IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661,p-ISSN: 2278-8727, Volume 21, Issue 3, Ser. V (May - June 2019), PP 20-32
www.iosrjournals.org

Android Application Memory and Energy Performance: Systematic Literature Review

Abebaw Degu[1]
[1](*Department of Software Engineering, College of Computing/ Debre Berhan University, Ethiopia)*
Corresponding Author: Abebaw Degu

Abstract: *Android application is becoming critical in our daily lives by helping and enabling us to compute service intensive applications easily. These android applications run on smartphone mobile device which is relatively short of resources than computers. Android mobile applications require different approach for applications quality and efficient programming, analysis approach for better resource utilization such as memory and energy or battery, and for preventing and resolving resource leak behavior of apps. I have performed systematic literature review to categorize and to structure the research findings that has been published in the area of android application memory and energy performance, resource leaks, and performance testing techniques and challenges that they have reported. Thirty-one (31) empirical studies are mapped in to the classification scheme. Different research gaps that needs to be filled are identified and issues for practitioners are identified: there is a need to optimize memory and energy utilization; specific road-map or guideline to follow for application development, and studies on the issues of resource leaks due to various patterns, programming techniques, performance improvement, and source code analysis.*

Keywords: *Android, Performance, Resource leak, Energy, Memory*

Date of Submission: 10-06-2019 Date of acceptance: 26-06-2019

I. Introduction

Android is a Linux based operating system which is primarily designed for touch screen mobile devices such as smartphone devices, and tablets which are widely available on the market. Android is a powerful operating system that provides comfortable applications for users; and it is opensource which means it is free so that anyone can use and modify it.

Nowadays smartphones and tablet users and market needs are explosively increasing. Developers are trying to develop the application that satisfies customer's needs. Android platform app is currently having the largest market share and is expected to continue its explosive growth in the future [1]. But, most of the application are not usable to every phone as it consumes more energy and memory.

Our daily lives almost have become dependent on mobile devices such as smartphones and tablets. These devices enable us to use apps that provides us helpful and service intensive computation which makes our lives easy. Unfortunately, these devices, unlike computers are limited in terms of their battery power and memory capacity. Despite its limited resources, some programming practices contributes its part for inefficient use of our device's memory and energy.

To maximize the usability of the apps that are going to be developed, developers do not have sufficient guidance to the most useful practices and to improve the energy utilization of their implementations [2]. Even though, there is the advancement in hardware and battery technologies poorly designed apps are not stopped from needlessly draining the battery.

Mobile apps have been in the 1990s as they were created by Nokia and Motorola [3]. These apps have led to the development of various apps in different categories and hosted in different App stores. The apps from the most popular App stores, Google play and Apple store, have been downloaded more than 125 billion times, but performance issues are among the bottlenecks that affects app's rating [4]. The developers of these apps are currently facing the challenges of memory and energy as they receive complaints from users.

As far as I know, there are currently no available comprehensive systematic review studies in mobile and smartphone application performance testing and analysis regarding memory and energy. My systematic literature review provides a comprehensive and in-depth mapping study using a well-defined methodology to build anew classification scheme and structures the research area of mobile application performance analysis and testing. Additionally, my mapping study collects,interprets and analyzes all related evidence for empirical studiesaddressing challenges, approaches, methods or techniques of testing mobile and smart-phone applications.

This study also aims to highlight important research gaps inthe areas of mobile application performance analysis and testing regarding to memory and energy utilization. A total of 31 studies were selected for mysystematic study after going through three (3) filtration steps. I havepresented the synthesis of evidence based on FIVE (5) classification sub-categories: (i) performance testing, (ii) resource leak, (iii) memory, (iv) energy, and (v) general category. Severalresearch gaps are also reported and discussed. The remainder of this paper is organized as follows:Section 2 describes briefly the methodologyof my systematic literature review. Section 3 presents the results from themapping study followed by a discussion in Section 4. Finally,Section 5 concludes my work.

II. Methodology

The research methodology I have used is systematic literature review (SLR) which is based on the guidelines provided by Kitchenham et al., [5] and Petersen et al., [6]. This systematic review is inspired by the systematic review which is presented by Zein et al., [7]. The protocol that I have defined to undergo the review process is described in Figure 1.

- First, the research question that intrigues (fascinates) this SLR is identified and defined.
- Then, the different set of keywords that enables us to find the largest possible set of publication within the scope of the SLR is itemized.
- To limit my study to some very relevant publications, I applied exclusion criteria on the searched results, thus filter out the possible papers that are less likely relevant for my topic area.
- Then after, additional keywords are itemized from filtered papers, in this case, keywords enable us to categorize the possible studies on addressed problems contribution and focus areas of the researchers.
- Finally, the data is extracted and mapped to the appropriate sub-category of the study which is created for this purpose.

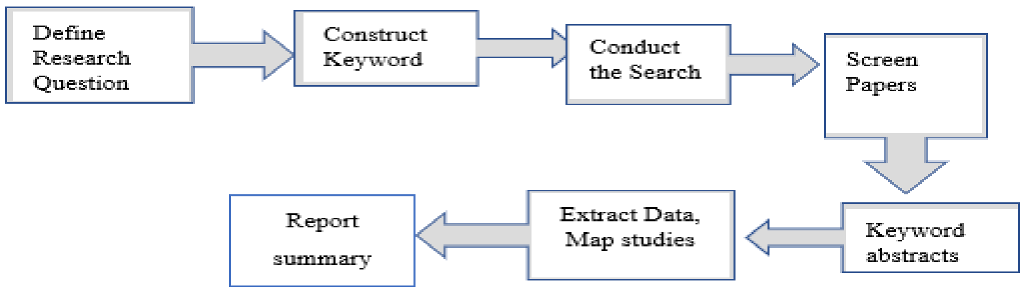

Fig.1: Systematic literature review process

2.1 Research Questions

The study emphasizes on identifying the research gaps, outstanding challenges, and to recommend where future study fits to extend current body of knowledge. Hence, I need to investigate the contribution of previous studies on android application performance testing specifically of memory, energy, and resource leaks reported to date.

The systematic literature reviewwill investigate the following research questions:

RQ1: what are the studies that empirically investigate android application performance enhancement techniques and programming challenges?

1.1. What are the research approaches do these studies applied and what are their contribution?

1.2. What kind of applications (Open source or commercial) do these studies used to evaluate their suggestions?

RQ2: How the analysis of the studies is designed and implemented?

- Through this question, the depth of analysis developed by researchers will be investigated thoroughly.

2.1 What are the fundamental techniques used by the researchers of energy and memory performance or issues of android applications?

RQ3: what are the challenges that doesn't seem to be satisfied and need to be addressed?

- With this requirement questions, I will survey that has an impact on memory and energy and have not yet benefited from significant research effort.

