The Impact of Automatic Crash Reports on Bug Triaging and Development in Mozilla

Iftekhar Ahmed
Oregon State University
School of Electrical Engineering and Computer Science
Corvallis, USA
ahmedi@onid.oregonstate.edu

Nitin Mohan
Oregon State University
School of Electrical Engineering and Computer Science
Corvallis, USA
nitinmohan.osu@gmail.com

Carlos Jensen
Oregon State University
School of Electrical Engineering and Computer Science
Corvallis, USA
cjensen@eecs.oregonstate.edu

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 deploy automatic crash reporting tools, which capture stack traces and memory dumps when a crash occurs. These systems potentially generate large volumes of data, which may overwhelm developers, and their presence may 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 fewer than 0.00009% of crash reports 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.

Categories and Subject Descriptors

D.2.5 [**Software Engineering**]: Testing and Debugging – *Debugging aids, Tracing.*

General Terms

Management, Measurement, Reliability.

Keywords

Free/Open Source Software, FOSS, Open Bug Reporting, Debugging, Testing, Automatic Crash reporting

1. INTRODUCTION

Free/Open Source Software (FOSS) projects often follow different development practices than traditional closed source projects. One of the reasons for such 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 [25] drives FOSS projects to involve as many people as possible in bug reporting and triaging. While there are advantages to broad involvement, there are also downsides. Reports submitted by less experienced users can be incomplete or inaccurate [6]. Users may not use the right terms to describe a bug, which can make it hard for developers to find the bug. A study by Davidson et al. [12] found that as the size of the reporting community grows, so does the ratio of duplicate reports. Though duplicate reports are not always problematic [6, 23], duplicates represent a waste of time and effort. Though projects publish guidelines for submitting bug reports, training and coordinating contributors is often an overwhelming task.

OpenSym '14, August 27 - 29 2014, Berlin, Germany
Copyright 2014 ACM 978-1-4503-3016-9/14/08 $15.00.
http://dx.doi.org/10.1145/2641580.2641585

There have been a number of studies examining the bug triaging processes of FOSS projects. Bettenburg et al. [7] 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. [8] 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 details are in bug triaging.

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 crash 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, this does not necessarily translate to more actionable information. The majority of crash reports refer to a small number of common problems.

Furthermore, a crash reporting system could lead users to stop submitting traditional bug reports, feeling that they have already contributed. This would thus lead 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, the latter of which also tend to include usability issues and missing features. To this end our research questions are:

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 leading FOSS project: 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 4 describes the results of our study, including excerpts from interviews with developers and users of these systems. Section 5 discusses our findings and the pros and cons of using these systems, as well as lessons to enhancing these tools. Section 6 concludes with a summary of the key findings and future work.

2. RELATED WORK

Automatic crash reporting systems have been used in many closed-source systems and companies [5]. The most famous is the Windows Error Reporting (WER) system by Microsoft, described by Kinshumann et al. in [21]. The author found 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. [20] 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 studies of Mozilla's automatic crash reporting system. Kim et al [18] focused on prioritizing debugging efforts by predicting top crashes. Dhaliwal et al. [13] proposed a grouping approach to crash report triaging. They show that effective grouping can reduce the time to fix bugs by 5%. Khomh et al. [17] proposed the use of crash entropy values to prioritize crash types during triaging. These studies focused on a small subset of crash reports.

There has been a lot of work on automating and improving the bug triaging process [1, 2, 4, 14, 19, 23, 26]. Bug triaging refers to the steps taken to manage a bug from the time it is reported to the time the bug is resolved. Anvik [2] discussed a semi-automated approach for assigning bugs to developers through a recommender system. Anvik et al. [4] 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. [24] proposed a vocabulary-based approach where developer expertise and bug vocabularies were matched. Tamrawi et al. [26] 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. Researchers have also come up with different ways of visualizing bug related information [11]. Jeong et al. [16] 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 [23] 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. [6] 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. [9, 10] 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. [12] 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. [3] studied duplicates in Firefox and Eclipse and found that these were common and that there is a need for tools to detect these. Researchers have also used machine learning and natural language processing approaches to identify duplicate bugs [15, 27].

The quality of bug reports in FOSS projects is another important topic. Bettenburg et al. [7] 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. [8] 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. [22] 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. 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.

