Performance Evaluation of Android Applications: a Case Study

Thiago Soares Fernandes, Érika Cota and Álvaro Freitas Moreira
PPGC - Institute of Informatics - UFRGS
Porto Alegre, Brazil - Po Box 15064 - Zip 91501-970
email: {tsfernandes, erika, alvaro.moreira}@inf.ufrgs.br

***Abstract*—Mobile applications development is guided by the careful attention to a number of non-functional requirements, where a good user experience is the ultimate goal. As part of this effort, the developer needs to efficiently use the physical resources of the mobile device, such as CPU time and battery. However, up until now, there is no structured method for performance evaluation in this domain that provides actionable information to the software developer. We present two case studies regarding the performance evaluation process for mobile applications and its impact on development.**

I. Introduction

Performance of mobile devices and applications has increased aims to satisfy the demands of users [1]. In mobile applications, an excellent "*user experience*" is the ultimate goal for developers as the user can very easily change to another app in the market. A number of factors define the concept of user experience in this domain: functionality, usability, response time, response to errors, among others. Even though the term performance is usually related to execution and response time, it is reasonable to say that, in the mobile domain, the overall performance of an application includes not only the notion of time, but the efficient usage of hardware resources, including memory, battery, and so on.

The evaluation of computer systems regarding the satisfaction of non-functional requirements has been explored in the literature and many analysis techniques have been reported. Such techniques are based on distinct concepts such as benchmarking, workloads and Quality of Service (QoS). These concepts have been established for computation platforms (both embedded and non-embedded ones) and some application domains. However, to the best of our knowledge, a structured and effective assessment method for the evaluation of non-functional requirements of mobile systems (comprising the software and the target platform) has not been established yet.

Knowing the importance of the effects of non-functional requirements on the acceptance or rejection of an application and assuming that such an analysis would be useful for the mobile systems developers, this paper presents two case studies on the performance evaluation in the mobile domain. In the first study, we apply a structured performance evaluation method to a mobile system showing the challenges and solutions for such a task. In the second study we evaluate the hypothesis that mobile application developers can use the results of a performance evaluation to support their design decisions along the development cycle.

This work is partially funded by CNPq and FAPERGS

The paper is organized as follows: Section II discusses related works on the topic of performance evaluation in mobile systems. Section III presents a case study of mobile application developed on Android emulator and describes step-by-step the evaluation process. Section IV presents the second case study to analyze how can code interventions and metrics affect application performance in mobile systems. Section V concludes the paper.

II. Related Work

Performance evaluation of a system (software or hardware) aims at obtaining data related to the performance of the system in a desired environment or execution scenario [2].

Current performance evaluation in mobile systems can be classified into two groups: i) analysis of the resource consumption and its influence on the performance of applications and on the mobile operating system [3]–[6]; ii) performance comparison of mobile code produced using distinct programming languages (and respective features) [7]–[9]. These approaches focus on the mobile device and its components (operating system and storage) alone. With this specific goal in mind, the evaluation is performed without a systematic procedure for the definition of workloads and without deeper analysis (including statistical ones) that could lead to more generic conclusions [2].

A benchmark suite is a collection of real and synthetic applications used to simulate the use of one or more resources intensively. Using the data gathered from these exercises it is possible to generate indexes for comparing different systems (benchmarking [2]). Android's Play Store presents more than 70 applications classified as benchmark. These applications provide different coverage factors that are exercised in a device for later comparison. In general the available applications focus on benchmarking metrics related to CPU, GPU, memory, and data storage. The user can download these applications and see how his device behaves with respect to a global ranking of devices.

Academic researches use the idea of benchmarks for a generic comparison among platforms (processors and peripherals) [1], [10], [11], embedded processors alone [12], [13], or even programming languages resources used in application development [14], [15]. Most proposed suites consider a few aspects of the whole mobile development stack and deal with the processor evaluation only, thus leaving several aspects still uncovered.

For mobile systems, the term QoS is mostly associated to the quality of the telecommunication service in terms of performance, but also reliability and usability [16]–[19].