2.2 Search Datasets

My data search is based on the repositories, which help us to find relevant publications. To find data sets that are relevant to SLR, I first select four well-known electronic repositories; namely, ACM Digital Library, IEEE Xplore Digital Library, arXiv, and Science Direct.

2.3 Search Strategy

Studies need to be directly related to android application focusing on resource leak, performance enhancement, performance testing techniques, challenges, methods or approaches.

The strategy is adopted as suggested by kitchenham et al., [5]:

- Search for synonyms and alternate words.
- Use Boolean OR to associate alternate spellings and synonyms.
- Use Boolean AND to combine major terms together.

For each try, the search string is evaluated based on how the returned result is relevant for my focus area i.e. android application performance analysis or testing specifically for memory and battery consumption. Based on my initial research review I set 12 papers as a reference to evaluate the effectiveness of search strings to retrieve those studies. The search strings which retrieves the most relevant papers to my study and returned the maximum number of the previously known or selected studies will be chosen. The strings which retrieves less and less relevant to my study will be excluded from the list.

Table 1*:* Search strings applied on IEEEXplore

No of try	Search String	No of studies missed	Returned results
Try 1	((android) AND (application OR app) AND (memory) AND (performance OR “resource leak”) AND (test OR analysis))	4	28
Try 2	((“android app” OR “android application”) AND (memory) AND (performance OR “resource leak”))	9	74
Try 3	((android) AND (application OR app) AND (memory) AND (performance) AND (test OR analysis OR “automated test”))	10	44
Try 4	((android) AND (application OR app) AND (memory OR automated testing OR battery OR energy) AND (performance))	8	22
Try 5	((android) OR (performance) OR (“resource leak”) OR (memory) OR (energy))	0	691

On the other hand, several studies published using the term “app” instead of “application”. Hence, I incorporated it in my search string. In addition to this, the term “energy” is used in other fields and which is generic that I need to be specific and strict for android application battery usage. This needs to be done carefully in order not to struggle with the bulkiness of the results retrieved.

2.4 Study Selection Criteria

Studies that considered are which is supported by empirical evaluation or data i.e. if a research paper proposes a new method, testing approach or programming tips, it should show some sort of evidence that supports the proposed approach for its effectiveness.

In my systematic literature review the following inclusion criteria are used:

- Studies must be related to android application performance testing specifically battery and memory utilization or enhancement, memory and energy or battery challenges, and limitations.
- Studies must be supported with empirical data.

To exclude the irrelevant papers the following exclusion criteria are used:

- Non-English publications are filtered and excluded.
- Because my search terms include “performance” and “energy” to collect most papers, the collected set includes papers about “hardware device performance”, “android OS performance”, and other fields energy, I exclude such non-android application related papers.
- Papers that present the opinion without any supporting empirical evidence.

2.5 Study Selection Process

The search process is applied on all databases that has been previously identified. The initial year of publication were specified to 2012 during search, and restricted to studies related to computer science. The search for literature is covered up until 2017.

The paper selection process was iterative and incremental. Hence, it passes through different phases. Initially, the database searching using search string. then after, in the first phase, the resulting paper will be filtered based on their title and abstract. In this step, papers that are not related to android application performance testing and enhancement are excluded from the list.

In the second phase, papers are filtered by applying selection criteria through reading introduction, methodology, and conclusion. In this step papers may be excluded because it is not empirical or did not confirm to study selection criteria. Final step involves complete and thorough reading of a paper.

2.6 Classification Scheme (abstract keywording)

The keywording process was inspired by Petersen et al., [6] and consisted of two phases which is applied to the final selected papers.

Firstly, the researcher reads abstract of selected papers and searches for keywords that shows the findings or results of the study. When abstracts are written with poor quality or too short for choosing significant keywords, researcher can read introduction and conclusion as well.

Secondly, the set of keywords identified from different papers combined to understand the context and contribution of the paper. This helps researchers to identify sub-categories for classification scheme.

III. Results

3.1 Search Results:

It was clear that searching for and retrieving empirical studies in mobile and smartphone applications memory and energy performance or resource leaks needs careful search string construction. The term energy is used in various disciplines. That is why there was the inclusion of android to make it specific and lots of trial in search of appropriate search string.

Remaining papers for systematic literature review after passing through each filtration steps are described in the following table:

Table 2: Papers after each filtration steps

Repositories	Initial Search Result	Phase 1	Phase 2
IEEEXplore	2,352	37	21
ACM	625	21	5
arXiv	15	6	3
ScienceDirect	234	8	2
Total	3,226	72	31

Initially the search results returned 3,226 papers from all sources. After doing so, the three filtration steps which is stated below is applied.

- Initially, apply advanced command search
- Then, filter papers based on title and abstracts
- Finally, apply selection criteria by reading introduction, method and conclusions of the study.

Totally, 31 papers were included after applying filtration steps and inclusion/exclusion criteria. Out of 31 studies, 21 (67.74%) came from IEEEXplore, 5 studies (16.13%) came from ACM Digital Library, 3 studies (9.67%) from arXiv, and 2 studies (6.45%) are from ScienceDirect. The distribution of the included papers per publication year is depicted with a graph below.

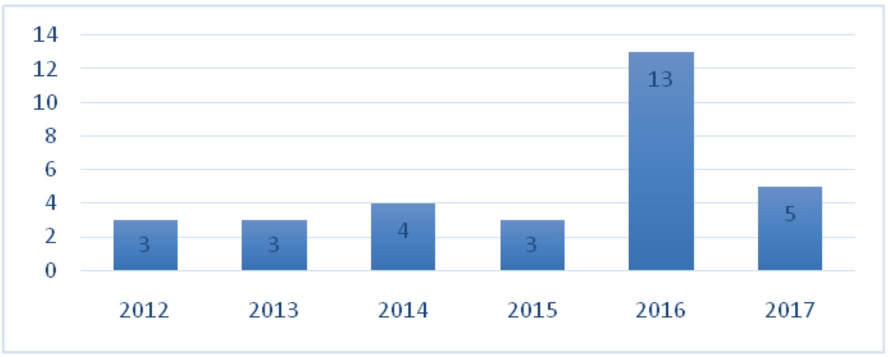

Fig.2: Papers per publication year

3.2 Classification Scheme

This abstract keywording (classification scheme) process consists of the following three main categories:

i. Nature (structure) of the topic or evidence provided
ii. Contribution of the study
iii. Objects involved in the study (e.g. type of applications used for evaluation i.e. opensource or commercial)

Using structure of the topic which is the first category and thematic analysis, I grouped the study into five sub-categories: classification sub-categories: (i) performance testing, (ii) resource leak, (iii) memory, (iv) energy, and (v) general category

The second classification which is contribution facet, classifies the studies in to contribution type namely Framework, Method, Tool, evaluation and metrics.

