A Quantitative and Qualitative Investigation of Performance-Related Commits in Android Apps

Teerath Das*, Massimiliano Di Penta†, Ivano Malavolta‡
*Gran Sasso Science Institute, L'Aquila, Italy - teerath.das@gssi.infn.it
†University of Sannio, Benevento, Italy - dipenta@unisannio.it
‡Vrije Universiteit Amsterdam, The Netherlands - i.malavolta@vu.nl

***Abstract*—Performance is nowadays becoming a crucial issue for mobile apps, as they are often implementing computational-intensive features, are being used for mission-critical tasks, and, last but not least, a pleasant user experience often is a key factor to determine the success of an app. This paper reports a study aimed at preliminarily investigating to what extent developers take care of performance issues in their commits, and explicitly document that. The study has been conducted on commits of 2,443 open source Android apps, of which 180 turned out to contain a total of 457 documented performance problems. We classified performance-related commits using a card sorting approach, and found that the most predominant kinds of performance-related changes include GUI-related changes, fixing code smells, network-related code, and memory management.**

***Index Terms*—Android, Mobile Performance issues, App Store Mining.**

I. Introduction

Mobile applications are nowadays gaining a huge popularity and importance. From a purely economical standpoint, their market is incredibly increasing, and it has been estimated it will reach about $70 billion in annual revenue by 2017 [3]. Besides that, it is possible to observe two phenomenon. For some operating systems—such as Android—the number of available devices is increasing, each one having its specific characteristics, *e.g.,* in terms of CPU, memory, Graphical Processing Unit (GPU), and screen. In general, the hardware market is releasing devices with better and better performance, nowadays comparable to desktop computers. In such a context, it may not be infrequent that an app undergoes fixes with the aim of dealing with performance problems. These problems may be due to the addition of a new feature in a scenario where developers mainly focus on ensuring an early release. Or else, they could happen for an improper usage of the Android APIs on, in general, because of bad design/implementation choices.

So far, performance issues have been investigated in Web applications [1], heterogeneous environments [5], or large-scale applications [14]. Also, recently Zaman *et al.* have conducted a qualitative study of performance bugs [20]. To the best of our knowledge, there is only work on the analysis of mobile app performance bugs by Liu *et al.* [12], limited to the analysis of 70 real bugs.

This paper reports a study aimed at conducting a quantitative and qualitative characterization of performance-related commits for Android apps. First—and similarly to what previously done in a work on energy-related commits [15]—we identify, using regular expressions, commits explicitly referring to performance-related issues. In other words, instead of using static source code analysis—as done by Liu *et al.* [12], and instead of using dynamic analysis—which would require appropriate execution profiles, and not practicable on large-scale—we rely on documented performance related changes, as previously done by Ray *et al.* [17] for the analysis of bug categories on GitHub.

We report the distribution of such commits, also analyzing how do they vary across app categories. Then, using the card sorting approach, we produce a taxonomy of performance related concerns, and qualitatively describe some examples of commits belonging to these concerns. With respect to work such as the one of Liu *et al.* [12], our study has been conducted in the large, featuring the analysis of 68,025 commits from 2,443 open source Android apps. Of such commits, 457 (0.67%) of them, belonging to 180 apps, turned out to be documented, performance-related commits. They generally affected any kinds of apps, although categories in which user experience was very important—*e.g.,* health and fitness, or photography—were slightly more affected than others. The card sorting categorization revealed how the most frequent kinds of performance-related commits were about GUI-related changes, removing (performance-affecting) code smells, and dealing with network or memory-related problems.

In summary, the main **contributions** of this paper can be summarized as: (i) results of a study investigating performance-related commits in 180 open source Android apps; (ii) a taxonomy of the main kinds of performance-related problems, obtained by applying card sorting [19]; and (iii) the study replication package, featuring a dataset of categorized performance-related commits[1].

II. Study Design

The *goal* of this study is to investigate performance-related commits in Android apps, with the *purpose* of understanding their nature and their relationship with projects' characteristics, such as project domain or size. The study *context* consists of 2,443 open-source apps and their evolution history. The study aims at addressing the following research questions:

RQ_1: To what *extent* developers consider performance issues of Android apps?

[1]http://cs.gssi.infn.it/ICSME_2016

RQ_2: What are the *concerns* that developers have when dealing with performance issues of Android apps?

