GreenSource: a large-scale collection of Android code, tests and energy metrics

Rui Rua\textsuperscript{1} HASLab/INESC TEC, Portugal
Universidade do Minho, Portugal
Braga, Portugal
rui.a.rua@inesctec.pt

Marco Couto\textsuperscript{2} HASLab/INESC TEC, Portugal
Universidade do Minho, Portugal
Braga, Portugal
marco.l.couto@inesctec.pt

João Saraiva\textsuperscript{3} HASLab/INESC TEC, Portugal
Universidade do Minho, Portugal
Braga, Portugal
saraiva@di.uminho.pt

Abstract—This paper presents the GreenSource infrastructure: a large body of open source code, executable Android applications, and curated dataset containing energy code metrics. The dataset contains energy metrics obtained by both static analysing the applications’ source code and by executing them with available test inputs. To automate the execution of the applications we developed the AnaDroid tool which instruments its code, compiles and executes it with test inputs in any Android device, while collecting energy metrics. GreenSource includes all Android applications included in the MUSE Java source code repository, while AnaDroid implements all Android’s energy greedy features described in the literature. GreenSource aims at characterizing energy consumption in the Android ecosystem, providing both Android developers and researchers a setting to reason about energy efficient Android software development.

Index Terms—Energy Consumption, Android, Source Code Metrics.

I. INTRODUCTION

E NERGY consumption has become a main concern for software developers of non-wired/mobile devices. Such devices have become powerful computing devices, with multiple and complex CPUs, offering more functionalities than regular personal computers, such as GPS-based location, camera, and activity sensors. While the applications rely on those features, they also demand more from the devices’ batteries, which are known to have limited capacities. Hence, developers are now seeking help to learn more about energy-aware development strategies [1], [2].

To address the developers demands, researchers started developing tools and techniques to analyze the energy efficiency of software components, such as data structures [3]–[6], APIs [7], [8], source code patterns [9]–[12] and even languages [13], [14]. Nevertheless, the validation of such tools and techniques is not always performed over a large collection of software artifacts, hence their conclusions cannot be completely generalized.

In order to provide a suitable setting for evaluating software analysis and optimization techniques, software engineering researchers have defined both benchmark infrastructures [15], and large-scale open source code repositories [16]–[18], which are freely available so that researchers can evaluate their works. Even so, there is still a lack of benchmarking infrastructures and/or repositories that can be used to validate energy-related works in Android ecosystem, which is by far the most used OS for mobile devices, having in 2018 84.8% of the world’s devices running its platform\textsuperscript{1}.

This paper presents the GreenSource\textsuperscript{2} infrastructure: a large body of open source Android applications tailored for energy analysis and optimization. GreenSource consists of three main components: (1) a large collection of open source, executable Android applications, (2) a benchmarking framework, called AnaDroid\textsuperscript{3}, to test such applications under different usage scenarios and collect structural and energy-related metrics, and (3) a large scale repository of metrics obtained from executing the applications using AnaDroid.

Each application in GreenSource can be analyzed, optimized, instrumented with energy measuring code, and executed with the provided test scenarios. The metrics database stores both static code metrics (e.g., which well-known energy-greedy APIs [7] or patterns [10] it uses, or how many lines of code it has, etc.), and dynamic metrics (i.e., obtained from testing the applications, such as the total energy consumed, or the resources/sensors usage. The context of each execution is also stored in the database (e.g., mobile device, operating system, number of running processes, etc.).

GreenSource provides a common ground for energy-aware software analysis and development: one can execute an application under a certain context, and the resulting metrics can be compared with previously obtained ones, both in the same or different contexts. This provides potentially interesting data mining analysis, such as comparing energy efficiency among different platform versions, devices, or even among application’s versions. In addition, the full infrastructure was developed to be extended with new applications and inputs, 

\textsuperscript{1}https://www.idc.com/promo/smartphone-market-share/os
\textsuperscript{2}GreenSource support webpage: http://greenlab.di.uminho.pt/greensource
\textsuperscript{3}AnaDroid’s source code: https://github.com/RRua/AnaDroid
new static and dynamic green metrics, testing devices, or even energy profiling tools.