Hence, the term is more related to the network infrastructure that is used by mobile applications and devices rather than to the mobile application and/or device themselves.

III. Case study I

In this section, we present a structured evaluation of non-functional requirements for an internet browser developed for Android. This application was chosen for two reasons: 1) it is present in the great majority of mobile devices thereby being a representative case study; 2) it requires a native implementation (as opposed to html-based applications) where platform-specific tuning makes sense. The basic evaluation methodology presented in [2] was adapted to the problem in hand and the Android Emulator was used as target platform. According to that methodology, the following steps must be defined: objective of the evaluation, system under test, component under study, services, metrics, parameters, factors, workload characterization, experiments and analysis of results. Each step is detailed below.

A. Objective

The aim of this study is to evaluate the performance of a mobile application with the goal of using this information to improve the overall performance of the application in a given device. The following aspects should be used for the performance assessment of the application: CPU time, memory footprint, battery usage, communication demand, and the total execution time of the application for displaying web pages. Data related to each aspect must be collected during the execution of the application in at least two target platforms.

B. Methodology

1) System Under Test(SUT) and Component Under Study (CUS): The first concepts that must be defined in the evaluation process are the system under test (SUT) and the component under study (CUS). The SUT in this case is the display, in the mobile device, of web pages hosted by a local server. This, SUT includes the device, the host, the webServer, etc. The CUS is an Android implementation of an internet browser that receives an address in the web, searches the page on the server and displays it on the mobile device screen. The CUS is the focus of the evaluation.

In this case study we used the Tint Browser 1.7.1[1], which is an Android application for web browsing and is available under open source license[2]. The initial structure of the application has about 310 classes organized into 7 major packages. The application still needs an additional external package (TintBrowserAddonFrameworkLibrary) which consists of 19 classes.

2) Services: Services are functionalities available in the CUS. A typical web browser has features such as addition and manipulation of user preferences, manipulation of favorite pages, management of downloads and page views, etc. These features can be included in a specific browser version using small applications called add-ons. However the main feature, or service, of a web browser is to display web pages.

[1]https://play.google.com/store/apps/details?id=org.tint

[2]https://github.com/Anasthase/TintBrowser

3) Metrics: Metrics are criteria used to compare the performance [2]. To evaluate the defined service in the CUS we use metrics that may indicate the overall performance of the application during execution of this functionality. In this evaluation three metrics are used, all of which are low better metrics, i.e. lower values indicate better performance.

1) Response time: the time in seconds (s) that the browser takes from the beginning of its execution to the end of loading the entire web page content.
2) CPU time: the time in seconds that the process of viewing the webpage occupied the CPU resource. Since the CPU may perform other tasks while loading a webpage, this metric isolates the time taken only by the application being evaluated.
3) Memory usage: the volume in megabytes (MB) that the process of displaying the page used in the RAM of the execution environment.

We note that in the original goal (Step III-A), other metrics such as battery usage and communication demand were mentioned. However, the Android Emulator does not support those features. When the actual device is available, the developer can consider including additional metrics.

Data related to the defined metrics are collected through instrumentation of the application code. For CPU time and memory consumption, methods available in the Android API can be used. The CPU time can be collected by calling *android.os.Process.getElapsedCpuTime()* method whereas memory consumption can be collected using the libraries *android.os.Debug.MemoryInfo* and *ActivityManager* class through *activity_manager.getProcessMemoryInfo(process_pid)*. This last method provides the amount of memory released when a process is finalized [20]. For information about the response time, a counter is included in the code. This counter is reset when the application calls the *onCreate()* method and read when the application calls the *onDestroy()* method. These methods are responsible, respectively, for the beginning and ending of the life cycle of an Android activity.

4) Parameters: Parameters are aspects that can affect the performance. The Android emulator allows different configurations for some peripherals such as CPU type, size of the device screen, etc. However, due to characteristics of the emulator itself, it is known that the emulation of an Intel x86 CPU presents better performance than the ARM CPU, when emulation is performed on an INTEL-based host processor. This is due to the compatibility of the emulated CPU with the architecture of the computer where the emulator is running, and may not correctly represent the performance of a CPU type over another. Because of this, the CPU is considered a fixed parameter and will not be changed in the experiments.