More specifically, RQ_1 aims at assessing the frequency in which app developers consider performance issues of the app, whereas RQ_2 aims at classifying the specific concerns that developers have when considering the performance issues of the app being developed (*e.g.,* fast access to file system, reactivity of the user interface, etc.)

The **context** of our study consists of a set of open-source Android apps distributed in the Google Play store. We decided to analyze mobile apps in the Google Play Store because of its large market share in terms of both distributed apps and sold smartphones with respect to other platforms such as Apple iOS, Windows Phone, BlackBerry [9], [2]. Since we are targeting mobile apps that have been designed and developed as real projects with real users and we also aim at accessing the performance concerns managed by their developers, the **objects** of our study are Android apps that (i) are freely distributed in the Google Play Store; and (ii) have their versioning history hosted on GitHub.

Fig. 1 presents the process we followed for identifying our target population, together with the number of apps considered at each step. Basically, (i) we mined the well-known FDroid open source apps repository for extracting all those Android apps in which the description page contains both a link to a GitHub repository and a link to a Google Play page; (ii) we performed a custom search on GitHub by targeting all the repositories in which the $\mathrm{readme.md}$ file contains a link to a Google Play page, and (iii) we collected all the apps enlisted in the community-maintained list of free and open-source Android apps on Wikipedia[2]. After a merging and duplicates removal activities we obtained 4,287 mobile apps. At this point we filtered out (i) all those apps whose GitHub repository does not contain an Android manifest file as they clearly do not refer to real Android apps, (ii) all those apps for which the corresponding Google Play page is not existing anymore (*i.e.,* they have been removed from the store for some reason), and (iii) all those apps in which the Android file is not in the root directory of an Android app (those cases happen when the manifest file actually refers to an Android library, to the binaries of some other app, etc.). The final population resulting from this process is a set of 2,443 mobile apps, each of them represented by its GitHub and Google Play identifiers.

The variables considered to address RQ_1 are the (i) *pCommits*, the number of performance-related commits in the GitHub repository of the app, as compared to the overall number of *commits*, and (ii) the app *category* on Google Play. As for RQ_2, we considered the different kinds of performance *concerns* being dealt in performance-related commits.

We extracted the *commits* and *pCommits* using a script that considers only the folder containing the source code and resources of the mobile app, excluding back end, documentation, test or mockups. In each repository we identified the folder

[2]http://wikipedia.org/wiki/List_of_free_and_open-source_Android_applications

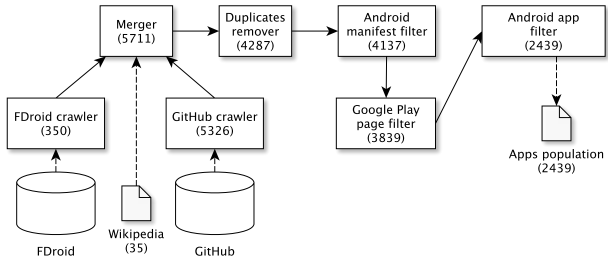
Merger (5711)
Duplicates remover (4287)
Android manifest filter (4137)
Android app filter (2439)
FDroid crawler (350)
GitHub crawler (5326)
Google Play page filter (3839)
Apps population (2439)
FDroid
Wikipedia (35)
GitHub

Fig. 1. Apps identification process.

of the app by the presence of an Android manifest file. The mining script identifies a commit as performance-related if it matches at least one of the following keywords: wait, slow, fast, lag, tim, minor, stuck, $\mathrm{instant}$, $\mathrm{respons}$, react, speed, latenc, $\mathrm{perform}$, $\mathrm{throughput}$, hang, memory, leak. Those keywords have been identified by considering, analyzing, and combining mining strategies in previous empirical studies on software performance in the literature (both mobile and not mobile-specific) [18], [20], [8], [13]. The mining script considers all the possible combinations of both lower and upper cases of each keyword. By applying this pattern matching, we identified a set of 535 candidate performance-related commits. Such commits were manually analyzed, and 78 of them were identified as false positives. This produced a final set of 457 performance-related commits.