The main contributions of this paper are the three software artifacts we have developed:

- The collection of testable Android applications, which at the moment contains 609 projects;
- GreenSource, consisting of the combination of the metrics database and a RESTFUL-based query engine to access the data;
- The AnaDroid framework, which comprises the applications’ source code analysis, source code instrumentation and transformation, and context-independent execution phases;

The remaining of this paper is organized as follows: Section II contains a description of the tools and materials used in our work; in Section III we explain the methodologies followed to construct GreenSource and the information contained in it; Section IV contains an explanation of the collected data, how is it structured an how can it be accessed/used; Finally, in Section V we present our conclusions and future work directions.

II. MATERIAL AND TOOLS

In this section, we describe the different components used to construct our GreenSource dataset. First, we explain how we obtained our body of source code of executable Android applications. Then, we explain the TrepnLib library, which is used by AnaDroid to collect information regarding energy consumption and resources usage of the tests. Then, we present our framework AnaDroid, which is responsible for executing the applications while obtaining both the source code and green metrics for our dataset.

A. Data provenance

Collecting a significant number of Android applications was deemed necessary to perform our study and build the GreenSource repository. The goal was to gather the largest possible number of real-world, open source, and executable Android applications.

After a thorough research, we considered the MUSE repository [17], [18]: a very large collection of Java projects obtained from different online platforms, such as GitHub or GitLab. This repository has an associated database, which contains information regarding the static structure of each of its projects. Thus, in MUSE is possible to use such database to filter projects with some properties, like for example projects that have at least 50 different classes, or that at least one class with a specific import statement.

To extract all Android applications from MUSE we define a filter to select the projects that have at least a class with imports from the Android API. Hence, we made sure that we were only selecting Android applications. Then, we performed a build check to determine which applications could be successfully built and executed, resulting in a set of 609 executable applications.

B. Trepn and TrepnLib

In order to monitor the energy consumed by an application we rely on the Trepn Profiler [19]: a software-based artifact developed by the mobile device manufacturer Qualcomm that works on Snapdragon chipset-based Android devices. It is a monitoring tool which can be used to profile hardware usage (e.g. GPS, WiFi, etc.), resources usage (e.g. memory and CPU), and energy consumption of the whole system in a given time interval. Moreover, Trepn reports accurate measurements on Qualcomm-based Android devices [20]. Trepn needs to be explicitly started and stopped, and after starting it collects profiling samples at a rate that can’t be adjusted to less than 100 ms. In the same spirit as previous works [21]–[23], we used Trepn to gather energy consumption values for each execution of an application.

Trepn profiling mechanism can also be controlled by using Java methods, which can easily be integrated in any Android application’s source code, or via Android debugging tools such as Android Debug Bridge (ADB). Thus, it is possible to start and stop profiling exactly during the execution of specific portions of code, such as methods, or during the execution of an application. To support both test and method based scenarios, we developed the Android Library TrepnLib. This library provides an API that allows to instrument the application source code with API calls, so that when executed it profiles the instrumented Java class methods. To monitor test execution scenarios, this library also provides functions to start and stop the profiling process.

C. AnaDroid Framework

In order to analyse the GreenSource repository, we have developed the AnaDroid tool. The AnaDroid workflow is depicted in Figure 1.

The entry point is the application project (i.e., source code and assets). It starts by analyzing the source code to compute static (green) code metrics, such as the APIs used by methods and classes, number of declared variables, arguments, among others. Then it instruments the source code to include the necessary TrepnLib instructions to monitor energy consumption at runtime.

After the code is fully instrumented, AnaDroid builds the application and generates the corresponding APK file, which is used to install the application on a connected Android device. Once the installation finishes, AnaDroid runs a series of usage scenarios over the application and collects the results for them, storing them in the GreenSource green metric database, described in the next section.

D. GreenSource Backend

The GreenSource repository contains an backend API, which main task is to store the information collected for each tested application in a relational database (which simplified schema is depicted in Figure 24), and allow access to it. It is accessible through a web server that provides RESTful API,

4The complete database schema is available here: https://bit.ly/2tafvdM.
providing a uniform form of communication that allow CRUD operations through HTTP requests.

The contained database is populated every time AnaDroid executes tests over an application. The database structure was inspired by the SourcererDB [24], a previous works that presented an approach to store metrics and relations between Java source code elements and It has been carefully designed to be expandable for future refinements and expansions. For instance, every test result (i.e., an entry in the TestResults table) is always associated to an application, a device on which is executed, and to a test description (i.e., an entry in the Tests table), which has an associated testing tool/framework and a profiler. This way, our infrastructure considers extensions to support different testing frameworks, devices and energy profilers, as well as new application and test metrics.