As indicated by A. Shahrokni et.al. [8] a contribution type framework can be defined as a detailed method that touches various aspects by focusing on many research questions and areas. On the other hand, a method deals

with specific goal and narrow purpose or research question. As the name indicates tool contribution is a category in which the study that provides tool as a solution is classified. Metrics in the other way measures relevant variables in memory and energy performance analysis or resource leaks of android applications. Lastly, the evaluation contribution is studies that evaluate the system or methods of specific approach.
The third classification category is “objects used or involved in the study” that indicates the answer for **sub_RQ1.2** and constitutes the type of applications used (commercial/opensource) for evaluation of proposed study solution. I was also motivated to see the extent of the studies and applications. Real world applications are trustworthy and reliable

3.3 Answers for Research Questions

RQ1: What are the studies that empirically investigate android application performance enhancement techniques and programming challenges?

The studies used in this systematic literature review is categorized and grouped based on the classification scheme stated in section 3.2. Those categories used to structure the topics are: *performance testing, resource leak*, *memory*, *energy*, and *performance in general/related.*

Studies challenge that are not related to any one of the first four categories are grouped into a general sub-category. Out of 31 studies, seven (7) studies are under memory, eleven (11) studies under energy, four (4) studies under resource leak, five (5) studies under performance testing, and four (4) studies are under general sub-category. Table 3 shows the included studies for each sub-category. From the statistics found in this sub-category, it is easy to notice that most of the studies are related to energy 35.48% (i.e. 11 out of 31).

Table 3:Included studies in each category

Performance Testing	Resource Leak	Memory	Energy	General Category
R4	**R13**	**R17**	**R24**	**R35**
R9	**R14**	**R18**	**R25**	**R36**
R10	**R15**	**R19**	**R26**	**R37**
R11	**R16**	**R20**	**R27**	**R38**
R12		**R21**	**R28**	
		R22	**R29**	
		R23	**R30**	
			R31	
			R32	
			R33	
			R34	

3.3.1 Performance Testing:

I found five studies that investigates related to performance testing of android applications (**R4, R9, R10, R11, R12**) where **R** stands for Reference.

According to Abogharaf et al. [9], in the context of smartphone applications, energy performance testing can be conducted for evaluation of energy consumption of an application on a single device or for comparing the energy consumption of different smartphone running the same application. The study indicates that energy performance testing focuses on specific parameters that controls the hardware (e.g. GPS) and most likely causes the energy bugs that mostly affect the energy consumption. To evaluate the energy performance of the applications, the study proposes a methodology to select test configurations and conduct an experiment.

Das et al. [10] investigates the extent to which developers handle the performance issues in their commits and documenting those issues. The study conducted quantitative and qualitative analysis and found that performance related issues found mainly in apps user interfaces. Other than this, bad code smells, network-related codes, and memory managements are predominant performance related commits found by the analysis. The study also proposes to use these categories as a checklist for developers while developing an app to see if they are considering all performance related issues.

On the other hand, the empirical study conducted by Hecht et al. [11] assesses the positive performance impacts of correcting android code smells. The study focuses specifically on the three android performance code smells namely; *Member Ignoring Method, Internal Getter/Setter, and Hash Map Usage*. The evaluation has been done using metrics such as *number of delayed frames, frame time, and number of garbage collection calls.* The study concludes that there is up to 12.4% on UI metrics and 3.6% on memory related metrics improvements when correcting *Member Ignoring Method* and abovementioned three android code smells respectively. The study also proposes to use this results as a guide while refactoring the code to improve the quality and performance of the app.

Linares-Vasquez et al. [4] states in their survey that performance bottlenecks are the major and a cause for frequent complains that affects app’s rating and chance of success. The study categorizes tools and practices used by application developers while addressing performance issues. The study concludes that developers depend on manual testing and analysis to detect performance bottleneck, prefers tools for profiling and debugging of their apps. In addition to this, the study suggests that for detecting performance bottlenecks there is a need of a tool that detects automatically, and developers need to be careful about quality attribute tradeoffs such as responsiveness vs memory management and usage of APIs.

Another empirical study by Linares-Vasquez et al. [12] investigates android application developers' practices towards their application performance improvements. The study tries to assess the use of micro-optimization and its impact on memory/CPU usage. Finally, it concludes that micro-optimization techniques are not common in open source android applications and developers are rarely uses it to improve the performance of their apps. The study indicates the reason for developers neglecting micro-optimization as follows: most android developers are not aware of micro-optimization, thinking as apps are not worthy of optimization, considering micro-optimization as a wastage of time, and believing optimization will not have an impact on the performance of the apps. On the other hand, the study argues that micro-optimization has a significant impact on performance of the apps. As a limitation the study doesn't address android specific GUI micro-optimization.

3.3.2 Resource Leak:

Resource leaks are caused due to improper management of resources [13] and missing of releasing operation [14] which results performance degradation, crashes, slowdowns, and negative user experience. I found 4 studies (**R13, R14, R15, R16**) that have stated and reported their evidence on resource leaks in android applications.

To automatically detect resource leaks in the byte code of android applications, Wu et al. [14] presents a static analysis tool called Relda2. The study focuses on the byte code of the apps and consists of three parts. These parts are: preprocess module, analysis module, and bug report modules to convert APK files in to DEX byte-code and construct FCG, for recording resource request and release operation, and to decide whether the app has a resource leak respectively. The study addresses scalability and precision tradeoffs using flow sensitive and flow insensitive analysis techniques. In the first techniques flow information are ignored so that apps can be analyzed quickly. Whereas the later technique is used in the study to eliminate false negatives.

An approach presented by Yan et al. [13] aims at testing resource leaks based on GUI model that specifically focuses on coverage criteria. The study is based on the neutral cycles in which the GUI events should not increase the resource consumption and should have neutral effect. The authors argue that resource leaks based on the model based systematic testing can successfully uncover various leaks in different applications. Finally, the study concludes this systematic testing is effective as well as feasible for identification of resource leak defects and suggests having analysis techniques that can detect methods related to resource allocation and reclamation automatically.

To quantify the strength and weakness, and to compare resource leak bugs in android applications effectively Liu et al. [15] develops an initial benchmark called DROIDLEAKS, which is a collection of bugs. For each bug, the following data are collected: name of infected app, leaked resource type, bug location, bug fixing patches, and bug report. The researchers identify the root cause of resource leaks and implements static checker to reveal common patterns of leaks and mistake made by developers. The study concludes that the checker used can effectively detect resource leak bugs and suggests for quantitatively compare the strength and weakness of resource leaks.

