step was to collect basic statistics about the size of the problem, including the number of bugs reported per month, the number of reporters, the number of developers, the percentage of duplicates (as marked by developers), and how these bugs are dealt with for each project (see Table 2).

Many projects invest time in screening reports before they are assigned to developers. In part, what they screen for are duplicates, but also whether the bug is for an older release of the software and to determine the appropriate owner for the bug. This process is more rigorous for some projects than others.

Table 2. Descriptive Project Statistics. Weighted Averages (Weighted With Total Number Of Bugs) Reported With (Std. Dev) Where Appropriate. Averages For Project Groups (Small, Medium Large) Given In Bold. Consumer Oriented Projects Are Shaded.

	Code Contributors	**Developers**	**Reporters**	**Developers/ Reporters**	**Bugs/month**	**Bugs/Dev**	**% Duplicate (total duplicates)**	**% of duplicates ID before assignment**	**Time to first assignment (Days)**	**Average time to close bug**	**Avg time to ID duplicates once assigned (Days)**
Small	**28.75**	**9**	**369.75**	**0.02**	**10.86**	**96.67**	**5.69**	**46.93**	**1.25**	**190.99**	**133.07**
Sudo	5	3	329	0.01	3.59	135.33	4.68 (38)	0.00	0.00 (0.00)*	136.36	65.58 (192.27)
ClamAV	10	10	555	0.02	32.81	157.50	6.22 (180)	45.56	0.01 (0.17)	58.74	25.07 (61.69)
Open Watcom	30	19	338	0.06	8.47	51.32	6.05 (110)	46.36	0.25 (3.26)	469.86	356.44 (517.14)
Nouveau	70	4	257	0.02	11.90	122.00	4.09 (20)	100	8.30 (25.21)	106.11	67.45 (96.07)
Medium	**342.5**	**434.75**	**11,559**	**0.04**	**427.51**	**169.29**	**14.15**	**19.02**	**1.66**	**320.92**	**218.75**
Apache HTTP	102	15	3,447	0.004	51.85	352.60	12.45 (1,371)	51.93	81.63 (269)	421.31	370 (413.40)
Mandriva Linux	162	222	7,419	0.03	569.20	248.70	13.29 (9,765)	24.83	0.05 (2.51)	217.18	102.88 (190.84)
Gcc	429	304	12,267	0.02	335.59	142.40	13.74 (5,948)	89.36	1.25 (20.87)	240.42	10.29 (131.10)
Fedora	677	1,198	23,102	0.05	1,512.75	159.10	14.54 (28,172)	1.61	0.01 (0.68)	366.47	290.57 (430.28)
Large	**2,571**	**1,630**	**37,598**	**0.04**	**1,889.43**	**140.25**	**19.57**	**25.06**	**0.18**	**332.55**	**518.98**
Mozilla Core	1,010	3,413	110,201	0.03	3,361.15	162.49	24.70 (137,001)	31.61	0.16 (6.29)	638.32	629.25 (809.41)
Wine	1,181	132	8,908	0.01	195.34	177.58	12.51 (6,172)	52.50	3.05 (29.60)	497.68	330.61 (483.64)
Linux Kernel 2.6	6,758	665	7,487	0.09	176.43	24.67	6.51 (1,068)	0.93	0.01 (0.02)	207.49	107.58 (198.14)
Eclipse	1,336	2,310	26,495	0.09	3,019.3	138.55	11.86 (37,958)	0.00	0.0 (0.02)	264.87	147.12 (331.67)

In terms of our first hypothesis: “The more active the bug repository, the more duplicates we see.” This does not seem to be true. Medium and large projects see a statistically significant jump in duplicates compared to small projects (p=0.009, F=10.37, df=1), but there was no statistically significant difference between medium and large projects (p=0.92, F=0.01, df=1). This may indicate a threshold between small and medium projects where reporters get overwhelmed. The Linux Kernel project, an exception to this rule shows us that effective management practices can significantly lower the rate of duplicates.

Some results were surprising. The rate of reporters to developers (people assigned bugs) fell into a relatively narrow range. The size of the project did not seem to affect this ratio (p=0.33, F=1.2485, df=2), nor did the consumer vs. technical focus of the project.

One exception was the Linux Kernel project, which follows very rigorous procedures for bug reporting and has the lowest reporter to developer ratio in our study (10:1). Another exception was the Apache httpd project, with a reporter to developer ratio in excess of 232:1. One thing that became apparent when looking at the data was that project culture and project management practices had a strong effect on how well projects deal with bugs and duplicates. One therefore should be careful when examining statistics and observe the community before making assertions.

In terms of our second hypothesis: “Consumer-oriented projects will see a larger number of duplicates, as they have more inexperienced contributors.” Surprisingly, was not supported in the statistical analysis. The rate of duplicate bug reports does not appear to be statistically linked with the focus of projects (consumer vs. technical) (p=0.34, F=0.99, df=1). That is, projects with a large number of non-technical users are no more likely to be burdened with more duplicate bugs than those with a large number of technical users.

In terms of our third hypothesis: “The more bugs, the longer it takes people to find the duplicate bugs (time needed to mark a bug as duplicate).” It was supported. The projects that had less than 10% duplicate reports were those that spent the least amount time closing duplicates that had slipped past the first screening, regardless of how rigorous that screening had been. Screening in this case means marking bugs as duplicates before they were assigned.

Excluding small projects, where the assignment of bugs to developers can be trivial due to the small number of developers, we do not see a big difference in the time spent before assigning bugs to developers and the success rate of screening duplicates in projects of different sizes. The data are inconclusive about a link between screening success and review time (correlation: p=0.4247, t=0.8322, df = 10, coeff=0.2545). Screening time probably does not account for the large variance in time used by some projects (such as Apache httpd) in assigning and resolving bugs.

This is where practices surrounding bug repositories for things like feature requests can skew the data. Another issue to keep in mind is that duplicate bugs may have been handled differently by projects. As previous research has shown, duplicate reports may provide helpful information (Bettenburg et al., 2008).

2.5.2 Bug Triaging Practices

We found differences in how projects manage and triage bugs, as well as how they use Bugzilla itself. For example, some projects log feature requests together with bug reports in Bugzilla. The process of triaging bugs varied across projects as well. Furthermore, because Bugzilla is FOSS (and therefore customizable), some projects changed the “Status” and “Resolution” categories to better fit their needs (see Table 3). The ability to customize is a core advantage of FOSS, and allows projects to support and define a custom processes. However, customization can make it difficult for developers working across projects (a common practice) to adapt to the idiosyncratic practices of a specific project.

Table 3. Bugzilla Status And Resolution States. Synonymous States Were Collapsed For The Purpose Of Analysis. Consumer Oriented Project Titles Are Shaded.

	Small				Medium				Large			
	Sudo	ClamAV 🟩	Open Watcom	Nouveau 🟩	Apache httpd	Mandriva 🟩	gcc	Fedora 🟩	Mozilla Core 🟩	Wine 🟩	Linux Kernel	Eclipse
Status												
Unconfirmed	X	X	X	X	X	X	X		X	X		X
New	X	X	X	X	X	X	X	X	X	X	X	X
Assigned	X	X	X	X	X	X	X	X	X	X	X	X
Reopened	X	X	X	X	X	X	X	X	X	X	X	X
Resolved	X	X	X	X	X	X	X		X	X	X	X
Verified	X	X	X	X	X	X	X	X	X	X	X	X
Closed	X	X	X	X	X		X	X	X	X	X	X
Needinfo		X		X	X			X	X		X	
Resolution												
Fixed	X	X	X	X	X	X	X		X	X		X
Invalid	X	X	X	X	X	X	X		X	X	X	X
Wontfix	X	X	X	X	X	X	X	X	X	X	X	X
Later	X	X	X		X			X			X	
Reminder	X	X	X		X							
Duplicate	X	X	X	X	X	X	X	X	X	X	X	X
Worksforme	X	X	X	X	X	X	X	X	X	X	X	X
Moved	X	X	X	X	X	X	X		X	X	X	X
Expired						X	X		X	X		
Notabug				X				X				
Notourbug				X		X		X				X

Table 4 gives an overview of the dynamics of these projects. As we can see, there are deep differences in terms of the relatively large number of reporters as seen in Table 2, and that most reporting is done by a small group of people. The majority of reporters post only one bug and a relatively small number of participants do the majority of the work. This is consistent with what Calvacanti et al. (2010) found. This held true across projects of all sizes.

Table 4. Bug Triaging Practices. Breakdown Of How Many People Engage In Extended Bug Reporting, Assigning Of Bugs To Developers, Reasigning Of Bugs To Developers, And Who Marks Bugs As Resolved (*Used “Closed” State For Fedora). Consumer Oriented Projects Are Shaded.

