An Approach to Analyze Power Consumption on Apps for Android OS Based on Software Reverse Engineering

E. Caballero-Espinosa, A. Payne, T. Atkison
Computer Science Department, University of Alabama, Tuscaloosa, AL, USA

Abstract—Managing power consumption efficiently in mobile devices is a challenge for software engineers who design and develop operating systems or applications for this market. Studies have been conducted to offer tools and techniques aimed to accomplish this goal by implementing simulation, apps, and sophisticated power meters. However, they do not provide a practical methodology for those students, practitioners or newcomers who want to analyze source code of apps and reduce power consumption. In this paper, a new approach is presented applying Software Reverse Engineering (SRE) to optimize the source code of mobile applications in Android OS and increase the battery duration. The results demonstrate that the proposed lightweight methodology reduces power consumption on a mobile application up to 20%.

Keywords: Apps, Mobile Devices, Power Consumption Analysis, Software Reverse Engineering, Source Code Analysis.

1. Introduction

Handheld devices, such as smartphones and tablets, have become important tools for human beings not only for communication but to support daily basic activities. They include organizing lists of tasks, geolocation, and reading news from the Internet. However, when these apps are executed by users, the instructions of the source code generate processes that must be attended to by the processor. This constant interaction generates a degree of power consumption. Therefore, techniques and methods are needed to minimize the power consumption on mobile devices.

The research presented in this paper will attempt to answer the following question: Can a flexible method be suggested to debug the source code of any Android mobile app by means of a Software Reverse Engineering (SRE) approach to aid in a power consumption reduction? This effort is important to the SRE field because it has been identified as an emerging goal [1]. Therefore, a methodology based on SRE will be proposed in this research to analyze the source code of a mobile application and identify potential changes that could contribute to reducing the power consumption.

In addition, this analysis is crucial in the context of mobile devices and sensor networks [2], [3]. Moreover, the results will offer hints to recommend changes for future maintenance tasks making the power consumption performance more efficient. In addition, this study will provide additional information to the current comprehension of the mobile applications, which is not accessible by using other methods.

The remainder of this paper is structured as follows. In Section 2, a succinct background about SRE will be presented. In Section 3, the methodology presented in this research will be described. In Section 4, the results will be presented. In Section 5, the limitations and threats to validity will be discussed, as well as the implications to practitioners and researchers by suggesting future work. Finally, conclusions will be drawn in Section 6.

2. Background

2.1 SRE and Legal Aspects

Chikofsky and Cross [4] define SRE as the process of analyzing a subject system to identify the system’s components and their inter-relationships and create representations of the system in another form or at a higher level of abstraction. In other words, SRE is a process of examination, not a process of change or replication. Furthermore, SRE inverts the traditional software life cycle taking place on the design stage. Therefore, when a software engineer performs reverse engineering, software artifacts like source code, executables, project documentation, or test cases are the source of information [1], [4].

Despite the fact that researchers and practitioners applied SRE to analyze software products for different purposes, such as teaching and fixing bugs, it is very important to keep in mind some practical legal aspects before undertaking a SRE project. Behrens and Levary [5] recommend straightforward steps in order to avoid legal conflicts like Sega Enterprise Ltd. vs Accolade, Inc. First, the attorney should advise the client to obtain an authorized copy of the software. The second step indicates that attorneys should advise their clients to be sure that reverse engineering is the only mechanism to get information from the software under evaluation. The third step indicates that engineers should be advised to reverse engineer only the portions of the original program needed to decipher the precise functional elements required for the new program. The next step suggests that software developers should be sure to divide their reverse engineering efforts between two groups of engineers, one group to reverse engineer the program, the other to develop the new software. Finally, software developers should conduct research on the product to be reverse engineered to ensure that patent law does not provide protection...
for the particular process or function that is to be reverse engineered and used in a new program.