Another important parameter for the performance of a web browser is the network that will be used to access the pages. The network structure influences the response time due to the location and configuration of the server hardware that hosts a particular web page. Aiming at controlling these aspects, all pages of the experiments were downloaded and stored on a local Apache server.

5) Factors: Factors are parameters that will be varied in the performance evaluation. During performance assessment, the

Parameters				Factors	
Emulator		Environment			
CPU Type:	Intel x86	Web Server: Apache version 2		RAM size (MS):	384 MB
Screen size:	4.65" Galaxy Nexus				1024 MB
Local Disk in the emulator:	128 MB			Heap size (HS):	32 MB
Version of the browser application:	Tint Browser 1.7.1				64 MB
No card slot				OS version (AV):	Android 4.0.3
					Android 4.2.2

TABLE I. PARAMETERS AND FACTORS DEFINED FOR THE PROPOSED PERFORMANCE EVALUATION

values of some experimental environment factors are changed and then the behavior of the CUS is evaluated using the defined metrics. The factors used in this study are detailed below.

The Android emulator allows the modification of some aspects of the target hardware only. This feature allows us to partially evaluate the impact of hardware on the application performance. Indeed, only CPU and memory-related metrics can be evaluated in this environment. In our experiments, three factors are considered: the size of the device RAM, the heap size of Dalvik virtual machine, and the version of the operating system.

Table I summarizes the parameters and factors defined for the proposed evaluation.

6) Workload characterization: The chosen workload plays a major role during performance evaluation. The workload must be such that meaningful operating scenarios of the CUS are exercised. However, evaluating all possible scenarios (all possible web page contents, for instance) is normally an impractical task. A good workload characterization (listing a smaller number of scenarios that could be used to represent the entire execution environment of CUS) enables the execution and replication of the experiments in a timely manner [2]. There are techniques to classify components of a workload for performance evaluation so that the selection of scenarios does not affect the validity of the analysis.

We used the clustering technique [2], to group web pages with similar resources needs or contents and reduce the number of scenarios to be executed in the experiments. Our clustering process was based on the 100 most visited websites in 2011 according to the Alexa site[3], obtained from Mobile Web Standards 2011-05 Dataset[4]. From the initial list provided by Alexa, a few web pages with inappropriate content were removed. After the filtering process, we built a dataset containing information of CPU time and memory used to view these websites hosted on a local server. Each site was accessed ten times while the metrics defined in SectionIII-B3 were collected. This data set was then analyzed in the R statistical tool [21] using the Mclust library to group the sites according to memory consumption and CPU time. This analysis generated four clusters represented in Fig. 1.

From the defined clusters, four pages (one for each group) are used in the experiments:

Group 1: www.google.com.br
Group 2: www.amazon.com
Group 3: www.uol.com.br
Group 4: www.huffingtonpost.com

7) Statistical relevance: We try to reduce the influence of external factors in the measured data by controlling as much as possible the experimental setup. Still, some variables are beyond control. For instance, one cannot control embedded Android processes such as garbage collector or other internal processes of the operating system. Similarly, intrinsic latency of the web server is out of control. Therefore, to reduce the noise such variables can cause in the measured data, each experiment is repeated 30 times and the average result is analyzed.

