Security Versus Performance Bugs: A Case Study on Firefox

Shahed Zaman, Bram Adams, Ahmed E. Hassan
Software Analysis and Intelligence Lab (SAIL)
School of Computing, Queen's University, Canada
{zaman,bram,ahmed}@cs.queensu.ca

ABSTRACT

A good understanding of the impact of different types of bugs on various project aspects is essential to improve software quality research and practice. For instance, we would expect that security bugs are fixed faster than other types of bugs due to their critical nature. However, prior research has often treated all bugs as similar when studying various aspects of software quality (e.g., predicting the time to fix a bug), or has focused on one particular type of bug (e.g., security bugs) with little comparison to other types. In this paper, we study how different types of bugs (performance and security bugs) differ from each other and from the rest of the bugs in a software project. Through a case study on the Firefox project, we find that security bugs are fixed and triaged much faster, but are reopened and tossed more frequently. Furthermore, we also find that security bugs involve more developers and impact more files in a project. Our work is the first work to ever empirically study performance bugs and compare it to frequently-studied security bugs. Our findings highlight the importance of considering the different types of bugs in software quality research and practice.

Categories and Subject Descriptors

D.2.9 [**Software Engineering**]: Management

General Terms

Performance, Security, Management

Keywords

Performance bugs, security bugs, empirical study, Firefox, Bugzilla.

1. INTRODUCTION

Previous studies show that maintenance and evolution activities represent over 90% of the software development cost [1, 2]. Improving quality assurance effort in software projects is an important task to keep the existing customers satisfied and to compete in a competitive market. Research has focused on improving the quality assurance of software projects. For example, software defect prediction models use various code, process, social structure, geographical distribution and organizational structure metrics to predict the number of defects and their locations [1, 3, 4, 5, 6, 7, 8, 9]. Other work focuses on predicting the time it takes to fix a bug [10, 11, 12], and which developer should fix a bug [13].

One common theme in the aforementioned quality assurance research is that most of them treat all bugs equally, i.e., they do not distinguish between different kinds of bugs. For example, one would expect bugs that are labeled as security risks to experience a different treatment than typos in code comments or user interface quirks. Yet, most techniques build generic models for generic bugs.

We believe that a thorough understanding of the various aspects associated with the different types of bugs is needed. In this paper, we empirically study the characteristics and differences among security, performance and other bugs in the Firefox open source project to show that these characteristics are different from each other in practice. We pick security and performance bugs, since they are important non-functional types of bugs. Security bugs are of high risk, and performance issues are very commonly occurring in the field [14]. Yet both types have never been compared against each other to see if there are differences and if these differences are likely to impact or affect current work in the area of software quality assurance. We address the following three research questions:

Q1). How fast are bugs fixed?

On average, security bugs are fixed 2.8 times faster than performance bugs, but they are reopened 2.5 times more than performance bugs and approximately 4.5 times more than other bugs.

On average, security bugs are triaged 3.64 times faster than performance bugs and 3.38 times faster than the rest of the bugs, but security bugs are tossed 2.67 times more than performance and 4.7 times more than the rest of the bugs.

Q2). Who fixes bugs?

On average, security bugs are assigned 2.39 times more developers than performance and 3.51 times more developers than the rest of the bugs. Performance and security bugs are fixed by more experienced developers.

MSR '11, May 21-22, 2011, Waikiki, Honolulu, HI, USA

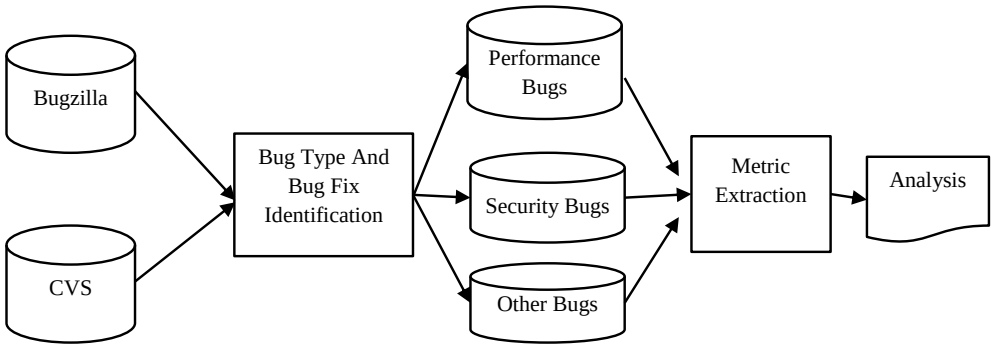

Figure 1: Overview of our approach to study the differences in characteristics of security, performance and other bugs

Q3). What are the characteristics of bug fixes?

> On average, security bug fixes are 1.48 times more complex than performance bug fixes and 1.6 times more complex than the fixes for rest of the bug.
>
> On average, fixing a performance bug requires change in 2.6 times more files than for security bugs.

Overview of the paper: Section 2 discusses our approach. Section 3 is divided into three sub-sections according to the studied research questions. Each sub-section of Section 3 discusses a motivation, approach and the results of the specific question. In section 4, we discuss the work related to our study. Finally, Section 5 discusses threats to validity and Section 6 presents the conclusion of the paper.

2. STUDY DESIGN

This section presents the design of our case study to compare security, performance and other bugs in Firefox. Figure 1 shows an overview of our approach. First, we extract the necessary data from the bug repository (Bugzilla) and source code repository (CVS). Then, we classify the bug reports related to performance and security. Using CVS data, we also identify the bug fix information. For every type of bug, we calculate several metrics, then statistically compare the metrics across the types of bugs (performance, security and other). This section elaborates on each of these steps.

2.1 Choice of Subject System

In our case study, we study the Firefox web browser, since Firefox is one of the most popular web browsers and runs on different platforms and system environments. Performance and security are two major software quality requirements for a browser like Firefox. Bugzilla, the issue tracking system used by Mozilla, contains and documents all reported bugs. We start our approach with Mozilla's Bugzilla bug data from September, 1994 to August 15, 2010.

First, we had to identify the bugs that are related to the Firefox web browser, since Mozilla's bug tracking system manages multiple projects, including Firefox. Bugzilla contains 567,595 bug reports of different Mozilla components, like Core, Firefox, SeaMonkey, and Thunderbird. For Firefox, we took bug reports that are related to Core (shared components used by Firefox and other Mozilla software, including the handling of Web content) and to Firefox.

