A Systematic Mapping on Energy Efficiency Testing in Android Applications

Jaziel S. Moreira
Federal University of Campina Grande
jaziel@copin.ufcg.edu.br

Everton L. G. Alves
Federal University of Campina Grande
everton@computacao.ufcg.edu.br

Wilkerson L. Andrade
Federal University of Campina Grande
wilkerson@computacao.ufcg.edu.br

ABSTRACT

Android devices include a wide range of features and functionalities. However, they are limited by their battery capacity. Energy efficiency has become a critical non-functional requirement for Android applications. Most applications use multiple hardware elements that may consume a great amount of energy. Moreover, energy faults and bad resource management may aggravate this issue. Several works have proposed solutions to help developers deal with energy consumption issues. In this work, we present a systematic mapping study on energy efficiency testing for Android applications. From a starting set of 1525 papers, we narrowed our investigation to 32 relevant ones. The most common research topics were *Fine-grained Estimation* with nine studies, followed by *Test Generation* and *Classification*, both with six studies. We also found that most apply only dynamic solutions and use software-based strategies to estimate energy consumption. Finally, we discuss a series of open problems that should be addressed by future research.

CCS CONCEPTS

• **Software and its engineering → Software verification and validation**.

KEYWORDS

Android Application, Testing, Energy Efficiency

ACM Reference Format:
Jaziel S. Moreira, Everton L. G. Alves, and Wilkerson L. Andrade. 2020. A Systematic Mapping on Energy Efficiency Testing in Android Applications. In *19th Brazilian Symposium on Software Quality (SBQS'20), December 1–4, 2020, São Luís, Brazil.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3439961.3439964

1 INTRODUCTION

Android devices have become popular over the years, due to their large set of functionalities and mobility. According to a recent report[1], Android is the most used mobile operational system, dominating 86% of the market share. Although computational performance and storage capabilities of mobile devices have considerably increased over the years, battery constraints remain an issue [14], specially due to an increasing demand for computationally intensive applications [7].

According to Ferrari et al. [14], heavy battery usage is one of the main reasons for poor reviews of Android applications. Users often want to avoid constant recharging. In this sense, efficient power usage is an important, and still growing, non-functional requirement for mobile projects [35].

However, current Android applications use multiple hardware elements, such as WiFi, Bluetooth, GPS, and sensors, which often reflect on a great energy consumption [18]. Moreover, code-related issues might also lead to unexpected energy usage. Energy faults and bad resource management are examples of energy inefficiency issues.

Tests can be used to avoid unnecessary power consumption [39]. However, energy inefficiencies, although common, are not easy to diagnose. They may be related to several aspects of the application's code and/or device characteristics, such as sensors usage, service management, and resources management [35].

Several researchers have worked on solutions for assessing application's energy consumption (e.g., [17, 21, 29]). The proposed solutions cover different fields of study, such as energy-aware test generation, hardware-assisted energy measurement, and estimations with machine learning. Therefore, it is important to investigate the literature in order to better assess the available solutions.

In this paper, we present a Systematic Literature Mapping [30] focused on identifying and characterizing the main approaches, tools, and frameworks for testing Android applications aiming at detecting energy faults, as well as for measuring energy consumption. For that, we follow a systematic procedure and provide a structured overview of the current state-of-art of this research field.

This paper is organized as follows. In Section 2, we present the methodology applied in this paper. Section 3 presents the data collected in the study. In Section 4, we present and discuss the results achieved. In Section 5, we present the open problems that still need to be solved. In Section 6, we discuss related work. In Section 7 we discuss about the threats to validity and their potential impact. Finally, in Section 8 presents our conclusions.

2 RESEARCH METHODOLOGY

To conduct the systematic mapping, we follow the guidelines presented by Petersen et al. [30], which comprises five phases: (i) defining research questions; (ii) conduct the search; (iii) select relevant papers; (iv) build a classification scheme; and (v) extract the data and analyze them. In the next sections, we present how those phases were conducted.

[1] https://www.idc.com/promo/smartphone-market-share/os

SBQS'20, December 1–4, 2020, São Luís, Brazil

ACM ISBN 978-1-4503-8923-5/20/12…$15.00
https://doi.org/10.1145/3439961.3439964

2.1 Goal and Research Questions

The main goal of this systematic mapping study is to obtain knowledge about the current state-of-art scenario of approaches for energy efficiency testing in Android applications. Thus, we intend to identify research reports in order to characterize the field of study and identify open problems. To achieve this goal, we defined the research questions in Table 1.

2.2 Searching Strategy

To run this systematic mapping, we selected primary studies from four well-known digital libraries: IEEE Xplore[2], ACM Digital Library[3], Springer Link[4] and Wiley[5]. Those libraries are known as repositories for relevant academic papers and have been used as sources by other systematic mappings [15, 27, 38].

To build a proper search string, Kitchenham et al. [3] suggest keywords indicating population, intervention, and outcome. Table 2 presents the final set of keywords. The main focus of this systematic mapping is energy efficiency testing of Android application. Therefore, for the *Population*, we set the keywords *Android* and *Application*, with *App* and *Service* as synonyms for the last one. Since we are focusing on testing evaluation, we set the keyword *Test* for intervention, with *Verification* as synonym. The expected outcome are works related to energy efficiency, thus we set the keywords *Energy* and *Efficiency*. In this context, *Energy* can be referred as *Power* or *Battery*, and *Efficiency* can be referred as *Economy*, *Usage*, or *Consumption*.

To work with a representative search string, we ran several search trials on the ACM Digital Library. The resulting set was manually inspected and used to refine the string. Our search string goes as follow:

(android) AND (application OR app OR service) AND (test OR verification) AND (energy OR power OR battery) AND (efficiency OR economy OR usage OR consumption)

2.3 Study Selection

Regarding power usage of Android applications, works may vary from measuring energy consumption and finding energy faults. Therefore, we defined selection criteria to help to select relevant papers to answer our research questions. To be selected, a paper must be:

- **Inclusion criteria (IC1):** A study that defines/presents a method, process, approach or instrument for measuring energy consumption of Android applications;
- **Inclusion criteria (IC2):** A study that defines/presents a method, process, approach or instrument for detecting energy faults on Android applications;
- **Inclusion criteria (IC3):** A study that compares methods, processes, approaches, or instruments for evaluating energy efficiency on Android applications.