Apart from testing, identifying or detecting of resource leak bugs, fixing those bugs are labor intensive task for developers. To eliminate reported resource leaks, Liu et al. [16] proposed a light-weight static analysis approach and inserted patch code in to the byte code of apps. The static analysis of the study is based on Relda2 and fixing is done at the byte-code level. The authors argue the reason for choosing byte-code is that apps are released in the form of byte code instead of source code as well as byte code is easier to find. The study inserts additional few codes (low instrumentation) to release leaked resources in the app.

3.3.3 Memory Utilization:

Under this sub-category, I found 7 studies that have reported on the memory utilization of the android applications (**R17, R18, R19, R20, R21, R22, R23**).

The memory behavior of embedded devices in which the application is designed as multicore architecture and multithreaded nature of the application is typical for triggering garbage collection frequently. This performance bottleneck nature of memory paves the way for Chang et al. [17] to design a profiling tool. The authors indicate that their profiling tool can locate causes of garbage collection and garbage collection type. In addition to this it finds the location of the contention and which method or thread causes it.

To improve the application loading time Vimal et al. [18] proposes a memory management schemes using user's application usage pattern to decide on killing of apps, and setting the background cache limit.

Authors argue that if an app is frequently used, then number of kill will be lowered. The study concludes that using extensive caching helps to reduce the average free memory on the device.

Heap size is limited [18] [19] and dependent on the devices. The study conducted by Yoo et al. [19] studies about garbage collection performance and states that abnormal memory cancelation causes memory leakage. In addition to this, the author argues that garbage collection has a negative impact on performance level. The study suggests Dalvik VM as significant aspect for application performance.

On the other hand, Hamanaka et al. [20] proposes a methodology to choose GC implementation of applications based on application state and size. This method reduces process terminations triggered by *LowMemoryKiller.* The focus of the study is on LowMemoryKiller and ART GCs. The study method is all about reducing process memory and number of terminations. Finally, the authors show a method how to choose GC for applications with larger objects.

The study by Shahriar et al. [21] proposed fuzz testing approach to detect memory leaks by inputting huge amounts of data and making an application crash. Authors states the approach effectively discover memory leaks in android applications. The study also suggests considering extra memory leak patterns such as inner class, context, and database connection leaks. Implementing wrappers and changing resource types such as text and video files also suggested in the study.

Santhanakrishnan et al. [22] studies on detection of memory leaks using code patterns at an object level depend on object references. The study able to identify parts of the code which is responsible for the memory leak. Authors proposes to detect memory leaks due to multithreaded activities, large data, and event-based memory leaks.

Another study by Lim et al. [23] assesses to enhance memory performance by suggesting memory partitioning scheme targeting the avoidance of Out-of-Memory Killer and Low Memory Killer. The study provides the memory portioning framework at the operating system level of android. The presented approach prevents LMK and OOMK from killing application processes due to insufficient memory of the system. Beyond this, it focuses on page reclamation and non-page reclamation of non-critical and critical applications respectively.

3.3.4 Energy Utilization:

Based on data extraction, eleven (11) out of 31 studies were specifically related to energy consumption and utilization of android applications (**R24, R25, R26, R27, R28, R29, R30, R31, R32, R33, R34**).

Banerjee et al. [24] investigates and detects about the energy hotspots/bugs in android applications based on user interactions that may cause energy bugs/hotspots. A scenario of consuming abnormally high energy despite low hardware resource utilization is defined as energy hotspot while the scenario in android application that prevents smartphone devices of being idle after completing execution is outlined in the study as energy bugs. The study provides a systematic definition of energy hotspots/bugs and the exploration as well as detection in android applications. The source code in the studied framework is not used for detection purposes rather than to obtain code coverage metrics. Finally, the study suggests computing root cause of energy wastage and to investigate automated refactoring to reduce energy wastage of application.

Similarly, Li et al. [25] proposes a source-level generic energy optimization framework that considers hotspots in its source code optimization algorithm. The authors understand source code's energy feature and optimized the source code presented a framework by realizing energy features of the source code.

The empirical studies by Carette et al. [26] assesses the positive impact of code smell correction on energy consumption. For evaluation and correction of code smells on energy consumptions, the authors propose tooled approach called HOT-PEPPER. This tool can correct these android code smells: *Internal Getter/Setter, Member Ignoring Method, and HashMap Usage* as used by [11]. Not only these android code smells but also three picture smells (compression, bad picture format, and bitmap format) is investigated in the study. The study argues that the correction of the three code smells can reduce the energy consumption of apps by 4.83%. finally, the study suggests using non-intrusive method to collect energy metrics.

Oliviera et al. [27] on the other hand investigates development approach impacts (such as Java, JavaScript, and C++) on energy consumption in android applications. The study finding indicates that combining different approaches are significant for performance and energy improvement. It also indicates JavaScript as a most energy efficient approaches than java.

The study by Chen et al. [28] investigates programming languages, runtime, implementation choices, and compilers impact on energy consumption of android apps. As [27], this study assesses the impact of programming languages (C/C++/Java) on energy consumption. Runtimes such as ART Vs Dalvik, implementation choices (serial Vs parallel and recursion Vs iteration), and thin client design impacts on energy utilization efficiency is investigated. The study concludes native code is better in terms of energy efficiency and performance in Dalvik runtime whereas in ART native code and Java are comparable in terms of energy and performance efficiency.

Dutta et al. [29] proposes an approach that leverages data caching techniques to reduce energy consumption of mobile devices and implemented on android OS. The proposed method consists response caching and object caching approach. The study result indicates that object caching, on average can reduce energy usage by 45% whereas response caching can reduce up to 20% when compared to no-cache case.

To analyze the energy efficiency and calculate the energy usage by applications energy profiler can be used. Bakker [30] compares different available energy profilers that uses for optimizing application for their energy consumption.

Ferrari et al. [31] studies a way for developers to easily identify energy leaks in apps. The study uses POEM (Portable Open Source Energy Monitor), which enables developers to detect possible leaks by performing analysis on the call graphs, basic blocks, and android API calls.

A study by Banerjee et al. [32] provides a framework called EnergyPatch which uses dynamic and static analysis techniques to detect, validate, and repairing android application bugs. The framework consists three phases: detection, validation, and repair.

Xu et al. [33] proposes static analysis techniques called state-taint analysis in their study which considers resource protocols for resource bug detection. The analysis is implemented as prototype tool called state droid. The data for experimentation using the tool is obtained from test dataset from GreenDroid and Relda. The author indicates that the tool can detect energy leak patterns precisely. On the other hand, Saarinen et al. [34] studies the impacts of offloading towards the reduction of energy consumption of mobile devices.

3.3.5 General Category:

This section contains investigation and discussion of studies with contribution that do not have a focus in the categories of performance testing, resource leak, memory utilization, and energy utilization of android applications. I identified four (4) which can be classified under this category (**R35, R36, R37, R38**).