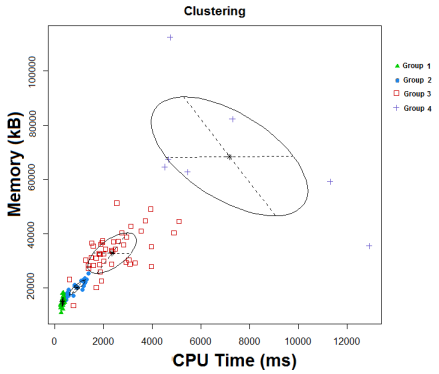


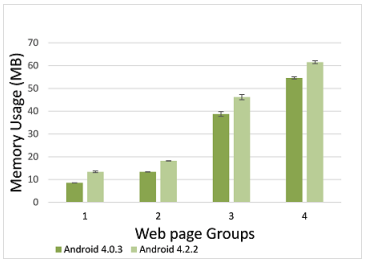
Fig. 1. Clustering representation

C. Experiments

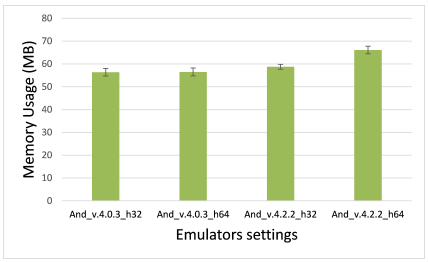
The experiments were performed with the Android emulator version 22.3. The emulator is executed in a computer with 4 GB of RAM, Intel i7 2.3 Ghz with Ubuntu 64bit operating system. This same computer has Apache 2 web page server installed and was used to host the pages selected for the experiments. As mentioned, the webpages were downloaded and stored locally to mitigate the impact of the network fluctuation on measured data. To perform the experiments, we used the Android Debug Bridge (ADB), which enables the creation of a new process in the operating system when for each instantiation of the application.

The defined experimental setup (workload, factors and repetitions) defines a total of 960 executions (4 groups of web pages X 8 emulator's settings X 30 repetitions). We used then one technique that identifies the factors that actually affect the performance to further reduce the number of required experiments. The technique is called Experimental Design, and in this study we used 2^k Factorial Design [2], to identify the most relevant factors that must be exercised. From Table I we have 3 factors (k=3) that became experimental design variables (MS - Memory Size; HS - Heap Size; AV - Android version). Each factor can affect the metrics either individually or combined with another one. Thus, seven combinations of these factors were evaluated. Table II presents the average impact percentage of each factor (response time, memory usage and cpu time) and each combination of factors in evaluated metrics. Eight

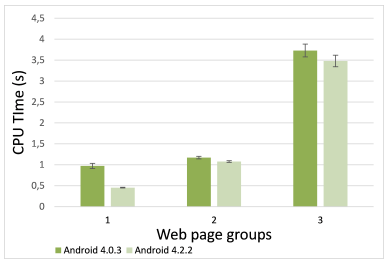
[3]http://www.alexa.com/topsites
[4]http://www.csg.uzh.ch/publications/data/mobilewebstandards.html



(a) Memory usage



(b) Memory usage in Group 4



(c) CPU time

Fig. 2. Experimental Results

	MS	HS	AV	MS-HS	MS-AV	HS-AV	MS-HS-AV
Response time	0,4705	1,445	97,161	0,042	0,479	0,217	0,544
Memory usage	0,215	2,122	84,05	0,500	0,061	10,614	2,435
CPU time	3,737	16,885	59,926	3,982	8,081	0,406	6,981

TABLE II. EFFECT OF FACTORS ON EVALUATED METRICS

	Android version	Heap (MB)
Emulator 1	Android 4.0.3	64
Emulator 2	Android 4.2.2	64
Emulator 3	Android 4.0.3	32
Emulator 4	Android 4.2.2	32

TABLE III. EMULATION ENVIRONMENT SETTINGS

possible configurations of the emulator were considered but with a lower number of repetitions (10) of each experiment.

From Table II one can notice that the operating system is the factor that impacts the most on the performance, followed by the heap size. Although not visible in Table II, the effect of the heap size is specially important for pages on Group 4, not much for the other groups of pages. Thus, four different experimental settings for the Android emulator were used in the experiments, as presented in Table III. These settings represent typical phones using two different versions of the operating system. Through this mechanism the number of experiments reduced from 960 to 360 executions (62% reduction), that is: [4 (groups of web pages) X 2 (emulator's settings) X 30 (repetitions) for page Groups 1-3 plus 1 (Group 4) X 2 (emulator's settings) X 30 (repetitions) to Group 4].