Project Name	From Table 2			Who Reported Bugs More Than Once?				Who Assigns Bugs?				Who Reassigns Bugs?				Who Marks Bugs As Resolved?*			
					How Many Bugs Reported?				How Many Times Per Person?				How Many Times Per Person?				How Many Times Per Person?		
	Reporters	**% Duplicate**	**% of dups ID before assign.**	# People	Median	Mean	St. Dev	# People	Median	Mean	St. Dev.	# People	Median	Mean	St. Dev.	# People	Median	Mean	St. Dev.
Small																			
Sudo	329	4.68	0.00	32	2	3.31	3.03	324	1	1.21	0.99	1	4	4.00	0.00	2	194	194	272.94
ClamAV	555	6.22	45.56	120	3	4.85	6.78	320	1	5.35	30.03	21	1	4.62	6.38	6	1.5	283.67	691.41
Open Watcom	338	6.05	46.36	102	3	5.13	8.21	82	1	3.01	6.24	11	2	11.36	16.97	1	781	781	0.00
Nouveau	257	4.09	100	79	3	3.81	3.11	225	1	1.88	2.23	7	2	2.29	2.14	20	1	17.45	73.33
Medium																			
Apache httpd	3,447	12.45	51.93	688	2	2.72	2.17	56	1	1.43	0.93	35	1	1.17	0.45	23	2	212.91	1,008.90
Mandriva	7,419	13.29	24.83	3,266	3	10.89	33.28	6,343	1	8.79	58.07	988	2	18.97	146.72	90	1	588.69	5,546.10
Gcc	12,267	13.74	89.36	3,518	3	6.09	16.83	3,815	1	6.65	72.38	195	2	10.28	52.84	64	1	495.41	3,914.80
Fedora	23,102	14.54	1.61	10,652	4	9.52	28.03	23,979	1	7.70	39.80	3,948	2	12.47	91.58	760	1	224.97	6,085
Large																			
Mozilla Core	110,201	24.70	31.61	31,111	3	6.05	16.11	59,551	1	6.94	68.71	6,832	2	34.34	241.71	1,534	1	337.21	12980.8
Wine	8,908	12.51	52.50	2,645	3	5.11	9.62	597	1	2.84	8.28	115	1	6.18	16.93	94	1	212.5	2,028.7
Kernel 2.6	7,487	6.51	0.93	1,960	2	4.11	6.47	7,469	1	2.12	13.05	1,260	1	6.79	46.29	15	2	339.27	1,304.8
Eclipse	26,495	11.86	0.00	12,158	4	25.004	133.3	27,371	1	11.24	87.28	5,704	2	49.31	350.09	430	1	709.49	14,609.9

More surprising is that most projects allow reporters to assign bugs to developers. However, appearances are sometimes deceiving. Many projects have dummy-accounts for groups to hold bugs until someone has a chance to review these and assign them to the right person. As we see, the group of people who reassigns bugs is much smaller than the group of reporters, though it is larger than the group of developers for most of the projects. The cause for this is twofold: Code contributors are not all part of the developer group (someone who has a bug assigned to them) but can sometimes reassign bugs to others. Furthermore, many projects have non-development users help triage bugs, such as the bug wrangler group in Mozilla. Both of these reasons help to inflate the number of people who reassign bugs.

One of the interesting findings is how much projects differ on core leadership practices such as who is authorized to mark bugs as closed. We see great variety from large projects like the Linux Kernel project, where 15 people close all bugs, to the Mozilla project, where more than 1,500 people close bugs. For Mozilla, this represents almost half the developer population compared to 2% of the Kernel developer population performing this quality control.

2.6 Discussion

We gathered statistical data on bug reporting and triaging practices from a range of FOSS projects. While not all of the statistics have been discussed in depth (we include these for others seeking to explore these questions), we have been able to show that this is a complex space worthy of further study and improvement.

Turning to the research questions, we have found strong evidence for RQ1; duplicates are plentiful, though their impact, or perceived impact is not clear. For the medium and large projects, save the Linux Kernel, the duplicate rate was over 10%. That constitutes a potential waste of effort, both for developers and users. For Mozilla, this constitutes over 494 reports per month that someone has to write, and someone else has to identify and discard as duplicates.

An example of the range of approaches for management can be seen when comparing Mozilla to the Linux Kernel, where the first seeks to widen participation, but invests resources in managing the 25% duplicate rate, whereas the latter keeps duplicates down through policy and training, in a smaller and more centralized organization.

There are a number of reasons why duplicate reports are bad for both users and the project community. Because duplicate reports are a product of a lack of knowledge of the current state of the project, reporters are only adding additional information by accident. The lack of details on an individual bug report, or the need to search through and manually synthesize the information from multiple reports, may outweigh any benefit from multiple reports, which is contrary to what Bettenburg et al. (2008) propose.

One of the previous findings that inspired our study was that duplicate reports were not a serious concern for projects, specifically for the Mozilla project. It is not clear whether projects routinely reflect on duplicates, their impact on current operations and how to reduce them, or whether reducing them is desirable. Our study was quantitative so we cannot assess the true impact of these duplicates on project members.

Despite these issues, FOSS projects use open bug reporting successfully to bring in the user community and involving them in QA (reporter to developer ratios between 100:1 and 10:1). However, projects have a hard time sustaining participation, as most users contribute only one bug report. While some of these users may transition to developers (which is not captured in our data), we believe it to be highly unlikely that such a transition would occur so rapidly.

The practice of open bug reporting carries a cost. In addition to potential problems associated with duplicate reports, screening and bug triaging is required on behalf of projects to manage the large amount of reports. We see that dealing with an avalanche of untrained reporters may cause problems, especially for medium-sized projects.

These projects have the highest bug to developer ratios and virtually the same high duplicate rates as large projects, without access to the resources larger projects have.

To answer RQ2, involving non-technical users did not result in a larger number of duplicate bug reports, which is somewhat contrary to what was discussed by Cavalcanti et al., which determined that the expertise of participants of certain project was a factor in duplicate bug reporting. This was unexpected, as the prevailing theory was that duplication is in part due to poor practices amongst end-users. As we did not look at individual reporters, it is still possible that technically skilled users routinely write higher quality bug reports than end-users.

As we expected, the final stages of bug triaging are typically tightly controlled; closing bugs is handled by a small group of people in most projects. Because of the high number of bug reports, it would be worthwhile to study how to make the final steps of the bug triaging process more manageable.

Finally, our study shows some of the dangers associated with exclusively studying large projects like Mozilla and the Linux Kernel, as there are dramatic differences in terms of practices and resources. Looking at the data we gathered should convince the reader that we must be very careful about generalizing from studies of large projects.

2.6.1 Threats to Validity

While we analyzed a broad range of projects, it is always difficult to make generalizations about a diverse movement as FOSS. While we believe that our sample is good in that it includes projects of different sizes, and that both consumer-oriented and technical-oriented projects were represented, there were limitations to our methodology and selection criteria.

For technical reasons we only sampled projects that used Bugzilla. We did this to simplify and unify analysis, as we did not want to have to perform custom analysis for a host of different types of repositories, and deal with the incompatibilities that might emerge. This decision however may bias some of our findings, because many small projects do not use a complicated tracking system such as Bugzilla. This may mean projects that we did study could be different from other, more common small projects.

Furthermore, we found that projects used Bugzilla in different ways. Projects triage bugs differently, and some allow feature requests and bug reports to be recorded in the same Bugzilla instance. Our analysis of individual bug reports may have been affected by this; the average response time to bugs may have been confounded by the inclusion of more long-term feature requests. There is concern about the effect of automated bug reporting on the bug reporting repository. Conversations with Fedora developers shed light on the issue, and showed that Automated Bug Reporting Tools (ABRT) do not artificially increase the number of duplicate bug reports. However, there has not been a discussion about the possibility of ABRT possibly reducing the number of duplicate bug reports, or of its possible merits in actively engaging end-users. In future work, we plan to investigate the impact of ABRT on the bug repository.

External to our analysis, the Bugzilla repositories may not accurately reflect the true state and workflow of projects. For example, if the triager did not follow the sequence of steps they claimed (i.e., not claiming bugs until they are addressed), the bug history may be inaccurate. Additionally, these are live, active projects, and therefore the numbers presented in this paper represent a snapshot in time. It is likely that these numbers have already changed, and will continue to change.

Finally, we chose to classify projects as small, medium or large based on the number of code contributors. This is only one of many possible ways of analyzing projects, and though we believe this is a valid classification given that we were interested in examining how projects were able to cope with the influx of new contributors, others may be equally valid. Classifying these projects by the size of their code-base for instance would have led to a different grouping of projects in our sample.

2.7 Conclusion and Future Work

Open bug reporting has a positive effect on participation, engages users in QA, and is fundamental to realizing Linus' Law; "Given a large enough beta-tester and co-developer base, almost every problem will be characterized quickly and the fix will be obvious to someone" (Raymond, 2000). While open bug reporting does engage a large group of users, most of their involvement is shallow, meaning that most only report one or a handful of bugs. While this is better than no help, the FOSS community would be richer if it managed to keep reporters involved.