2.2 Bug Type Classification

Bugzilla does not have a built-in categorization or tagging for performance and security bugs. Firefox uses the keyword 'perf' for performance-related bugs, but this tagging is not mandatory and we found many bugs that do not have any such tag.

Hence, to classify performance bugs, we had to use heuristics. We looked for the keywords 'perf', 'slow', and 'hang' in the bug report title and keyword field. Although the keyword 'perf' gave us bug reports that had 'performance' in their title, we found that there are many bug reports containing the word 'perfect', 'performing' or 'performed', without having any relation to performance. We had to automatically exclude these keywords from the list using regular expressions. Using these heuristics, we found 7,603 performance bugs.

Since heuristics are not perfect, we performed a statistical sampling to estimate their precision and recall. To calculate precision with a 95% confidence level and a confidence interval of 10, we randomly sampled and checked 95 of the 7,603 bugs [15]. All 95 randomly selected samples were indeed performance bugs, yielding a precision of 100 $\pm$ 5%. To calculate recall, we did statistical sampling on the 287,595 non-performance bugs. For the same 95% confidence level and confidence interval of 10, we randomly sampled and checked 96 bugs. 19 out of these 96 samples bug reports were found to be related to performance, yielding a recall of 80 $\pm$ 5%.

To identify the security bugs, we use the Mozilla Foundation Security Advisory (MFSA) [16]. MFSA contains links to security bugs for every issued advisory. MFSA has been issuing security advisories for Mozilla's products since 2005. Based on the MFSA, we extracted a dataset of 847 security and 294,351 non-security bugs. As security bugs were marked by the experts of the security advisory team of Firefox, we did not estimate the precision and recall as we did for performance bugs.

2.3 Distribution of Bugs

Figure 2 shows the number of reported bugs over time in log scale. The earliest security bug in our data was reported in May, 2003. Hence, we do not study bugs reported prior to May, 2003. 2003 was also the year in which the Firefox project started. This leaves us with 4,293 performance bugs, 847 security bugs and 178,531 other bugs for our study (May, 2003-August, 2010. Performance and security bugs have 11

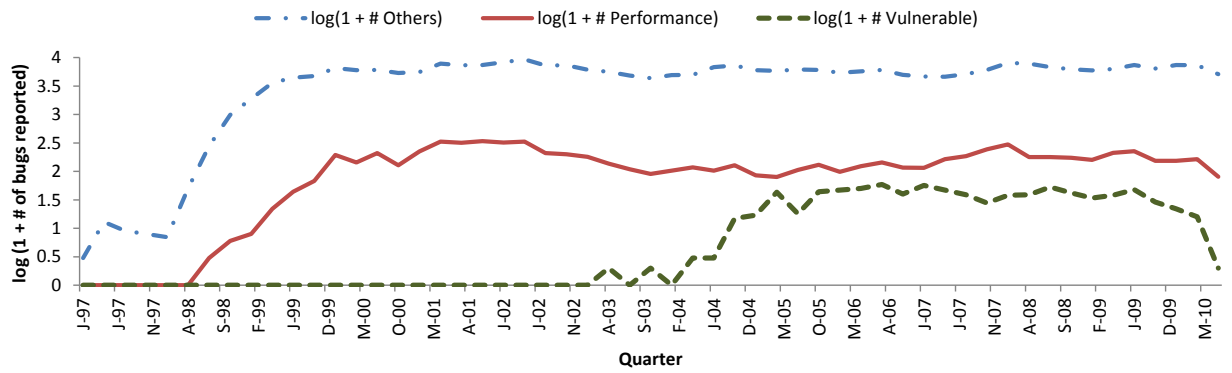

Figure 2: Distribution of the number of bugs reported (quarterly) for security, performance and other bugs.

bugs in common. Out of these 11 bugs, 7 had security threat and caused the browser to hang/crash. We kept these 11 bugs in security and performance groups in our study.

To better understand the composition of each group of bugs, we analyzed a topic model [17]. Topics are collections of words that co-occur frequently of the bugs in a corpus of text. These topics describe the major themes that span a corpus [18], which provide a means of automatically summarizing and organizing the various data of a software project. Recently, researchers have applied topic models to various aspects of a software project, including source code [19, 20, 21, 22] and documentation [23, 24]. Statistical topic models, such as latent Dirichlet allocation (LDA) [17], can automatically discover a set of topics within a corpus.

We ran LDA on the bug report comments to find out the topics that describe the major themes that span these performance and security bugs. For security bugs, as the number of security bugs was very small, the recovered LDA topics (such as "build-tinderbox related", "window-javascript related", "user-cookie related") could not be grouped together which could constitute a sub-category.

From the LDA output (due to space constraints, we have put the results online [25]) of performance bug comments, we find two major sub-categories of performance bugs: (1) Plugin-related (like java applet plugin and the flash plugin) and (2) GUI-related (user interface related libraries, like gfx and gecko) performance issues. Other smaller types of performance issues (like, topics related to network, history, bookmark, speed, IO, memory, file system, build and document structure) add up to the majority of performance bugs. Figure 4 shows the distribution over time of plugin and user interface related bugs and their relation to all performance bugs. This figure is in log scale, so the peaks here donote large changes in the number of bugs reported. Between the release of version 2.0 and 3.0 of Firefox (shown in Figure 4), there was an increase in GUI-related performance bugs. Version 2.0 included updates that were related to GUI such as a tabbed browsing environment, and an extensions manager. Study of the bug reports at a finer level of granularity can be helpful to explain the effect of different software quality measures.

3. CASE STUDY RESULTS

Each subsection below discusses one of the three research questions that we studied using the Firefox data. For each question, we present the motivation behind the research question, the approach and a discussion of our findings. Table I summarizes the results of each of the three studied questions.

3.1 How Fast are Bugs Fixed?

Motivation: A project manager typically wants some types of bugs, like security and performance bugs to be assigned faster than others, because of their inherent importance in software quality assurance. In addition, he wants to pick the best developer for that bug, such that later on the bug does not need to be tossed to someone else to fix [27]. Finally, the bug needs to be fixed completely, such that it does not need to be reopened afterwards, prolonging the total time to fix the bug. Our study measures these characteristics for Firefox performance and security bugs.