The power profiling techniques in which applications energy efficiency on android devices is explained by Metri et al. [35]. They also propose Soft Power Monitor (SoftPowerMon) which is helpful for application and platform developers to debug apps from energy perspective and develop efficient systems respectively. The study highlights the importance of SoftPowerMon as it answers why instead of how much power is consumed on a device.

The study by Hammad et al. [36] shows the way how smartphone mobile devices power consumption can be managed by killing unnecessary applications running in the background, and adjusting CPU frequency. The study uses task killer applications and setting CPU frequencies based on real load as method of investigation.

Das et al. [37] presents a way of performance enhancement of smartphone devices and reduce energy consumption through offloading computation intensive parts of apps to the server and executing the rest on smartphone devices. The study describes APPS (Accelerating Performance and Power Saving) which supports thread migration of android apps and class level offloading. It also supports dynamic byte code migration and runtime execution offloading of computation intensive applications. The study is implemented on N Queens problem and there is an improvement on performance.

A case study based report by Cheon[38] is to show best assumption of java programming may not always be good for android programming. As of the author, apart from other OS android devices are resource constrained. Hence, the study points that android programmers should write their code considering resources and performances in mind. The study suggests using profiling tools to gain insight, find, and fix problems within an application.

3.4 Research Approaches and Contributions

1.1. What are the research approaches do these studies applied and what are their contribution?

The contributions are inspired from [8] in which it is classified into methods, tools, evaluations, and frameworks. Metrics is a contribution which presents guidelines for different issues of applications performance testing. An evaluation provides assessment of a method by software applications or assessment of a software application using a method. Methods are the contribution that has specific goal and research question. Finally, a framework which is detailed method or techniques that has a broad purpose and addresses multiple research questions.

I have found four (4) studies presenting frameworks for enhancing performance of applications and detections of energy consumptions (R12, R17, R18, R25). In my systematic literature review, I found that most of the contributions (13 out of 31, 41.9%) were provided as methods. On the other hand, the contributions in terms of metrics and tools are (8 and 6 studies) respectively.

To summarize the research approaches, most of the research studies were conducted using a validation research approach (i.e. finding or testing the truth of something) which accounts (24 out of 31, 77.42%). On the other hand, the number of research studies with evaluation research is (7 out of 31, 22.58%) relatively small.

Based on this statistic, I can conclude that there is a need for more evaluation research that evaluates how memory and energy performance analysis and testing of these studies are effective by asserting or fixing the value or worth of it with the help of mathematical calculation and rating.

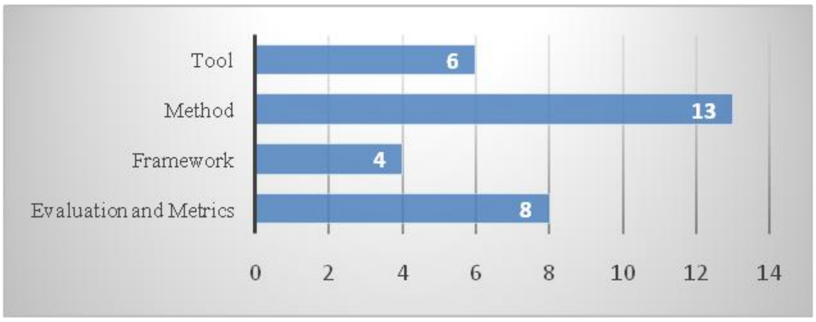

Fig.3:Contribution of the study

3.5 Objects involved in the study

1.2 What kind of applications (Open source or commercial) do these studies used to evaluate their suggestions?

All the studies under the performance testing and energy (5 and 11 respectively) had their evaluation done on real world or open source application. Whereas, studies under memory (6 out of 7) and general categories (3 out of 4) were conducted on real world applications. on the other hand, the studies under resource leak categories (4 out of 5) were implemented on real world apps while one of the studies in this category is conducted on both industrial and real world or opensource applications. One of the studies under memory category is conducted by creating a benchmark application instead of real world or industrial application.

3.6 Extent or Depth of the Study

RQ2: How the analysis of the studies is designed and implemented?

3.6.1 Depth/ Extent of the Analysis

I investigated how the analysis of the study is explained in the literature and how the study is implemented. Specifically, I analyzed targets of the study and analysis methods and techniques that they apply.

3.6.1.1 Distribution and Targets of Analysis used by Examined Studies

The targets of the study that developers are used when trying to solve memory, energy, resource leak, and performance of android applications in their approaches are listed in the following graph. In the study, source code is the most used targets for investigation in the study.

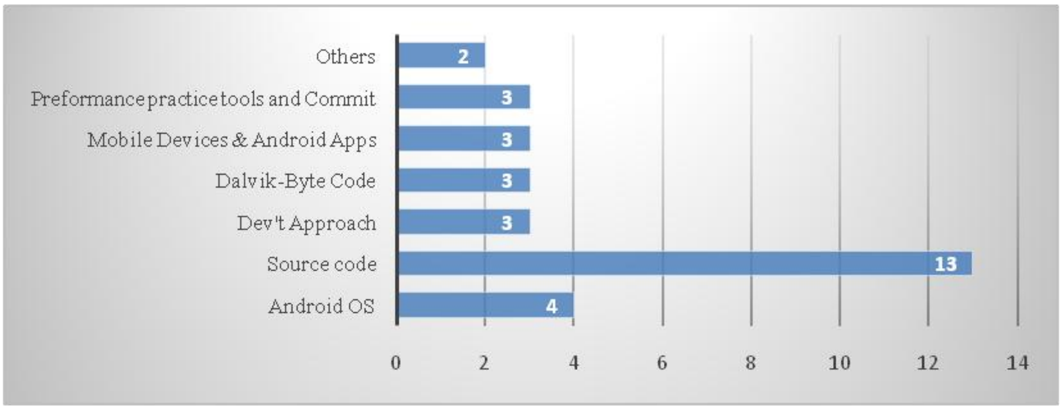

Fig.4: Distributions and targets of analysis used by examined studies

3.6.1.2 Fundamental Analysis Methods

In my study I have identified nine (9) fundamental techniques that are used, often in conjunction with one another, in the publications.

Abstract Interpretation: it is a theory of estimating/ approximating the semantics of the program in which the quality of the analysis is guaranteed and helps in avoiding false negative results. *Symbolic Execution:* it is an approach for generating possible program inputs, locating infeasible paths. Symbolic values can be taken as for producing possible inputs that fulfills all conditional branches inside the given path. It can also be taken as test cases to find the given path for repeatable dynamic analysis. If no input is produced, then it is said to be infeasible.

Taint Analysis: this approach consists static and dynamic analysis. It is a kind of information flow analysis where objects are tainted and tracked using data flow analysis.
Code Instrumentation: is used to address the challenges of static analysis in android applications. It also consists code tracing, debugging and structured exception handling, profiling, performance counters and logging or tracking of major events in the executions of apps.