Another important result is that consumer-oriented projects, which presumably have a greater proportion of non-FOSS trained non-technical reporters, did not have a significantly higher duplicate rate compared to technical projects. One would assume projects with a technical focus (such as Apache) would have more technical users familiar with good bug reporting practices. Although unexpected, this shows that there is room for additional scaffolding and support for reporters of all types. While current work on automatic duplicate detection and triaging is important, the Linux Kernel project shows that through proper training and management, the rate of duplication can be controlled. Therefore, it may be worth investing effort in more effective training materials and interactive scaffolding.

Despite a surprisingly high duplicate rate for some medium and large-scale projects, the communities seem able to deal with these with relative efficiency, screening a large number of these before assignment. It therefore remains to be seen how much of a burden these duplicates really pose to these communities.

In future work, we plan on interviewing and surveying developers, maintainers, as well as first time reporters to see how we can help them avoid duplicates where possible.

2.8 Acknowledgements

We thank all our participants and helpers at the projects studied for their time in helping us with data collection, parsing and interpretation, especially John Poelstra and Jiri Moskovcak from Fedora, and Andrew Ko from University of Washington. We also thank the HCI group at Oregon State University for help in drafting and revising this paper.

2.9 References

Anvik, J. “Automating bug report assignment.” In Proc. of the 28th Int. Conf. on Software Engineering (Shanghai, China, May 20 - 28, 2006). ICSE '06. ACM, New York, NY.

Anvik, J., Hiew, L., and Murphy, G. C. “Coping with an open bug repository.” In Proc. of the 2005 OOPSLA Workshop on Eclipse Technology Exchange (San Diego, California, October 16-17, 2005). ACM, NY NY, 35-39.

Anvik, J., Hiew, L., and Murphy, G. C. “Who should fix this bug?” In Proc. of the 28th international Conference on Software Engineering (Shanghai, China, May 20 - 28, 2006). ICSE '06. ACM, New York, NY, 361-370.

Bettenburg, N., Just, S., Schröter, A., Weiss, C., Premraj, R., and Zimmermann, T. “What makes a good bug report?” In Proc. of the 16th ACM SIGSOFT Int. Symp. on Foundations of Software Engineering (Atlanta, GA, November 09 - 14, 2008). SIGSOFT '08/FSE-16. ACM, New York, NY.

Bettenburg, N., Just, S., Schröter, A., Weiß, C., Premraj, R., and Zimmermann, T. “Quality of bug reports in Eclipse.” In Proc of the 2007 OOPSLA Workshop on Eclipse Technology Exchange (Montreal, Canada, Oct. 21-25, 2007). ACM, NY.

Bettenburg, N., Premraj, R., Zimmermann, T., Kim, S.,"Duplicate bug reports considered harmful … really?" IEEE International Conference on Software Maintenance,. ICSM 2008, pp.337-345, Sept. 28-Oct. 4 2008.

Cavalcanti, Y.C., Anselmo, P.M.S.N., Almeida, E.S., Cunha, C.E.A, Lucredio, D., Meira, S.R.L . “One Step More to Understand the Bug Report Duplication Problem.” In Proceedings of the 2010 Brazilian Symposium on Software Engineering (SBES '10). Washington, DC, USA.

Cavalcanti, Y.C., Almeida, E.S., Cunha, C.E.A, Lucredio, D., Meira, S.R.L "An Initial Study on the Bug Report Duplication Problem," 14th European Conference on Software Maintenance and Reengineering, 15-18 Mar. 2010.

Chung, E., Jensen, C., Yatani, K., Kuechler, V., and Truong, K. N.. “Drawing and sketching in Open Source design”, in IEEE Symposium on Visual Languages and Human-Centric Computing, 2010. VL/HCC 2010.

Ducheneaut, N. “Socialization in an Open Source Software Community: A Socio-Technical Analysis.” Computer Supported Coop. Work 14, 4 (Aug. 2005), 323-368.

Ghosh, R.A. and Prakash, V.V. “The Orbiten Free Software Survey.” First Monday, 5(7), July 2000, http://www.firstmonday.org/issues/issue5_7/ghosh/

Green, C., Tollinger, I., Ratterman, C., Pyrzak, G., Eiser, A., Castro, L., and Vera, A. “Leveraging open-source software in the design and development process.” In Proc. of the 27th Int. Conf. on Human Factors in Computing Systems (Boston, MA, Apr. 04 - 09, 2009). CHI '09. ACM, New York, NY.

Hars, A., Ou, S.., "Working for free? Motivations of participating in open source projects," System Sciences, 2001. Proceedings of the 34th Annual Hawaii International Conference on , vol., no., pp. 9 pp., 3-6 Jan. 2001.

Jalbert, N.; Weimer, W. "Automated duplicate detection for bug tracking systems," IEEE Int. Conf. on Dependable Sys. and Networks With FTCS and DCC, 24-27 June 2008

Jensen, C.; Scacchi, W. "Role Migration and Advancement Processes in OSSD Projects: A Comparative Case Study," Int Conf. on Software Engineering, ICSE’07. 20-26 May 2007.

Jeong, G., Kim, S., and Zimmermann, T. “Improving bug triage with bug tossing graphs.” In Proc. of the the 7th Joint Meeting of the European Software Engineering Conf. and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (Amsterdam, The Netherlands, Aug. 24 - 28, 2009). ESEC/FSE '09. ACM, New York, NY, 111-120.

Ko, A. J. and Chilana, P. K. 2010. “How power users help and hinder open bug reporting.” In Proc. of the 28th Int. Conf on Human Factors in Computing Systems (Atlanta, GA, April 10 - 15, 2010). CHI '10. ACM, New York, NY.

Krishnamurthy, S. “Cave or Community?: An Empirical Examination of 100 Mature Open Source Projects.” First Monday, 2002. URL: http://ssrn.com/abstract=667402

Mockus, A., Fielding, R. T., and Herbsleb, J. D. “Two case studies of open source software development: Apache and Mozilla.” ACM Transactions on Software Engineering Methodology 11, 3 (Jul. 2002), 309-346.

Panjer, L. D. “Predicting Eclipse Bug Lifetimes.” In Proceedings of the 4th int. Workshop on Mining Software Repositories. Int. Conference on Software Engineering. IEEE Computer Society, Washington, DC, (May 20 - 26, 2007)

Park, Y. 2008. Supporting the Learning Process of Open Source Novices: An Evaluation of Code and Project History Visualization Tools. Thesis.

Raymond, Eric S. "The Cathedral and the Bazaar." Computers & Mathematics with Applications 39.3-4 (2000).

von Krogh, G.,Spaeth, S., Lakhani, K. R. “Community, joining, and specialization in open source software innovation: a case study.” Research Policy, Volume 32, Issue 7, Open Source Software Development, July 2003,

Wang, X., Zhang, L., Xie, T., Anvik, J., and Sun, J. “An approach to detecting duplicate bug reports using natural language and execution information.” In Proc. of the 30th Int Conf on Software Engineering (Leipzig, Germany, May 10 - 18, 2008). ICSE '08. ACM, New York, NY.

Weiss, C.; Premraj, R.; Zimmermann, T.; Zeller, A."How Long Will It Take to Fix This Bug?," 4th Int. Workshop on Mining Software Repositories, MSR '07, 20-26 May 2007

Yunwen, Y., Kishida, K. “Toward an understanding of the motivation of open source software developers.” In Proc. of the 25th Int. Conf. on Software Engineering, 3-10 May 2003.

3 Second manuscript: The Impact of Automatic Crash Reports on Bug Triaging and Development in Mozilla

Nitin Mohan, Carlos Jensen

School of EECS
Oregon State University
Corvallis, Oregon, 97331, USA
{mohanni, cjensen} @ eecs.oregonstate.edu

Proceedings of the 2012 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), Innsbruck, Austria.
(Submission in Process)

3.1 Abstract

Free/Open Source Software projects often rely on users submitting bug reports. However, reports submitted by novice users may lack information critical to developers, and the process may be intimidating and difficult. To gather more and better data, projects can deploy automatic crash reporting tools, which generate stack traces and memory dumps when a process crashes. These systems potentially generate large volumes of data, which may overwhelm developers and discourage users from submitting traditional bug reports. In this paper, we examine Mozilla's automatic crash reporting system and how it affects their bug triaging process. We find that of all crash reports, less than 0.00009% end up in a bug report, but as many as 2.33% of bug reports have data from crash reports added. Feedback from developers shows that despite some problems, these systems are valuable. We conclude with a discussion of the pros and cons of automatic crash reporting systems.

3.2 Introduction

Free/Open Source Software (FOSS) projects often adopt different development practices to traditional closed source projects. Among the reasons for these differences is that FOSS contributors are often volunteers working together across the world. The lack of physical colocation, resources, and often ad-hoc project planning, calls for different development and project management practices, including bug triaging. Effective bug reporting and triaging is vital to any software project. The idea that enough eyes make all bugs shallow (Raymond, 2000) drives FOSS projects to involve everyone in bug triaging.

