A Qualitative Study on Performance Bugs

Shahed Zaman*, Bram Adams† and Ahmed E. Hassan*
*SAIL, Queen's University, Canada
{zaman,ahmed}@cs.queensu.ca
†MCIS, École Polytechnique de Montréal, Canada
bram.adams@polymtl.ca

***Abstract*—Software performance is one of the important qualities that makes software stand out in a competitive market. However, in earlier work we found that performance bugs take more time to fix, need to be fixed by more experienced developers and require changes to more code than non-performance bugs. In order to be able to improve the resolution of performance bugs, a better understanding is needed of the current practice and shortcomings of reporting, reproducing, tracking and fixing performance bugs. This paper qualitatively studies a random sample of 400 performance and non-performance bug reports of Mozilla Firefox and Google Chrome across four dimensions (Impact, Context, Fix and Fix validation). We found that developers and users face problems in reproducing performance bugs and have to spend more time discussing performance bugs than other kinds of bugs. Sometimes performance regressions are tolerated as a trade-off to improve something else.**

***Keywords*-Performance bugs, qualitative study, Mozilla Firefox, Chromium.**

I. Introduction

Software performance is one of the most influential non-functional requirements [1], with the power to make or break a software system in today's competitive market. It basically measures how fast and efficiently a software system can complete a certain computing task [2]. Resolving performance bugs, i.e., performance problems like high resource utilization or slow response time is as important as adding new features to the system to keep the users happy and loyal to the software system.

Despite being an important software quality problem, there has not been significant research on performance bugs. Research has focused especially on improving the overall software quality by doing qualitative and quantitative studies on software bugs in general. For example, software defect prediction models use metrics derived from software bugs in general to predict the number of defects and the locations where to fix them [3] [4] [5]. Other work focuses on different types of bugs like security bugs, or usability bugs [6] [7] [8] [9].

However, performance bugs are different and require special care. Our earlier work on performance bugs found that performance bugs take more time (at least 32% more on average) to fix, are fixed by more experienced developers and require changes to larger parts of the code than non-performance bugs [10]. Unfortunately, our analysis primarily considered high-level quantitative data like the overall bug fix time, the number of developers involved, and the size of the bug fix, but ignored qualitative process data like the content of discussions and the identity of people involved in the bug fix process. Although we were able to pinpoint differences between performance and other kinds of bugs, we could not fully explain these findings or make concrete suggestions to improve performance bug handling.

In order to understand the differences in the actual process of fixing performance and non-performance bugs, and potentially identify concrete ways to improve performance bug resolution, this paper qualitatively studies performance bug reports, comments and attached patches of the Mozilla Firefox and Google Chrome web browsers. More in particular, we studied 100 performance and 100 non-performance bug reports and comments from each project (Mozilla Firefox and Google Chrome, 400 bugs in total) to discover the knowledge, communication and collaboration specific to fixing performance problems in a software system. We qualitatively compared the sampled performance and non-performance bugs across four dimensions and 19 sub-dimensions. These are our major findings for each dimension:

1) *Impact on the stakeholder.*
   Performance bugs suffer from tracking problems. For example, Chrome has seven times more regression bugs that are performance-related than non performance-related. In Firefox, 19% of the performance bugs were fixed out of the blue without any concrete link to the patch or change that fixed them. Furthermore, in 8% of the performance bugs of Firefox, bug commenters became frustrated enough to (threaten to) switch to another browser, compared to only 1% for non-performance bugs.
2) *Context of the bug.*
   Despite the tracking problems and low reproducibility of performance bugs, 34% of performance bugs in Firefox and 36% in Chrome provided concrete measurements of e.g., CPU usage, disk I/O, memory usage and the time to perform an operation in their report or comments. In addition, twice (32%) as many

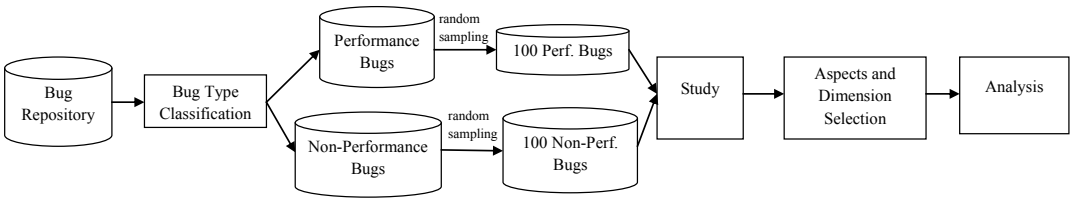

Figure 1. Overview of our approach to qualitatively study the performance bugs of a system.

performance bugs as non-performance bugs in Firefox provided at least one test case in their bug report or comment.

3) *The bug fix.*
Fixing performance bugs turns out to be a more collaborative activity than for non-performance bugs. 47% of the performance bugs in Firefox and 41% in Chrome required bug comments to co-ordinate the bug analysis, while only 25% of the non-performance bugs required them. Some performance regression bugs are not fixed on purpose.

4) *Bug Fix Validation.*
We found no significant difference in how performance bug fixes are reviewed and validated compared to non-performance bugs.

Overview of the paper: Section II explains our case study approach. Section III is divided into four sub-sections according to the dimensions considered in our study. Each sub-section of Section III discusses our findings for the different sub-dimensions found in the specific dimension. In section IV, we summarize the findings of this paper with respect to our previous quantitative study on Firefox performance bugs [10]. In section V, we discuss the threats to validity of our study. Finally, Section VI discusses related work and Section VII presents the conclusion of the paper.