Approach: Once a bug is reported to the bug repository, it browses through a complex life cycle (Figure 3), from the UNCONFIRMED state to the CLOSED state. Figure 3 shows the life cycle of a bug report in Bugzilla. In every state, it takes some time for the bug report to go to the next state.

We used four metrics for this question. First, we calculate the time to fix, i.e., the time between bug assignment (ASSIGNED) and the bug fix date (FIXED and then RESOLVED). This time was calculated for all bugs that went to the ASSIGNED state first and then from that, the bug went to the RESOLVED state followed by FIXED (bolded arrow in Figure 3). Since some bugs are closed prematurely and have to be re-opened later (e.g., because of an incomplete fix), we measure both the total time to fix (total time of each assignment to final fix) and the average time of each fix attempt. We used a t-test to compare the total and average times and the cumulative density function of the metrics for the three groups (performance, security and other bugs). As shown in sub-section 2.3, there is a huge difference in the number of security, performance and other bugs. Since

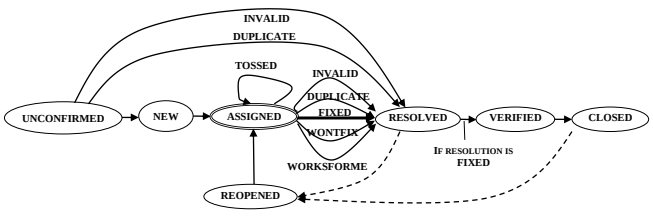

Figure 3: Life cycle of a bug [26].

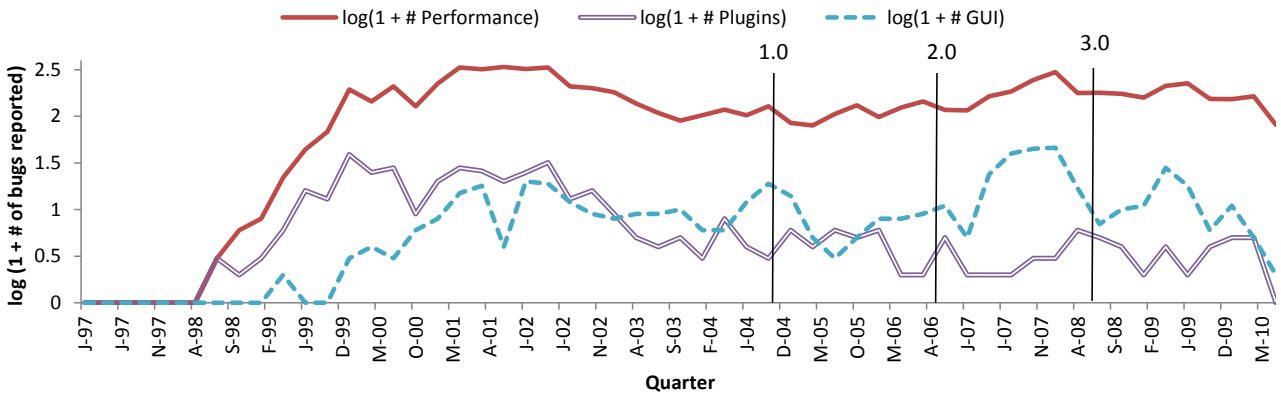

Figure 4: Comparison of the distribution of the number of bugs reported for all performance bugs and the two sub-categories of Plugin and GUI-related performance bugs.

we use the Wilcoxon t-test, this difference does not have an impact on our analysis.

The second metric that we use for comparison is the number of times a bug is reopened. In the ideal case, one would want a bug to be FIXED on the first try. In practice, a bug may be CLOSED, then REOPENED again. Reopened bugs take a considerably longer time to resolve, and hence increase the maintenance cost. For example, in the Eclipse platform 3.0 project reopened bugs take more than twice the time to resolve as a non-reopened bug [1]. In addition, an increased bug resolution time consumes valuable time from the already-busy developers, and reopened bugs may negatively impact the overall end-user's experience and trust in the quality of a software product.

The third metric that we used for our comparison is the bug triage time. Triage time is considered separately from time to fix, as triaging and bug fixing are usually done by different people and comprise two different processes. Bug Triaging is the process of reviewing a reported bug in order to assign the right developer to fix it. Each project has a different strategy for this. Due to the volume of reports, reports submitted to the Mozilla bug repository are triaged by quality assurance volunteers, rather than by the developers [13]. Ideally, one would want a bug to be triaged as fast as possible. We calculate the difference between bug reporting (UNCONFIRMED) and first assignment (ASSIGNED) as bug triage time. Although bug triage time affects the total bug fix time, one would also want to take the time to select the right person.

Our fourth metric measures the number of bug tosses, i.e., the number of times a bug was reassigned before being resolved completely [27]. Tossing may occur due to triaging inaccuracy, i.e., the developer assigned does not have the expertise to fix the time. Another reason for tossing could be bug complexity, i.e., the assigned developer could only fix part of the bug and needed another developer to fix the rest of that complex bug.

Findings: Performance bugs take the largest time to fix. Figure 5 plots the "total time between every assignment and fix". We used log-scale here to better show the differences between the three kinds of bugs. We can see a distinct difference for larger values. We ran t-tests to decide if the differences in mean between these three groups are statistically significant or not. We found a statistically

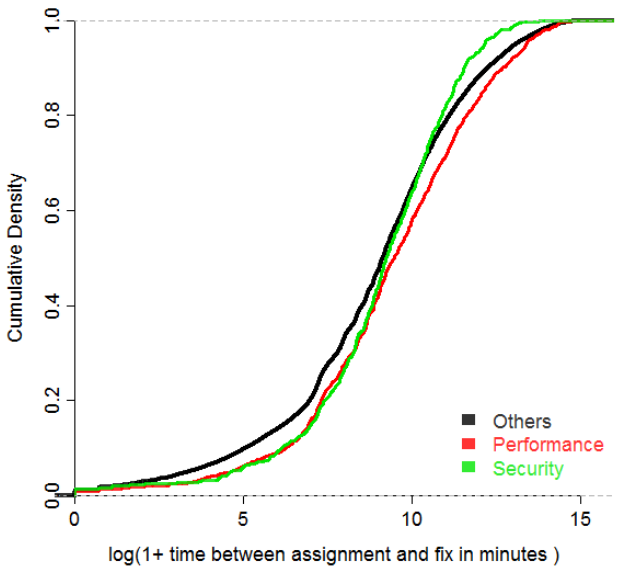

Figure 5: CDF (cumulative density function) of time between bug assignment and fix (total in log scale)

significant ($p < 0.05$) difference between security and performance bugs. Security bugs generally take less time to fix than performance bugs. Other bugs take less time to fix than Performance bugs. However, the difference between security and other bugs was not found to be statistically significant($p = 0.0538$, it is close though). According to the CDF plot, mean values and t-test result, we find that security bugs take the most amount of time to fix, followed by performance bugs and other bugs.