During the searching phase, it is important to exclude irrelevant, duplicated, and/or incomplete studies. For that, we defined the following exclusion criteria:

- **Exclusion criteria (EC1):** A paper not written in English;
- **Exclusion criteria (EC2):** A Systematic mapping/review paper;
- **Exclusion criteria (EC3):** A short paper (less than five pages);
- **Exclusion criteria (EC4):** A paper that is not available online;
- **Exclusion criteria (EC5):** A duplicated study (same paper or updated version);
- **Exclusion criteria (EC6):** A paper that its main topic is not the energy efficiency of Android applications.
- **Exclusion criteria (EC7):** A paper that does not use testing.

As for the study selection process, we adopted an iterative and incremental approach:

- **Phase 1:** We searched for studies on the selected digital libraries using the searching string;
- **Phase 2:** As a preliminary selection, we filtered relevant papers using the inclusion and exclusion criteria considering their titles and abstracts;
- **Phase 3:** We filtered the selected papers based on their introduction and conclusion sections;
- **Phase 4:** The remaining papers were read in full to exclude the ones whose main topics were not related to energy efficiency of Android applications (EC6) or testing (EC7).

For the *Phase 1*, the three authors were involved in defining the search string and carrying out the preliminary selection. The first author analyzed the articles returned by the search considering the inclusion and exclusion criteria. Periodic meetings were held to validate the inclusions/exclusions, as well as to address possible differences in classification. In case of divergences, the classification of the majority was used.

2.4 Abstract Keywording

We built a classification scheme based on the abstract keywording process defined by Petersen et al. [30]. The process takes into account the studies selected after all phases previously defined and it is composed by two steps:

- Collecting keywords;
- Clustering keywords into categories.

The first step consists of reading the abstracts of all studies and collecting keywords that describe each paper. For papers whose abstract is too poor, we also analyzed the introduction and conclusion sections. The second step consists of grouping keywords from different papers into categories that reflect high-level descriptions of the nature and contributions of the studies. After running the keywording phase, we ended up with the following categories: *Test Generation*, *Resource Management Inefficiency*, *Fine-grained Estimation*, *Simulated Runtime Measurement*, and *Classification*. Studies that could not be grouped were placed in the category *General*, as recommended by Zein et al [38]. Table 3 presents the description of each category.

Naturally, is expected some of the categories to overlap. For instance, solutions for *Fine-grained Estimation* and/or *Simulated Runtime Measurement* can be used to find *Resource Management Inefficiency*. The same goes for *Test Generation* approaches, whose

[2]https://ieeexplore.ieee.org/
[3]https://dl.acm.org/
[4]https://link.springer.com/
[5]https://onlinelibrary.wiley.com/

Table 1: Research Questions

Research Question	Rationale
RQ1: What are the most common research topics on energy efficiency testing in Android applications and how they changed over time?	This question was designed to help us better understand the field and identify hot and open topics.
RQ2: What are the proposed solutions and their availability?	Since energy consumption and testing are aspects that often influence the developers daily routine, with this question we plan to understand what are the state-of-the-art solutions for supporting them.
RQ3: How were the proposed solutions evaluated?	This question aims at analysing the methods used to evaluate each solution.
RQ4: Do the solutions combine testing with other approaches?	With this question we plan to identify which other strategies are used to complement the testing.
RQ5: Do the studies used Hardware or Software based solutions to estimate energy consumption?	Performing an energy test usually requires energy consumption estimation. Thus, this question was designed to find which strategy is mostly used for that.

Table 2: Keyword and Synonyms Used in the Search String.

Term	Keyword	Synonyms
Population	Android	-
	Application	App, Service
Intervention	Test	Verification
Outcome	Energy	Power, Battery
	Efficiency	Economy, Usage, Consumption

Table 3: Categories Description.

Categories	Description
Test Generation	The main contribution is an approach for energy test generation.
Resource Management Inefficiency	The main contribution is an approach for detecting and/or fixing inefficient resource usage.
Fine-grained Estimation	The main contribution is an approach for estimating the energy consumption for fine-grained code sections.
Simulated Runtime Measurement	The main contribution is an approach for estimating energy consumption based on simulated executions.
Classification	The main contribution is an approach for classifying tests, applications or it characteristics.
General	Others.

test can be used to exercise different source code paths. However, most solutions belong to a single category.

3 RESULTS

In this section, we present the main results from our systematic mapping. In Subsection 3.1, we report the results from the study selection process. Subsection 3.2 presents general information about the studies in our final set. In Subsections 3.3 to 3.7, we answer our research questions.

3.1 Searching Results

Our initial search (*Phase 1*) returned a total of 1525 studies. After the preliminary selection (*Phase 2*), 82 studies were collected for the next phase. In *Phase 3*, by analyzing the introduction and the conclusion of each paper, we selected 40 studies. Finally, in *Phase 4*, after reading the remaining papers, we selected 32 studies to compose the final set. Details of the selection per library are shown in Table 4, while Table 5 presents the results considering each phase. Moreover, our final data is available for analysis [6].

3.2 General Results

Initially, we extracted general information from the selected papers, such as type of publication vehicle (e.g., journal, conference), authors affiliation (country), and the distribution of the papers throughout time.

3.2.1 Publication Vehicle. Figure 1 summarizes the results regarding publication vehicles. As we can see, most papers were published in Conference Proceedings (68.7%). Journal papers correspond to

[6] https://www.dropbox.com/s/t8u36o1qms5syaf/Papers_Details.csv

Table 4: Study Search Result by Library.

Digital Library	Search Result
IEEE Xplore	157
ACM Digital Library	48
Springer Link	641
Wiley	679
Total Studies	1525
Excluded Studies	1493
Final Set	32

Table 5: Study Search Result by Phase.

Phase	Description	Included	Excluded
Phase 1	Search Studies	1525	-
Phase 2	Preliminary Selection	82	1443
Phase 3	Introduction and Conclusion	40	42
Phase 4	Full Reading	32	8

21.9%, while Workshop papers correspond to only 9.4%. One deserves attention, the *Working Conference on Mining Software Repositories* [11, 17], with two papers from our final set. As for Journals, the *IEEE Access* is the only one with more than one paper [1, 41].

Different venues have different publication requirements. Papers published on workshops are often early-state studies, with partial and/or initial findings. Conference papers, on the other hand, usually report studies with limited or summarised results. As for Journal papers, they present complete and more detailed research, which often infer to stronger results. Thus, since most studies are Conference papers, we can infer that, in this field of study, solutions are mostly limited and there is room to reduce threats regarding conclusion and external validity.

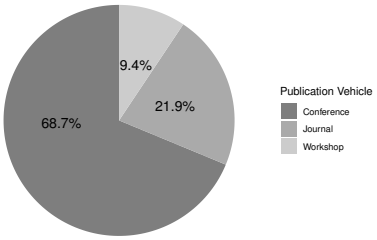

Figure 1: Publication vehicles.