We extracted the *category* variable by mining the web page of the Google Play store of each app. Then, we identified the *concerns* by applying the open card sorting technique [19] to categorize performance-related commits into relevant groups; we performed the card sorting in two phases: in the first phase we tag each commit with its representative keywords (*e.g., read from file system*, *swipe lag*) and in the second phase we group commits into meaningful groups with a descriptive title (*e.g., UI issues*, *file system issues*). To minimize bias, this activity has been performed by two researchers and the results have been checked by a third researcher; this activity resulted in a set of 10 categories (see Section III).

III. Results

In this section we answer each research question by presenting the results of the study described in Section II

A. RQ_1 - To what extent developers consider performance issues of Android apps?

To answer this question, we count the frequency of performance-related commits (see the $pCommits$ variable) with respect to all the commits of our dataset. Firstly, we can observe that 7.5% of the apps in our dataset have at least one performance-related commit (180 distinct apps over a total of 2,443). Among them, we identified 457 performance-related commits (0.67%) over a total of 68,025 commits.

Table I shows the distribution apps, commits, and performance-related commits across categories. Performance-related commits are more frequent in very different categories in terms of application context [4]; for example, the category

TABLE I
DISTRIBUTION OF ALL COMMITS AND PERFORMANCE-RELATED COMMITS ACROSS CATEGORIES

Category	#Apps	#Commits	#pCommits
Comics	4	216	5 (2.31%)
Customization	85	1,140	18 (1.58%)
Weather	21	1,016	15 (1.48%)
Health and Fitness	69	1,158	14 (1.21%)
Photography	36	5,533	67 (1.21%)
Tools	573	11,426	97 (0.85%)
News & Magazines	43	4,361	36 (0.83%)
Communication	91	2,632	21 (0.80%)
Productivity	216	3,952	28 (0.71%)
Games	350	5,703	39 (0.68%)
Shopping	13	457	3 (0.66%)
Libraries & Demo	70	640	4 (0.63%)
Travel & Local	71	2,888	16 (0.55%)
Media & Video	49	1,343	7 (0.52%)
Music and audio	62	1,226	6 (0.49%)
Social	69	7,843	37 (0.47%)
Medicine	8	704	3 (0.43%)
Finance	57	1,117	4 (0.36%)
Business	35	1,184	4 (0.34%)
Education	200	5,477	18 (0.33%)
Entertainment	128	3,089	8 (0.26%)
Transportation	70	1,916	4 (0.21%)
Books & Reference	41	1,638	2 (0.12%)
Lifestyle	82	1,366	1 (0.07%)

TABLE II
APPS WITH THE HIGHEST NUMBER OF PERFORMANCE-RELATED COMMITS

Google Play ID (GitHub)	Category	#pCommits
com.almalence.opencam (almalence/OpenCamera)	Photography	31 (8.7%)
com.gopro.smarty (M66B/XPrivacy)	Photography	24 (2.5%)
com.newsblur (samuelclay/News-Blur)	News	18 (13.56%)
ca.cumulonimbus.barometernetwork (Cbsoftware/pressureNET)	Weather	14 (7.27%)
org.wordpress.android (wordpress-mobile/WordPress-Android)	Social	13 (4.75%)
net.usikkert.kouchat.android (blurpy/kouchat-android)	Communication	11 (4.72%)
com.pacoapp.paco (google/paco)	Health	10 (13.20%)
com.eleybourn.bookcatalogue (eleybourn/Book-Catalogue)	Productivity	10 (1.9%)
org.qii.weiciyuan (makings/mst)	Social	9 (4.57%)
org.quantumbadger.redreader (bamptonm/RedReader)	News	9 (8.87%)

with the highest percentage of performance-related commits is *Comics*, that primarily contains apps with an immersive user experience and long usage sessions, followed by the *Customization* and *Weather* categories, that primarily contain utility, task-based apps with very short usage sessions, etc. This observation may be an indication that performance issues are somehow orthogonal across apps, independently of their specific application context and user experience requirements.

Nevertheless, it is interesting to note that in our dataset there are some apps with a relatively high number of performance-related commits, the top-10 being listed in Table II. Note that we considered the absolute number of performance-related commits instead of percentages (also reported), as we want to focus on the absolute number of performance-related tasks. As one can notice, these apps are not tied to some very specific categories. We manually analyzed the commit messages of the app with the highest number of performance-related commits (*i.e.,* com.almalence.opencam), it has 17 commits concerning memory consumption, 9 commits about user interface responsiveness, 4 commits about images optimization, and other mixed types of commits; being it a photography app, the nature of those commits is aligned with the features provided by the app. We also manually analyzed all the other top-10 apps and we found that the types of their performance-related commits vary without exhibiting any specific pattern.