While there are advantages to involving users in bug triaging there are also possible downsides. Reports submitted by less experienced users can be incomplete or inaccurate (Bettenburg et al., 2008). Users may not use the right keywords to describe a bug, which can make it harder for developers to find the bug. A study by Davidson et al. (2011) also found that as the size of the reporting community grows, so does the ratio of duplicate bugs. Though duplicate bug reports are not always problematic

(Bettenburg et al., 2008 and Ko et al., 2010) duplicates potentially represent a waste of time and effort. Though projects publish guidelines for submitting bug reports, training and coordinating contributors is often an overwhelming task.

There have been a number of studies examining the bug triaging processes of FOSS projects. Bettenburg et al. (2008) surveyed 466 developers and users of the Apache, Mozilla and Eclipse projects and found a mismatch between what users reported and what developers found useful in bug reports. Breu et al. (2010) analyzed questions posed in 600 bug reports in the Mozilla and Eclipse projects to understand how developers and reporters collaborate. Both studies found a need for better ways to handle bugs and enhancing the quality of bug reports.

To gather more data, some projects have turned to automatic crash reporting systems. These systems are invoked when a process crashes. They gather stack traces, memory dumps, identifying the thread that caused the crash, product information, etc., and prompt users to submit these. Automatic crash reporting tools often ask users to add more descriptive information about the crash in order to assist developers in the triaging process, but there is no data on how many users provide such details, or how useful these are to the bug triaging process.

The terms “crash report” and “bug report” have distinct and different meanings. “Crash reports” refer to automatic error information gathered when a process crashes or quits unexpectedly. A “bug report” refers to a report filed manually by a user or developer about a fault or flaw of any type experienced with the software.

We are interested in understanding how automated bug reporting fits into current bug reporting and triaging practices, and if and how they add value to developers. To the best of our knowledge, no such study has been done. These are important questions, as deploying a crash reporting system is not without risks or costs. While these systems increase the volume of raw data available to developers, they might not necessarily make more information available to developers. The majority of crash reports refer to a small number of common problems. Furthermore, such a system could lead users to stop submitting traditional bug reports, feeling that they have already contributed, thus leading to a net loss of information for developers. This is especially true, as the issues covered by crash reports do not fully overlap with those in bug reports, including usability issues and missing features. To this end our research questions were:

RQ1. What impact do automatic crash reporting systems have on FOSS projects?

RQ2. What overhead do automatic crash reporting tools add to the bug triaging process?

RQ3. Do crash reporting systems discourage participation in the bug reporting process?

Given that there is a lot of diversity within the FOSS community, and there is no such thing as a "typical" FOSS project, this paper is intended to be a first investigation into these questions within the context of one of the leading FOSS projects, Mozilla.

The rest of the paper is organized as follows: We start with a review of research on bug triaging. Next we describe our methodology, and the Mozilla systems we studied. Section IV describes the results of our study, including excerpts from interviews with developers and users of these systems. Section V discusses our findings and the pros and cons of using these systems, as well as lessons to enhancing these tools. Section VI concludes with a summary of the key findings and future work.

3.3 Related Work

Automatic crash reporting systems have been used in many closed source systems (Apple (Apple, 2010), Windows (Kinshumann et al., 2011)). The most famous is the Windows Error Reporting (WER) system by Microsoft, described by Kinshumann et al. in [21]. The author mentions that "a bug reported by WER is about 5 times more likely to be fixed than a bug reported directly by a human". Kim et al. (2011) studied the WER system and provided "Crash Graphs" which present a high-level aggregated view of multiple crashes belonging in the same bucket.

There have been a few recent studies on Mozilla's automatic crash reporting system. Kim et al (2011) focused on prioritizing debugging efforts by predicting top crashes. Dhaliwal et al. (2011) proposed a grouping approach to group crash report triaging. They show that effective grouping of crash reports can reduce bug fixing time by 5%. Khomh et al. (2011) proposed the use of crash entropy values to prioritize crash types during triaging. These studies focus on a small subset of crash reports.

There has been quite a lot of work on automating and improving the bug triaging process. Bug triaging refers to the steps taken to manage a bug from the time the bug is reported to the time the bug is resolved. Anvik (2006) discussed a semi-automated approach to assigning bugs to developers through a recommender system. Anvik et al. (2006) proposed another text-based categorization that achieved between 57% and 64% accuracy for assignment of bug reports in the Eclipse and Firefox projects. Matter et al. (2009) proposed a vocabulary-based approach where developer expertise and bug vocabularies were matched. Tamrawi et al. (2011) designed a tool called “Bugzie” which offered a fuzzy set-based approach to automated bug assignment, and achieved 68% accuracy in predicting the 5 most suited developers. Jeong et al. (2009) created a tool that visualized “bug tossing,” showing how bug ownership got passed from developer to developer within a project in order to identify ‘tricky’ bugs and effective contributors.

Another topic examined by researchers has been duplicate bug reports. Ko and Chilana (2010) studied bug reports in the Mozilla project and found that though there was a large number of duplicate reports, these were often seen as helpful by developers. Duplicates could reflect the severity and priority of a bug. Bettenburg et al. (2008) studied the Eclipse project and found that most developers did not consider duplicate bug reports to be a serious problem.

Other studies have found problems with duplicate reports. Cavalcanti et al. (2010) studied 8 FOSS projects and found that duplicate reports negatively impacted the overall development process. They also identified factors that affect the frequency of bug duplication. Davidson et al. (2011) studied this problem in 12 FOSS projects of different size and focus. They found that medium-sized projects are most affected – they have to deal with the same number of duplicates as the large projects, but without their resources. However, they did not find a relationship between duplicates and whether the user base was more or less technical. Anvik et al. (2005) studied duplicates in Firefox and Eclipse and found that these were common and that there is a need for tools to detect these. Jalbert and Weimer (2008) presented a machine learning tool that identified duplicate bugs.

The quality of bug reports in FOSS projects is another important topic. Bettenburg et al. (2008) surveyed developers and users of the Apache, Mozilla and Eclipse projects and compiled a list of information that developers look for in a bug report. Based on this inventory, they developed a bug reporting system called CUEZILLA. This system provides a quality metric for bug reports and points to information that would enhance the quality of the report. Breu et al. (2010) analyzed 600 bug reports from the Mozilla and Eclipse projects and the information requests developers made of reporters, and found that there was a need for tools to structure and guide the reporting and information exchange process. Ko et al. (2006) examined the language of nearly 200,000 bug report titles to understand how people describe bugs. They also identified a need for tools that help reporters submit more structured reports, which could be automatically parsed.

3.4 Methodology

Our goal was to analyze the impact of crash reporting tools on bug triaging in FOSS projects. More specifically, we wanted to determine whether such systems lead to a net gain or loss in information, as they could discourage users from submitting more meaningful bug reports.

For our research, we examined Mozilla’s crash reporting system because a) Mozilla products have a large user base and an active developer community, b) the data needed for this study is publicly available, c) this is an extensively studied project, which allowed us to set our findings in context, and d) they have used a crash reporting system for an extended period of time, allowing procedures to develop and be adopted within the project.

3.4.1 The Breakpad/Socorro Crash Reporting System

Mozilla started using their current custom crash reporting system in 2008, coinciding with the release of Firefox 3. Currently, only their Firefox, SeaMonkey and Thunderbird projects use this system. It has two components – Breakpad and Socorro. Breakpad is an open source project started by Google. It runs as a thread in every instance of the Mozilla process. It is invoked when a crash occurs in any Mozillla's processes, collects the call stack and memory dumps from the process, finds the thread that crashed and sends the information to Socorro. The system prompts the users for additional information, which they can provide if they wish. Socorro is a python-based server system that aggregates and performs statistical analysis on the crash reports submitted to Mozilla. The Mozilla QA team processes these and either adds new bugs or amends existing ones.

3.4.2 Analysis

We collected daily crash report logs from March 2010 to October 2011. Due to the volume of data the system generates, older logs are not kept. We gathered bug information and bug revision histories from the start of the Mozilla project to October 2011 from their bug tracking system. Some of the reports were unavailable for analysis due to permission issues, internal database errors or malformed content. However, these only accounted for 5% of all bugs in the database.

To further evaluate the usefulness of Mozilla's crash reporting system we supplement the quantitative data with interviews of developers who worked directly with the system. A total of 5 developers participated in our study - 2 Socorro/Breakpad developers and 3 members of the Mozilla's QA team responsible for processing the reports. By examining perspectives of developers and users we can better judge the impact of this system and identify design changes that would improve such systems.

3.5 Results

A previous study of 12 FOSS projects (Davidson et al., 2011) found that Mozilla had a very active bug repository (around 3,361 new bugs reported per month) compared to other projects. They also found that the more active the bug repository, the more duplicates there were. They found that Mozilla was especially affected, with 24.7% of bugs submitted being marked as duplicates, significantly more than other projects studied. We were interested in finding the reason for this high duplicate rate, and whether the automatic crash reporting system lessened or amplified the problem.