The experiments are automated through the execution of shell scripts and each script is executed separately so that there is no competition for the page server. For each page view, the emulator is started without any previous user data. The Tint application is installed and the command to view the page is executed. At the end of the page loading, the application is closed and the database of resource consumption and runtime is stored in a .csv file. Each group page is accessed 30 times.

D. Results

Figure 2 presents a sample of the results obtained from the experiments. Each graph represents one metric that is being evaluated and the bars indicate the value of the arithmetic mean of 30 replicates of each experiment. The plot also presents the dispersion index of each data set. The semi inter-quartile range is used as dispersion index due to the nature of the data (with the presence of outliers).

From Fig. 2(a) one can observe that the newest version of Android used a greater amount of memory to process and display the four groups of pages. This change is significant because when we take into account the sampling error there is no overlapping bars. Fig. 2(b) shows that the operating system alone does not determine memory usage for all pages. Indeed, using Android version 4.0.3 there is no difference when heap size is changed. On the other hand, this difference appears when a newer Android version is used and one can notice an increase in memory usage directly related to the size the heap.

Results for CPU time (Fig. 2(c)) show two distinct behaviors. For Group 1 pages, there is a considerable reduction in CPU time in Android 4.2.2. However, for pages in intermediate groups, especially in Group 3, the CPU time reduction is not significant when Android version changes. This behavior may be related to the structure of the pages used in the experiments.

The newer Android version leads to a better response time for the application. The reduction in the total time required to completely display the pages is more significant on pages of Groups 1 and 4. In this metric, pages in Group 2 also show a significant reduction in time for version 4.2.2 in respect of the previous operating system. However, the improvement in response time for pages of Group 3 is not significant.

IV. CASE STUDY II

This case study evaluates whether and how performance evaluation can guide the mobile software engineer in tuning the app for a given device. We used the clustering data of the case study and developed another study where the code of the application was modified to reach desirable metric values and the impact of these modifications was related with application performance. This study was performed using a real device, a Samsung Galaxy S3 GT-I9300.

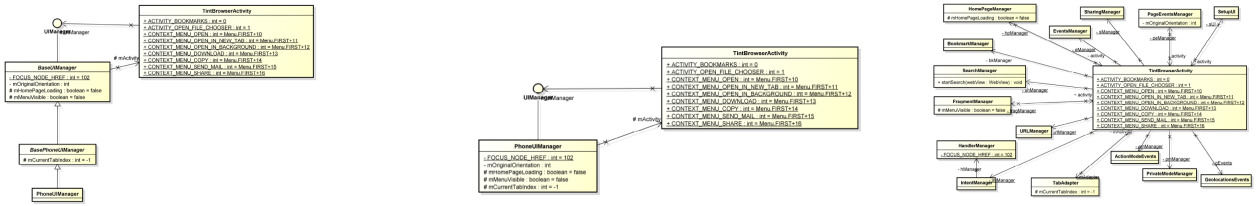
A. Code Metrics

To evaluate the Tint web browser, its code was studied and the main classes used for displaying web pages were identified (Figure 3(a)).

Code metrics were collected from the Tint Browser Project using the tool Understand[5]. In this study we use three metrics that, in the software engineering domain, are standard indicators of internal code quality:

- *CountClassCoupled (CC)*: quantifies the number of other classes coupled to some specified class. The lower the better.
- *MaxCyclomatic (Cyc)*: quantifies the maximum cyclomatic complexity of all nested functions or methods. The lower the better.

[5]http://www.scitools.com/



(a) Original (b) Refactoring 1 (c) Refactoring 2

Fig. 3. Class diagram of Tint Browser

- *PercentLackofCohesion (LC)*: quantifies the lack of cohesion between the features developed for a class. A lower percentage is better.

Then, we implemented two code interventions (called here refactoring) aiming, respectively, at improving and worsen the code quality with respect to the chosen metrics. Only the code related to the main browser service being evaluated, as defined in III-B2 is considered.

B. Code refactoring

Table IV shows the code metrics (M1, M2, and M3 as defined above) for, respectively, the original version (VO), the improved version (R1) and the worsen version (R2) of the Tint Browser Project.