2.2 Android Applications and Power Consumption Analysis

The usage of mobile devices such as smartphones and tablets has increased dramatically in the past decade [6]. They have also become indispensable personal gadgets to support activities in almost every aspect of our lives [7]. Because of the fast advancement of mobile technologies, platforms, as well as the enthusiasm of individual developers, a wide range of mobile applications, commonly named apps, has been created to serve and make daily life more convenient. These apps are distributed as special files, called Application Package File (APK), with the .apk file extension [8].

Ongkosit and Takada [9] state that responsiveness has been an important type of quality factor in Android apps since it directly affects user experience. However, Couto et al., [10] argue that the Android ecosystem was designed to support all different mobile (and non-mobile) devices (ranging from smart-watches to TVs). As a result, a power consumption model for Android needs to consider all the main hardware components and their different states (for example, CPU frequency, percentage of use, etc.).

Previous studies have made important contributions to the analysis of power consumption in mobile-system devices and Android OS by using a range of methods and tools. They include simulating a variety of environments or test scenarios simultaneously [11], Arduino Duemilanove Board [12], apps like PowerTutor [13], commercial devices like Power Monitor [14], and web-based platforms for measurement and analysis of power consumption on multiple mobile devices [15].

Furthermore, other researchers have analyzed power consumption of apps based on instructions executed. Couto et al. [16] applied API-based software for monitoring and analyzing power consumption for the Android ecosystem. Hao et al. [17] combined program analysis and per-instruction energy modeling. However, they suggest neither a SRE method nor insights about code optimization to reduce the power consumption of an app.

3. Methodology

In this section, the experimental setup and how the study was conducted will be discussed.

3.1 Context

The focus of this research will be in developing and providing a methodology for power consumption analysis based on SRE. This proposal is an option for those app developers who can afford to acquire neither expensive nor affordable instruments like Monsoon Power Monitor [18], for tracking power consumption of their apps in order to make changes to their source code.

3.2 Ethics

To put into practice the concepts and legal aspects of SRE, an app developer was contacted in order to get an authorized copy of an app’s source code. The source code was the subject of this research. In this case, the developer was informed about the goal of the study, the techniques that would be applied, and what sections of the code would be used in the results section. After the explanation, the developer agreed to share the app’s source code.

3.3 Study Object and Subsystem

The source code used was from a music-player prototype that works by gesture recognition (Figure 1), which was developed in Android Studio 2.3.1. This programming environment is an Integrated Development Environment (IDE) for Android app development, based on IntelliJ IDEA [19]. The source code contains 350 lines, and three classes comprise this mobile application, such as MainActivity, MainInstructions, and POP. This app allows users to manage music files and execute the main functions. They include play (shaking), forward (passing the right hand in front of the phone), backward (passing the left hand in front of the phone), stop (double tap) and volume control (swipe the screen up or down). In the case of the MainActivity class, only the first three functions will be evaluated because the actions to activate volume control and stop require swapping the screen or continuous taps. Consequently, sophisticated instruments are needed to measure these power consumption cases.
Table 1

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor</td>
<td>Quad-core, 1300 MHz, ARM Cortex-A7, 28 nm</td>
</tr>
<tr>
<td>LCD Display</td>
<td>Super AMOLED</td>
</tr>
<tr>
<td>Wi-Fi</td>
<td>IEEE 802.11 b/g/n</td>
</tr>
<tr>
<td>GPS</td>
<td>A-GPS</td>
</tr>
<tr>
<td>Cellular</td>
<td>Cricket-Mobile USA: GSM/UMT/LTE/HSPA</td>
</tr>
<tr>
<td>Audio</td>
<td>Built-in microphone and speaker</td>
</tr>
<tr>
<td>Battery</td>
<td>Internal Rechargeable Li-ion: 2600 mAh</td>
</tr>
<tr>
<td>Operating System</td>
<td>Android 6.0 Marshmallow</td>
</tr>
</tbody>
</table>

Murmurria et al. [20] define subsystems as each dependent on various factors that determine their power consumption. This includes the amount of time they are kept alive, but more importantly, the state of the hardware, and other environmental factors. Table 1, based on Zhang et al. [21], details the Samsung Galaxy Express Prime device used for monitoring the power consumption of the study object.