3.5.1 Quantitative Results

First, we quantitatively analyzed the crash report logs from March 2010 to October 2011. We aggregated basic statistics, listed in Table 5, and compared to the activity in the bug reporting system over the same period.

Table 5. Mozilla Crash Reports (March 2010 - October 2011) And Bug Reports (July 1998 - October 2011) * Crash signatures added to database June 9, 2011

	Breakpad/ Socorro	**Bug Reports**
Avg. # of reports per month	96,131,054.5	4,048.4
% Duplicate	88.19%	22.68%
Avg. # of crash reports turned to bug reports per month	89.2*	
Avg. # of bug reports associated with crash report data per month		94.5
Days for crash reports to be associated with bug report (Avg)	230.87*	

Mozilla on average receives 96 million crash reports per month, they outnumber bug reports by more than 20,000:1. While these are very large numbers, one should keep in mind that there were an estimated 350 million Firefox users by early 2010, and between 15 and 20 million Thunderbird users. Of these 96 million crash reports Mozilla only processes a sample of 10%, biased towards reports with user-provided details. 88.19% of this sample is trivially classified as duplicates. This still leaves 1,135,308 reports to process per month. While this is a dramatic reduction, it is still a huge set to work through.

Remaining reports are manually classified as either duplicates, not critical, or not actionable. Of the remaining reports, 89.2 per month will be turned into one or more bug reports (data is limited to the period after June 9, 2011 when the project started tracking crash signatures in bug reports). As we explain below, that monthly average is heavily skewed. Of all crash reports, this accounts for only 0.00009% that are finally associated with a bug report, or 0.008% of unique crash reports sampled. However, if we turn this around, 2.334% of bug reports are either created or augmented with crash report data. Therefore, though there is a lot of waste, crash reports add significant value to Mozilla’s QA.

The introduction of a crash reporting system, and the volumes of data these can generate do come at a price. Developing effective strategies and tools to triage the data are essential to leverage these systems.

Figure 2 shows a plot of the report date of a crash against the date when these were associated with a bug (a new bug was created, or an existing bug was amended). Again, the data is limited to the period after June 9 2011, when the project started tracking these associations. In the 4.5 months for which we have data, the QA team matched 402 crash reports, or 89.2 per month. More importantly, though the majority of matched reports are recent (median 197.5 days), we see that a significant number have been in the queue for close to two years. Given that Mozilla has had six major releases in that time-frame, it shows that crash reports can help identify deep and fundamental bugs that can haunt software project for years. There is therefore a strong need to develop tools to not just help view reports more easily, but also help the QA team analyze the data more easily.

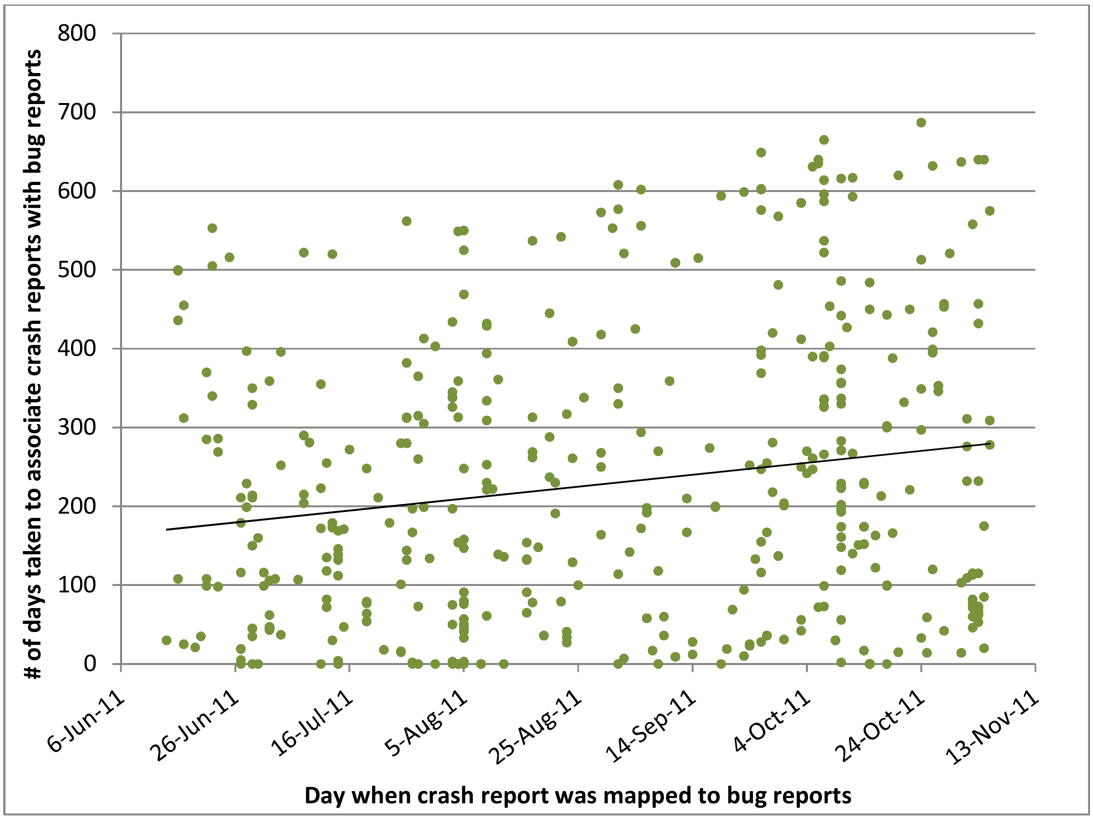

Figure 2, Time taken to associate crash reports with bug reports

Bugs and crashes are of course cyclical and affected by the development activity taking place at the time. When new versions of the software are released, we expect to see spikes (see Figure 3) (Kim et al., 2011). The match is not perfect however; adoption is not immediate, and there may be differences in quality control between releases. Also, because Mozilla’s products are platforms for other software (plugins and extensions), problems can spike as those are refreshed. From our conversations with developers, such spikes are not uncommon.

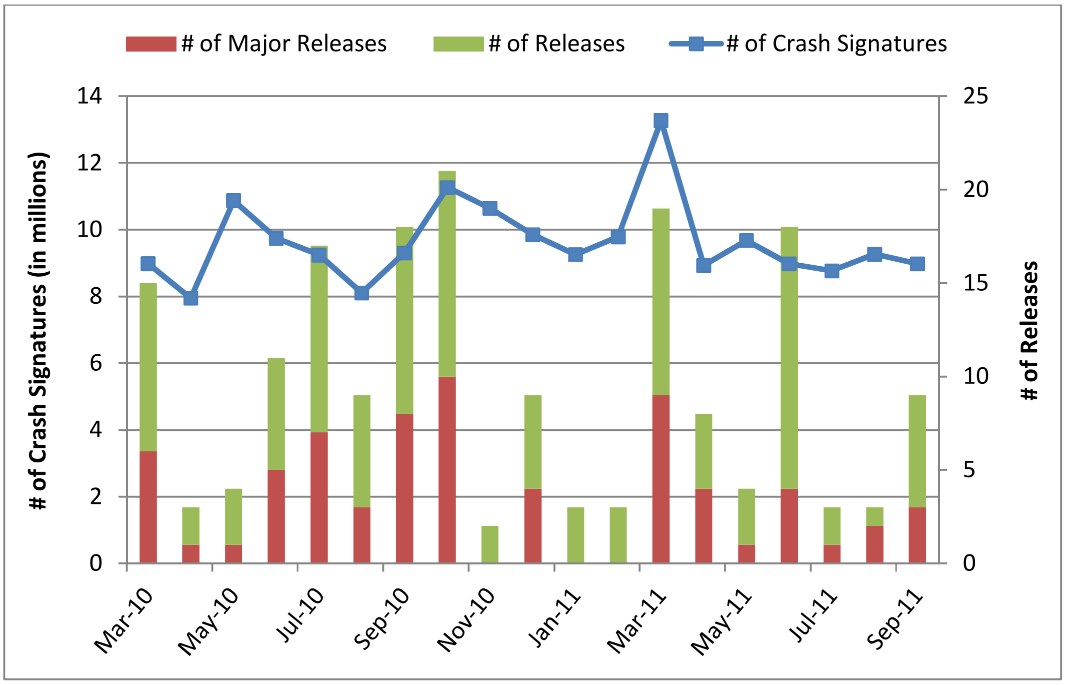

Figure 3, Crash signatures vs. software releases in Mozilla

In Figure 4 we can see long-term trends for bug reporting and duplication rates. The automatic crash reporting system was introduced in June 2008 (first red vertical line), and they switched to a rapid release cycle in April 2011 (2nd red line). It is important to note that though there is a strong downward trend in duplicate rates, this may be artificially inflated because identifying some duplicates can take a long time. The duplicate numbers should therefore be interpreted with caution.