Class	VO			R1			R2		
	CC	Cyc	LC	CC	Cyc	LC	CC	Cyc	LC
BaseUIManager	25	21	94	-	-	-	-	-	-
TintBrowserActivity	26	13	92	26	13	92	-	-	-
PhoneUIManager	18	18	84	37	21	94	-	-	-
BasePhoneUIManager	22	6	86	-	-	-	-	-	-
SetupUI	-	-	-	-	-	-	10	2	82
TabAdapter	-	-	-	-	-	-	27	6	79
FragmentManager	-	-	-	-	-	-	14	5	74
HandlerManager	-	-	-	-	-	-	5	3	62
ActionModeEvents	-	-	-	-	-	-	1	2	50
PageEventsManager	-	-	-	-	-	-	15	6	50
HomePageManager	-	-	-	-	-	-	4	1	40
SharingManager	-	-	-	-	-	-	2	2	33
HttpManager	-	-	-	-	-	-	2	1	0
IntentManager	-	-	-	-	-	-	16	22	0
PrivateModeManager	-	-	-	-	-	-	7	2	0
SearchManager	-	-	-	-	-	-	0	2	0
URLManager	-	-	-	-	-	-	9	3	0
EventsManager	-	-	-	-	-	-	0	1	0
GeolocationsEvents	-	-	-	-	-	-	3	2	0

TABLE IV. CODE METRICS FOR THREE VERSIONS OF THE TINT BROWSER PROJECT

The first refactoring (R1) stands from the common knowledge that an embedded application with a small number of classes performing all functions - low cohesion - have a better performance than the same application developed with a larger number of classes with a better definition of their duties - High cohesion [22], [23]. The corresponding class diagram after this refactoring is shown in Figure 3(b). The code interventions assigned then all functionality of page displaying to a single class. This modification raised the lack of cohesion of this class from 84% to 94%.

The second refactoring (R2) dissolved the single class in a more cohesive set of classes, with 7 fully cohesive classes, 4 classes with up to 50% of cohesion and 4 classes with cohesion still below 84% which was the value of this metric in the original code. The corresponding class diagram after this refactoring is shown in Figure 3(c).

C. Performance Evaluation Results

The same performance evaluation method used in Section III was repeated, but using a real device as target platform. This change allowed the inclusion of the energy consumption as an evaluation metric. All other definitions required in the method remained the same. Metrics in the real device were collected using LittleEye[6], a tool that monitors the execution of an application and provides information on the consumption of CPU, memory, network, and energy used.

Figure 4 shows the energy consumption of each version of the application during the execution of the experiment (30 executions of the application). CPU and memory usage were also collected during the experiments but are not shown here due to space restrictions. The figure depicts the data collected through a 1-minute execution, which represents 5 consecutive executions of the same version of the application and under the same conditions.

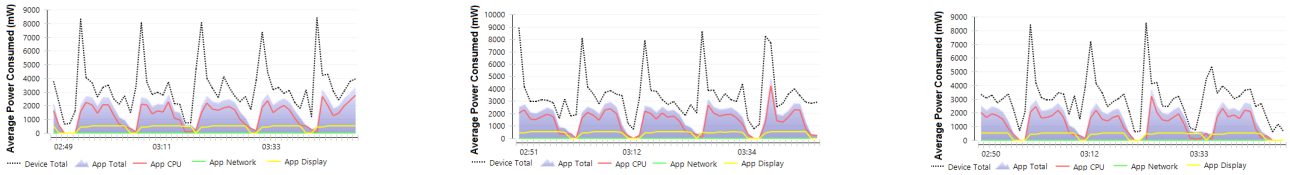
Surprisingly, one can observe a large variation between two consecutive executions of the application. As a consequence, despite the impact in the code metrics, one cannot infer the impact of code refactoring on the performance of the application in the target device.

