Coping with Duplicate Bug Reports in Free/Open Source Software Projects

Jennifer L. Davidson, Nitin Mohan, Carlos Jensen
School of Electrical Engineering and Computer Science
Oregon State University
Corvallis, USA
{davidsje, mohanni, cjensen}@eecs.oregonstate.edu

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.

Keywords- Free/Open Source Software; FOSS; Open Bug Reporting; Bug tracking; Bug duplication; Bug triage

I. 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" [22]. 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. [8], 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 [17] 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. [6] 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 [12, 18]. 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 [12, 18], 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.

II. Related Work

A. *FOSS Workflow and Project Management*

FOSS projects are volunteer-driven where people join to address common needs, for skill development, and to gain experience [13, 26]. Most communication and collaboration is done using text-based channels, including IRC, wikis, forums, mailing lists and bug repositories [9, 12]. 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 [23]. 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 [15]. 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 [10]. In most FOSS projects, the bulk of the code is contributed by a small percentage of the contributors [11, 19].

B. *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. [4] 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. also proposed a version called quZilla that would provide immediate feedback to the user about his/her bug report in the context of Eclipse [5].

Calvacanti et al. [7, 8] 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 [17] 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. [2] 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 [24] and automatic bug assignment [1, 3]. Jalbert and Weimer provided a classifier to detect duplicate bug reports as they were being reported [14]. While automatic duplicate detection is a useful approach to dealing with duplicate bugs, Bettenburg et al. [6] 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. [16] 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. [23] and Panjer [20] 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.

III. 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 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[1]. We used the information on Open Watcom's webpage listing "contributors"[2]. 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[3] 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.

A. 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 [17]. 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.

[1]http://www.gratisoft.us/sudo/authors.html

[2] http://openwatcom.com/index.php/Open_Watcom:About

[3] http://dash.eclipse.org/dash/commits/web-app/commit-count-loc.php

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.

TABLE I. 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

IV. RESULTS

A. *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.

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.

TABLE II. 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)

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 [6].

B. 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 III. 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. [7] found. This held true across projects of all sizes.

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.

TABLE IV. 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?*			
	Reporters	**% Duplicate**	**% of dups ID before assign.**	# People	How Many Bugs Reported?			# People	How Many Times Per Person?			# People	How Many Times Per Person?			# People	How Many Times Per Person?		
					Median	Mean	St. Dev		Median	Mean	St. Dev.		Median	Mean	St. Dev.		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

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.

V. 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. propose in [4].

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 in [7], 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.

A. 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.

VI. 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" [22]. 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.

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.

References

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

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

[11] 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/

[12] 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.

[13] 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.

[14] 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

[15] 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.

[16] 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.

[17] 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.

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

[19] 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.

[20] 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)

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

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

[23] 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,

[24] 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.

[25] 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

[26] 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.