B. RQ_2 - *What are the concerns that developers have when dealing with performance issues of Android apps?*

Table III shows the 10 categories resulting from the card sorting, together with an example of representative commit message for each category (we randomly took it from our dataset), the frequency and percentage of commits belonging to each category. It is important to note that the sum of all frequencies (586) is higher than the total number of identified commits because each commit message can belong to more than one category. More specifically, there are 422 commits with 1 category, 62 commits with 2 categories, and 2 commits with 3 categories. In general, *Android developers primarily focus on addressing one performance-related concern at a time (422 times over 486), rather than addressing more than one in combination (64 times over 486).* In the following we discuss each identified category of concerns. All together, those categories give an indication about what are the main concerns that developers perceive, consider, and address when dealing with mobile apps performance.

The most frequent concern that developers have when dealing with performance issues of Android apps is about the responsiveness of the *user interface*, *e.g.,* in terms of swipe lags, screen layout drawing, lists scrolling responsiveness (27.35% of commits). This is an indication of the fact that developers know that end users perception of app performance is of paramount importance and that end users just expect mobile apps to properly work (*e.g.,* without delays, with few bugs, with a natural user experience), independently of the implemented technical solutions, used tools or libraries [4]. Examples of UI concerns include: use of Android's recycler views instead of plain list views[3], render images in slices, prefer Android's asynchronous tasks.

Another recurrent target of app developers is to fix existing performance-related *code smells* (22.53% of commits), *i.e.,* symptoms of poor design and implementation choices, mainly due to time constraints of the project [6]. Examples of fixed code smells include inefficient usage of regular expressions, recurrent computations of constant data, usage of deprecated decryption algorithms.

In a relatively large number of commits (21.66%) developers mention only *generic concerns* about app performance, without detailing what the problem is and how it has been potentially fixed. This is a clear behavioral anti-pattern because generic commit messages make very difficult to know the

[3]https://developer.android.com/training/material/lists-cards.html

TABLE III
CATEGORIES OF IDENTIFIED CONCERNS.

Category	Representative commit message	#pCommits
User interface	▲ *FIX layouts for better rendering performance*	125 (27.35%)
Code smells	▲ *fixed String concatenation performance issue*	103 (22.53%)
Generic concerns	▲ *Performance and error handling improvements*	99 (21.66%)
Networking	▲ *Use a socket connection when scanning for hosts instead of isReachable. Set performance options to prefer fast connection. Enable TCP_NODELAY*	59 (12.91%)
Memory	▲ *Fixed major memory leak; should improve responsiveness on older devices*	50 (10.94%)
Loading time	▲ *Made initial load WAY faster*	34 (7.43%)
Images	▲ *Draw all static objects on one image in android to optimize performance*	20 (4.37%)
Local database	▲ *Added indexes for posts.postid and posts.blogID to improve performance of several lookups*	12 (2.62%)
File system	▲ *Separate file loading and vault initialization to enhance performance*	10 (2.18%)
Sensors	▲ *Performance fix: Closing GPS service as soon as lat/-long has been determined.*	5 (1.09%)

nature of a performed change (*e.g.,* it may fix a bug, implement a new feature, improve code quality, etc.), its effects, to find when a bug has been introduced, etc.

Networking is one of the area in which performance-related commits occur a lot in mobile apps. For example, making HTTP requests is one of the most energy consuming operations in Android [10]. Our analysis shows that app developers take special care of networking operations (12.91% of commits) in their apps and refine their code in order to mitigate the impact of networking operations on the overall performance of the app. Examples of implemented solutions include: reduce as much as possible the frequency of calls when doing long polling, avoid making multiple requests in parallel, check the status of the connection before making a request to a server.

Keeping small the *memory* footprint of a mobile app is one of the key solutions for improving its performance[4], specially for low-end devices. Developers are aware of this interaction between memory usage and performance (10.94% of commits) and apply solutions like stopping auxiliary services when available memory gets low, avoiding to load potentially unused data, etc.

[4]https://developer.android.com/training/best-performance.html

Loading time of app screens is of paramount importance for its success, specially when considering the startup screen of the app. Developers are focusing on this key aspect of their apps (7.43% of commits) and apply solutions like reducing the information to be parsed when starting an activity, caching methods requiring a restart, etc.