Offloading: is a technique of migrating executions, processes, methods, threads and tasks to different machines.
Cache Management: it provides in-memory storage and management for your data. It helps to organize the data in data regions with configurable behavior.

Benchmarking: it is a way of creating sample applications for later reference or other things that helps to evaluate the newly suggested approaches performance or effectiveness as well as efficiency. *Code Optimization:* it is a technique which includes code refactoring, and other rearrangement techniques that helps in enhancing performance of android applications.

Survey: is a technique of looking in to previously done empirical studies and performing some statistical and numerical analysis on founded performance related commits and development approaches.

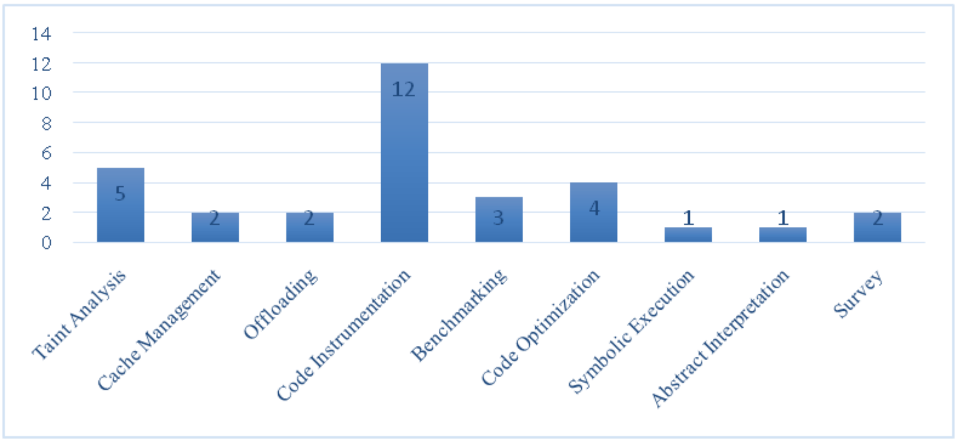

Fig.5: Fundamental analysis techniques used in the study

Looking at the summary of the analysis techniques in figure, code instrumentation is widely used and applied techniques in memory and energy performance analysis of android applications.

3.7 Suggested Areas for Future Studies

RQ3: what are the challenges that doesn't seem to be satisfied and need to be addressed?

In answering this research question, I investigated and present possible areas that needs further research based on the findings from my systematic literature review. Here, the suggested research gaps that needs further investigation and the study is categorized with the classification that I used in the study. The areas that are suggested for further investigation is based on my interpretation of the results or data found as well as the number of research available in each area.

- **Performance Testing:** under this category I have identified lit of issues that needs to be addressed in the future as follows:

i. Application contents impact towards test case selection
ii. Investigating various code smells and mobile apps as suggested by [11] [26] [10].
iii. Investigate the effect of performance changes in source code by deep analysis of source code.
iv. Produce a guideline that helps developers for detecting and performance optimization opportunities.
v. Empirically verify the benefits of optimization practices in android apps.
vi. Tradeoff between quality attributes (such as responsiveness Vs Memory management) need to be considered.

- **Resource Leak:** I found the suggested issues that needs further investigation in this category as follows:

i. Compare the strength and weakness of existing resource leak detection techniques quantitatively.
ii. Analysis of native memory and specific resources (such as Database cursors and Bitmaps).
iii. Automated/ static/ dynamic discovery/ analysis of code that allocates and reclaims important resources.
iv. Improving resource management through software abstractions and patterns.
v. Correlating repeated behavior with heap growth (better diagnosis techniques).

- **Memory Utilization:** list of issues that needs to be addressed are:

i. Generate leak reports in the test classes by installing or inserting it into the source code during development (for the detection of more classes of code patterns) such as (memory leak due to large data, multithreaded activities, and event-based memory leaks).
ii. Develop more memory leak patterns (e.g. context, inner class, and Database connection leaks)
iii. Convert leak patterns into test cases automatically to allow the detection of memory leak vulnerabilities early.
iv. For fuzzing [21] specifically in resource fuzzing, alter resource file types (such as video and text files) and in API fuzzing, implementing wrappers for memory release related method calls for returning anomalous values and test apps behavior.

- **Energy Utilization:** while assessing the energy utilization of android applications, the possible issues that needs further study is identified as follows:

i. Profilers are assessed using the functionality they provided by Bakker et al. [30] but there is no assessment that considers the usability of profilers.
ii. Investigating automated refactoring of applications to reduce energy wastage.
iii. To compute root cause of energy wastage automatically, need to apply energy stressing input scenario.
iv. Energy aware debugging
v. Working on a non-intrusive method to collect energy metrics.
vi. **General Category:** like the previous one, this section has also list of studies that needs attention for future studies.
vii. While offloading, different security aspects such as encryption techniques should be considered.
viii. Critical code sections such as onDraw() method of a class should be investigated for memory and energy utilization.

IV. Discussion

The purpose of this systematic literature review was to build a classification scheme and to collect, interpret and analyze all evidences related to android application memory and energy utilization, resource leaks of apps, and android applications performance testing approaches techniques and challenges. Currently, there are no comprehensive systematic literature review which is done in this important and hot area. As a result, a thorough and unbiased systematic literature review towards the area could contribute to the body of knowledge of android application memory and energy performance optimization.

This study indicates several research gaps that needs to be filled by further research and investigation. Based on my observation and analysis, most of the research approaches are applied validation research (i.e. 24 out of 31). Android application developers could get difficulty while choosing source code analysis tools, programming techniques, and approaches which is available in the sub-category of performance testing, memory, energy, and resource leak. This is because of unavailability of clear road-map for developers on how and which tools, techniques, approaches to apply to efficiently utilize the available resources in smartphones.

4.1 Threats to Validity

In this systematic literature review, several factors must be taken in to consideration while generalizing results. Firstly, during literature identification process, I only considered studies that has been electronically published. This may probably have neglected studies either journal or conference proceedings that were not published online.

Other possible key threats to the validity of the result is related to bias in the selection of studies and inaccurate data extraction. Several search strings were tested to choose the most appropriate search string which is used in the review. However, there is no guarantee whether all studies were retrieved possibly there would be a possible study that were omitted. Despite those issues, I have piloted the selection of search terms which could possibly minimize missing of certain studies or evidences.

In addition to this I excluded studies that was not empirically evaluated. This will exclude studies that do not have an empirical testing components or experimental evaluations to validate the papers' experiments and conclusions. My assumption was the studies with empirical evaluation provide certain level of validation of the stated benefits and limitations of the testing and analysis approaches discussed. Moreover, the studies may have been affected by bias when selecting articles

V. Conclusions and Future Work

Based on the results of my systematic literature review, this work presents a review of empirical knowledge in the field of android application memory and energy utilization as well as performance testing and analysis of apps. A total of 3226 studies from four online databases were analyzed and passes through three filtration steps. A total of 31 studies were finally found after the filtration and mapped to my classification scheme in my survey based on predefined inclusion/exclusion criteria.