### 3.4 Power Consumption Measurement

Currently, most power measurements are based on hardware meters used to confirm power side channels and obtain specialized measurements on mobile phones. Nonetheless, these instruments are still not applicable for this study because they are apparently not a standard auxiliary accessory for smartphones [18]. However, Android is an operating system that makes accessible power statistics. In general, instant power numbers can be calculated based on voltage and current readings of BMU (Battery Monitoring Unit).

For this research, PowerTutor was used to obtain measurements before and after possible changes in the source code. This app has been widely used [21] and is a free app available on Google Play (Figure 2). In addition, it provides accurate estimates of energy consumption in real time to the hardware components, including CPU and LCD display, as well as GPS, Wi-Fi, audio and mobile network interfaces. For apps, it calculates how much power each app uses by simulating hardware state as though the app was running alone [21].

### 3.5 Optimizing the Performance Based on Reverse Engineering

Regarding source code, Couto et al. [10] recommend monitoring application methods, since they are the logical code unit used by programmers to structure the functionality of their applications. Nonetheless, classes will still be monitored since the app prototype only contains three main classes.

In order to optimize the source code, Binh et al. [22] propose reverse engineering and the substitution of equivalent expressions as an improved local optimization method for high-level source code. They also highlight that the technique is generic because it is independent of the processor. In addition, by using this proposal, the app only needs to calculate an equivalent expression once and the execution time would decrease. They also propose the use of direct acyclic graph (DAG).

The complete optimization process of a source code by replacing equivalent expressions and using a DAG is as follows.

#### 3.5.1 Identifying and replacing equivalent expressions

This stage is based on mathematical properties. The first step is to scrutinize each logical unit to determine expressions and subexpressions. After that, the equivalent expressions should be identified. Finally, the equivalent expressions should be replaced by a representative one. Table 2 illustrates this step.

#### 3.5.2 Constructing a DAG

A DAG for each logic unit can be described as follows: leaf nodes are labeled with unique identifiers like variable or constants; internal nodes are labeled with operation symbols; arcs represent relationships between operations and operand; and internal nodes represent computed values held by the identifiers of these nodes, i.e. Figure 3. When common subexpressions, dead code, and unused variables are removed, this action may improve the app performance and minimize power consumption levels. Figure 4 illustrates the procedure adapted to this case study.
3.6 Getting started

With the research question in mind, the construction of the pilot case study was designed and a protocol created. Then, a developer was contacted to get the source code of an app following SRE legal practices. It is important to indicate that only the app, description and programming environment were requested for this experiment, so software requirements or specific features were not asked for to avoid threats to validity.

After setting up the programming environment (Android Studio), the app was imported and its execution verified according to the descriptions provided by the developer. The subsystem was, then, set by enabling its development options like USB debugging, and a demo of this app was installed for the experiments (Figure 5). Joule, an amount energy equivalent to one watt per 1 second, was chosen as the unit for power consumption measurements from PowerTutor.

For the source code analysis, a flexible design was created to allow both engineers and developers to optimize the source code using reverse engineering techniques like replacing equivalent expressions and constructing DAGs. For this first approach, an app was investigated because it is more feasible to get and evaluate its source code than robust software.

4. Results

The analysis identified three main classes as logic units on the source code. Two of them control basic functions, such as the help bottom and the welcome screen that also shows a concise tutorial. However, MainActivity, the main class, contains the instructions of each gesture and other actions. Table 3 shows the description of each class.

As for power consumption, PowerTutor was used to obtain measurements during the execution of the app before and after modifications to the source code in order to track power consumption for this study object. However, PowerTutor provides accumulative measurements according to the processes executed by each app; as a result, this information is not shown by instruction or line of code executed. In this case, power consumption is estimated for each class or function in the source code by subtracting the measurement of the next action to the measurement of the action executed before. An action is defined as a tap or gesture that activates the instructions in the class.

\[
P_{C_{\text{action}}} = m_{n+1} - m_{n-1} \tag{1}\]
Table 4

<table>
<thead>
<tr>
<th>Classes</th>
<th>Power Consumption (mJ)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MainActivity</td>
<td>167</td>
</tr>
<tr>
<td>MainInstructions</td>
<td>88</td>
</tr>
<tr>
<td>POP</td>
<td>78</td>
</tr>
</tbody>
</table>

Fig. 6

Code analyzer option on Android Studio.