We implemented two simple applications to check whether there was something overshadowing the changes arising from interventions in the code, such as the use of Android libraries. The first one is a Quicksort implementation using no specific Android library. The second one is an app the simply displays a single, fixed webpage (huffingtonpost.com) but uses Webkit library to do this. Resulting measurements for memory usage can be seen in Figure 5. The error bar shows the variation of the data in sequential executions. One can observe a very small variation in the measurement between two executions of the Quicksort app and a much higher variation for the Webkit-based app. Similar behaviors were observed for the CPU and energy usage as well. These experiments pointed the Webkit library as responsible for the variation on metrics and this could be overshadowing the refactoring impacts on application performance.

V. CONCLUSION

Performance evaluations shall be performed in a systematic way, enabling its reproduction, analysis, and use. This paper presented a systematic performance evaluation of a representative Android application and used this analysis to evaluate the actual design space of a mobile application. In this study we observed that better use of assessment to eliminate

[6]http://http://www.littleeye.co/



(a) Original (b) Refactoring 1 (c) Refactoring 2

Fig. 4. Energy consumption for each version of the Tint browser

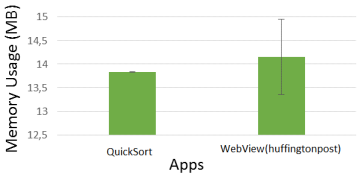


Fig. 5. Variation in memory usage for Quicksort and Webkit-based apps

possible combinations of factors that do not significantly alter the behavior of the application can make the evaluation more efficient. We also pointed the difficulties to perform a reliable and meaningful assessment. This study shows that performance assessment in mobile domain is still an ad-hoc and time consuming task. From the results, it was possible to observe the influence of code libraries in application behavior and difficulty for the developer to improve the efficiency of a library-based application. We are currently working on a semi-automatic method for performance evaluation in the mobile domain, aiming at tackling the difficulties shown in this study and helping the developer in tuning the application and focus on a better user experience.

REFERENCES

[1] J.-M. Kim and J.-S. Kim, "Androbench: Benchmarking the storage performance of android-based mobile devices," in *Frontiers in Computer Education* (S. Sambath and E. Zhu, eds.), vol. 133 of *Advances in Intelligent and Soft Computing*, pp. 667–674, Springer Berlin Heidelberg, 2012.

[2] R. Jain, *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. Wiley Professional Computing, Wiley, 1991.

[3] H. Kim, N. Agrawal, and C. Ungureanu, "Examining storage performance on mobile devices," *Proceedings of the 3rd ACM SOSP Workshop on Networking, Systems, and Applications on Mobile Handhelds - MobiHeld '11*, pp. 1–6, 2011.

[4] D. Kayande and U. Shrawankar, "Performance analysis for improved RAM utilization for Android applications," *2012 CSI Sixth International Conference on Software Engineering (CONSEG)*, pp. 1–6, Sept. 2012.

[5] J. Wang, P. Yin, and H. Zhang, "Analysis and measurement on factors influencing the performance of mobile platform," *Proceedings of 2012 2nd International Conference on Computer Science and Network Technology*, pp. 116–119, Dec. 2012.

[6] D. T. Nguyen, "Evaluating impact of storage on smartphone energy efficiency," *Proceedings of the 2013 ACM conference on Pervasive and ubiquitous computing adjunct publication - UbiComp '13 Adjunct*, pp. 319–324, 2013.

[7] S. Lee and J. W. Jeon, "Evaluating performance of android platform using native c for embedded systems," in *Control Automation and Systems (ICCAS), 2010 International Conference on*, pp. 1160–1163, Oct 2010.

[8] Y.-J. Kim, S.-J. Cho, K.-J. Kim, E.-H. Hwang, S.-H. Yoon, and J.-W. Jeon, "Benchmarking java application using jni and native c application on android," in *Control, Automation and Systems (ICCAS), 2012 12th International Conference on*, pp. 284–288, Oct 2012.

[9] C.-M. Lin, J.-H. Lin, C.-R. Dow, and C.-M. Wen, "Benchmark Dalvik and Native Code for Android System," in *2011 Second International Conference on Innovations in Bio-inspired Computing and Applications*, pp. 320–323, IEEE, Dec. 2011.