We also considered the average time to see how fast the developer responds to fix the bug. Considering the average time (each fix attempt's time), security bugs are fixed fastest, i.e., developers seem to respond quickly to them. On the other hand, performance bugs take the largest time for each developer. One possible explanation for this is that they often require changes across different subsystems of the architecture (see Q3). The difficulty of fixing such bugs might cause them to be postponed to future releases/milestones.

Security bugs are reopened most. Figure 6 shows the number of times a bug is reopened. As we can see, most

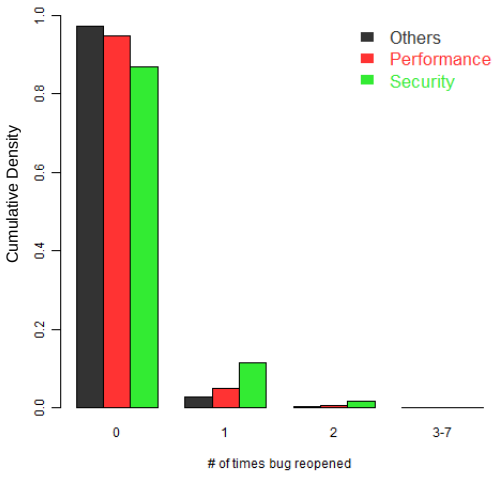

Figure 6: Histogram of number of times bug reopened

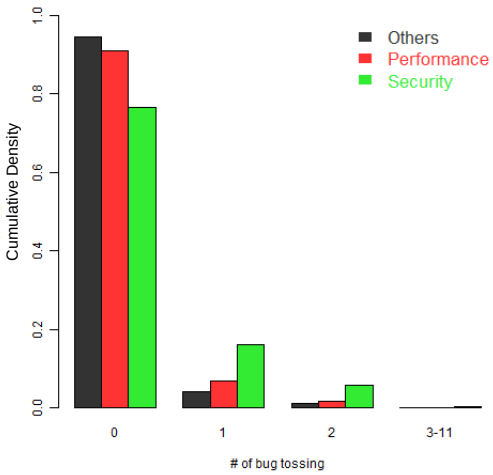

Figure 7: Histograms of number of bug tossing comparison

of the bugs are never reopened (count_reopened = 0), but more than 10% of the security bugs are reopened at least once. Pairwise t-test comparison of the distribution of the number of times bugs are reopened shows that the difference between the three groups is statistically significant ($p < 0.05$). On average, security bugs tend to be reopened more than twice as often as performance bugs and performance bugs tend to be reopened almost twice as often as other bugs. There are various possible explanations for this. Security bugs might be more complex to solve and test than others. Alternatively, while trying to fix the security bugs as fast as possible (as seen above), there is a possibility that developers are not able to do sufficient security testing for these bugs, which leads the bug to be reopened later. We plan to conduct a grounded theory study to better understand the rationale for those findings.

Security bugs are triaged faster than performance and other bugs. Statistical comparison of bug triage time reveals that performance bugs are not statistically different from other bugs in terms of bug triage time ($p = 0.15 > 0.05$), but security bugs have a smaller (70% smaller on average) triage time than others. A possible explanation for this might be that, to avoid the possibility of a security loophole being exploited, security bugs are kept secret (hidden) until the bug is fixed. Different software projects have different disclosure strategies [28].

Fast assignment is not identical to correct assignment. Our comparison of the number of bug tosses reveals that **security bugs are tossed more frequently than performance and other.** As shown in Figure 7 and Table I, security bugs are tossed more than performance (166.7% more) and other bugs (371.2% more). These differences among the three groups are significant. One common reason for tossing is incorrect assignment to a developer who does not own the defective source or may not have the expertise to fix the bug [27]. In any case, tossing events slow down the process and increase the total time to fix a bug.

While security bugs are re-opened and tossed more frequently, they are fixed and triaged the fastest.

3.2 Who Fixes Bugs?

Motivation: The task of finding the person with the most expertise to fix a bug is critical. In an empirical study on finding experts in a software development company, Ackerman and Halverson [29] observed that experience was the primary criterion engineers used to determine expertise [30]. For example, performance bugs are critical in the sense that they require thorough knowledge of the software system, compilation tool chain, execution bottlenecks and memory layout. Similarly, fixing a security bug requires the understanding of possible security loopholes in the source code. In our case study, we are particularly interested in finding any relation between the performance or security bug fixers and their development experience.

Approach: To measure the number of developers whose expertise was needed to fix a bug, we count the number of unique developers assigned to a bug in its life time. For example, if a developer was assigned twice to the same bug (because of tossing or re-opening) we count this developer as one developer assignment.

We measure the experience of a developer fixing a particular bug using the following two metrics:

- Number of previously fixed bugs by the developer.
- Experience in days, i.e., the number of days from the first bug fix of the developer to the current bug's fix date.

For both metrics, if a bug was fixed by more than one developer (because of tossing and/or re-opening), we take the average experience of these developers.

A bug tracking system like Bugzilla uses an email address as developer identification, yet a developer often has more than one email address [31]. For example, a bug could be fixed by John_Doe@mozilla.org and another bug could be fixed by John_Doe@gmail.com years ago, although both bug fixers are the same person. For all bugs, We use only the name part of the email address to deal with email aliasing, a technique commonly used for bug report and email mining [32, 33]

Findings: Security bugs require more developers to fix than performance and other bugs. This follows from Figure 8 and a t-test on the data in Table I (p

Table 1: Mean and median metric values for the three research questions.