For Equation 1, let $PC_{actions}$ be the power consumption measurement estimated for each action. Then, $m$ is the measurement obtained from PowerTutor, and $n$ is the number of taps required to generate one event on the app. For example, the first tap activates the welcome screen and generates a measurement of 75 mJ. The next action is playing the song by shaking the device and generates an accumulative measurement of 185 mJ. Applying Equation 1, $n$ is equal to two, the measurement for the welcome screen (MainInstructions class) is 70 mJ and 110 mJ for playing the song (gesture recognition on the MainActivity class). Table 4 shows the first set of measurements before modifications to the source code.

Analysis of the source code was performed using the Code Inspector tool available on Android Studio tool bar (Figure 6). After running the tool, the inspection results displayed a number of warnings without significant impact on the execution of the study object. Table 5 shows the results of this inspection. However, each warning is evaluated to find common subexpressions, dead code, or unused variables that could affect the power consumption.

After analysis, clusters 1 and 5 were determined to not require changes because these warnings pinpoint phrases or variables written in Spanish. In addition, the warnings about declaration suggested some variables could be private; nonetheless, no changes were made since they do not represent a technical error on Java. The same was determined for cluster 6. Regarding cluster 4, an unused import found during the code inspection was deleted. For cluster 7, both control-flow warnings were simplified. Because the source code did not have common mathematical subexpressions, this reverse-engineering technique could not be applied.

Upon finishing the analysis and optimization process, the study object was compiled and tested on the subsystem again. Table 6 shows the second set of power-consumption measurements estimated by PowerTutor. The new measurements indicate a decrease of power consumption in runtime.

Regarding the MainActivity class, the three actions under evaluation also presented improvements. Table 7 shows the results.

5. Discussions

In this section, limitations, and validity of the results will be discussed, as well as the implications for practitioners and researchers.

5.1 Validity and Limitations

To increase the credibility, an app developer was contacted to obtain the study object. Particular requirements or features were not requested. Moreover, PowerTutor was used to obtain accurate power consumption measures. However, one limitation regarding this tool is the accumulative format to show the power consumption measurements for each app. Because of that, a simple equation was defined to estimate the power consumption of each class or functions in the source code based on the actions that generate an event. Therefore, it could cast threats to validity.

This research is a small scale case study that involves one app. Therefore, the results are not generalizable to other apps since developers could have taken into account one or more contexts, used different programming languages, and written thousands of lines of code. In fact, the study was a design decision because it is a need identified in a previous publication [1] and has not been investigated from a SRE perspective. Furthermore, the existing literature about power consumption analysis on both devices and apps did not cover the topic applying SRE. This study also contributes to the “green computing” perspective by finding alternative methods to reduce utilization of power resources.

5.2 Implications for Practitioners

This study was designed to provide a flexible methodology based on SRE. The results show that identifying and adjusting issues like unused variables, dead code or equivalent expressions can reduce power consumption. The following steps are a guideline to replicate our approach for other mobile applications for Android OS. Figure 7 shows the flowchart of this guideline.

1) Read the practical legal aspects of SRE.
2) Contact the app developer and apply the practical legal aspects of SRE.
3) Set the programming language environment on a computer and the subsystem for app tests on your mobile phone.
4) Identify logic units of interest in the source code like methods or classes and create a register, see Table 6.
5) Download PowerTutor on the subsystem.
Table 5
RESULTS OF THE SOURCE-CODE INSPECTION.