3.2.2 Publication per Country. Figure 2 shows the regional context where the researches performed. For papers whose authors are from different countries, we considered the majority and authors position, respectively. The US is, by far, the leading country in energy efficiency testing in Android research. Researchers from the US have published 11 papers in the area, which is more than twice the second place, China and Canada, with four papers each. After that, comes the Republic of Singapore with three studies. Researchers from several other countries also published papers in the topic (Switzerland, Republic of Korea, Portugal, Pakistan, Korea, Japan, Germany, England, Brazil, and Algeria). We can then conclude that, though stronger in the US, research on this topic is not restricted to a single region.

3.2.3 Publication Year. Although we did not apply any time filter in our search, the papers from our final set were all published in a recent time frame (2011 to 2019). This was expected since Android 1 was released in September of 2008. However, it also shows that

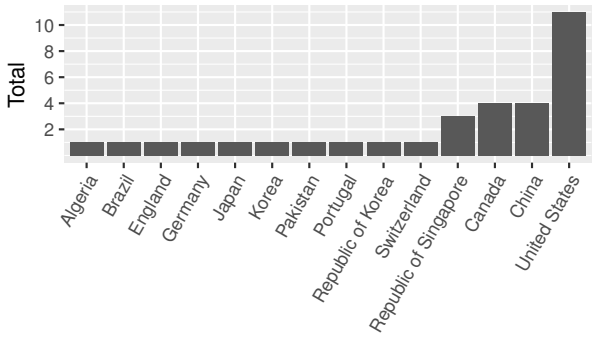

Figure 2: Publication per Country.

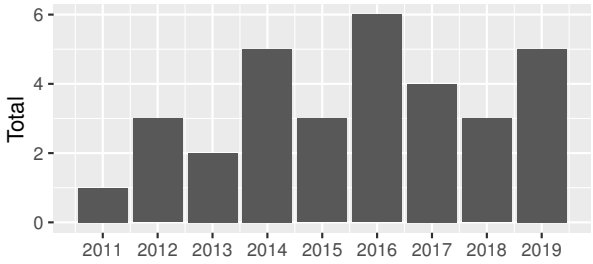

Figure 3: Publications per year.

concerns about energy efficiency in Android applications emerged since its early days. Figure 3 shows the distribution of the papers throughout the years. The number of studies published in the area has grown over the years and reached its peak in 2016, when 6 articles were published. Since 2014, at least three new articles have been published in the area of energy efficiency testing, indicating that the topic is still relevant.

3.3 What are the most common research topics on energy efficiency testing in Android applications and how they changed over time?

To answer this question, we grouped the studies from our final set according to the classification scheme defined in Section 2.4. Most papers were categorized as one of the five topics described in Table 3. 18.8% of them were related to *Test Generation*, 6.2% to *Resource Management Inefficiency*, 28% to *Fine-grained Estimation*, 9.4% to *Simulated Runtime Measurement*, and 18.8% related to *Classification*. Finally, 18.8% of the studies could not be grouped and were placed in the *General* category. Table 6 summary of papers by topics (categories), while Figure 4 shows the distribution of each topic over time.

Table 6: Study Search Results by Category

Category	Studies	Total
Test Generation	[4] [5] [18] [22] [39] [40]	6
Resource Management Inefficiency	[19] [20]	2
Fine-grained Estimation	[6] [7] [12] [14] [16] [21] [29] [34] [36]	9
Simulated Runtime Measurement	[8] [26] [35]	3
Classification	[2] [9] [10] [13] [31] [41]	6
General	[1] [11] [17] [24] [25] [33]	6

By analyzing the topics distribution over time, we can see that the first studies in the energy efficiency of Android applications focused on the *Fine-grained Estimation* since it is important to quickly find energy faults location in the source code. However, in the last two years, no study on this topic was found. The first studies focused on *Fine-grained Estimation* needed to perform tests on actual Android

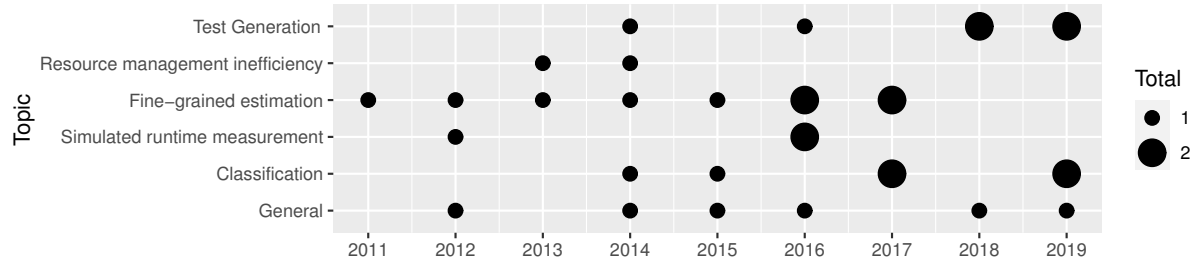

Figure 4: Topics Distribution Over Time.

devices, which is expensive and time-consuming. Thus, studies on *Simulated Runtime Measurement* started to appear right after fine-grained ones, which makes sense, since simulated execution allows developers to test applications without the need of an actual device that constrains the test activity to a device configuration (e.g., hardware components and a specific Android version).

Studies on *Resource Management Inefficiency* have been published only in 2013 and 2014. Since then, only the work of Zhang et al. [39] aimed at improving resource management. However, since its main focus was the test generation algorithm, we classified it as *Test Generation*. Energy-aware techniques for *Test Generation* and *Classification* works have been published since 2014. Moreover, studies in the diverse *General* category have been published since 2012.

3.4 What are the proposed solutions and their availability?

Most of the studies present their solutions as a tool or framework. The solutions proposed vary from machine learning training data, to tools/frameworks with different goals. However, most of them were not available. Only ten studies made their solution available (two as free to download [13, 18], and eight as open-source projects [4, 6–8, 11, 17, 26, 40]). On the other hand, 22 papers [1, 2, 5, 9, 10, 12, 14, 16, 19–22, 24, 25, 29, 31, 33–36, 39, 41] (68.8%) did not present information regarding their solutions availability.

Though some papers provide detailed information on how to implement their solution, it is always recommended for a software engineering work to provide a tool, or prototype, to be used/tested by developers in practice. Moreover, since no solution is available, other research cannot use it for comparison or empirical studies. More details on solutions and availability can be found in our data set.