Metric			Mean			Median		
			Security	Performance	Other	Security	Performance	Other
Time	Time between assignment and fix (total time in minutes)		41,588.81	116,501.93	88,011.43	10,211	13,016	9,078
	# of times bug reopened		0.15	0.06	0.033	0	0	0
	Bug Triage time (in minutes)		57,347.80	208,983.55	193,700.40	4,098.03	8,904.97	6,714.02
	# of bug tossing		0.344	0.129	0.073	0	0	0
Person	# of developers assigned		1.203	0.503	0.343	1	0	0
	Experience	# of days	1,238.92	1,011.87	926.639	1,178	946	820
		# of prior bugs fixed	808.448	777.424	660.249	505	472	389
Bug Fix	# of lines changed		222.966	401.459	195.203	75.5	66	26
	# of files changed		3.236	8.396	4.527	1	2	2
	Complexity (entropy)		0.814	0.55	0.51	0.903	0.628	0.627

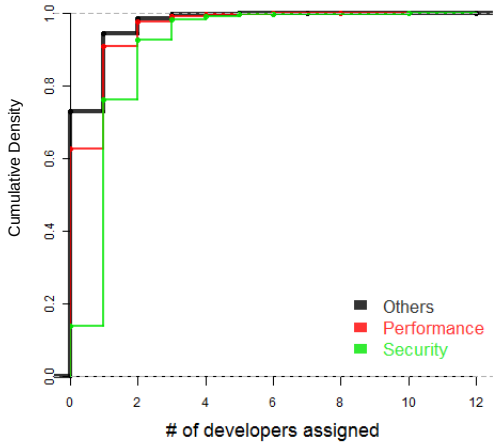

Figure 8: CDF of number of developers assigned

< 0.05). From the CDF plot in Figure 8, we see that 73% of other bugs and more than 62% of performance bugs are never assigned, compared to 14% of security bugs. Manual inspection of security bugs without developer assignment reveals that they are fixed very quickly, even before being assigned to someone. This is probably due to the security bug disclosure strategy followed by Firefox [28]. Alternatively, some security bugs are fixed before they are entered in the bug repository by the developer. On the other hand, performance and other bugs are often not assigned because they are found to be a 'duplicate' of other bug reports that were fixed already, or found to be 'invalid' or 'resolved'. **Security or performance bug fixers are more experienced than fixers of other bugs.** From Figure 9 and Table I, we can say, on average, security bug fixers and performance bug fixers have fixed a similar number of bugs before, but fixed significantly more number of bugs than other bug fixers. In terms of experience days, on average, security bug fixers are 22.45% more experienced than performance bug fixers, and performance bug fixers are 9.19% more experienced than other bug fixers (for both cases, the t-test holds with p value < 0.05). In short, it appears that security and performance bugs require more experience to fix.

On average, more developers are assigned to security bugs. Performance and security bugs require more experienced bug fixers than other bugs.

3.3 Characteristics of the Bug Fix

Motivation: So far, we have seen that security bugs require more bug fixers and are reopened more often than performance and other bugs. One possible reason for these findings might be the complexity of the bug fix. Consider an example of two bugs. In the fix of the first bug (the more complex bug), the developer had to change over a dozen files. When asked about the steps required to fix the feature, she or he may not recall half of them. For a bug fix for which the developer had to change only one file, recalling the changes required for that bug is much easier. In general, if we have a bug that required change across all or most of the files of a software system, developers will have a hard time keeping track of all these changes.

Approach: To quantify complexity, we used three metrics:

– Total number of lines added/deleted.

– Total number of files changed for fixing a bug.

– Bug fix Entropy [34].

Bug tracking systems and bug reports do not contain the source code of a bug fix. To get the actual data related to a bug fix, one needs to link the bug report to the code repository (CVS). For this, we used the algorithm described by Sliwerski et al. [35]

We implemented their algorithm for the Firefox CVS code repository. As this approach relies on the developer revision comments having a bug id, we did not get the fix information for all of the bugs: 303 performance, 174 security and 7,800 other bugs could be linked to their source code changes.

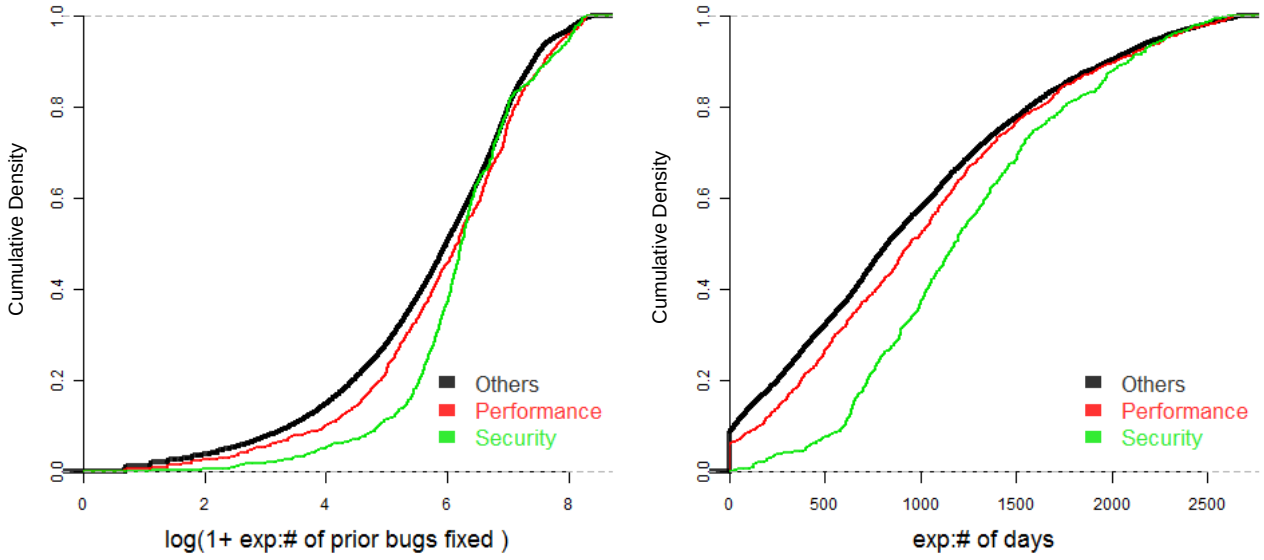

Figure 9: CDF of developer experience

Length of fix (number of lines added/deleted/edited) and spread across files (number of files changed) are straightforward to measure. To measure the bug fix entropy, we used the normalized Shannon Entropy, which is defined as: $H_n(P) = -\sum_{k=1}^{n}(p_k * \log_n p_k)$, where $p_k \geq 0, \forall_k \in 1, 2, ..., n$ and $\sum_{k=1}^{n} p_k = 1$. For a distribution P where all elements have the same probability of occurrence ($p_k = \frac{1}{n}, \forall_k \in 1, 2, ..., n$), we achieve maximum entropy. On the other hand, for a distribution P where only one element i has a probability of occurrence, we achieve minimal entropy (0).