Concerns with a frequency lower than 5% are: *images* management (4.37%), interaction with *local databases* (2.62%), *file system* access (2.18%), and interaction with device *sensors* (1.09%). Those concerns have been less targeted by app developers; nevertheless, they are representative examples of potentially relevant aspects to verify when considering the performance of an Android app. Examples of implemented solutions include:

- Images: load images directly in the required size, render images one at a time;
- Local databases: perform queries in an asynchronous task, add indexes to specific fields;
- File system: separate file loading from other activities, use buffered streams for file decryption;
- Sensors: filter sensor data, limit the use of the GPS service.

IV. THREATS TO VALIDITY

Threats to *construct validity* are mainly related to the use of keyword matching to identify performance-related commits. Under this perspective, we are assuming that if a commit message contains specific keywords is describing a change in the source code of the app related to its performance. Therefore, we are aware that such approach may miss undocumented performance-related commits. False positives have been instead avoided by performing a manual analysis of the identified commits.

Reliability validity threats concern the possibility of replicating this study. We mitigated this possible threat by making the replication package with all extracted data, mining, and analysis scripts available to interested researchers.

Threats to *external validity* mainly concern the generalization of our results that relate to the representativeness of the apps considered in this work. We reduced this threat by considering a relatively large data set (i.e., 2,443 apps) and by selecting apps that have been developed in the context of real projects (i.e., all selected apps have been distributed in the Google Play store and available to the public). Another potential threat to external validity is that fact that we consider only freely available apps; this is an acceptable bias because free apps represent more than 75% of all Google Play store apps and they are downloaded more often [7]. Also, we analyzed only commit messages written in the English language; this potential bias can be considered as acceptable as English is the dominant language used by Android developers.

V. RELATED WORK

Liu *et al.* [12] have conducted an empirical study comprising of 8 popular and large Android applications with the aim of finding performance-related bugs. They manually identified 70 performance bugs, which they further characterized into three different categories. According to their research, GUI lagging is the most common category with 75.7% of bugs followed by energy leak with 14.3% and memory bloat with 11.4%. They also implemented a static code analyzer to detect two types of performance bugs anti-patterns. In our work, we are doing a study of 2,443 open source repositories, mainly focusing on documented performance related commits. Our core aim is to find out key issues which cause the degradation of performance in android applications. Also, we performed a fine-grained classification of such commits through card sorting, which resulted in a set of 10 different categories.

Moura *et al.* [15] followed an approach similar to ours, however focusing on energy aware commits instead of performance-related commits. There is also some work done on identifying performance bugs from developer's perspective. Linares-Vsquez *et al.* [11] surveyed 485 GitHub developers to investigate on the performance issues encountered by developers and their best practices to deal with such issues. The use of multi-threading mechanisms resulted to be a widely adopted solution to deal with performance bottlenecks in mobile apps.

Other studies focused on detection of performance bugs through automated tools [15][16]. In our study, we relied, instead, on documented (by means of commit messages) performance-related changes.

VI. Outlook and Future Work

This paper reported results of a study aimed at identifying documented performance-related commits in Android apps. By analyzing commits of 2,443 apps, we discovered a total of 457 performance-related commits spread across 180 apps. We performed a qualitative analysis of such commits using card sorting, and identified a total of 10 commit categories. Overall, performance commits mainly related to issues found in the app user interfaces. Other than that, we also found frequent commits aimed at removing some code bad smells or at improving some part of the app logic and, finally, dealing with lags in the networking connection. Last, but not least, we found improvements related to I/O towards file system or databases, and related to the access to sensors.

As a preliminary result, the concern categories we identified can be used as checklist by developers in order to see if they are considering all major performance-related aspects of their mobile app.

Future work aims at performing a deeper analysis on the commits' source code to (i) study performance regression problems, (ii) investigate what kinds of performance smells typically occur in mobile apps and how they are identified and solved by developers, and (iii) to investigate on the measure of effect of performance improving changes in the source code of mobile apps. Finally, further research is needed to find out if the results apply to other app markets and mobile platforms.