[10] M. Guthaus, J. Ringenberg, D. Ernst, T. Austin, T. Mudge, and R. Brown, "Mibench: A free, commercially representative embedded benchmark suite," in *Workload Characterization, 2001. WWC-4. 2001 IEEE International Workshop on*, pp. 3–14, Dec 2001.

[11] A. Gutierrez, R. G. Dreslinski, T. F. Wenisch, T. Mudge, A. Saidi, C. Emmons, and N. Paver, "Full-system analysis and characterization of interactive smartphone applications," in *Proceedings of the 2011 IEEE International Symposium on Workload Characterization*, IISWC '11, (Washington, DC, USA), pp. 81–90, IEEE Computer Society, 2011.

[12] C. Lee, M. Potkonjak, and W. Mangione-Smith, "Mediabench: a tool for evaluating and synthesizing multimedia and communications systems," in *Microarchitecture, 1997. Proceedings., Thirtieth Annual IEEE/ACM International Symposium on*, pp. 330–335, Dec 1997.

[13] N. V. Uti and R. Fox, "Testing the Computational Capabilities of Mobile Device Processors: Some Interesting Benchmark Results," *2010 IEEE/ACIS 9th International Conference on Computer and Information Science*, pp. 477–481, Aug. 2010.

[14] G. Chen, M. Kandemir, N. Vijaykrishnan, and M. Irwin, "PennBench: a benchmark suite for embedded Java," in *2002 IEEE International Workshop on Workload Characterization*, pp. 71–80, IEEE, Nov. 2002.

[15] M. Schoeberl, T. B. Preuß er, and S. Uhrig, "The embedded Java benchmark suite JemBench," *Proceedings of the 8th International Workshop on Java Technologies for Real-Time and Embedded Systems - JTRES '10*, pp. 120–127, 2010.

[16] K. Wac, "Towards qos-awareness of context-aware mobile applications and services," in *On the Move to Meaningful Internet Systems 2005: OTM 2005 Workshops* (R. Meersman, Z. Tari, and P. Herrero, eds.), vol. 3762 of *Lecture Notes in Computer Science*, pp. 751–760, Springer Berlin Heidelberg, 2005.

[17] K. Wac, A. van Halteren, and D. Konstantas, "Qos-predictions service: Infrastructural support for proactive qos- and context-aware mobile services (position paper)," in *On the Move to Meaningful Internet Systems 2006: OTM 2006 Workshops* (R. Meersman, Z. Tari, and P. Herrero, eds.), vol. 4278 of *Lecture Notes in Computer Science*, pp. 1924–1933, Springer Berlin Heidelberg, 2006.

[18] Y. Ye, N. Jain, L. Xia, S. Joshi, I.-L. Yen, F. B. Bastani, K. L. Cureton, and M. K. Bowler, "A Framework for QoS and Power Management in a Service Cloud Environment with Mobile Devices," *2010 Fifth IEEE International Symposium on Service Oriented System Engineering*, pp. 236–243, June 2010.

[19] Y. Ye, L. Xiao, I.-L. Yen, and F. Bastani, "Leveraging Service Clouds for Power and QoS Management for Mobile Devices," *2011 IEEE 4th International Conference on Cloud Computing*, pp. 235–242, July 2011.

[20] G. Android, "Dalvik Technical Information."

[21] V. A. Bloomfield, *Using R for Numerical Analysis in Science and Engineering*. Chapman & Hall/CRC, 2014.

[22] J. Mattos, E. Specht, B. Neves, and L. Carro, "Making object oriented efficient for embedded system applications," in *Proceedings of the 18th Symposium on Integrated Circuits and Systems Design*, SBCCI05, (Florianpolis), pp. 104–109, IEEE Computer Society, 2005.

[23] M. e. a. Oliveira, "Impact of quality metrics for sw product on embedded system properties," in *Proceedings of the 5th International Workshop on Model-based Methodologies for Pervasive and Embedded Software*, MOMPES08, (Budapest), pp. 68–77, IEEE Computer Society, 2008.