For example, to fix a bug, three files A(3 lines), B(2 lines) and C(2 lines) are changed. We calculate the number of lines changed as the sum of the number of lines added and deleted. We define a file change probability distribution P as the probability that $file_i$ is changed for a bug's fix. For each file, we count the total number of lines changed in that file and divide by the total number of lines changed for all files. Hence, in our example, we have $p(file_A) = \frac{3}{7}, p(file_B) = \frac{2}{7}, p(file_C) = \frac{2}{7}$. Finally we can calculate the normalized Shannon Entropy for that bug's fix $H_n(P) = 0.982$. If another bug had the same total number of lines (7) changed in seven files, then Entropy would be, $H_n(P) = 1$. This means that the former bug fix is more concentrated, and hence less complex.

Findings: Performance bug fixes change more lines than fixes for other bugs. From Figure 11, Table I and t-test, we find that for fix size: number of lines changed, except for the pair of performance and other bugs , the difference was not statistically significant.

For the fix spread or number of files changed to fix a bug, we found that **fixing a performance bug requires change in more files than fixing a security bug.**

A more surprising result was found when we compared the entropy values. **Security bug fixes have the highest entropy**. There is a huge difference between security bugs and the other two groups of bugs. On average, security bug

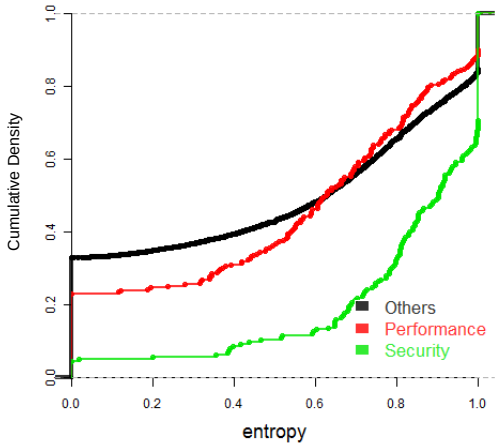

Figure 10: CDF of entropy

fixes have 48% more entropy than performance bugs, and 59.6% more entropy than other bugs. This means that security bugs are much more complex to fix than performance and other bugs. For example, we found that in the extreme case, for bug id 289940, there were 296 changes in the code repository. Further investigation showed that this bug revealed a security flaw that was extremely invasive: *"...The big part of this change is to mark all our internal events as trusted, where applicable. This will be done by changing the constructors of the various ns*Event classes .."*

Security bug fixes are more complex than performance and other bugs, yet they affect fewer files than performance bugs.

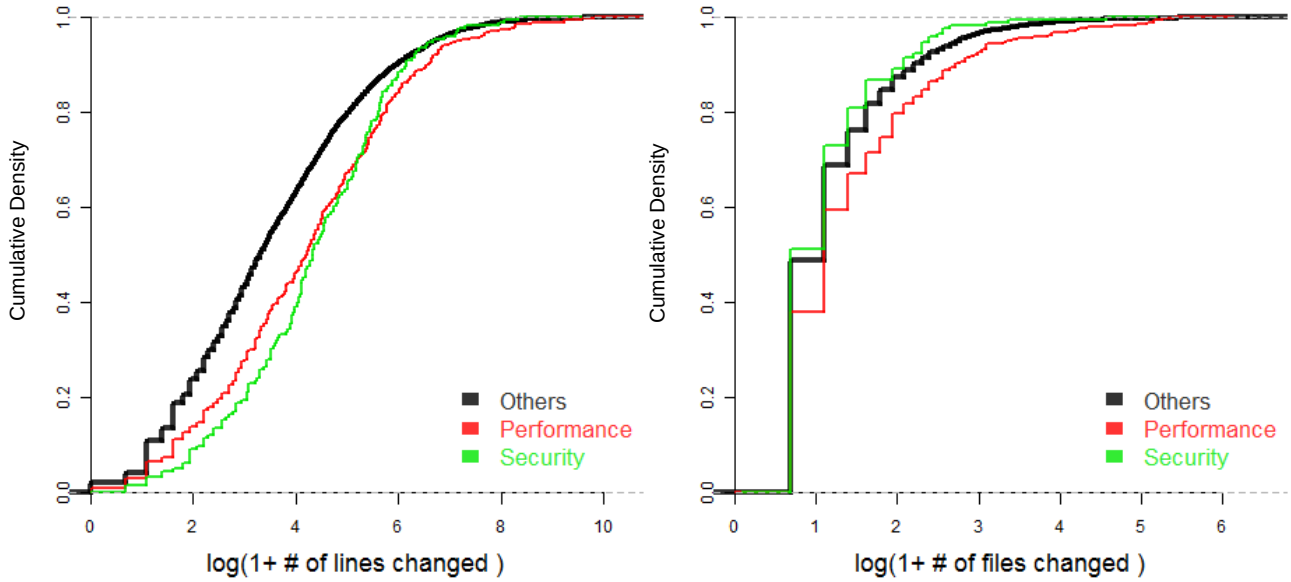

Figure 11: CDF of fix size

4. RELATED WORK

We discuss related work in the areas of bug classification. The literature contains several bug classification taxonomies [36]. The draft of the IEEE Standard Classification for Software Anomalies is a standard classification of software defects [37]. In this classification, software security and performance represent two out of six types of problems based on the effect of defects. Hamill et al. identify the common trends in software fault and failure data [38] by identifying fault types according to the source of the fault, i.e., design fault, data problem or requirements fault. Ahsan et al. propose an automatic bug triaging system based on a categorization of bug reports using text mining [39]. Gegick et al identify security bug reports via text mining [40]. We propose the use of classification of the bug reports according to their effect type (i.e., effects performance, security etc.) to improve software quality.