3.1 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.

3.2 Analysis

We collected historical data for the project in the form of daily crash logs, spanning from March 2010 to October 2011. We were particularly interested in this time-period as we wanted to see if and how bug reporting changed over that period of time. 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.

4. RESULTS

A previous study of 12 FOSS projects [12] 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.

4.1 Quantitative Results

First, we quantitatively analyzed the crash report logs from March 2010 to October 2011. We aggregated basic statistics, listed in Table 1, and compared to the activity in the bug reporting system over the same period.

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 [28], and between 15 and 20 million Thunderbird users [29]. 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 classified as duplicates using fuzzy matching techniques. 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.

Table 1. 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*	

Remaining reports are manually classified by the QA group 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 the name suggests, crash signatures are unique identifiers of system crashes that captures potentially important technical information for both debugging and simple categorization and identification of duplicate reports. As we explain below, that monthly average is heavily skewed. Of all crash reports, the number that leads to bug reports account for only 0.00009%, 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.

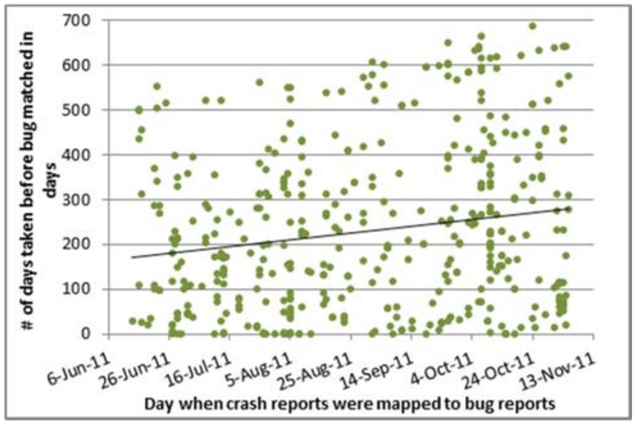

Figure 1. Time taken to associate crash reports with bug reports

Figure 1 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 projects 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.

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 2) [18]. 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.

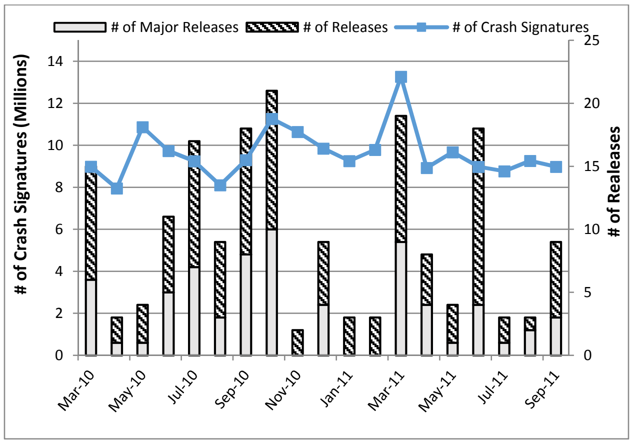

Figure 2. Crash signatures vs. software releases in Mozilla

In Figure 3 we can see long-term trends for bug reporting and duplication rates. The automatic crash reporting system was introduced in June 2008 (first stripped vertical line), and they switched to a rapid release cycle in April 2011 (2nd stripped vertical 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.

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 2, 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 2. 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)	ANOVA(df=1, F=47.965, p<0.00001)	ANOVA(df=1, F=1.4081, p=0.2427)
% Duplicates	ANOVA (df=1, F=96.333, p<0.00001)	ANOVA (df=1, F=126.89, p<0.00001)	ANOVA (df=1, F=15.187, p=0.00038)

As we see in Figure 3. 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<0.00001). We can therefore conclude that though introducing the crash reporting system may have been disruptive, these issues were quickly worked out.