II. Case Study Approach

This section presents the design of our qualitative study on performance bugs. Figure 1 shows an overview of our approach. For a particular project, we first extract the necessary data from its bug repository. Then, we identify the bug reports related to performance and randomly sample 100 performance and 100 non-performance bugs. We then studied the sampled bugs to identify the different dimensions and their sub-dimensions that can be used to compare the bug fixing process of performance and non-performance bugs. Finally, we perform our analysis based on these sub-dimensions. This section elaborates on each of these steps.

A. Choice of Subject System

In our case study, we study the Mozilla Firefox and Google chrome web browsers, since these are the two most popular web browsers with publicly available bug tracking system data. We focus on the domain of web browsers, since for web browsers that support multiple platforms and system environments, performance is one of the major software quality requirements. For that reason, performance bugs are reported in both systems and are tagged explicitly in the issue tracking systems of both projects. We considered Mozilla's Bugzilla data from September 9, 1994 to August 15, 2010 and Chrome's issue tracker data from August 30, 2008 to June 10, 2010.

For Firefox, we first had to identify the bugs that are related to the Firefox web browser, since Mozilla's bug tracking system manages multiple projects. It contains 567,595 bug reports for all different Mozilla components combined, like Mozilla 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 the Firefox component. We have reused the same dataset for Firefox as we had in our previous work [10]. For the Chrome web browser, our data was extracted from the Chrome issue tracker dump provided in the MSR challenge 2011 [11].

B. Bug Type Classification

In Bugzilla (the issue tracking system for Mozilla Firefox), the "keyword" field is tagged with "perf" for performance bugs. In Google's issue tracker for Chrome, the "label" field is tagged as "performance" for performance related bugs. However, in both projects, this tagging is not mandatory, and we found many performance bugs that do not have any such tag.

Hence, to identify 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 (as intended), we found that there are many bug reports containing the word 'perfect', 'performing' or 'performed', without having any relation to performance problems. We had to automatically exclude these keywords from the list using regular expressions. In previous work [10], we used this heuristics-based approach on the Firefox bug data and classified 7,603 bugs (4.14% out of 183,660 bugs) as performance-related. Manual analysis of a random sample of 95 bugs classified as performance-related and 96 bugs classified as non-performance related showed that our heuristics had a precision of 100 $\pm$ 10% and recall of 83.3 $\pm$ 10% (see confusion matrix in Table II). The sample sizes were chosen to obtain a confidence interval of 10.

Table I
TAXONOMY USED TO QUALITATIVELY STUDY THE BUG REPORTS OF PERFORMANCE AND NON-PERFORMANCE BUGS. BOLD SUB-DIMENSIONS SHOW A STATISTICALLY SIGNIFICANT DIFFERENCE IN AT LEAST ONE PROJECT.

Impact on the User	**Context of the bug**	**The Fix**	**Fix Validation**
Regression **Blocking** **WorksForMe after a long time** **People talking about switching**	**Measurement Used** **Has test cases** - by reporter ·with report - later **Contains stack traces** - in report - in follow-up Has reproducible info - in report - in follow-up **Problem in reproducing** Duplicate Bug Improvement suggestion **Reported by a project member**	**Problem discussion in comments** **Dependent on other bug** **Other bugs depend on this bug** Reporter has some hint on the actual fix Patch uploaded by - reporter ·with report ·later - Others	Discussion about the patch Technical Non-Technical Review Super-review

Using the same approach for the Google Chrome issue tracking system data, we classified 510 bugs as being performance-related (1.13% out of 44,997 bugs). In order to estimate precision and recall for this dataset, we again performed a statistical sampling with a 95% confidence level and a confidence interval of 10. We randomly selected and checked 81 performance (out of 510 bugs classified as being performance-related) and 96 non-performance (out of 44,487 bugs classified as being non-performance) bugs [12]. Out of these 81 bugs classified as being performance-related, 73 were actual performance bugs, yielding a precision of 90.12 $\pm$ 10%. Out of the sampled 96 bugs classified as being non-performance, 94 were indeed non-performance bugs, yielding a recall of 97.33 $\pm$ 10%. In other words, the heuristics seem to apply to Chrome as well.

Table II
CONFUSION MATRIX FOR THE DATASETS

Actual $\rightarrow$	Firefox		Chrome	
Prediction $\downarrow$	**P**	**NP**	**P**	**NP**
P	95	0	73	8
NP	19	77	2	94

C. Selection of Samples

For our 4 datasets of performance and non-performance bugs of Firefox and Chrome, the largest one is the set of Firefox non-performance bugs, with 176,057 bugs. With a 95% confidence level and 10% confidence interval, the sample size should be 96 bug reports. This means that if we find a sub-dimension to hold for n% of the bugs (out of these 96 non-performance Firefox bugs), we can say with 95% certainty that n $\pm$ 10% of the bugs contains that sub-dimension. The datasets with less bugs required less than 96 bug reports to be sampled to ensure this 10% confidence interval in the result. However, to simplify, we randomly sampled 100 bug reports from each dataset, resulting in a confidence interval of 10% or less for each dataset.

D. Identification of bug report and comment dimensions

Before being able to compare performance and non-performance bugs, we first need to establish the criteria to compare on. The different fields and characteristics recorded by bug repositories are a good start, but our study requires knowledge of how testers and developers collaborate on and think about (non-)performance bugs. This knowledge is part of the natural language content of bug comments and discussions, but we are not aware of any taxonomy or other work for analyzing qualitative bug report data. Hence, we first performed a manual study of the sampled data to identify such a taxonomy.