3.5 How were the proposed solutions evaluated?

To evaluate their solution, the studies relied on either an empirical investigation or case study. Figure 5.a show the distribution of the evaluation method used. 27 papers [1, 4–11, 13, 14, 16, 18–22, 24, 26, 29, 33–36, 39–41] (84.4%) applied some sort of empirical investigation (e.g., controlled experiment), which gives more confidence about their conclusions since variables and noise are controlled. The other five papers [2, 12, 17, 25, 31] (15.6%) relied on case studies for evaluation. Through providing a more realistic scenario, one may raise questions due to the lack of control.

3.6 Do the solutions combine testing with other approaches?

Energy efficiency testing can be done alone (dynamic solution) or combined with static analysis through (hybrid solution). Figure 5.b shows the frequency of each in the papers from our final set. Most studies focused on dynamic execution of the application under test. From the 32 papers in our final data set, 22 papers (68.8%) used only dynamic verification to analyse energy efficiency [1, 2, 5, 8, 9, 11, 13, 16–20, 22, 24–26, 29, 35, 36, 36, 39, 41]. For instance, the solution proposed by Abbasi et al. [1] finds application tail energy bugs by analysing the execution log of the application under analysis.

On the other hand, 10 papers (31.2%) [4, 7, 10, 12, 14, 21, 31, 33, 34, 40] adopted a hybrid approach, combining static and dynamic analysis. For instance, the solution proposed by Banerjee et al. [4] uses static analyses to find potential faults, then it confirm the found faults through dynamic testing.

3.7 Do the studies used Hardware or Software based solutions to estimate energy consumption?

To test the energy efficiency in Android applications, often it is necessary to estimate how much power the application consumes. Thus, to get more accurate information, most researchers relied on physical power meters, and/or software-based solutions. Figure 5.c shows the frequency of papers that used each alternative.

From the 35 papers in our set, only one [17] depended only on *Hardware* to measure power usage. Physical power meters can measure consumption with very high accuracy, however it often is described as impractical by developers. The approach proposed by Hindle et al. [17] is a DIY framework developed aiming to reduce equipment costs.

On the other hand, 18 studies [8–13, 16, 19, 24–26, 29, 33–36, 40, 41] relied solely on *Software* solutions. Software-based approaches allow consumption estimation without the need for a physical power meter. Some of the approaches (e.g., [33, 40]) calculate energy consumption using components power profiles. Others uses machine learning to estimate energy consumption based on previous executions (e.g., [9, 41]).

Three studies [2, 14, 21] uses *Both*, by applying a combination of hardware and software to estimate energy consumption. Since hardware meters measures the whole device consumption, software solutions can be used to give more details about the power used. For instance, Li et al. [21] propose an approach where a physical power meter measures the device's power consumption, and a software

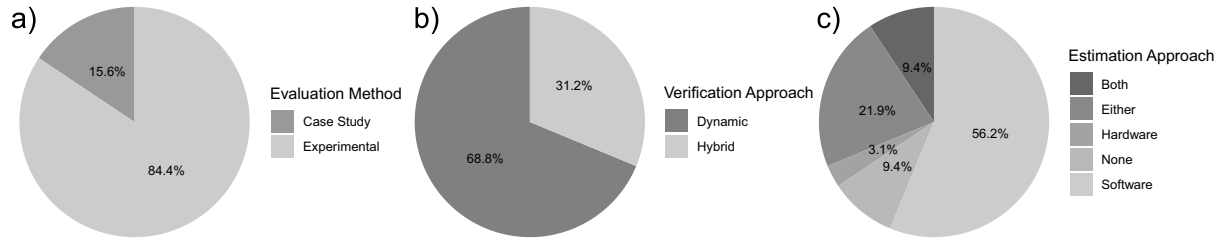

Figure 5: Evaluation Methods, Verification Approach, and Estimation Approach.

uses machine learning to get the energy consumption at source code line level.

Seven approaches [4, 5, 7, 18, 22, 31, 39] need the energy estimation of the application under test, but the user can choose *Either* software or hardware solution. For instance, [39] propose a framework for automatic test generation. The tests then needs to be run to measure the energy consumption during its execution, but it is up to the developer to setup a measuring environment.

Finally, three approaches [1, 6, 20] do not need to estimate energy consumption to detect energy inefficiencies (i.e. *None*). These solution can find energy faults by identifying certain patterns previously defined. Such patterns can be identified during dynamic testing [6, 20], or through the analysis of execution logs after the application execution [1].

4 DISCUSSIONS

In this section, we discuss, for each category/topic, the studies and the data presented in Section 3.

4.1 Test Generation

Evaluating energy efficiency in Android applications is not an easy task. To identify possible inefficiency faults, one may directly measure power usage or run tests. Regarding testing, a test suite needs to have good coverage in order to find abnormal energy usage, since it is necessary to exercise the faulty code in order to expose the fault. For instance, if an application includes an energy fault in a module responsible for an specific functionality, it is necessary to explore this functionality to reveal the fault. However, unlike functional testing, where code coverage is often an indicator testing quality, in energy testing, it does not tell us much about a suite's effectiveness. Instead, API and context coverage are better indicators, since they indicate how many of the energy greedy resources are used. Thus, energy-aware test generation tools are important since the generated tests focus on covering energetic behavior.

Zhang et al. [39] propose an approach to automatically generate tests that detect resource leaking faults responsible for unnecessary power usage. It is based on neutral sequences of GUI events. A neutral sequence of GUI events is a sequence of events in which it is expected that resource usage remains the same, without any resource growth after multiple repetitions. Zhang et al. [40] also propose a framework for generating tests for watch faces for Android Wear. The framework first applies static analysis to find potential inefficiencies based on several energy-inefficiency patterns of watch face behavior. The potential inefficiencies are used as input for test generation.

Some studies also propose frameworks for generating energy tests considering the whole application. Banerjee et al. [5] and Liu et al. [22] propose frameworks for automatic test generation. The test generation process takes into account both the control flow and window transition graphs. The generated tests are sequences of virtual user interactions (e.g., a screen touch) that are likely to lead to energy faults or energy hot spots (abnormal consumption). Jabbarvand et al.[18] propose a framework for test generation that uses a set of models representing the application's functional behavior, and the contextual conditions impacting the application's power behavior. Moreover, the approach relies on an evolutionary algorithm to evolve the generated tests.

Banerjee et al. [4] propose a framework that generates energy tests and repairs energy faults in Android applications. First, it uses static analysis to search for energy faulty paths. Then, it runs the program paths using symbolic execution. If a fault is reported, it generates repair expressions for validated energy faults.