<table>
<thead>
<tr>
<th>#</th>
<th>Cluster</th>
<th>Type</th>
<th>Quantity</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Internationalization</td>
<td>Text View</td>
<td>6</td>
<td>String literal in 'setText' can not be translated. Use Android resources instead.</td>
</tr>
<tr>
<td>2</td>
<td>Declaration</td>
<td>Declaration access</td>
<td>8</td>
<td>They can be private.</td>
</tr>
<tr>
<td>3</td>
<td>Imports</td>
<td>Unused import</td>
<td>1</td>
<td>Unused import 'import android.view.keyEvent;'</td>
</tr>
<tr>
<td>4</td>
<td>Probable bugs</td>
<td>Unused assignment</td>
<td>1</td>
<td>Variable 'listOfSensorOnDevice' is never used.</td>
</tr>
<tr>
<td>5</td>
<td>Spelling</td>
<td>Typo</td>
<td>10</td>
<td>Words written in Spanish language like ‘Detenido’ - stoped.</td>
</tr>
<tr>
<td>6</td>
<td>Verbose or redundant</td>
<td>Redundant type cast</td>
<td>2</td>
<td>Casting ‘this’ to ‘SensorEventListener’ is redundant.</td>
</tr>
<tr>
<td>7</td>
<td>Control flow</td>
<td>If statements can be simplified</td>
<td>2</td>
<td>if (prueba==true) {...</td>
</tr>
</tbody>
</table>

Table 6
SECOND SET OF POWER-CONSUMPTION MEASUREMENTS.

<table>
<thead>
<tr>
<th>Classes</th>
<th>Power Consumption (mJ)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Before</td>
</tr>
<tr>
<td>MainActivities</td>
<td>167</td>
</tr>
<tr>
<td>MainInstructions</td>
<td>88</td>
</tr>
<tr>
<td>POP</td>
<td>78</td>
</tr>
</tbody>
</table>

Table 7
MEASUREMENTS ON MainActivity FUNCTIONS.

<table>
<thead>
<tr>
<th>Classes</th>
<th>Power Consumption (mJ)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Before</td>
</tr>
<tr>
<td>Play</td>
<td>69</td>
</tr>
<tr>
<td>Forward</td>
<td>47</td>
</tr>
<tr>
<td>Backward</td>
<td>51</td>
</tr>
</tbody>
</table>

6) Take measurements of the app in runtime on the subsystem and apply Equation 1 if it is needed.
7) Create a table to register the measurements obtained from PowerTutor.
8) Optimize the source code by applying Section 3.5 described in the methodology section.
9) Set the subsystem with the new version of the app after source code modifications.
10) Take the second set of measurements and compare them against the first set.

5.3 Implication for Research

Although this case study is a small-scale design, it provides interesting results and a flexible methodology to analyze the power consumption on apps following SRE concepts and techniques. In this research, measurements of power consumption were presented for logic units on source code. Future research could estimate the power consumption of the most implemented logic units from the most popular IDEs to develop apps, such as Android Studio. From these results, it would be possible to generate metrics about power patterns and power consumption for each logic unit. It would also provide elements for a robust security analysis, and allow developers to create programming strategies to prevent power-side channel attacks [18]. Furthermore, the methodology could be applied to analyze the source code of robust apps from different contexts and use software to automate this task. It will also include experiments on diverse mobile devices and processors. In this case, CodeSurfer and Bauhaus could be acquired for this purpose because the source code of these apps is comprised of thousands of lines of code and it is a time consuming task for humans.

6. Conclusions

The results of research conducted to study power consumption on mobile applications were presented based on source code analysis by following SRE techniques. From these results, a preliminary methodology was built explaining how to apply practical legal aspects of SRE and perform a power consumption analysis using PowerTutor. These results answered the research question of this study. Furthermore, the results provide a complete description of how a practitioner can apply SRE to evaluate a source code and apply changes to make an app more efficient in terms of CPU use, power consumption and lines of code. As far as we know, this is the first study to address this topic in SRE.

While this methodology may not be complete, this approach can be modified and enriched by including other application contexts, size of source code, programming environment, and affordable SRE tools, not only for researchers but also practitioners. Therefore, the proposed approach provides a well-founded starting point to keep studying this topic from a SRE perspective.

References

Fig. 7
GUIDELINE TO APPLY POWER CONSUMPTION ANALYSIS BASED ON SRE.