The first author studied each sampled bug report and tagged it with any word or phrase describing the intent or content of the bug comments and discussions. Each time new tags were added, older reports were revisited until a stable set of 19 tags was identified with which we could tag all reports. Afterwards, we analyzed the tags and grouped them into 4 clusters ("dimensions"): the impact of the bugs on stakeholders, the available context of the bug, the actual bug fix and the validation of the bug fix. The resulting dimensions and "sub-dimensions" (tags) are shown in Table I and are used to compare performance and non-performance bugs on. We discuss the sub-dimensions in more detail in our study results.

In order to determine, for a particular system, whether the percentage of performance bugs related to a sub-dimension (e.g., Blocking) is statistically significantly higher than the percentage of non-performance bugs related to that sub-dimension, we used the "joint confidence interval" or "comparative error". This measure is often used to compare two independent samples [13]. If the difference between the percentage of performance and non-performance bugs for a sub-dimension is greater than their calculated comparative error, then the difference is considered to be statistically significant.

III. Study Results

Each subsection below discusses one of the four dimensions in our study on using Mozilla Firefox and Google Chrome data. For each dimension, we present the description of the dimension, the sub-dimensions and a discussion of our findings. All differences between performance and non-performance bugs mentioned in our findings are statistically significant unless stated otherwise. Any percentage value mentioned in our findings has a maximum confidence interval of 10%.

A. Impact on the stakeholder

Since different stakeholders are involved with a software system, e.g., developers, users, testers and managers, a bug can impact any of them. This impact can vary widely, from blocking a software release, to an annoying regression bug that pops up again, or to small annoyances in user experience. For example, release blocking bugs typically require immediate attention and priority from the developer assigned, while a small hiccup during start-up might be forgiven. In the worst case, failure to adequately fix a major bug can incite users to switch to a competitor's product when tired of a software problem. To understand the impact and severity of performance bugs and how stakeholders deal with that, we need to study this dimension.

Sub-dimensions:

Regression: A regression bug is caused by a certain event (like software upgrade, another bug fix patch, or daylight saving time switch) that reintroduces a previously fixed problem to the system [14]. This problem can be a performance problem or any other problem, like functionality or security-related. It typically is (re-)discovered via a failing test. Both bug tracking systems support the tagging of regression bugs, but do not enforce it. We identified the regression bugs from existing tagging and by studying the report details. The patch, build or version from which the regression started is typically also indicated.

Blocking: By "blocking release", we refer to bugs that prevent a release from being shipped, typically because they are showstopper bugs. These bugs are flagged in the 'Flags' field of a bug report in Firefox and labeled as 'ReleaseBlock' in Chrome. That label or flag also mentions the specific release version that is blocked. For example, in Chrome, the label mentions either 'Beta', 'Dev' or 'Stable' meaning that the bug blocks that release channel. If while working on a bug report, the project contributor feels that the bug has to be fixed before that release, she tags it as a release blocker.

WorksForMe after a long time: In our study, we found many bugs that were reported, confirmed to be valid, then closed after a long time with a WFM (Works For Me), Fixed or Won't Fix status. By "closed after a long time", we mean that a bug report is closed after several months and in many cases, more than a year. The 'WFM' (WorksForMe) status is specific to the Mozilla project and indicates that the problem mentioned by the reporter was not reproducible by the project contributor or others. Instead of 'WFM', the Chromium project uses 'Won't Fix' or 'Invalid' as a closing status for such bugs.

People talking about switching: We counted the number of bugs where in the bug comments, at least one user mentioned being unhappy and thinking about switching to another browser because of that bug. Out of the millions of users of these browser, a comment from a few people may not seem to be significant. However, since bug reporters are tech-savvy users, we can expect that each of them voices the opinion of many more other users in total.

Findings:

22% of the performance and 3% of the non-performance bugs are regression bugs [Chrome].

A similar phenomenon occurred in Firefox, but the difference was not significant (19% vs. 13%). Problems related to non-performance regressions include browser crashes, missing functionality, and unexpected behavior.

Since performance regressions are common problems, both projects have a dedicated framework to detect and report them. In Firefox, regression bugs are reported like any other performance bug, then identified and/or confirmed (and usually tagged) as a regression by the reporter and/or contributors. In Chrome, most of the regression bugs found in our study are manually reported by project members who detected the regression in the automated compile/test cycle. The report contains a link to automatically generated graphs that show the performance deviation between the build under test and the previous build. The reporter also provides some notes specifying the specific metric that showed the deviation as there are many metrics plotted in those graphs. Most of the Firefox regression bugs were reported after the software affected the user, while Chrome's bug reports are showing an early detection practice.

Not all regressions are unexpected.

We found regression bugs that, although unexpected by the reporter, turned out to be known and tolerated by developers. Often, the performance bug was tolerated because of a trade-off with a more essential performance feature. In Chrome, we found the following comment in a regression bug (bug # 7992, comment # 3) that was closed with "WontFix" status.

"*I talked to dglazkov and darin. This regression is expected and OK. dglazkov mentioned rebaselining the test, but I'm not sure that matters.*
The ... is a heavy user of the WebCore cache, and recent changes tilted the tuning toward memory usage instead of speed. If you look at the memory usage, you'll see a corresponding drop where the perf slows down."