4.2 Resource management inefficiency

Although Android devices provide limited batteries, they often include several resources that are energy greedy. These resources are hardware components available in the device (e.g., GPS, Bluetooth), and software features provided by the Android operating (e.g., wake locks). The bad management of these resources may have a huge impact on battery lifetime. Thus, studies conducted by Kamiyama [19] and Kim et al. [20] focus on identifying improper uses of such resources.

Wake locks are mechanisms provided by the Android system that prevents the device to enter into a sleeping state, which enables the application to promptly execute. The downside of this mechanism is that, when not properly used, it leads the application to use more energy than it should. For instance, when no critical task is been executed, the device reduces the processor's usage (e.g., its frequency) and, consequently, the energy consumption. However, a misused wake lock can prevent the device to enter into this energy-saving state. In this sense, Kim et al. [20] propose an approach to detect wake lock anomalies at runtime. For that, their approach continually tracks wake lock acquisitions and releases, and, if an application stops without releasing the wake lock, it points out an issue.

Kamiyama et al. [19] propose an easy-to-use testing tool that evaluates the energy-efficiency of an application by analyzing the relationship between hardware components behavior and power usage. For that, it runs training programs on a device to collect data, and runs multiple regression analyses to generate a power model that can estimate consumption without an actual power meter.

4.3 Fine-grained Estimation

Over the years, developers have become more aware of the impact of energy efficiency in Android applications, mainly because battery usage is a major issue for users. Therefore, different approaches focus on measuring power usage. However, knowing how much energy an application consumes is often not enough. To improve power usage, it is necessary to link code elements and consumption. Therefore, there is a need for a finer-grained consume estimation.

Pathak et al. [29] propose an approach for fine-grained estimation that considers both component use and low-level power optimizations programmed in drivers. This approach consists of a system-call-based power modeling that encompasses utilization-based and non-utilization-based energy behavior. It uses a finite state machine (FSM) to represent system-call power modeling, and a suite to specify the FSM transition rules. First, physical power meter is needed for generating training data. After that, it can be used without this external device.

Hao et al. [16] propose an approach for estimating energy consumption at code level by profiling the application's bytecode. For that, it runs a test case suite to generate the execution trace of the application under analysis. Then, it analyses the execution trace with the CPU's energy cost function (which is provided in the CPU profile) to estimate consumption. The down-side of this approach is that it analyses only CPU consumption, besides requiring the CPU's profile with energy cost functions for each instruction type.

The approaches proposed by Li et al. [21] and by Ferrari [14] take the whole device into account. However, to get acceptable accuracy, both solutions need physical power meters. They require the application to be instrumented for capturing time stamps and/or execution traces. Then, the instrumented application is run on the measurement platform to record power samples. Moreover, since the approach proposed by Li et al.[21] aims at estimating energy consumption at code level, it uses statistic analysis to adjust high energy samples not related to the application (e.g., thread switch) and to calculate energy consumption per code line.

The approach proposed by Westfield et al. [36] estimates energy consumption at method level without a power meter. In fact, it runs on emulators. This approach can also provide hardware power usage to improve energy analysis.

To find unusual power usage, Couto et al. [12] propose an approach that applies statistical analysis based on fault-localization techniques. It also relates power consumption to the source code. For that, it uses an algorithm to dynamically calibrate a power model to any Android device.

Instead of analyzing a specific application, the approach proposed by Wang et al. [34] aims at analyzing the energy consumption of the entire device. First, it performs static analysis to instrument each application, then, it requires the device to be connected to a computer through a USB cable for collecting resources usage data. Finally, it uses an online power model to estimate energy consumption. The main feature of this approach is its output visualization. The user interface provides an instant energy curve where the tester can choose a specific aspect to analyze, different components (e.g., CPU, screen), or any of the listed applications/processes.

Duo to insufficient and/or unreliable information, debugging failures can be a challenging. To address this issue, Banerjee et al. [6] propose a framework that reports detailed field debugging logs from user applications. The framework is based on two automated tools, on the user side, an instrumentation utility logs energy inefficiencies using fault patterns considering fault models [5] and reports them; on the developer side, the reported faults are automatically localized and patch locations are suggested.

To reduce the time and efforts needed to get a power usage estimation, Bangash et al. [7] propose a methodology for relating software energy consumption to software structural metrics. The goal is to no longer need exhaustive tests to measure the energy effects of every change. First, it extracts structural properties of Android applications from source code, and it measures the energy consumption of use-cases through test executions. Then, it applies Spearman $p-rank$ [37] to correlate the structural information with energy consumption.

4.4 Simulated Runtime Measurement

Estimating energy usage in Android applications demands time and resources. It is necessary to have an Android device and instrument the application. Moreover, to properly test changes made in the software, it is necessary to reinstall the whole application. Depending on the component under test, the effort needed can be ever worst. For instance, for testing geolocation components one might need to walk long distances. To cope with this problem, Bareth [8] proposes an environment to test energy efficiency of location services through simulation. The approach is based on an Android emulator. The simulator provides the location data for the application while logs the process, then it uses an energy consumption model, device energy profiles, and logs to compute statistics.

Wang et al. [35] and Ju et al. [26] propose solutions for full-system simulation. Both solutions simulate the runtime behavior of the application under test. Wang et al.'s solution has as goal to automatically monitor sensory data and wake lock usage. On the other hand, Ju et al.'s solution [26], even though provides full-system simulation, it focuses on network testing, including simulated server system connected to a simulated Ethernet. This configuration allows different networking simulations, such as network throughput, delays, and errors.

4.5 Classification

This category present classification approaches for different goals: to select the best test cases, to identify application's characteristics, to improve energy analysis for developers, to rank application based on its general consumption, or to help final users to better manage their batteries.

Executing energy test is an expensive labor-intensive task, which affects negatively the application evolution. Thus, to reduce the cost and effort needed to execute such tests, Sahin et al. [31] propose an approach to select tests based on the code changes. For that, it analyses the impact of the changes on energy greedy APIs. Based on the test coverage provided, it determines the tests that should be executed.

Abousaleh et al.[2] propose an approach to determine the in-app mode power profile from recorded energy measurements. It combines well-known signal processing approaches to analyze recorded data, such as moving average filtering, exemplar template

generation, and matched filtering. On the other hand, Beghoura et al. [9] aim at determining whether an application can be classified as "Green Software". For that, the authors introduce green efficiency as software quality, defining a set of green requirements. They also built a tool to evaluate the green efficiency of an application.

Zhu et al. [41] propose a system-call power modeling approach to estimate the energy consumption of each valid commit in a given versioning history. The authors used cross-validation to select and rank the importance of different system calls. Then, they evaluate seven machine learning models and select linear regression with the Lasso regularization [32].