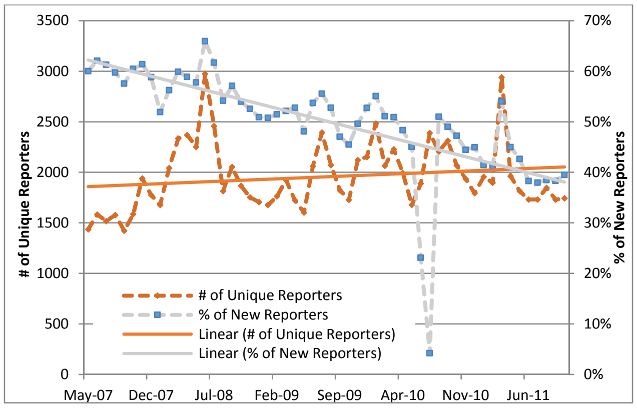

Figure 4. Number of unique bug reporters and new bug reporters

As we see in Figure 4, 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<0.00001). 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 [12].

Though there is a declining trend in terms of first-time bug submitters, it is not unexpected. As the community grows we expect it to approach a saturation point in terms of the number of people with both the ability and interest in contributing. We also expect that as the community grows, communication and coordination problems grow as well, potentially 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).

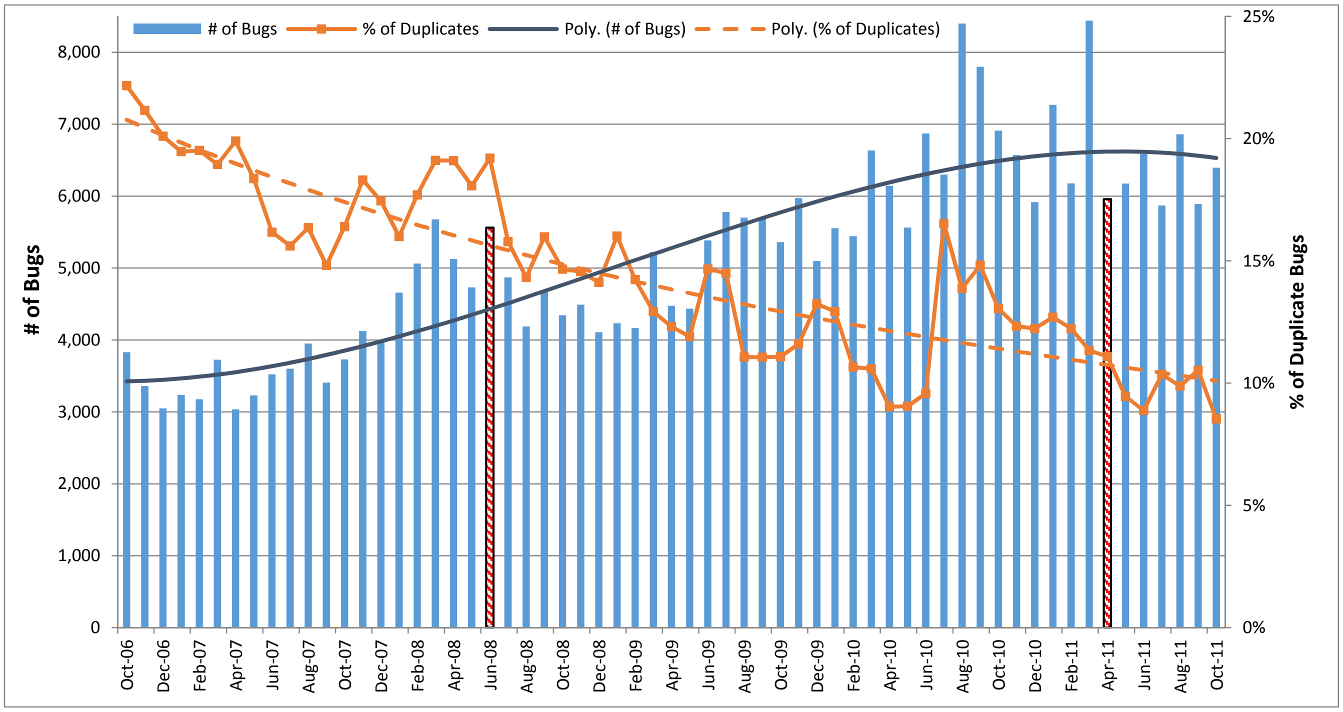

Figure 3. Temporal view of bug activity in Mozilla. First stripped vertical bar indicates the introduction of the crash reporting system, and the second indicates transition to rapid release cycle

4.2 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.