Performance improvement may cause regression.

We found different bug comments where a performance regression was introduced to the system while trying to optimize the performance. For example, in Firefox bug #449826,

the bug was reported as a possible performance improvement suggestion, but the resulting performance ended up being much slower and still open since September, 2008.

14% of the performance bugs and 4% of the non-performance bugs are release blocking [Firefox].

A similar finding, but not significant, was found in Chrome (8% vs. 4%). Release blocking bugs have the highest priority to the developers.

19% of the performance and 6% of the non-performance bugs are marked as WFM/Fixed/Won'tFix after a long time [Firefox]

Although the final outcome of these three groups was different, in all of them bugs have a traceability problem as we do not know how the bug was fixed, what caused the bugs, or why this bug will not be fixed. The reporter and other commenters who participated in the discussion of the bug report did not receive a response for a long time.

There are various reasons behind the popularity of these kind of bug reports. For example, these bug reports might have been insufficient to work on a fix, or the bug might have been too complex. Ideally, some note on the state of the bug should have been recorded. This type of bug was also found in Chrome without any significant difference (10% perf. vs. 9% non-perf.)

8% of the performance and 1% of the non-performance bugs drove at least one user to threaten to switch to another browser [Firefox].

In Chrome, 7% of performance and 2% of non-performance bugs are of this type and the difference was not statistically significant. These percentages do not include comments by users who are just frustrated with a bug. 7 users threatening to leave Firefox might not seem much, but for each user that reports a bug, X other non-contributor users encounter the same issue and feel the same pain. For example, in a comment on Firefox bug #355386, (comment #10), the user said:

"*... This bug is really bothering us and forcing us [his company] to recommend IE +SVG Adobe plugin to customers...* ".

Surprisingly, this bug was only closed with WFM status after more than 2 years. In the Chromium project, a commenter said (bug #441, comment #6):

"*I no longer use Chrome/Chromium for anything much and can no longer recommend to any of my friends or acquaintances or random people in the street that they use it. I still try it regularly in the hopes that the issue has been sorted out but as soon as the problem shows up I switch back to one of the others.*". Again, in this case, the problem was found to be not happening (WFM) anymore and closed with the status 'WontFix' after more than 10 months.

Users and developers compare the performance with other browsers.

In both projects, we found that users and developers compare their product performance with other competitors. While submitting a bug report in the Chromium project, the default bug report template suggests the reporter to provide information if the same problem was observed in other browsers. Mozilla bug reports usually also contain a comparison with other browsers. This is a common and expected practice irrespective of the bug type. It came as a surprise to us, however, to find bug reports in Google Chromium declared by the developers as "WontFix" only because the other popular browsers have the same performance issue or their browser at least was not slower than others. Users were not amused by this. For example, in Chrome, we found a comment (bug # 441, comment # 15) saying:

"*Thats a poor reason for closing a verifiable bug in something other than the rendering engine.*
Just becuase firefox consumes 100% cpu on something as basic as scrolling doesnt mean chrome should."

One acceptable case of this pattern, however, happens when comparing Chrome functionality to that of the Safari browser. Since both share the same layout engine "WebKit", many performance bugs had their root cause in webkit. In that case, the bug is identified as a "webkit" bug, reported to the webkit project's issue tracking system and the corresponding link is added to the Chrome bug report as a comment.

B. Context of the Bug

This second dimension captures the context available about a bug when reported in an issue tracking system. Bettenburg et al. did a survey on developers and users where they identified the factors that are considered to be helpful by the developers and reporters [15]. Both developers and reporters accepted that 'steps to reproduce' in bug reports are the most helpful information. Test cases and stack traces were next in the list. In the context of performance bugs, we need to know if these kinds of information are used more/less/similarly in order to understand the performance bug reporting process. Access to more relevant context can reduce the performance bug fix time and thus the overall quality of software.

Sub-dimensions:

Measurement used: We counted the number of bugs that include any kind of measurements in their report. By "measurement", we mean the use of any numerical value (e.g., CPU usage, disk I/O, or memory usage) in the bug report or comments to describe the problem.

Has test cases: Test cases are provided by the bug reporter or any other commenter to help reproduce the problem. We found different forms of test cases, i.e., specific web links, and attached or copy-pasted HTML/JavaScript files. In Chrome, very few bugs had test cases attached to them.

Instead, they use problematic web links as test case. We found the use of attachments more convenient as web links may become unavailable or are expected to be changed.

Contains stacktrace: Stack traces are also provided by the reporter and sometimes by other people in the bug comments. We counted the number of bugs where a stack trace was provided in the report or in the comments, since we also found bugs where the developer had to request for a stack trace to the bug reporter.

Has reproducible info and problem in reproducing: It is common practice in bug reporting to provide a step by step process to reproduce a bug. This process should include any non-default preference, hardware requirement and software configuration needed to trigger the bug. However, we found many bugs where despite having the steps to reproduce the bug, people had substantial problems reproducing the bug and requested more information or suggestions for reproducing the bug. We counted the number of bugs where commenters mentioned such problems in reproducing.

Reported by a project member: Chromium bug reports contain information about whether the bug is reported by a project member or someone else. In Firefox bug reports, we could not see any such information. Hence, we counted the number of performance and non-performance bugs reported by a project member only for Chrome.