Application repositories often provide a series of high-level information about the applications, such as size, description, and user evaluation. However, they lack energy-consumption information. Therefore, Behrouz et al. [10] propose an approach that ranks applications based on their energy consumption. The approach consists of using automatically generated tests to estimate general energy consumption and static analysis for paths not covered by tests. Finally, it gives a score to the application, which is used for ranking purposes.

Dao et al. [13] propose an approach to identify energy-hungry applications according to their power usage. The approach collects resource usage, such as screen, network, and CPU usage in the given time. Then, it classifies high-energy applications from this data through a three phases process: categorize applications, handle co-existing applications, and dealing with multi-modal applications.

4.6 General

Duo to the variety of components that composes Android devices, there are many ways to evaluate energy efficiency and find energy faults. For instance, Metri et al. [25] propose an approach where a developer can have an overview of their applications' energy efficiency. Their tool runs on a host computer and accesses the platform under test through a USB connection. Then, it collects several processor information, such as sleeping states, power state, tick events, among others.

Hindle et al., [17] propose an approach that enables systematic power measurement through physical instrumentation. The approach uses a Raspberry Pi for managing tests on Android devices. The Raspberry Pi is responsible for uploading and collecting data from the device and executing the tests. An Arduino with an INA219 chip is used to measure power consumption. All components, including the phone, are powered by an external power supply. The approach also provides a web interface to manage the test suite and visualizing measurements.

The display is known as an energy-greedy component. Therefore, improving its energy consumption often inputs a positive impact on battery life. In this sense, Wan et al. [33] propose an approach that enables the detection of display energy hotspots related to user interfaces. It uses a color transformation technique to transform user interface to its energy-efficient version, then it compares both versions using a cost function specific for each Android device. The difference is due to the OLED display, whose consumption is based on the colors displayed.

Even though energy efficiency is an important non-functional requirement, relying on physical power meters is not always an option. Therefore, developers often work only with statistical power models. For a power model to work properly it needs to be trained with real data. However, as power meters are not available, there is no way to get real energy consumption data to train these models. To address this issue, Chowdhury et al. [11] provide an oracle that enables energy estimation without previous measurements. The oracle contains resource usage patterns for a large number of versions of different applications and can be used to estimate the power usage of other applications.

During the application's execution, it is expected for the device to consume more energy than before and return to its previous consumption level. However, the energy consumption in the post-execution state is often higher than the pre-execution state. Abbasi et al. [1] define this characteristic as application tail energy bugs (ATEBs) and discuss causes. Moreover, The authors propose a tool able to detect ATEBs without the need for a physical power meter by comparing kernel log information from before and after the application's execution. On the down side, the tool only considers services and wake locks.

Most solutions proposed aim at evaluating the energy consumption of an application after implementing it, this also goes to applications in a mobile cloud computing (MCC) environment. However, applications running in such environments must consider issues, such as internet connection problems. Bad decisions during the design phase can have a significant impact on application development. Thus, Mendonça et al. [24] propose a model-based approach to estimate energy consumption and performance of mobile applications in a mobile cloud computing environment during early development stages. The approach uses DSPN models [23] to represent the MCC infrastructure and mobile application.

5 OPEN PROBLEMS

The solutions previously discussed address important issues. However, we envision a series of open problems that still need to be solved. Regarding *Test Generation*, we find the need for an energy-aware test prioritization strategy. Generated test suites often include a great number of tests, which may be infeasible to run [39]. By reorganizing the suites according to energy-aware heuristics, developers my run a smaller set of test cases and still find energy faults. Other problems in this category are related to model different contexts when generating tests (e.g., services), and reduce the time for test generation.

For the *Resource Management Inefficiency* category, we can list open problems that are worth mentioning. Kim et al. [20] suggest the development of a hybrid solution able to pinpoint the cause of wake locks anomalies by combining their approach with static analysis. Pinpointing the wake locks anomalies at source-level can reduce the effort of fixing the bugs.

Concerning *Fine-grained Estimation*, when measuring energy consumption one must instrument the application under test. However, to improve accuracy, it is required to increase the number of logging points in the application, which increases the tracing overhead. Therefore, there is a need for a solution to improve accuracy while reducing tracing overhead. Moreover, profiling tools estimate energy consumption for a very limited number of hardware

components. Thus, it is important to analyze and profile different components in order to have more accurate estimations.

As for *Simulated Runtime Measurement* topic, the current approaches do not support several new Android features (e.g., concurrent executions). Thus, simulation tools need to keep evolving. Another problem to be investigated is how to perform time-lapse simulations, which consist of performing tests as if time was faster than in the real world. For instance, to simulate a GPS tracking of a two-hour hiking in a few minutes.

For *Classification*, most approaches rank the applications under analysis based on their general consumption. However, user's typical use can impact the amount of energy an application consumes. For instance, a user might not frequently use the most energy-expensive features of an application. Therefore, by profiling the user's typical use, we could better classify the applications.

Finally, in the *General* category, Abbasi et al. [1] point the need for automatic support for identifying tail energy bugs for other components, such as listeners, audio, and wireless services. Moreover, to help developers deciding what ATEBs have higher fixing priority, it is necessary to develop a power model to estimate the energy cost of different types of ATEBs.

6 RELATED WORK

With the increasing demand for reliable Android applications, testing techniques needed to be revisited for this specific domain.

To identify state-of-art approaches, Méndez-Porras et al. [28] conducted a systematic mapping to characterize testing approaches for mobile applications. This study focuses on automated techniques and identified 83 studies, encompassing model-based testing, capture/replay, model-learning testing, systematic testing, fuzz testing, random testing, and scripted based testing.

However, since the systematic mapping conducted by Méndez-Porras et al. [28] was limited to test automation, many other studies that focused on other aspects were left out. To fill this gap, Zein et al. [38] conducted a more generic systematic mapping, including mobile testing related studies from different sub-areas. The work mapped 79 studies, divided into four main categories: test automation, usability, context-awareness, and security testing. Moreover, there were also studies mapped as *general* since they do not fit in any of these categories.

The works previously listed are essential to the field and provide substantial contributions. However, these studies focused only on functional and traditional non-functional testing, and do not consider energetic aspects. Since mobile devices are constrained by limited batteries, it is also important to test energy efficiency to ensure that there is no energy waste as shown by several studies [18] [17] [21] [29]. In this sense, we conducted a systematic mapping aiming to examine the extent, range, and nature of research activity in android applications energy testing in order to identify gaps in the existing literature.

7 THREATS TO VALIDITY

Here we list potential threats to our conclusions. First, the set of keywords defined for the initial search might not be representative of the domain. To mitigate this threat, we run several search trials to get the most representative keyword set.