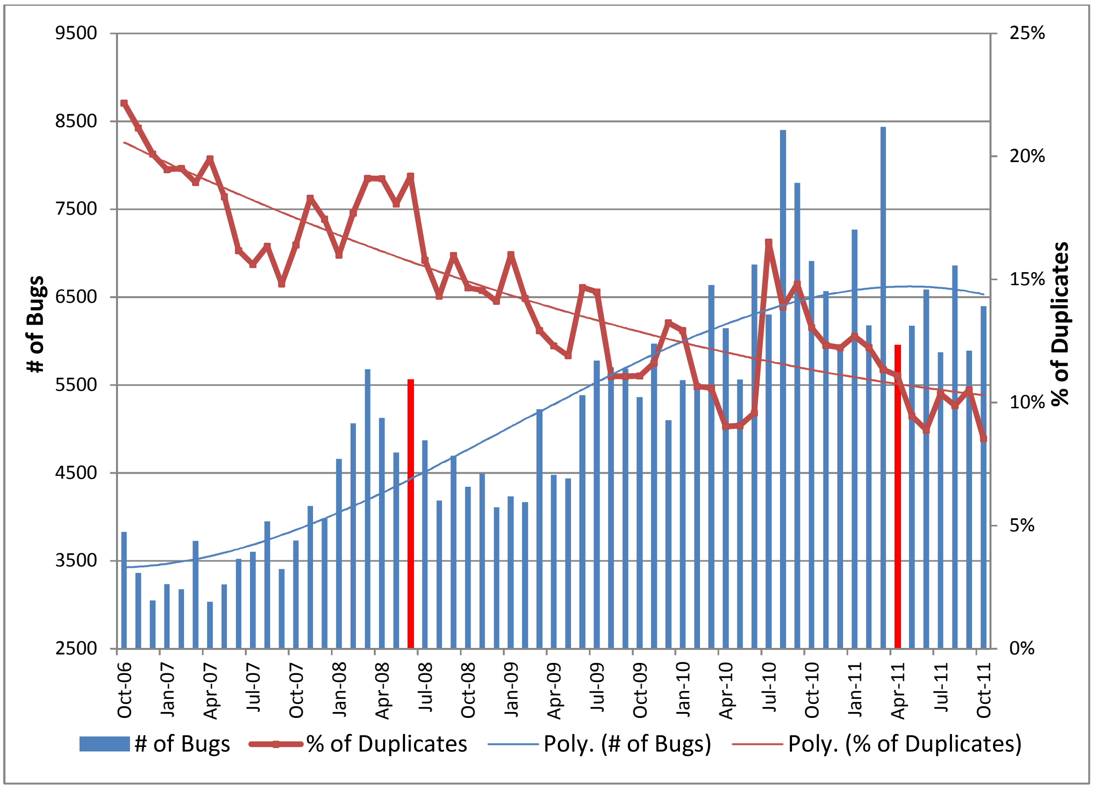

Figure 4, Temporal of view of bug activity in Mozilla. First red vertical bar indicates the introduction of the crash reporting system, and the second the transition to rapid release cycles

That said; we see a strong positive development in terms of reducing the number of duplicate bug reports within the project. As we can see from Table 6, this development has been statistically significant across the three project “periods”. In terms of data quality, we can therefore say that it does not appear that the introduction of the crash reporting system has interrupted a positive trend that was already in effect, the reduction of duplicate bug reports in Mozilla. While this is perhaps not surprising given the small number of crash reports that are turned into bug reports, it is a positive nonetheless.

Table 6. ANOVA Results. The Mozilla Project (October 2006 To October 2011 And Introduction Of Key Changes (Crash Reporting System June 2008 & Rapid Release Cycle April 2011)

	Pre vs Post-Crash System	Pre-Crash vs Rapid Release	Post-Crash vs Rapid Release
# of Bugs	**ANOVA(df=1, F=33.199, p<0.00001) Tukey(p<0.0001)**	**ANOVA(df=1, F=47.965, p<0.00001) Tukey(p=0.0001)**	**ANOVA(df=1, F=1.4081, p=0.2427) Tukey(p=0.408)**
% Duplicates	**ANOVA(df=1, F=96.333, p<0.00001) Tukey(p<0.0001)**	**ANOVA(df=1, F=126.89, p<0.00001) Tukey(p<0.0001)**	**ANOVA(df=1, F=15.187, p=0.00038) Tukey(p=0.0008)**

Another positive development is that though there was a slight dip in the number of bug reports immediately after the introduction of the crash reporting system, activity has since picked back up. We see an increasing trend in the number of bugs reported per month after the introduction of the automatic reporting system (ANOVA: df=1, F=33.199, p<0.0001). We can therefore conclude that though introducing the crash reporting system may have been disruptive, these issues were worked out.

As we see in Figure 5, the community of bug reporters has been continuously growing, and the community renews itself with new members, though the renewal rate seems to be in decline (ANOVA: df=1, F=41.01, p<0.0001). It is also worth nothing from this chart that though the rate of new reporters is relatively high, the growth of the regular commenter community is relatively slow. Most new contributors leave after posting a single bug report, as others have shown (Davidson et al., 2011).

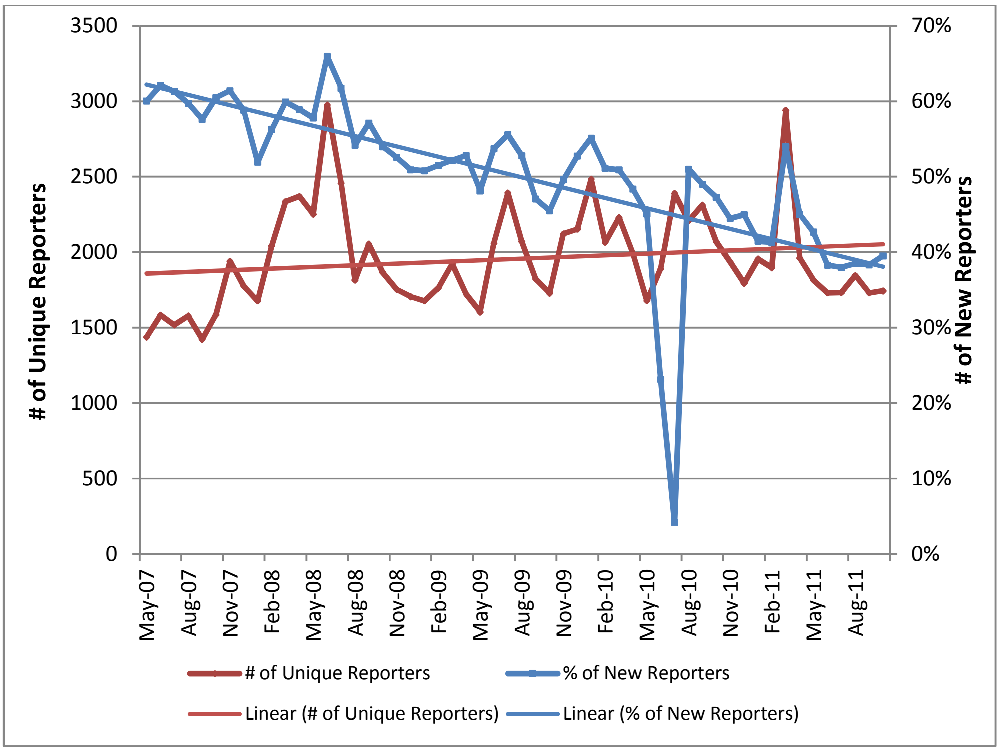

Figure 5, Number of unique bug reporters and new contributors.

Though there is a declining trend in terms of first-time bug submitters, it is not unexpected. As the community grows it approaches a saturation point in terms of the number of people with both the ability and interest in contributing. As the community grows, communication and coordination problems grow as well, discouraging further growth.

The data therefore seems to show no long-term negative effects of the introduction of the crash reporting system in terms of participation or data quality (here measured as duplicate reporting rates).

3.5.2 Qualitative Results

To supplement our statistical findings, we interviewed five developers working for Mozilla. Two participants were involved in developing the Breakpad and Socorro systems, and the other three worked for the QA team that processes the crash reports submitted to Socorro. All our participants were employed full-time at Mozilla and three had some formal background in computer science.

Participants were asked to give their opinions and share their experiences with the current system and with working with crash reports to debug Mozilla projects. This included but was not limited to what challenges they face in using or developing the crash reporting systems, pros and cons of using crash reports to drive debugging, and features that they would like to see in the system in the future. Some questions were posed as open-ended questions, and others as likert-scale alternatives. The results of these are presented in Figure 6.