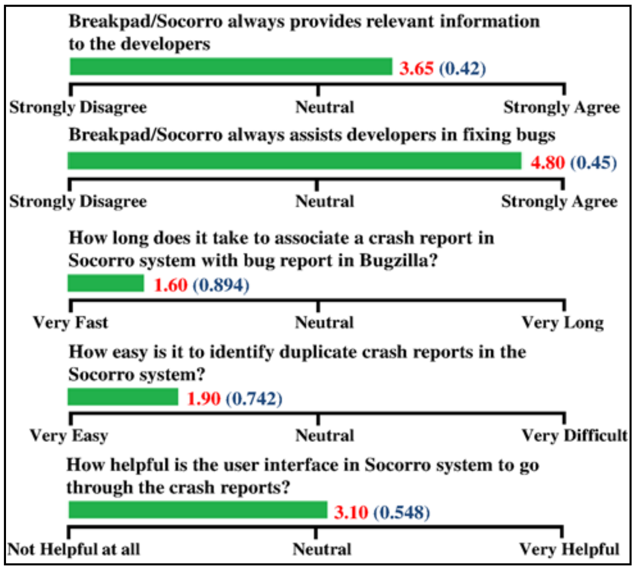

Figure 5, Interview Results. Values in "red" are the means and values in "blue" (inside the parenthesis) are the standard deviations for each response

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 5.

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. Our participants did find the crash reporting system to be very helpful, as they felt it was effective in 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 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. 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."

Though the participants had positive feedback about the automatic crash reporting system, they were not 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."

5. 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 4 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 3 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, full time 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.

6. 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. Having a more realistic control condition would likely make failings or limitations of the current system more apparent.

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 and opportunity to improve these systems.

8. ACKNOWLEDGMENTS

We would like to thank the Oregon State University HCI group for their input and feedback on the research.

9. REFERENCES

[1] 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., pp. 216–221.

[2] Anvik, J. "Automating bug report assignment." In *Proceedings of the 28th international conference on Software engineering,* Shanghai, China, 2006, pp. 937–940.

[3] Anvik, J., Hiew, L. and Murphy, G.C. "Coping with an open bug repository" In *Proceedings OOPSLA workshop on Eclipse technology eXchange*, San Diego, California, 2005, pp. 35–39.

[4] 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.

[5] Apple, "Technical Note TN2123: CrashReporter," 2010.

[6] 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.

[7] 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.

[8] 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.

[9] 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)*, pp. 148–157.

[10] 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.

[11] 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, pp. 113–120.

[12] Davidson, J.L., Mohan, N. and Jensen, C. "Coping with duplicate bug reports in free/open source software projects." In *Proceedings of* IEEE Symposium on Visual Languages and Human-Centric Computing, 2011, pp. 101–108.

[13] Dhaliwal, T., Khomh, F., Zou, Y. "Classifying field crash reports for fixing bugs: A case study of Mozilla Firefox." In *Proceedings of Software Maintenance (ICSM), 2011* pp.333-342,

[14] 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.

[15] Jalbert, N. and Weimer, W. "Automated duplicate detection for bug tracking systems." In *International Conference on Dependable Systems and Networks With FTCS and DCC*, 2008, pp. 52–61.

[16] Jeong, G., Kim, S. and Zimmermann, T. "Improving bug triage with bug tossing graphs." In *Proceedings of 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.

[17] Khomh, F., Chan, B., Zou, y., Hassan, A.E., "An Entropy Evaluation Approach for Triaging Field Crashes: A Case Study of Mozilla Firefox." In *(WCRE), 18th Working Conference on Reverse Engineering*, 2011,, pp.261-270.

[18] 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." In *Software Engineering, IEEE Transactions on* , vol.37, no.3, 2011, pp.430-447,

[19] 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,, pp. 81–90.

[20] Kim, S., Zimmermann, T., Nagappan, N. "Crash graphs: An aggregated view of multiple crashes to improve crash triage." In *Proceedings of 41st International Conference on Dependable Systems & Networks (DSN), 2011,*, pp.486-493.

[21] 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." In. *ACM Commun*, vol. 54, no. 7,2011, pp. 111–116, .

[22] 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., pp. 127–134.

[23] 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.

[24] 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., pp. 131–140.

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

[26] 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.

[27] 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.

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

[29] http://blog.mozilla.com/thunderbird/2011/11