The classification scheme contained three categories: the first is main category for structuring the topics and consisting classification sub-categories: (i) performance testing, (ii) resource leak, (iii) memory, (iv) energy, and (v) general category. The sub-categories in this scheme represents the focus area of the studies that has been addressed. After classification, at each sub-category techniques or method applied in each study were discussed. The second classification represents the contribution of the study. This classification represents the contribution of the study namely, Framework, Method, Tool, evaluation and metrics. The third classification represents objects involved in the study (e.g. type of applications used for evaluation i.e. opensource or commercial).

Several research gaps were identified as illustrated in the Discussion section. Based on the finding of this paper, there is a lack of studies that focus on memory and energy memory utilization optimization. In addition, more studies should address the important issues of resource leaks, programming techniques, performance enhancement and source code analysis tools.

It is also noted that practitioners will find it difficult to choose from several programming techniques and analysis tools and that there is a need for a clear road-map to guide them. Additionally, I recommend on future studies that can systematically compare proposed optimization solutions under categories of energy, memory and resource leaks using opensource and industrial applications. For future work, analysis and optimization framework will be developed consisting solutions to address specific research gaps tracked in this systematic literature review.

References

[1]. A. Lewerentz and J. Lindvall, *Performance and Energy Optimization for the Android Platform,* 2012.

[2]. D. Li and W. G. J. Halfond, "An investigation into energy-saving programming practices for android smartphone app development," in *Proceedings of the 3rd International Workshop on Green and Sustainable Software*, 2014.

[3]. M. Nagappan and E. Shihab, "Future trends in software engineering research for mobile apps," in *Software Analysis, Evolution, and Reengineering (SANER), 2016 IEEE 23rd International Conference on*, 2016.

[4]. M. Linares-V{\'a}squez, C. Vendome, Q. Luo and D. Poshyvanyk, "How developers detect and fix performance bottlenecks in android apps," in *Software Maintenance and Evolution (ICSME), 2015 IEEE International Conference on*, 2015.

[5]. S. Keele and others, "Guidelines for performing systematic literature reviews in software engineering," in *Technical report, Ver. 2.3 EBSE Technical Report. EBSE*, sn, 2007.

[6]. K. Petersen, R. Feldt, S. Mujtaba and M. Mattsson, "Systematic Mapping Studies in Software Engineering.," in *EASE*, 2008.

[7]. S. Zein, N. Salleh and J. Grundy, "A systematic mapping study of mobile application testing techniques," *Journal of Systems and Software,* vol. 117, pp. 334-356, 2016.

[8]. A. Shahrokni and R. Feldt, "A systematic review of software robustness," *Information and Software Technology,* no. 55, pp. 1-17, 2013.

[9]. A. Abogharaf, R. Palit, K. Naik and A. Singh, "A methodology for energy performance testing of smartphone applications," in *Proceedings of the 7th International Workshop on Automation of Software Test*, 2012.

[10]. T. Das, M. Di Penta and I. Malavolta, "A Quantitative and Qualitative Investigation of Performance-Related Commits in Android Apps," in *Software Maintenance and Evolution (ICSME), 2016 IEEE International Conference on*, 2016.

[11]. G. Hecht, N. Moha and R. Rouvoy, "An Empirical Study of the Performance Impacts of Android Code Smells," in *ACM International Conference on Mobile Software Engineering and Systems*, 2016.

[12]. C. V. M. T. D. P. Mario Linares-Vásquez, "How developers micro-optimize Android apps," *The Journal of Systems and Software,* no. 130, pp. 1-23, 2017.

[13]. D. Yan, S. Yang and A. Rountev, "Systematic testing for resource leaks in Android applications," in *Software Reliability Engineering (ISSRE), 2013 IEEE 24th International Symposium on*, 2013.

[14]. T. Wu, J. Liu, X. Deng, J. Yan and J. Zhang, "Relda2: an effective static analysis tool for resource leak detection in Android apps," in *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, 2016.

[15]. Y. Liu, L. Wei, C. Xu and S.-C. Cheung, "DroidLeaks: Benchmarking Resource Leak Bugs for Android Applications," *arXiv preprint arXiv:1611.08079*, 2016.

[16]. J. Liu, T. Wu, J. Yan and J. Zhang, "Fixing resource leaks in Android apps with light-weight static analysis and low-overhead instrumentation," in *Software Reliability Engineering (ISSRE), 2016 IEEE 27th International Symposium on*, 2016.

[17]. C.-Y. K. Y.-S. C. G.-Y. C. Kuei-Chung Chang, "Memory Behavior Profiler for Android Applications," in *IEEE 3rd Global Conference on Consumer Electronics (GCCE)*, 2014.

[18]. K. Vimal and A. Trivedi, "A memory management scheme for enhancing performance of applications on Android," in *Intelligent Computational Systems (RAICS), 2015 IEEE Recent Advances in*, 2015.

[19]. S.-j. K. M.-s. J. Hyun-Joo Yoo, "Study of Garbage Collection Performance on Dalvik VM Heap Considering Real-Time Response," in *International Conference on IT Convergence and Security (ICITCS)*, 2013.

[20]. S. Hamanaka, S. Kurihara, S. Fukuda, M. Oguchi and S. Yamaguchi, "Application state aware GC selection optimization in Android," in *Consumer Electronics-Taiwan (ICCE-TW), 2016 IEEE International Conference on*, 2016.

[21]. H. Shahriar, S. North and E. Mawangi, "Testing of memory leak in Android applications," in *High-Assurance Systems Engineering (HASE), 2014 IEEE 15th International Symposium on*, 2014.

[22]. G. Santhanakrishnan, C. Cargile and A. Olmsted, "Memory leak detection in android applications based on code patterns," in *Information Society (i-Society), 2016 International Conference on*, 2016.

[23]. G. Lim, C. Min and Y. I. Eom, "Enhancing application performance by memory partitioning in Android platforms," in *Consumer Electronics (ICCE), 2013 IEEE International Conference on*, 2013.

[24]. A. Banerjee, L. K. Chong, S. Chattopadhyay and A. Roychoudhury, "Detecting energy bugs and hotspots in mobile apps," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2014.

[25]. X. Li and J. P. Gallagher, "A source-level energy optimization framework for mobile applications," in *Source Code Analysis and Manipulation (SCAM), 2016 IEEE 16th International Working Conference on*, 2016.