III. METHODOLOGY

As stated before, to collect the data contained in our dataset we used AnaDroid to test the collected applications with different scenarios and store it in the GreenSource repository. In this section, we explain in detail the conditions under which the applications where tested, how we developed the tests, and how we ensured that the testing conditions were equally maintained for every application.

In GreenSource we can also collect the energy consumption of the application in a real usage of a mobile device. In fact, the context where the application executes on test input is stored in our dataset: processes running, brightness levels, networks traffics, etc. This allows us to reason about energy consumption in different devices and settings.

A. Experimental Setup

The tests conducted for every application were all performed in the same factory-reseted Nexus 5 device. The brightness level of such device was always in the lowest possible value, to ensure that the consumption not related to the computational effort of the application under test was reduced to the minimum. During each test execution, we ensured that only the application under test, Trepn and the OS-related applications were installed.

To exercise the applications, we designed 20 usage scenarios using the Android Application Exerciser Monkey [25]. We chose this tool because it is, to the best of our knowledge, the only one capable of automatically test any application without knowing its context, which is a necessary characteristic since we aimed at testing a large set of applications. Moreover, it was already proven that Monkey achieves the better compromise between coverage and setup effort [26], [27].

B. Experimental Procedure

Although our infrastructure is capable of monitoring energy at the method level, for the purpose of creating an uniform dataset and provide it as a contribution to this work, we focused on monitoring energy at the test level. To ensure the maximum possible coverage of the tests, after running the pre-defined set of 20 tests, we check the method coverage and see if it is greater or equal than 60%. If it is, we stop the test procedure and store the results. If not, we use a new set of 30 tests to execute the application, and we execute them one after another until we reach 60% method coverage or we run out of tests. This procedure is depicted in Algorithm 1

Before executing each test, our framework first opens the application, starts the Trepn profiling service, waits 5 seconds and then runs the usage scenario using the Monkey events. When Monkey finishes, it stops Trepn, waits 5 more seconds to cool-down the device, and stores the results.

```
for (app : projects builted)
    do
        grantAppPermissions(app);
        i = 0;
        while i ≤ 20 || (i ≤ 50 && coverage ≤ 60% )
            do
                startProfiling();
                getDeviceSystemState();
                runTest(app, i);
                getDeviceSystemState();
                stopProfiling();
                cleanAppCache(app);
                pullResults();
                i++;
        end
    end
sendResults();

end
```

Algorithm 1: Test Execution Procedure

IV. DATA DESCRIPTION

The process of running AnaDroid with a wide range of applications in a mobile device is a time consuming process. This process is influenced by both the performance of the development machine, the Android device on which the inputs are executed, and the complexity of the tested inputs.
Currently, the framework successfully produced results for a total of 4377 different test scenarios, executed over a total of 222 Android applications. We were able to gather 281811 metrics values regarding source code elements and tests (i.e., static and dynamic metrics), divided according to their type: 39375 test metrics, 241128 method metrics, and 1308 class metrics. We present the list of metrics considered so far in Table I.

The nature of the obtained results allows to visualize and draw comparisons on applications according to the executed inputs. We replicated the execution of the same test inputs, with the same sequence of UI-events generated for every application. The main entities and relations of the database schema designed for storing this results is represented by the diagram in figure 2.

Each execution of a test correspond to a new entry in the TestResults table. In this table we describe the attributes of the performed tests, as well as the state of the device before and after the test execution. This information is needed to fully characterize the testing conditions, and to evaluate if there were any major differences before/after the test that could be influencing the results (e.g., the resources usage). Table II contains a concrete example of such information calculated for test number 583, while Table III shows the metrics calculated for such test.