Duplicate bug: Bug reports contain information about duplicate bugs, i.e., bugs that are in fact identical to an already reported bug without the reporter noticing this. We found that duplicate bugs are afterwards identified and marked by the project contributors and sometimes by the reporter. Firefox has a field "duplicates" where all the duplicate bugs are listed. In Chrome, we identified duplicate bugs from the bug status "Duplicate" or from comments like, *"Issue ... has been merged into this issue"*. We counted the number of bugs with duplicates for all sampled performance and non-performance bugs in our study.

Improvement suggestion: We counted the number of bugs that were about improvement (both performance and non-performance) of the software, not about defects. We identified bugs related to improvement by checking if the report is about speed, resource utilization, usability or code quality improvement instead of a specific bug.

Findings:

34% of the Firefox and 36% of the Chrome performance bugs contain performance measurements.

More than one third of the performance bug reports and/or comments provided measurements of different performance metrics. We observed an interesting difference in the use of measurements between Chromium and Firefox. In Firefox, the measurements included in bug reports were numerical, i.e., CPU usage, disk I/O, memory usage, time to perform an operation, or profiler output. In addition to these numerical values, Chrome performance bug reports also contained the metric values calculated using Chromium's performance test scripts that run on "buildbot", the continuous integration server used by Chromium. Although Mozilla also had test suites for the "tinderbox" [16] continuous integration server, we could not find any bug reports in our sample that referred to this test suite.

As expected, we could not find any non-performance bug that had any kind of measurement mentioned in its bug report, since it is hard to quantify a functional bug.

32% of the performance and 16% of the non-performance bug reports or comments contain a test case [Firefox].

Most of the time (more than 70% of the time for both bug types and projects), test cases are uploaded by the reporter. In Chrome, the difference was not statistically significant (21% performance vs. 16% non-performance).

10% of the performance and 1% of the non-performance bugs contain a stack trace in the report or a comment [Chrome].

In Chrome, we could not find a statistically significant difference between performance (13%) and non-performance bugs (8%). Stack traces are normal for functional bugs that cause the system to crash. Sometimes, performance bugs make the software slow, which eventually causes a crash as well. Furthermore, performance bugs may also cause the software to hang, which requires the software to be manually aborted. Stack traces are used in such cases for debugging from the sequence of functions called just before the software hung or crashed.

15% of the performance and 6% of the non-performance bugs had comments related to problems with reproducing the bug [Firefox].

Insufficient information provided by the reporter or the unavailability of a test case is one of the most common reasons behind being unable to reproduce any bug. Moreover, for many performance bugs, the problem was caused by the use of a specific (or a specific version of an) extension or plugin of which the reporter could not collect sufficient information. We often found that in the bug comments, commenters guide the reporter by giving instructions and web links on how to collect the required context that will help the commenter and other people to reproduce the bug. Intermittent problems are also found to be hard to reproduce. In Chrome, 16% of the performance and 8% of the non-performance bugs had comments related to problems in reproducing, but this difference between performance and non-performance bugs was not statistically significant.

58% of the performance and 35% of the non-performance bugs are reported by a project member [Chrome].

One possible reason of this difference may be that performance regression bugs found by an automated test script

are always reported by a project member. We found that out of the 22% performance regression bugs in Chrome more than 86% were reported by a project member, which shows a larger involvement of project members in reporting performance bugs.

In both projects, many performance bugs are caused by external programs.

We found that 11% of the performance bugs in Firefox and 14% of the performance bugs in Chrome are caused by a program that was not developed within the project. Investigations of these bugs showed performance problems caused by an "add-on" or "plugin" (i.e., flash), a single web application/page (i.e., yahoo mail or gmail) or any other third party software in the system (i.e., antivirus).

In Firefox, comment # 1 of bug # 463051 in November, 2008 mentioned,

"*Report problems caused by external programs to their developers. Mozilla can't fix their bugs. ...*".

The bug was marked as resolved with resolution "invalid". In the next comment, which was submitted more than two years later, we found a comment saying,

"*We're now tracking such bugs. This doesn't mean it's something we can fix, merely something we hope to be able to point vendors to so they can investigate. This is an automated message.*".

After that comment in March, 2011, the bug was assigned a new QA (Quality Assurance) contact and since then, the status of that bug is "unconfirmed".

The Chromium issue repository kept track of these issues from the beginning, to support the QA team in collaborating with the users and/or third party developers facing the reported problem.

Non-performance bugs have more duplicate bugs reported than performance bugs [Firefox].

In Firefox, the difference was significant (26% performance vs. 40% non-performance), while we could not find a statistically significant difference in Chrome (24% performance vs. 32% non-performance).

Although steps to reproduce are used in most of the bug reports, we could not find any statistically significant difference between their use in performance and non-performance bugs. We also could not find any significant difference for the sub-dimension "improvement suggestion".

C. The Bug Fix

Process metrics like the response time of the fix process can be used to measure software quality [17]. However, in order to improve the performance bug fixing process and overall quality, we need to know how, given the context of the second dimension, a performance bug is fixed and what (if any) makes the performance bug fixing process different from other bugs in practice.

Sub-dimensions:

Problem discussion in comments: We studied the bug comments where people discussed the problems related to the bug report and the uploaded patch for that bug. To distinguish between problem discussion and patch discussion in Firefox, we considered all comments before the first patch upload as problem discussion, and all comments after the first patch upload as patch discussion (see Bug Fix Validation dimension). As Chrome uses a different tool for patch review than the issue tracking system, we considered all bug comments as problem discussion.