The second threat is the selection of libraries to search for studies. However, to carry this study, we choose four well-known digital libraries namely IEEE Xplore, ACM Digital Library, Springer Link, and Wiley. These libraries are known as repositories for relevant academic papers and have been used as sources by other systematic mappings.

Finally, there is also the threat related to the evaluation and classification of the papers returned in the search, which is influenced by the knowledge and understanding of the authors. To mitigate this threat, we define inclusion and exclusion criteria used to guide the paper selection. Moreover, the selected studies and their classification was validated by all three authors.

8 CONCLUSION

In this paper, we presented a systematic mapping study of the literature to obtain knowledge about the current state-of-art scenario regarding energy efficiency testing in Android applications. Our study focused on works that analyze approaches, tools, and frameworks for testing Android applications aiming at identifying energy faults, and measuring energy consumption.

Our search returned 1525 studies from four of the main digital libraries. After applying a filtering process, 32 studies were selected and mapped to our classification schema. The studies were categorized as *Test Generation*, *Resource Management Inefficiency*, *Fine-grained Estimation*, *Simulated Runtime Measurement*, *Classification* and *General*. The most researched topics were *Fine-grained Estimation* with nine studies, followed by *Test Generation* and *Classification*, both with six studies. We also found that most approaches apply only dynamic testing and use software-based solutions to estimate energy consumption. Finally, we discussed a series of open issues that should guide future research in this field.

ACKNOWLEDGMENTS

This research was partially supported by a cooperation between UFCG and Ingenico do Brasil LTDA, stimulated by the Brazilian Informatics Law n. 8.248, 1991. Second and third authors are supported by National Council for Scientific and Technological Development (CNPq)/Brazil (processes 429250/2018-5 and 315057/2018-1). First author was supported by UFCG/CAPES.

REFERENCES

[1] A. M. Abbasi, M. Al-Tekreeti, K. Naik, A. Nayak, P. Srivastava, and M. Zaman. 2018. Characterization and Detection of Tail Energy Bugs in Smartphones. *IEEE Access* 6 (2018), 65098–65108.

[2] M. Abousaleh, D. Yarish, D. Arora, S. W. Neville, and T. E. Darcie. 2014. Determining Per-Mode Battery Usage within Non-trivial Mobile Device Apps. In *2014 IEEE 28th International Conference on Advanced Information Networking and Applications*. 202–209.

[3] Kitchenham BA and Stuart Charters. 2007. Guidelines for performing Systematic Literature Reviews in Software Engineering. 2 (01 2007).

[4] A. Banerjee, L. K. Chong, C. Ballabriga, and A. Roychoudhury. 2018. EnergyPatch: Repairing Resource Leaks to Improve Energy-Efficiency of Android Apps. *IEEE Transactions on Software Engineering* 44, 5 (May 2018), 470–490.

[5] Abhijeet Banerjee, Lee Kee Chong, Sudipta Chattopadhyay, and Abhik Roychoudhury. 2014. Detecting Energy Bugs and Hotspots in Mobile Apps. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering* (Hong Kong, China) *(FSE 2014)*. Association for Computing Machinery, New York, NY, USA, 588–598.

[6] Abhijeet Banerjee, Hai-Feng Guo, and Abhik Roychoudhury. 2016. Debugging Energy-Efficiency Related Field Failures in Mobile Apps. In *Proceedings of the International Conference on Mobile Software Engineering and Systems* (Austin,

Texas) *(MOBILESoft '16)*. Association for Computing Machinery, New York, NY, USA, 127–138.

[7] A. A. Bangash, H. Sahar, and M. O. Beg. 2017. A Methodology for Relating Software Structure with Energy Consumption. In *2017 IEEE 17th International Working Conference on Source Code Analysis and Manipulation (SCAM)*. 111–120.

[8] U. Bareth. 2012. Simulating Power Consumption of Location Tracking Algorithms to Improve Energy-Efficiency of Smartphones. In *2012 IEEE 36th Annual Computer Software and Applications Conference*. 613–622.

[9] Mohamed Amine Beghoura, Abdelhak Boubetra, and Abdallah Boukerram. 2017. Green software requirements and measurement: random decision forests-based software energy consumption profiling. *Requirements Engineering* 22, 1 (2017), 27–40.

[10] R. J. Behrouz, A. Sadeghi, J. Garcia, S. Malek, and P. Ammann. 2015. EcoDroid: An Approach for Energy-Based Ranking of Android Apps. In *2015 IEEE/ACM 4th International Workshop on Green and Sustainable Software*. 8–14.

[11] Shaiful Alam Chowdhury and Abram Hindle. 2016. Greenoracle: Estimating software energy consumption with energy measurement corpora. In *2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR)*. IEEE, 49–60.

[12] Marco Couto, Tiago Carção, Jácome Cunha, João Paulo Fernandes, and João Saraiva. 2014. Detecting Anomalous Energy Consumption in Android Applications. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 8771 LNCS. Springer Verlag, 77–91.

[13] T. A. Dao, I. Singh, H. V. Madhyastha, S. V. Krishnamurthy, G. Cao, and P. Mohapatra. 2017. TIDE: A User-Centric Tool for Identifying Energy Hungry Applications on Smartphones. *IEEE/ACM Transactions on Networking* 25, 3 (June 2017), 1459–1474.

[14] A. Ferrari, D. Gallucci, D. Puccinelli, and S. Giordano. 2015. Detecting energy leaks in Android app with POEM. In *2015 IEEE International Conference on Pervasive Computing and Communication Workshops (PerCom Workshops)*. 421–426.

[15] Gustavo Girardon, Victor Costa, Rodrigo Machado, Maicon Bernardino, Guilherme Legramante, Fábio Paulo Basso, Elder de Macedo Rodrigues, and Anibal Neto. 2020. Testing as a Service (TaaS): A Systematic Literature Map. In *Proceedings of the 35th Annual ACM Symposium on Applied Computing* (Brno, Czech Republic) *(SAC '20)*. Association for Computing Machinery, New York, NY, USA, 1989–1996.

[16] S. Hao, D. Li, W. G. J. Halfond, and R. Govindan. 2012. Estimating Android applications' CPU energy usage via bytecode profiling. In *2012 First International Workshop on Green and Sustainable Software (GREENS)*. 1–7.

[17] Abram Hindle, Alex Wilson, Kent Rasmussen, E. Jed Barlow, Joshua Charles Campbell, and Stephen Romansky. 2014. GreenMiner: A Hardware Based Mining Software Repositories Software Energy Consumption Framework. In *Proceedings of the 11th Working Conference on Mining Software Repositories* (Hyderabad, India) *(MSR 2014)*. Association for Computing Machinery, New York, NY, USA, 12–21.