5. THREATS TO VALIDITY

Our empirical study focuses on one system (the Firefox web browser). It is not clear whether our results generalize to other open source systems (possibly in different domains), or to commercial systems.

We used heuristics on bug report titles and keywords to identify performance bugs. Since our study critically relies on bug classification, we statistically verified our heuristics for performance bug identification (section 2.2). A statistical sampling technique (with 95% confidence level and confidence interval of 10) showed that our heuristics have a high precision and recall.

Our data contains relatively few (847) security bugs in comparison to the large number of other (178,531) and performance (4,293) bugs. For the identification of security bugs, we are relying on the Mozilla Foundation Security Advisory and its links to Bugzilla. MFSA has been used in prior research as a source of security bug information [41, 42]. If there are any security bugs that were not linked to an MFSA advisory, they are treated as other bugs in this study. In future work, we also plan to use heuristics on bug comments and titles, similar to the case of performance bugs.

For collecting bug fix data using code repository revision comments, we used the information from revision comments using the algorithm described by Sliwerski et al. [35] (SZZ algorithm). Due to the limited number of revision comments with fix information and bug id, we could only get the associated fix code for 303 performance (20.2% of 1,503 performance bug fixes), 174 security (20.7% of 839 security bug fixes) and 7,800 other bug fixes (16.26% of 47,970 other bug fixes) for our study.

Finally, bugzilla fields for bug report time and bug history-related time do not necessarily correspond to the actual time.

6. CONCLUSION AND FUTURE WORK

In this empirical study, we analyzed the differences in time to fix, developer experience and bug fix characteristics between security, performance and the rest of the bugs to validate our conjecture that different type of bugs are different and hence quality assurance should take into account the bug type. We found that security bugs in Firefox behave differently than other. Security bugs require more developers with more experience, but need less triage time and are fixed faster than others. At the same time, security bug fixes are more complex than the fixes of performance and other bugs, and are reopened and tossed more than performance bugs.

Similar to security bugs, performance bugs require more experienced developers to fix, and more developers are involved in performance bugs than other bugs. In terms of bug triage time, performance bugs are not different from any other bugs. Furthermore, fixing a performance bug requires changes in more files, so performance bug fixers will need good knowledge of the system.

Different kinds of bugs have different characteristics, in particular regarding the time to fix, the person who fixes

bugs and the bug fix itself. Research on defect prediction and other models (time to fix, triaging model) should take this into account. Bug type-specific quality assurance could be used to shorten the time to fix, and optimize bug triaging.

We plan on performing case studies on other open source software systems from the same software domain and from other domains. From this, we will be able to generalize our findings. We also plan to consider additional types of non-functional bugs, like usability, serviceability, functionality and others [37].

7. ACKNOWLEDGEMENTS

The authors would like to thank Nicolas Bettenburg for sharing the Mozilla bugzilla bug database, Yasutaka Kamei for helping with his SZZ algorithm implementation and the linking of Bugzilla bugs to CVS changes, and Stephen Thomas for his help with the LDA analysis.

8. REFERENCES

[1] Emad Shihab, Akinori Ihara, Yasutaka Kamei, Walid M. Ibrahim, Masao Ohira, Bram Adams, Ahmed E. Hassan, and Ken-ichi Matsumoto. Predicting re-opened bugs: A case study on the eclipse project. In *Proceedings of the 17th Working Conference on Reverse Engineering*, WCRE '10, pages 249–258, Washington, DC, USA, 2010. IEEE Computer Society.

[2] L. Erlikh. Leveraging legacy system dollars for e-business. *IT Professional*, 2(3):17 –23, 2000.

[3] Thomas Zimmermann, Rahul Premraj, and Andreas Zeller. Predicting defects for eclipse. In *Proceedings of the Third International Workshop on Predictor Models in Software Engineering*, PROMISE '07, pages 9–, Washington, DC, USA, 2007. IEEE Computer Society.

[4] M. Cataldo, A. Mockus, J.A. Roberts, and J.D. Herbsleb. Software dependencies, work dependencies, and their impact on failures. *Software Engineering, IEEE Transactions on*, 35(6):864 –878, 2009.

[5] Marco D'Ambros, Michele Lanza, and Romain Robbes. On the relationship between change coupling and software defects. In *Proceedings of the 16th Working Conference on Reverse Engineering*, WCRE '09, pages 135–144, Washington, DC, USA, 2009. IEEE Computer Society.

[6] T.L. Graves, A.F. Karr, J.S. Marron, and H. Siy. Predicting fault incidence using software change history. *Software Engineering, IEEE Transactions on*, 26(7):653 –661, July 2000.

[7] Raimund Moser, Witold Pedrycz, and Giancarlo Succi. A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction. In *Proceedings of the 30th international conference on Software engineering*, ICSE '08, pages 181–190, New York, NY, USA, 2008. ACM.

[8] Christian Bird, Nachiappan Nagappan, Premkumar Devanbu, Harald Gall, and Brendan Murphy. Does distributed development affect software quality? an empirical case study of windows vista. In *Proceedings of the 31st International Conference on Software Engineering*, ICSE '09, pages 518–528, Washington, DC, USA, 2009. IEEE Computer Society.

[9] Nachiappan Nagappan, Brendan Murphy, and Victor Basili. The influence of organizational structure on software quality: an empirical case study. In *Proceedings of the 30th international conference on Software engineering*, ICSE '08, pages 521–530, New York, NY, USA, 2008. ACM.

[10] Lucas D. Panjer. Predicting eclipse bug lifetimes. In *Proceedings of the 4th International Workshop on Mining Software Repositories*, MSR '07, pages 29–, Washington, DC, USA, 2007. IEEE Computer Society.

[11] Cathrin Weiss, Rahul Premraj, Thomas Zimmermann, and Andreas Zeller. How long will it take to fix this bug? In *Proceedings of the 4th International Workshop on Mining Software Repositories*, MSR '07, pages 1–, Washington, DC, USA, 2007. IEEE Computer Society.

[12] Sunghun Kim and E. James Whitehead, Jr. How long did it take to fix bugs? In *Proceedings of the 3rd international workshop on Mining software repositories*, MSR '06, pages 173–174, New York, NY, USA, 2006. ACM.

[13] John Anvik, Lyndon Hiew, and Gail C. Murphy. Who should fix this bug? In *Proceedings of the 28th international conference on Software engineering*, ICSE '06, pages 361–370, New York, NY, USA, 2006. ACM.