References

[1] T. M. Ahmed, C. Bezemer, T. Chen, A. E. Hassan, and W. Shang. Studying the effectiveness of application performance management (APM) tools for detecting performance regressions for web applications: an experience report. In *Proceedings of the 13th International Conference on Mining Software Repositories, MSR 2016, Austin, TX, USA, May 14-22, 2016*, pages 1–12, 2016.

[2] Dan Crawley. VentureBeat report, 2014. http://venturebeat.com/2014/10/15/google-play-downloads-60-percent.

[3] Digi-Captial. Mobile internet report q1 2015. http://www.digi-capital.com/reports.

[4] B. Fling. *Mobile design and development: Practical concepts and techniques for creating mobile sites and Web apps*. O'Reilly Media, Inc., 2009.

[5] K. C. Foo, Z. M. Jiang, B. Adams, A. E. Hassan, Y. Zou, and P. Flora. An industrial case study on the automated detection of performance regressions in heterogeneous environments. In *37th IEEE/ACM International Conference on Software Engineering, ICSE 2015, Florence, Italy, May 16-24, 2015, Volume 2*, pages 159–168, 2015.

[6] M. Fowler. *Refactoring: improving the design of existing code*. Pearson Education India, 2009.

[7] I. Gartner. Gartner says free apps will account for nearly 90 percent of total mobile app store downloads in 2012, 2012. http://www.gartner.com/newsroom/id/2153215.

[8] G. Jin, L. Song, X. Shi, J. Scherpelz, and S. Lu. Understanding and detecting real-world performance bugs. *ACM SIGPLAN Notices*, 47(6):77–88, 2012.

[9] A. Lella, A. Lipsman, and B. Martin. The Global Mobile Report: How Multi-Platform Audiences and Engagement Compare in the US, Canada, UK and Beyond, 2015. comsCore white paper.

[10] D. Li, Y. Lyu, J. Gui, and W. G. Halfond. Automated Energy Optimization of HTTP Requests for Mobile Applications. In *Proceedings of the 38th International Conference on Software Engineering (ICSE)*, May 2016.

[11] M. Linares-Vasquez, C. Vendome, Q. Luo, and D. Poshyvanyk. How developers detect and fix performance bottlenecks in android apps. In *ICSME'15: Proceedings of the 31st International Conference on Software Maintenance and Evolution*, 2015.

[12] Y. Liu, C. Xu, and S. Cheung. Characterizing and detecting performance bugs for smartphone applications. In *36th International Conference on Software Engineering, ICSE '14, Hyderabad, India - May 31 - June 07, 2014*, pages 1013–1024, 2014.

[13] I. Malavolta, S. Ruberto, V. Terragni, and T. Soru. End users perception of hybrid mobile apps in the google play store. In *Mobile Services (MS), 2015 IEEE International Conference on*, pages 25–32. Institute of Electrical and Electronics Engineers (IEEE), June 2015.

[14] H. Malik, H. Hemmati, and A. E. Hassan. Automatic detection of performance deviations in the load testing of large scale systems. In *35th International Conference on Software Engineering, ICSE '13, San Francisco, CA, USA, May 18-26, 2013*, pages 1012–1021, 2013.

[15] I. Moura, G. Pinto, F. Ebert, and F. Castor. Mining energy-aware commits. In *12th IEEE/ACM Working Conference on Mining Software Repositories, MSR 2015, Florence, Italy, May 16-17, 2015*, pages 56–67, 2015.

[16] A. Nistor and L. Ravindranath. Suncat: Helping developers understand and predict performance problems in smartphone applications. In *ISSTA 2014: Proceedings of the 2014 International Symposium on Software Testing and Analysis*, pages 282–292, 2014.

[17] B. Ray, D. Posnett, V. Filkov, and P. T. Devanbu. A large scale study of programming languages and code quality in github. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, (FSE-22), Hong Kong, China, November 16 - 22, 2014*, pages 155–165, 2014.

[18] M. Selakovic and M. Pradel. Performance issues and optimizations in javascript: an empirical study. In *Proceedings of the 38th International Conference on Software Engineering*, pages 61–72. ACM, 2016.

[19] D. Spencer. *Card sorting: Designing usable categories*. Rosenfeld Media, 2009.

[20] S. Zaman, B. Adams, and A. E. Hassan. A qualitative study on performance bugs. In *9th IEEE Working Conference of Mining Software Repositories, MSR 2012, June 2-3, 2012, Zurich, Switzerland*, pages 199–208, 2012.