[18] R. Jabbarvand, J. Lin, and S. Malek. 2019. Search-Based Energy Testing of Android. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. 1119–1130.

[19] T. Kamiyama, H. Inamura, and K. Ohta. 2014. A model-based energy profiler using online logging for Android applications. In *2014 Seventh International Conference on Mobile Computing and Ubiquitous Networking (ICMU)*. 7–13.

[20] Kwanghwan Kim and Hojung Cha. 2013. WakeScope: Runtime WakeLock Anomaly Management Scheme for Android Platform. In *Proceedings of the Eleventh ACM International Conference on Embedded Software* (Montreal, Quebec, Canada) *(EMSOFT '13)*. IEEE Press, Article 27, 10 pages.

[21] Ding Li, Shuai Hao, William G. J. Halfond, and Ramesh Govindan. 2013. Calculating Source Line Level Energy Information for Android Applications. In *Proceedings of the 2013 International Symposium on Software Testing and Analysis* (Lugano, Switzerland) *(ISSTA 2013)*. Association for Computing Machinery, New York, NY, USA, 78–89.

[22] A. Liu, J. Xu, W. Wang, J. Yu, and H. Gao. 2019. Automated Testing of Energy Hotspots and Defects for Android Applications. In *2019 IEEE International Conference on Energy Internet (ICEI)*. 374–379.

[23] Marco Ajmone Marsan and Giovanni Chiola. 1986. On Petri Nets with Deterministic and Exponentially Distributed Firing Times. In *Advances in Petri Nets 1987, Covers the 7th European Workshop on Applications and Theory of Petri Nets*. Springer-Verlag, Berlin, Heidelberg, 132–145.

[24] Júlio Mendonça, Ermeson Andrade, and Ricardo Lima. 2019. Assessing mobile applications performance and energy consumption through experiments and Stochastic models. *Computing* 101, 12 (2019), 1789–1811.

[25] G. Metri, A. Agrawal, R. Peri, M. Brockmeyer, and Weisong Shi. 2012. A simplistic way for power profiling of mobile devices. In *2012 International Conference on Energy Aware Computing*. 1–6.

[26] Minho Ju, Hyeonggyu Kim, and S. Kim. 2016. MofySim: A mobile full-system simulation framework for energy consumption and performance analysis. In *2016 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 245–254.

[27] Brunna C. Mourão, Leila Karita, and Ivan do Carmo Machado. 2018. Green and Sustainable Software Engineering - a Systematic Mapping Study. In *Proceedings of the 17th Brazilian Symposium on Software Quality* (Curitiba, Brazil) *(SBQS)*. Association for Computing Machinery, New York, NY, USA, 121–130. https://doi.org/10.1145/3275245.3275258

[28] Abel Méndez-Porras, Christian Quesada-López, and Marcelo Jenkins. 2015. Automated testing of mobile applications: A systematic map and review. *CIBSE 2015 - XVIII Ibero-American Conference on Software Engineering*, 195–208.

[29] Abhinav Pathak, Y. Charlie Hu, Ming Zhang, Paramvir Bahl, and Yi-Min Wang. 2011. Fine-Grained Power Modeling for Smartphones Using System Call Tracing. In *Proceedings of the Sixth Conference on Computer Systems* (Salzburg, Austria) *(EuroSys '11)*. Association for Computing Machinery, New York, NY, USA, 153–168.

[30] Kai Petersen, Robert Feldt, Shahid Mujtaba, and Michael Mattsson. 2008. Systematic Mapping Studies in Software Engineering. In *Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering* (Italy) *(EASE'08)*. BCS Learning & Development Ltd., Swindon, GBR, 68–77.

[31] Cagri Sahin, Lori Pollock, and James Clause. 2019. Supporting software evolution through feedback on executing/skipping energy tests for proposed source code changes. *Journal of Software: Evolution and Process* 31, 4 (2019), e2155. https://doi.org/10.1002/smr.2155 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/smr.2155 e2155 smr.2155.

[32] Robert Tibshirani. 1996. Regression Shrinkage and Selection via the Lasso. *Journal of the Royal Statistical Society. Series B (Methodological)* 58, 1 (1996), 267–288.

[33] M. Wan, Y. Jin, D. Li, and W. G. J. Halfond. 2015. Detecting Display Energy Hotspots in Android Apps. In *2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)*. 1–10.

[34] C. Wang, Y. Guo, P. Shen, and X. Chen. 2017. E-Spector: Online energy inspection for Android applications. In *2017 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*. 1–6.

[35] Jue Wang, Yepang Liu, Chang Xu, Xiaoxing Ma, and Jian Lu. 2016. E-GreenDroid: Effective Energy Inefficiency Analysis for Android Applications. In *Proceedings of the 8th Asia-Pacific Symposium on Internetware* (Beijing, China) *(Internetware '16)*. Association for Computing Machinery, New York, NY, USA, 71–80.

[36] Benjamin Westfield and Anandha Gopalan. 2016. Orka: A New Technique to Profile the Energy Usage of Android Applications. In *Proceedings of the 5th International Conference on Smart Cities and Green ICT Systems* (Rome, Italy) *(SMARTGREENS 2016)*. SCITEPRESS - Science and Technology Publications, Lda, Setubal, PRT, 213–224.

[37] Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, and Anders Wesslén. 2012. *Experimentation in software engineering*. Springer Science & Business Media.

[38] Samer Zein, Norsaremah Salleh, and John Grundy. 2016. A systematic mapping study of mobile application testing techniques. *Journal of Systems and Software* 117 (2016), 334 – 356.

[39] Hailong Zhang, Haowei Wu, and Atanas Rountev. 2016. Automated Test Generation for Detection of Leaks in Android Applications. In *Proceedings of the 11th International Workshop on Automation of Software Test* (Austin, Texas) *(AST '16)*. Association for Computing Machinery, New York, NY, USA, 64–70.

[40] Hailong Zhang, Haowei Wu, and Atanas Rountev. 2018. Detection of Energy Inefficiencies in Android Wear Watch Faces. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Lake Buena Vista, FL, USA) *(ESEC/FSE 2018)*. Association for Computing Machinery, New York, NY, USA, 691–702.

[41] C. Zhu, Z. Zhu, Y. Xie, W. Jiang, and G. Zhang. 2019. Evaluation of Machine Learning Approaches for Android Energy Bugs Detection With Revision Commits. *IEEE Access* 7 (2019), 85241–85252.