[14] Zhen Ming Jiang. Automated analysis of load testing results. In *Proceedings of the 19th international symposium on Software testing and analysis*, ISSTA '10, pages 143–146, New York, NY, USA, 2010. ACM.

[15] Graham Kalton. *Introduction to Survey Sampling.* Sage Publications, Inc, 1983.

[16] Mozilla foundation security advisory. `http://www.mozilla.org/security/announce/`, February 2011.

[17] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3:993–1022, March 2003.

[18] Stephen W. Thomas, Bram Adams, Ahmed E. Hassan, and Dorothea Blostein. Validating the use of topic models for software evolution. In *Proceedings of the 2010 10th IEEE Working Conference on Source Code Analysis and Manipulation*, SCAM '10, pages 55–64, Washington, DC, USA, 2010. IEEE Computer Society.

[19] Jonathan I. Maletic and Andrian Marcus. Supporting program comprehension using semantic and structural information. In *Proceedings of the 23rd International Conference on Software Engineering*, ICSE '01, pages 103–112, Washington, DC, USA, 2001. IEEE Computer Society.

[20] Denys Poshyvanyk and Andrian Marcus. Combining formal concept analysis with information retrieval for concept location in source code. In *Proceedings of the 15th IEEE International Conference on Program Comprehension*, pages 37–48, Washington, DC, USA, 2007. IEEE Computer Society.

[21] Adrian Kuhn, Stéphane Ducasse, and Tudor Gírba. Semantic clustering: Identifying topics in source code. *Inf. Softw. Technol.*, 49:230–243, March 2007.

[22] Pierre F. Baldi, Cristina V. Lopes, Erik J. Linstead, and Sushil K. Bajracharya. A theory of aspects as latent topics. In *Proceedings of the 23rd ACM SIGPLAN conference on Object-oriented programming*

systems languages and applications, OOPSLA '08, pages 543–562, New York, NY, USA, 2008. ACM.

[23] Scott C. Deerwester, Susan T. Dumais, Thomas K. Landauer, George W. Furnas, and Richard A. Harshman. Indexing by Latent Semantic Analysis. *Journal of the American Society of Information Science*, 41(6):391–407, 1990.

[24] A. Hindle, M.W. Godfrey, and R.C. Holt. What's hot and what's not: Windowed developer topic analysis. In *Proceedings of the 25th IEEE International Conference on Software Maintenance*, ICSM '09, pages 339 –348, 2009.

[25] Firefox LDA outputs. `http://www.cs.queensu.ca/~zaman/fx-lda/`, February 2011.

[26] Andreas Zeller. *Why Programs Fail: A Guide to Systematic Debugging*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.

[27] Gaeul Jeong, Sunghun Kim, and Thomas Zimmermann. 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*, ESEC/FSE '09, pages 111–120, New York, NY, USA, 2009. ACM.

[28] Stephan Neuhaus and Thomas Zimmermann. Security trend analysis with cve topic models. In *Proceedings of the 21st International Symposium on Software Reliability Engineering*, ISSRE '10, pages 111–120, Washington, DC, USA, 2010. IEEE Computer Society.

[29] Mark S. Ackerman and Christine Halverson. Considering an organization's memory. In *Proceedings of the 1998 ACM conference on Computer supported cooperative work*, CSCW '98, pages 39–48, New York, NY, USA, 1998. ACM.

[30] A. Mockus and J.D. Herbsleb. Expertise browser: a quantitative approach to identifying expertise. In *Proceedings of the 24rd international conference on Software engineering*, ICSE '02, pages 503 – 512, 2002.

[31] C. R. Reis and R. P. de Mattos Fortes. An overview of the software engineering process and tools in the mozilla project. In *Proceedings of the Open Source Software Development Workshop*, pages 155–175, 2002.

[32] Christian Bird, Alex Gourley, and Prem Devanbu. Detecting patch submission and acceptance in oss projects. In *Proceedings of the 4th International Workshop on Mining Software Repositories*, MSR '07, pages 26–, Washington, DC, USA, 2007. IEEE Computer Society.

[33] Christian Bird, Alex Gourley, Prem Devanbu, Michael Gertz, and Anand Swaminathan. Mining email social networks. In *Proceedings of the 3rd international workshop on Mining software repositories*, MSR '06, pages 137–143, New York, NY, USA, 2006. ACM.

[34] Ahmed E. Hassan. Predicting faults using the complexity of code changes. In *Proceedings of the 31st International Conference on Software Engineering*, ICSE '09, pages 78–88, Washington, DC, USA, 2009. IEEE Computer Society.

[35] Jacek Śliwerski, Thomas Zimmermann, and Andreas Zeller. When do changes induce fixes? *SIGSOFT Softw. Eng. Notes*, 30:1–5, May 2005.

[36] Kai Pan, Sunghun Kim, and E. James Whitehead, Jr. Toward an understanding of bug fix patterns. *Empirical Softw. Engg.*, 14:286–315, June 2009.

[37] Draft standard for ieee standard classification for software anomalies. *IEEE Unapproved Draft Std P1044/D00003, Feb 2009*, 2009.

[38] M. Hamill and K. Goseva-Popstojanova. Common trends in software fault and failure data. *IEEE Transactions on Software Engineering*, 35(4):484 –496, 2009.

[39] Syed Nadeem Ahsan, Javed Ferzund, and Franz Wotawa. Automatic software bug triage system (bts) based on latent semantic indexing and support vector machine. In *Proceedings of the 4th International Conference on Software Engineering Advances*, ICSEA '09, pages 216–221, Washington, DC, USA, 2009. IEEE Computer Society.

[40] M. Gegick, P. Rotella, and Tao Xie. Identifying security bug reports via text mining: An industrial case study. In *Proceedings of the 7th international workshop on Mining software repositories*, pages 11 –20, May 2010.

[41] Yonghee Shin and Laurie Williams. An empirical model to predict security vulnerabilities using code complexity metrics. In *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*, ESEM '08, pages 315–317, New York, NY, USA, 2008. ACM.

[42] I. Chowdhury and M. Zulkernine. Using complexity, coupling, and cohesion metrics as early indicators of vulnerabilities. In *Special Issue on "Security and Dependability Assurance of Software Architectures," Journal of Systems Architecture*, 2010.