Dependent on other bug or other bug depends on the bug: We also studied the dependency information provided in the issue tracking system. When a bug's fix depends on the resolution of another bug, this dependency is provided in both issue tracking systems in two different fields ("Depends on" and "Blocking").

Reporter has some hint on the actual fix: We found many bugs (especially improvement suggestion bugs) where the reporter had an idea about the fix and provided a hint and sometimes the patch with the bug report. We counted the number of performance and non-performance bugs of this type.

Patch uploaded by: We found bug patches that were uploaded by its reporter, sometimes within a few minutes after reporting. We counted the number of bugs where the patch was uploaded by the reporter (immediately after reporting/later) and by others.

Findings:

Performance bugs are discussed more than non-performance bugs in order to develop a fix.

47% of the performance bugs of Firefox and 41% of the performance bugs in Chrome required a discussion in the bug comments, while only 25% of the non-performance bugs in both projects had such a discussion. By discussion, we mean the process of exchanging ideas in order to understand a problem, identify the possible fix, and reach a decision. We did not count those bugs that had only comments saying something like, "it happens to me too" rather than trying to find out the reason for that problem. As such, this finding suggests that performance bugs are harder to resolve, or at least require collaboration between multiple people.

Table III
NUMBER OF BLOCKING AND DEPENDENT BUGS (BOLD NUMBERS HAVE STATISTICALLY SIGNIFICANT DIFFERENCE).

Project	Type	Blocking	Depends on
Firefox	**Perf.**	**44**	**42**
	Non-perf.	**27**	**15**
Chrome	Perf.	2	6
	Non-perf.	6	4

Performance bugs are more dependent on other bugs and more performance bugs are blocking than non-performance bugs [Firefox].

Table III shows the dependency information found in our studied bugs. When 'B' blocks the resolution of bug 'A', bug 'B' is marked as blocking bug 'A' and bug 'A' is marked as depending on 'B'. Only Firefox is showing a difference between performance and non-performance bugs (42% vs. 15% for depends on and 44% vs. 27% for blocking) with statistical significance. Very few dependency information was found in the Chromium project and the difference was not statistically significant.

Patches are not always used for fixing a bug.

We found that many of the performance bugs in Chromium project are reported by a project member who observed a performance deviation in the automated performance test result. The performance test scripts running on "buildbot" compare the performance metric against a performance expectation baseline. The performance expectations are written in a file ("/trunk/src/tools/perf_expectations/perf_expectations.json") that has the expected 'improve' and 'regress' values. By comparing these values to the actual value, the performance regression testing script identifies both "performance regression" and "unexpected speedup" (bug # 18597, comment # 11). We found many patches related to performance bugs where this expectation baseline file was patched to readjust after a definitive performance speedup or an accepted regression. For accepted regressions, the corresponding bug is no longer a bug, but expected behavior. Moreover, some patches are only intended for troubleshooting. For example, in bug # 34926 comment # 9, a patch was uploaded to determine if another revision caused the performance deviation.

Although the sub-dimensions "reporter has some hint about the fix" and "patch uploaded by" were interesting, we could not find any significant difference between performance and non-performance bugs for them.

D. Bug Fix Validation

After a software fix has been proposed, it needs to be validated by reviewers to ensure software quality. Basically, reviewers comment on the approach, algorithm used, and code quality in order to ensure that the patch really fixes the bug and the fix does not introduce any new bug. Sometimes, depending on the importance of the bug and complexity of the fix, a more rigorous validation effort needs to be spent.

Sub-dimensions:

Discussion about the patch: After a patch is uploaded and linked to the issue tracking system, people can discuss the patch in order to validate its effect. In the Firefox Bugzilla repository, patch discussion appears in the bug comments of the bug report. In Chrome, this discussion can be found on the Chromium code review page for that patch (a separate page whose link is posted as a bug comment).

Some discussions focus solely on the code changes in the patch. We call these technical discussions. Other discussions are not source code specific, i.e., they discuss about whether the problem is still observed by others, and what changes people can see after the patch is used. We call these non-technical discussions.

Review and super-review: In both projects, the code was reviewed by one or more code reviewers. Moreover, we found the use of the term "super-review" in Mozilla. A super-review is performed by a super-reviewer (a list of reviewers indexed by area) and is required for certain types of critical changes mentioned in Mozilla's super-review policy [18].

Findings:

For none of the projects and sub-dimensions, performance bug reports behave statistically significantly different from non-performance bugs.

Table IV
COMPARISON OF FINDINGS ABOUT PERFORMANCE BUGS BETWEEN OUR PREVIOUS QUANTITATIVE STUDY [10] AND THIS STUDY.

Previous Quantitative Study	This Qualitative study
Require more time to Fix	1. WorksForMe (WFM) after a long time, 2. Problem in reproducing 3. More dependencies between bugs 4. Collaborative root cause analysis process
Fixed by more experienced developers	1. More release blocking 2. People switch to other systems
More lines of code changed	N/A

IV. DISCUSSION

In our previous work, we found that performance bugs take more time to fix, are fixed by more experienced developers, and that more lines of code are changed to fix performance bugs. Based on the qualitative findings in the current paper (summarized in Table IV), we are now able to, at the minimum, provide more context about our previous findings and, likely, provide a strong indication about the actual rationale behind those quantitative findings.