[26]. A. Carette, M. A. A. Younes, G. Hecht, N. Moha and R. Rouvoy, "Investigating the energy impact of android smells," in *Software Analysis, Evolution and Reengineering (SANER), 2017 IEEE 24th International Conference on*, 2017.

[27]. W. Oliveira, R. Oliveira and F. Castor, "A study on the energy consumption of Android app development approaches," in *Proceedings of the 14th International Conference on Mining Software Repositories*, 2017.

[28]. Z. Z. Xinbo Chen, "Android App Energy Efficiency: The Impact of Language, Runtime, Compiler and Implementation," in *IEEE International Conferences on Big Data and Cloud Computing (BDCloud), Social Computing and Networking (SocialCom), Sustainable Computing and Communications (SustainCom)*, 2016.

[29]. D. V. Kaushik Dutta, "Caching to Reduce Mobile App Energy Consumption," *ACM Transactions on the Web,* vol. 12, pp. 1-30, 2017.

[30]. A. Bakker, "Comparing energy profilers for android," in *21st Twente Student Conference on IT*, 2014.

[31]. D. G. D. P. a. S. G. Alan Ferrari, "Detecting Energy Leaks in Android App with POEM," in *IEEE International Workshop on the Impact of Human Mobility in Pervasive Systems and Applications*, 2015.

[32]. A. Banerjee, L. K. Chong, C. Ballabriga and A. Roychoudhury, "Energypatch: Repairing resource leaks to improve energy-efficiency of android apps," *IEEE Transactions on Software Engineering,* 2017.

[33]. C. W. S. Q. Zhiwu Xu, "State-taint analysis for detecting resource bugs," *Science of Computer Programming,* 2017.

[34]. M. S. Y. X. J. K. N. M. K. P. H. Aki Saarinen, "Can Offloading Save Energy for Popular Apps?," ACM, Istanbul, Turkey, 2012.

[35]. A. A. R. P. M. B. a. W. S. Grace Metri, "A Simplistic Way for Power Profiling of Mobile Devices," IEEE, 2012.

[36]. M. H. Memon, M. Hunain, A. Khan, R. A. Shaikh and I. Khan, "Power management for Android platform by Set CPU," in *Computing for Sustainable Global Development (INDIACom), 2016 3rd International Conference on*, 2016.

[37]. P. K. Das, S. Shome and A. K. Sarkar, "APPS: Accelerating Performance and Power Saving in Smartphones Using Code Offload," in *Advanced Computing (IACC), 2016 IEEE 6th International Conference on*, 2016.

[38]. Y. Cheon, "Are Java Programming Best Practices Also Best Practices for Android?," 2016.

[39]. T. Wu, J. Liu, Z. Xu, C. Guo, Y. Zhang, J. Yan and J. Zhang, "Light-weight, inter-procedural and callback-aware resource leak detection for android apps," *IEEE Transactions on Software Engineering,* vol. 42, pp. 1054-1076, 2016.

[40]. V. S. S. Rajan, A. Malini and K. Sundarakantham, "Performance evaluation of online mobile application using Test My App," in *Advanced Communication Control and Computing Technologies (ICACCCT), 2014 International Conference on*, 2014.

[41]. Y. Liu, C. Xu and S.-C. Cheung, "Where has my battery gone? Finding sensor related energy black holes in smartphone applications," in *Pervasive Computing and Communications (PerCom), 2013 IEEE International Conference on*, 2013.

[42]. B. Kitchenham, "Procedures for performing systematic reviews," *Keele, UK, Keele University,* vol. 33, pp. 1-26, 2004.

[43]. K. Kim and H. Cha, "Wakescope: Runtime wakelock anomaly management scheme for Android platform," in *Embedded Software (EMSOFT), 2013 Proceedings of the International Conference on*, 2013.

[44]. J. Jeong, H. Kim and J. Lee, "Transparently Exploiting Device-Reserved Memory for Application Performance in Mobile Systems," *IEEE Transactions on Mobile Computing,* vol. 15, pp. 2878-2891, 2016.

[45]. Y. Hu, J. Yan, D. Yan, Q. Lu and J. Yan, "Lightweight Energy Consumption Analysis and Prediction for Android Applications," *Science of Computer Programming,* 2017.

[46]. C. Guo, J. Zhang, J. Yan, Z. Zhang and Y. Zhang, "Characterizing and detecting resource leaks in Android applications," in *Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering*, 2013.

[47]. M. Gottschalk, M. Josefiok, J. Jelschen and A. Winter, "Removing Energy Code Smells with Reengineering Services.," *GI-Jahrestagung,* vol. 208, pp. 441-455, 2012.

[48]. D. Di Nucci, F. Palomba, A. Prota, A. Panichella, A. Zaidman and A. De Lucia, "Software-based energy profiling of android apps: Simple, efficient and reliable?," in *Software Analysis, Evolution and Reengineering (SANER), 2017 IEEE 24th International Conference on,* 2017.

[49]. Y. Cheon and A. Escobar De La Torre, "Impacts of Java Language Features on the Memory Performances of Android Apps," 2017.

[50]. A. Bartel, J. Klein, Y. Le Traon and M. Monperrus, "Dexpler: converting android dalvik bytecode to jimple for static analysis with soot," in *Proceedings of the ACM SIGPLAN International Workshop on State of the Art in Java Program analysis*, 2012.

[51]. D. Amalfitano, N. Amatucci, A. M. Memon, P. Tramontana and A. R. Fasolino, "A general framework for comparing automatic testing techniques of Android mobile apps," *Journal of Systems and Software,* vol. 125, pp. 322-343, 2017.

[52]. A. Dogtiev, "App Download and Usage Statistics 2017," 21 November 2017. [Online]. Available: http://webcache.googleusercontent.com/search?q=cache:http://www.businessofapps.com/data/app-statistics/. [Accessed 28 November 2017].

[53]. D. Moth, "85% of consumers favour apps over mobile websites," Econsultancy, 12 March 2013. [Online]. Available: https://econsultancy.com/blog/62326-85-of-consumers-favour-apps-over-mobile-websites. [Accessed 28 November 2017].

[54]. E. Sherman, "The top 6 reasons mobile apps crash: How to best avoid Murphy," Techbeacon, 2015. [Online]. Available: https://techbeacon.com/top-6-reasons-mobile-apps-crash. [Accessed 28 November 2017].

[55]. H. Fakhruddin, "Top 12 Reasons Why Users Frequently Uninstall Mobile Apps," Teknowledge, 5 January 2016. [Online]. Available: http://teks.co.in/site/blog/top-12-reasons-why-users-frequently-uninstall-mobile-apps/. [Accessed 28 November 2017].

Abebaw Degu. "Android Application Memory and Energy Performance: Systematic Literature Review " IOSR Journal of Computer Engineering (IOSR-JCE) 21.3 (2019): 20-32.