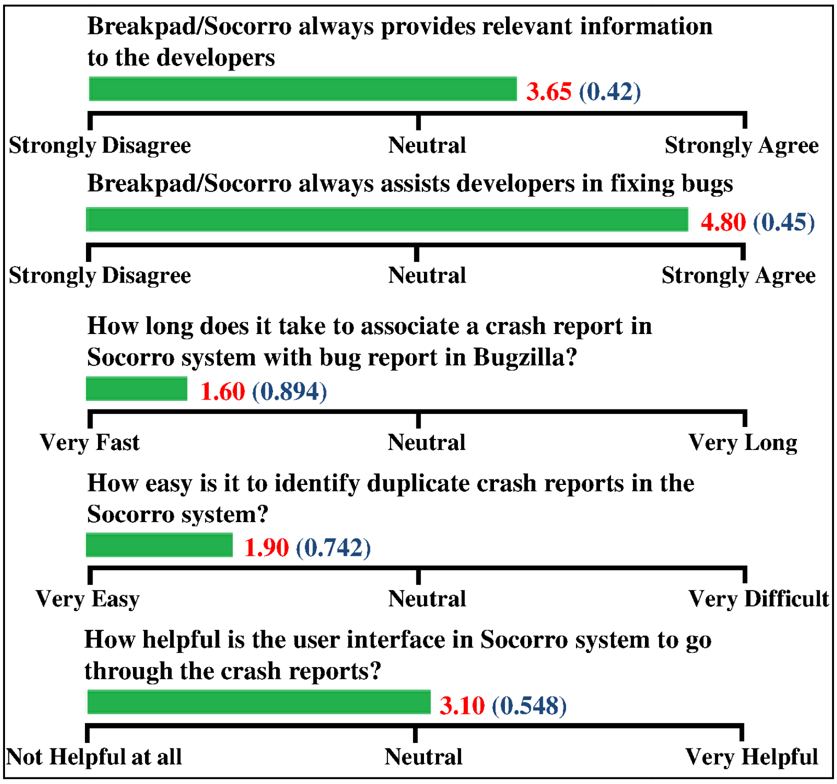

Figure 6, Interview Results. Values in “red” are the means and values in “blue” are the standard deviations for each response

There was strong agreement that the system – in its current incarnation and based on subjective experience – helps developers fix bugs, and helps them associate crash reports to bugs quickly and easily. Participants were more ambivalent about the usefulness of the user interface, and the relevance of the information shown to developers. This leads us to conclude that though the system is useful, there are still significant improvements to be made.

These positive reviews did not mean that participants were blind to the costs and risks of this system. When asked about the challenges to deploying and using the crash reporting system, a participant replied:

> *"It has a cost obviously. It's a lot of data to collect and report on. That can be a challenge to manage all that. We only report on a statistically valid subset of crashes. We only report on 10%. We collect 100% crashes so that's a lot of data coming in and it's really expensive and it's a challenge to make sure that the system is up and running."*
>
> *"I think it's pretty decent system overall. I wish it were easier to install and better and up-to-date documentation and installation utilities to help people. If the user has a new program and if they wish to support automatic crash reporting they have to dig deep into different websites and go through a lot of documentation to get it up and running."*

Our participants did find the crash reporting system to be very helpful, as they feel it is effective at helping developers find bugs and fix them:

> *"I would say it's doing the job it is intended to as far as I can tell from a developer's perspective."*

More importantly, participants felt that the system added unique capabilities without which certain types of bugs would have been difficult to detect:

> *"I always have a hard time as a QA person to strongly agree with a statement as my job is to find exceptions. If it wasn't for Breakpad, we would not be aware of some of the crashes that end up happening in the product. It would be definitely harder."*

Participants felt that Breakpad could do a better job collecting useful information in some situations, especially for newer platforms like Android devices.

> *"For android devices, it might not necessarily give the relevant information. [...] It is getting better for Android. Some of the other things are minor tweaks on the reporting end to make the information a bit more useful."*

3.6 Discussion

We started this research with three research questions:

RQ1. What impact do automatic crash reporting systems have on FOSS projects?

RQ2. What overhead do automatic crash reporting tools add to the bug triaging process?

RQ3. Do crash reporting systems discourage participation in the bug reporting process?

While we can't say anything about FOSS projects in general, we did get some compelling data for the Mozilla project, often held up as an exemplar in the FOSS community, and certainly one of the largest and most influential FOSS projects.

Starting from the bottom up (RQ3), we found no evidence that crash reporting systems discouraged participation in bug reporting, at least in the long term. Looking at Figure 5 we see that though new reporters as a portion of all bug reporters has been declining, this trend started before the introduction of the crash reporting system, and does not seem to have picked up speed since. Furthermore, the total number of bug reporters has continued to increase over time. Figure 4 shows that there was a slight decrease in the total number of bug reports shortly after the introduction of the system, but over the long term this number has also increased. Therefore we find no compelling evidence for crash reporting systems discouraging participation in bug reporting.

We did find a lot of evidence of the costs associated with adopting a crash reporting system (RQ2). The huge volume of data collected, and the relatively low number of bugs identified from the data is astounding. The costs, both monetary, as well as in time and effort needed to collect and sort through such vast amounts of data are significant, and thus adopting a crash reporting system is something that requires a significant commitment.

In all likelihood, for a moderate-sized FOSS project, implementing such a system will require a dedicated servers to receive reports, bandwidth charges, and because of the specialized skills required and the less appealing nature of the sleuthing work required, paid staff to try and process the data received. Our participants indicate that there is also a cost to incorporating these systems into their products due to either lacking documentation or tradeoffs in terms of implementation.

Much more work needs to be done to streamline the triaging and processing of data, or of extracting value from the data that these systems generate. The application of machine learning techniques to better match duplicates, better sampling techniques to ensure data is gathered about the most interesting/relevant crashes, and better diagnosis tools to help root out the underlying causes for crashes and turning these into bug reports.

Finally, turning to RQ1, all the developers we talked to unanimously think that the system provides real and significant value to the QA of Mozilla. Though only a tiny fraction of crash reports are actually used by the team, one of every 40 bug reports use data from the crash reports. These are bugs that would in all likelihood have been very difficult to track down without the information in the crash reports. In this sense, we can see that this system has a real and meaningful impact.

Because the implementation of these systems present both opportunities and challenges, it is important to identify best practices and optimize these systems. FOSS projects like the Kernel, Red Hat/Fedora, Ubuntu, etc. have deployed similar systems, and our next step will be to do an inventory of these.

That said, it is important to realize that deploying a crash reporting system is likely not an option for everyone. Many FOSS projects are not large enough to need such a complex system, or would be overwhelmed by the flood of data. In such cases these systems will likely prove counterproductive.

Threats to Validity

The data we gathered is just a snapshot in time for a single project. Considering the activity level and dynamism of the Mozilla project, a lot of things may have changed from the time we gathered our data and the time this paper goes to print.

Small improvements in the triaging process, or how crash reports are filtered can also have a big impact here, given the low “exploitation rate” of crash reports.

Given that we’ve only examined one project and the procedures they follow, we don’t know whether these will generalize to other FOSS projects. Mozilla is an outlier in the FOSS community, both because of its size as well as its top-down structure and reliance on professional employees. That said, Mozilla is often used as an exemplar, or a role model for other FOSS projects, and this knowledge will fit into the greater body of knowledge of how FOSS projects can and should be managed.

Without wanting to second-guess our participants, who after all have extensive experience using this system, it is possible that the ratings and stated opinions of our participants were biased by one of two factors: a) having a stake in the system (being paid to develop or use the system), and b) lacking exposure to other systems of this type. As one participant put it:

> *“I am not sure what alternatives we have. I think the advantages of having a crash reporting system at all is really great.”*

As we look for feedback and ideas for how to improve these systems, it is important to be aware of these limitations; our informants and users often compare these systems to no system, and thus excuse or ignore shortcomings.

3.7 Conclusion

We found that the Mozilla crash reporting system has had significant impact on the QA of their products, with 1 in 40 bug reports now being tied to or derived from crash reports. These systems come at a steep price however, as vast amounts of data tend to be generated, which is difficult to handle. The return on investment for these systems therefore has to be carefully considered for each project. We found no evidence to support the claim that these systems discourage participation, at least in the long term, and there is ample need for and opportunity for improvement of these systems.

3.8 Acknowledgements

We thank all the participants involved in our study for their valuable inputs and feedback. We really appreciate the efforts and help we received from the Breakpad/Socorro team at Mozilla for helping us with data collection. We especially thank Lars Lohn from Mozilla for all his help in making this study possible. We also thank the HCI research group at Oregon State University for helping in revising the paper.

3.9 References

Ahsan, S.N., Ferzund, J. and Wotawa, F. “Automatic Software Bug Triage System (BTS) Based on Latent Semantic Indexing and Support Vector Machine.” In Fourth International Conference on Software Engineering Advances, 2009. ICSEA ’09, pp. 216–221.

Anvik, J. “Automating bug report assignment.” In Proceedings of the 28th international conference on Software engineering, Shanghai, China, 2006, pp. 937–940.