Related to the finding that Firefox performance bugs take more time to fix, we found for Firefox that people face problems in reproducing performance bugs and have to discuss more to find the root cause of these bugs. These collaborative activities may require more time in a project of global scale like Firefox. Moreover, we also found that performance bugs have more dependencies on other bugs, which implies that performance bug fixing may be delayed because of the time to fix the dependent bug first. On the other hand, we also found indications that our measurements of the time to fix a bug might have been inflated. Indeed, 19% of the performance bugs in Firefox are fixed after a long time (in the order of months, sometimes years) without any

trace of related fixing process. They were basically fixed as a side-effect of some other bug. Since the corresponding bug report has been open (and probably forgotten) for so long, the unnatural fix time of these bugs may cause the average fixing time to be higher than it should be.

Related to the finding that performance bugs are fixed by more experienced developers in Firefox, we found in this study that a higher percentage of performance bugs is release blocking than non-performance bugs in Firefox. Release blocking bugs are known to be of the highest priority and this may prompt projects to assign experienced developers to fix performance bugs. Furthermore, the difficulty to reproduce performance bugs and fix them also suggests that experts are better equipped to fix performance bugs.

Finally, we did not find direct qualitative support for our finding that performance bug fixes are larger than non-performance fixes. However, our findings that performance bugs have many dependencies and require experienced developers to fix them could give some indications that performance bugs need more system-wide knowledge instead of knowledge about one component. More analysis is needed to further explore this possibility.

Our findings for Firefox and Chrome performance bugs were different in most of the cases. Possible explanations are the differences in development and quality assurance processes and policies of these two projects. Studies on other projects should be done to obtain a more complete picture of performance bugs.

V. Threats To Validity

Since this is a qualitative study, there may be some human factors and subjectivity in our bug analysis, since it is very difficult to prevent or detect researcher-induced bias. Although only the first author performed the qualitative analysis, he took care to consider all available bug data sources and to double-check findings when in doubt.

There may be more dimensions and/or more sub-dimensions in the selected dimensions other than the ones we selected, since our selection is based on the sampled data in Firefox and Chrome. We iteratively identified tags until a stable set of tags was found that could cover all sub-dimensions of sampled reports. Conceptually, the (sub-)domains cover most of the qualitative aspects of bug reports, and preliminary analysis on other projects supports the claim.

Our qualitative study focuses on two projects. It is not clear whether our findings generalize to other open source projects or to commercial projects. However, we have selected the two most popular open source projects (Mozilla Firefox and Chrome) from the browser domain, since performance bugs matter in this domain and we had access to all relevant data.

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. A statistical sampling technique (with 95% confidence level and confidence interval of 10) showed that our heuristics have a high precision and recall.

VI. Related Work

A. Bug Repositories and Qualitative Analysis

The use of issue tracking systems as a communication and collaboration medium among customers, project managers, quality assurance personnel, and programmers has been qualitatively studied by Bertram et al. [19]. From questionnaire and semi-structured interviews, they found issue trackers to be an ever-growing knowledge store where each stakeholder contributes knowledge and information via individual bugs and features. In our study, we tried to exploit this knowledge to learn about software performance bugs and their fixing process.

Bug reports from different open-source projects have also been qualitatively studied by Zibran et al. to identify and rank usability issues related to designing and developing software APIs [20]. Guo et al. qualitatively studied bug reports from the Microsoft Windows Vista operating system project to find out the primary reasons for bug reassignment [21]. They used a qualitative study on bug reports to provide support for their survey findings on Microsoft employees. Andrew et al. did a qualitative study on 100 bug reports from 3 different open-source software projects to find out how teams of distributed developers discuss and reach consensus in bug reports [22].

Bettenburg et al. performed a survey on developers and reporters related to software issue repositories in order to find out the qualities of a good bug report [15] [23]. In their survey, the developers and reporters identified and ranked different types of information provided in bug reports, i.e., steps to reproduce, test cases, and stack traces. Our study also found active use of this kind of information in bug reports to fix performance bugs.

B. Performance Bugs

There also has been research on detecting performance bugs. Jovian et al. worked on finding performance bugs by automatically monitoring the deployed application behavior in order to provide helpful information for performance problem tracing to the developer [24]. Narpurkar et al. worked on efficient remote profiling of mobile devices that can help users to dynamically provide information to developers to help isolate bugs, maintain and improve performance. These two papers focus on improving the performance bug detection and tracing process, while we study the bug reporting, tracing and fixing process followed in practice.

In our previous study [10], we quantitatively studied performance bugs and compared results with security and

other bugs of Firefox. Different metrics were considered in three dimensions, i.e., Time, People and Fix. In these dimensions, we found that performance bugs take more time to fix, are fixed by more experienced developers and require changes to more lines than non-performance bugs. These quantitative findings, although interesting, resulted in more questions than we started out with. Hence, in the current paper, we tried to understand what happens during the bug reporting, tracking and fixing to find an explanation of our quantitative results.

VII. Conclusion

Mozilla Firefox and Google Chrome are the two most popular open source web browser projects with millions of users and hundreds of developers and contributors around the world. We studied sampled performance bugs in these two projects to observe how project members collaborate to detect and fix performance bugs.

Performance bugs have different characteristics, in particular regarding the impact on users and developers, the context provided about them and the bug fix process. Our findings suggest that in order to improve the process of identifying, tracking and fixing performance bugs in these projects:

- Techniques should be developed to improve the quality of the "steps to reproduce", both in the performance bug reports as well as in the system as a whole.
- More optimized means to identify the root cause of performance bugs should be developed.
- Collaborative root cause analysis process should be better supported.
- The impact of changes on performance should be analyzed, e.g., by linking automated performance test results to commits, such that performance bugs no longer magically (dis)appear.