We are constantly increasing the number of metrics gathered from the analysis performed with the execution of the AnaDroid framework over the set of applications. The metrics considered so far are related to invoked source code portions (application, class or methods) and to the executed tests. We are targeting difficult objectives as identify the factors that have impact on the energy consumption of Android Applications. Thus, we are contributing with as many consumption-related metrics as possible.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>test_id</td>
<td>test identifier</td>
<td>583</td>
</tr>
<tr>
<td>timestamp</td>
<td>timestamp</td>
<td>2019-02-02T18:04:53</td>
</tr>
<tr>
<td>test_seed</td>
<td>test seed for generating events with monkey</td>
<td>40201</td>
</tr>
<tr>
<td>description</td>
<td>description</td>
<td>—</td>
</tr>
<tr>
<td>profiler</td>
<td>profiler tool test</td>
<td>trepp</td>
</tr>
<tr>
<td>init_mem (B)</td>
<td>memory allocated at the beginning of test</td>
<td>1069323264</td>
</tr>
<tr>
<td>end_mem (B)</td>
<td>memory allocated at the end of test</td>
<td>1066083328</td>
</tr>
<tr>
<td>nr_classes</td>
<td>number of processes running before the test</td>
<td>32</td>
</tr>
<tr>
<td>api_level</td>
<td>API level</td>
<td>23</td>
</tr>
<tr>
<td>android_version</td>
<td>Android Version of device</td>
<td>6.0.1</td>
</tr>
<tr>
<td>device_serial_nr</td>
<td>device serial number</td>
<td>066b51bd005cae0e</td>
</tr>
</tbody>
</table>

Table II

<table>
<thead>
<tr>
<th>Metric</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coverage</td>
<td>43 %</td>
</tr>
<tr>
<td>Time</td>
<td>4.926 s</td>
</tr>
<tr>
<td>Energy</td>
<td>8.324 J</td>
</tr>
<tr>
<td>Max CPU Load</td>
<td>44 %</td>
</tr>
<tr>
<td>AVG CPU Load</td>
<td>23.93 %</td>
</tr>
<tr>
<td>GPU frequency</td>
<td>1.307 MHz</td>
</tr>
<tr>
<td>Max memory</td>
<td>161064 B</td>
</tr>
<tr>
<td>Avg Memory</td>
<td>1579181.94 B</td>
</tr>
<tr>
<td>Avg Wifi RSSI</td>
<td>-65 dBm</td>
</tr>
</tbody>
</table>

Table III

Table II

<table>
<thead>
<tr>
<th>Metric</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>T / M LoC</td>
<td>281811</td>
</tr>
<tr>
<td>T / M Time</td>
<td>40201</td>
</tr>
<tr>
<td>T / M NrArgs</td>
<td>3418</td>
</tr>
<tr>
<td>T / M NrDeclaredVars</td>
<td>10705</td>
</tr>
<tr>
<td>T / M AvgMemory</td>
<td>1579181.94</td>
</tr>
<tr>
<td>T / M AvgWifi RSSI Level</td>
<td>-65 dBm</td>
</tr>
<tr>
<td>T / M Avg Memory Used</td>
<td>AvgMemory</td>
</tr>
<tr>
<td>T / M Max Memory Used</td>
<td>AvgMemory</td>
</tr>
<tr>
<td>T / M Tp</td>
<td>4.926 s</td>
</tr>
<tr>
<td>T / M AvgWifi RSSI Level</td>
<td>-65 dBm</td>
</tr>
<tr>
<td>T / M Avg Memory Used</td>
<td>AvgMemory</td>
</tr>
<tr>
<td>T / M Max Memory Used</td>
<td>AvgMemory</td>
</tr>
<tr>
<td>T / M Avg Wifi RSSI Level</td>
<td>-65 dBm</td>
</tr>
</tbody>
</table>

The table III provides an example of some of the metrics that we are gathering from the execution of the tests. The metrics provided in table are correspondent to the test presented in Table II.

V. Conclusions

The existence of a large body of open source code and executable Android applications presents a tremendous opportunity for green software research. In fact, we provide the full GreenSource infrastructure and curated datasets for other researchers to use. Moreover, we developed the AnaDroid tool to automate the process of running the applications with the provided usage scenarios, to instrument the source code with energy measurements calls, and to collect all green code metrics. These metrics include all Android’s energy greedy features published in the literature. Currently, GreenSource includes 609 different Android applications.

In order to increase the magnitude of the representativeness, we intend to steadily increase this set by extracting more applications from open source repositories. So far we have collected 281811 metrics produced by running AnaDroid in 222 applications with 4377 different inputs. This dataset is constantly being updated with more content.

We expect to soon have a dataset that characterize energy consumption on the Android ecosystem and relates such consumption to the source code of the applications. Our infrastructure can be easily extended with new applications, test inputs, and green metrics. This dataset will allow us and other researchers to answer questions such as: Which Android features have more impact on the energy consumption of an application? How different devices influence energy consumption of the source code? How is energy consumption affected by software and operating system evolution?
REFERENCES