Anvik, J., Hiew, L. and Murphy, G.C. “Coping with an open bug repository” In Proceedings of the 2005 OOPSLA workshop on Eclipse technology eXchange, San Diego, California, 2005, pp. 35–39.

Anvik, J., Hiew, L. and Murphy, G.C. “Who should fix this bug?” In Proceedings of the 28th international conference on Software engineering, Shanghai, China, 2006, pp. 361–370.

Apple, “Technical Note TN2123: CrashReporter,” 2010.

Bettenburg, N., Premraj, R., Zimmermann, T. and Kim, S. “Duplicate bug reports considered harmful … really?” In IEEE International Conference on Software Maintenance, 2008. ICSM 2008, pp. 337–345.

Bettenburg, N., Just, S., Schröter, A., Weiss, C., Premraj, R. and Zimmermann, T. “What makes a good bug report?” In Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering, Atlanta, Georgia, 2008, pp. 308–318.

Breu, S., Premraj, R., Sillito, J. and Zimmermann, T. “Information needs in bug reports: improving cooperation between developers and users.” In Proceedings of the 2010 ACM conference on Computer supported cooperative work, Savannah, Georgia, USA, 2010, pp. 301–310.

Cavalcanti, Y.C., Anselmo, P.M.S.N., Almeida, E.S., Cunha, C.E.A., Lucrédio, D. and Meira, S. R.L. “One Step More to Understand the Bug Report Duplication

Problem.” In Proceedings of the 2010 Brazilian Symposium on Software Engineering (SBES '10), pp. 148–157.

Cavalcanti, Y.C., Almeida, E.S., Cunha, C.E.A., Lucrédio, D., and Meira, S.R.L. “An Initial Study on the Bug Report Duplication Problem,” in 2010 14th European Conference on Software Maintenance and Reengineering (CSMR), 2010, pp. 264–267.

D’Ambros, M., Lanza, M. and Pinzger, M. “A Bug’s Life Visualizing a Bug Database.” In 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis, 2007. VISSOFT 2007, 2007, pp. 113–120.

Davidson, J.L., Mohan, N. and Jensen, C. “Coping with duplicate bug reports in free/open source software projects.” In 2011 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC '11), pp. 101–108.

Dhaliwal, T., Khomh, F., Zou, Y. "Classifying field crash reports for fixing bugs: A case study of Mozilla Firefox." Software Maintenance (ICSM), 2011 27th IEEE International Conference on , vol., no., pp.333-342, 25-30 Sept. 2011

Hooimeijer, P. and Weimer, W. “Modeling bug report quality.” In Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering, Atlanta, Georgia, USA, 2007, pp. 34–43.

Jalbert, N. and Weimer, W. “Automated duplicate detection for bug tracking systems.” In IEEE International Conference on Dependable Systems and Networks With FTCS and DCC, 2008. DSN 2008, pp. 52–61.

Jeong, G., Kim, S. and Zimmermann, T. “Improving bug triage with bug tossing graphs.” In Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering, Amsterdam, The Netherlands, 2009, pp. 111–120.

Khomh, F., Chan, B., Zou, y., Hassan, A.E., "An Entropy Evaluation Approach for Triaging Field Crashes: A Case Study of Mozilla Firefox." Reverse Engineering (WCRE), 2011 18th Working Conference on , vol., no., pp.261-270, 17-20 Oct. 2011

Kim, D., Wang, X., Kim, S., Zeller, A., Cheung, S.C., Park, S. "Which Crashes Should I Fix First? Predicting Top Crashes at an Early Stage to Prioritize Debugging Efforts." Software Engineering, IEEE Transactions on , vol.37, no.3, pp.430-447, May-June 2011

Kim, S., Zimmermann, T., Pan, K. and Whitehead, E.J. “Automatic Identification of Bug-Introducing Changes.” In 21st IEEE/ACM International Conference on Automated Software Engineering, 2006. ASE ’06, pp. 81–90.

Kim, S., Zimmermann, T., Nagappan, N. "Crash graphs: An aggregated view of multiple crashes to improve crash triage." Dependable Systems & Networks (DSN), 2011 IEEE/IFIP 41st International Conference on , vol., no., pp.486-493, 27-30 June 2011

Kinshumann, K., Glerum, K., Greenberg, S. Aul, G., Orgovan, V., Nichols, G., Grant, D., Loihle, G. and Hunt, G. “Debugging in the (very) large: ten years of implementation and experience.” Commun. ACM, vol. 54, no. 7, pp. 111–116, Jul. 2011.

Ko, A.J., Myers, B.A. and Chau, D.H. “A Linguistic Analysis of How People Describe Software Problems.” In IEEE Symposium on Visual Languages and Human-Centric Computing, 2006. VL/HCC 2006, pp. 127–134.

Ko, A.J. and Chilana, P.K. “How power users help and hinder open bug reporting.” In Proceedings of the 28th international conference on Human factors in computing systems, Atlanta, Georgia, USA, 2010, pp. 1665–1674.

Matter, D., Kuhn, A. and Nierstrasz, O. “Assigning bug reports using a vocabulary-based expertise model of developers.” In 6th IEEE International Working Conference on Mining Software Repositories, 2009. MSR ’09, pp. 131–140.

Raymond, Eric S. "The Cathedral and the Bazaar." Computers & Mathematics with Applications 39.3-4 (2000).

Tamrawi, A., Nguyen, T.T., Al-Kofahi, J. and Nguyen, T.N. “Fuzzy set-based automatic bug triaging: NIER track.” In Proceeding of the 33rd international conference on Software engineering, Waikiki, Honolulu, HI, USA, 2011, pp. 884–887.

Wang, X., Zhang, L., Xie, T., Anvik, J. and Sun, J. “An approach to detecting duplicate bug reports using natural language and execution information.” In Proceedings of the 30th international conference on Software engineering, Leipzig, Germany, 2008, pp. 461–470.

https://wiki.mozilla.org/images/e/ed/Analyst_report_ Q1_2010.pdf

http://blog.mozilla.com/thunderbird/2011/11

4. Conclusion

From the first manuscript, we can infer that the practice of open bug reporting incurs cost. We see that the users are interested in contributing to FOSS projects by reporting bugs and feedback. We can observe that the projects have to deal with overwhelming number of inexperienced reporters which could be a potential problem if the projects lack enough resources. Also, projects have a hard time sustaining user participation, as most users disappear after contributing just one bug report. This should convince the readers that the FOSS projects are in need of additional scaffolding that simplifies the bug management process and assists novice reporters in submitting quality feedback.

From the second manuscript, we find that large and active projects like Mozilla are aware of the importance of user feedback and prolonged user participation. The introduction of automatic crash reporting system has certainly helped improve Mozilla’s bug management practices. However, deploying such systems seems to be quite a hurdle since the project requires excellent infrastructure to deal with huge volumes of data.

Open bug reporting is certainly helpful and fundamental to realizing Linus’ Law; “Given a large enough beta-tester and co-developer base, almost every problem will be characterized quickly and the fix will be obvious to someone” (Raymond, 2000). FOSS communities would reap huge benefits if they keep the reporters involved and motivated.

Bibliography

Anvik, J. “Automating bug report assignment,” in Proceedings of the 28th international conference on Software engineering, Shanghai, China, 2006, pp. 937–940.

Anvik, J., Hiew, L., and Murphy, G. “Who should fix this bug?,” in Proceedings of the 28th international conference on Software engineering, Shanghai, China, 2006, pp. 361–370.

Chung, E., Jensen, C., Yatani, K., Kuechler, V., and Truong, K. N. “Drawing and sketching in Open Source design”, in IEEE Symposium on /Visual Languages and Human-Centric Computing, 2010. VL/HCC 2010.

Ghosh, A. “Economic impact of FLOSS on innovation and competitiveness of the EU ICT sector”, 2006.

Jensen, C., Scacchi, W. "Role Migration and Advancement Processes in OSSD Projects: A Comparative Case Study," Int Conf. on Software Engineering, ICSE’07. 20-26 May 2007.

King, S. “Joining Open Source Software Communities: An Analysis of Newbies’ First Interactions on Project Mailing Lists”, 2009, MS Thesis.

Matter, D., Kuhn, A., and Nierstrasz, O. “Assigning bug reports using a vocabulary-based expertise model of developers,” in 6th IEEE International Working Conference on Mining Software Repositories, MSR ’09, pp. 131–140.

Park, Y. “Supporting the Learning Process of Open Source Novices: An Evaluation of Code and Project History Visualization Tools”, 2008, MS Thesis.

Raymond, E. "Homesteading the Noosphere," 1998.

Wu, C., Gerlach, J., Young, C.. “An empirical analysis of open source software developers’ motivations and continuance intentions, Information & Management”, 2007, pp 253-262.

Yunwen, Y., Kishida, K. “Toward an understanding of the motivation of open source software developers.” In Proc. of the 25th Int. Conf. on Software Engineering, 3-10 May 2003.