We plan on performing case studies on other open source software systems from the same domain and from other domains to be able to generalize our findings.

References

[1] S. Balsamo, A. Di Marco, P. Inverardi, and M. Simeoni, "Model-based performance prediction in software development: a survey," *Software Engineering, IEEE Transactions on*, vol. 30, no. 5, pp. 295 – 310, may 2004.

[2] H. H. Liu, *Software Performance and Scalability: a Quantitative Approach*. Wiley & Sons, Inc., 2009.

[3] M. Cataldo, A. Mockus, J. Roberts, and J. Herbsleb, "Software dependencies, work dependencies, and their impact on failures," *Software Engineering, IEEE Transactions on*, vol. 35, no. 6, pp. 864 –878, nov.-dec. 2009.

[4] T. Graves, A. Karr, J. Marron, and H. Siy, "Predicting fault incidence using software change history," *Software Engineering, IEEE Transactions on*, vol. 26, no. 7, pp. 653 –661, jul 2000.

[5] R. Moser, W. Pedrycz, and G. Succi, "A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction," in *Proc. of the 30th international conference on Software engineering*, ser. ICSE '08, 2008, pp. 181–190.

[6] Y.-W. Huang, F. Yu, C. Hang, C.-H. Tsai, D.-T. Lee, and S.-Y. Kuo, "Securing web application code by static analysis and runtime protection," in *Proc. of the 13th international conference on World Wide Web*, ser. WWW '04, 2004, pp. 40–52.

[7] P. Anbalagan and M. Vouk, "An empirical study of security problem reports in Linux distributions," in *Proc. of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement*, ser. ESEM '09, 2009, pp. 481–484.

[8] "The whiteboard: Tracking usability issues: to bug or not to bug?" *interactions*, vol. 8, pp. 15–19, May 2001.

[9] F. Heller, L. Lichtschlag, M. Wittenhagen, T. Karrer, and J. Borchers, "Me hates this: exploring different levels of user feedback for (usability) bug reporting," in *Proc. of the 2011 annual conference extended abstracts on Human factors in computing systems*, ser. CHI EA '11, 2011, pp. 1357–1362.

[10] S. Zaman, B. Adams, and A. E. Hassan, "Security versus performance bugs: a case study on Firefox," in *Proc. of the 8th Working Conference on Mining Software Repositories*, ser. MSR '11, 2011, pp. 93–102.

[11] A. Schröter, "MSR challenge 2011: Eclipse, Netbeans, Firefox, and Chrome," in *Proc. of the 8th Working Conference on Mining Software Repositories*, ser. MSR '11, 2011, pp. 227–229.

[12] G. Kalton, *Introduction to Survey Sampling*. Sage Publications, Inc, September 1983.

[13] S. England, "Briefing Note: confidence intervals and statistical significance within the Active People Survey," unpublished.

[14] D. Nir, S. Tyszberowicz, and A. Yehudai, "Locating regression bugs," in *Hardware and Software: Verification and Testing*, ser. Lecture Notes in Computer Science, 2008, vol. 4899, pp. 218–234.

[15] N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, and T. Zimmermann, "What makes a good bug report?" in *Proc. of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, ser. SIGSOFT '08/FSE-16, 2008, pp. 308–318.

[16] C. R. Reis, "An overview of the software engineering process and tools in the Mozilla project," in *Open Source Software Development*, 2002, pp. 155–175.

[17] S. Kim and E. J. Whitehead, Jr., "How long did it take to fix bugs?" in *Proc. of the 2006 international workshop on Mining software repositories*, ser. MSR '06, 2006, pp. 173–174.

[18] "Mozilla super-review policy," http://www.mozilla.org/hacking/reviewers.html, February 2012.

[19] D. Bertram, A. Voida, S. Greenberg, and R. Walker, "Communication, collaboration, and bugs: the social nature of issue tracking in small, collocated teams," in *Proc. of the 2010 ACM conference on Computer supported cooperative work*, ser. CSCW '10, 2010, pp. 291–300.

[20] M. Zibran, F. Eishita, and C. Roy, "Useful, but usable? factors affecting the usability of APIs," in *Reverse Engineering (WCRE), 2011 18th Working Conference on*, oct. 2011, pp. 151 –155.

[21] P. J. Guo, T. Zimmermann, N. Nagappan, and B. Murphy, ""Not my bug!" and other reasons for software bug report reassignments," in *Proc. of the ACM 2011 conference on Computer supported cooperative work*, ser. CSCW '11, 2011, pp. 395–404.

[22] A. J. Ko and P. K. Chilana, "Design, discussion, and dissent in open bug reports," in *Proc. of the 2011 iConference*, ser. iConference '11, 2011, pp. 106–113.

[23] N. Bettenburg, S. Just, A. Schröter, C. Weiß, R. Premraj, and T. Zimmermann, "Quality of bug reports in Eclipse," in *Proc. of the 2007 OOPSLA workshop on eclipse technology eXchange*, ser. eclipse '07, 2007, pp. 21–25.

[24] M. Jovic, A. Adamoli, and M. Hauswirth, "Catch me if you can: performance bug detection in the wild," in *Proc. of the 2011 ACM international conference on Object oriented programming systems languages and applications*, ser. OOPSLA '11, 2011, pp. 155–170.