A Survey of Performance Optimization for Mobile Applications

Max Hort, Maria Kechagia, Federica Sarro, Mark Harman

Abstract—To ensure user satisfaction and success of mobile applications, it is important to provide highly performant applications. This is particularly important for resource-constrained systems such as mobile devices. Thereby, non-functional performance characteristics, such as energy and memory consumption, play an important role for user satisfaction. This paper provides a comprehensive survey of non-functional performance optimization for Android applications. We collected 156 unique publications, published between 2008 and 2020, that focus on the optimization of performance of mobile applications. We target our search at four performance characteristics: responsiveness, launch time, memory and energy consumption. For each performance characteristic, we categorize optimization approaches based on the method used in the corresponding publications. Furthermore, we identify research gaps in the literature for future work.

Index Terms—mobile applications, android, non-functional performance optimization, software optimization, literature survey.

✦

1 INTRODUCTION

The relevance of mobile (handheld) devices, such as the so-called *smartphones*, has been ever growing for the past ten years, reaching an estimate of 3.2 billion smartphone users in 2019.[1] Smartphones can nowadays be considered as the main information processing devices for users. With smartphones, users cannot only receive and make phone calls, but also execute similar tasks as those performed on personal computers (e.g., surf the internet, perform calculations, pay bills).

Even though mobile devices are powerful, they represent resource-constrained devices making the development of applications that can run on them (*mobile applications*) challenging. This means that the functionality and performance of mobile applications depend on the characteristics of mobile phones (e.g., their physical memory, processors, battery) and on the current execution context (e.g., how many applications run at the same time on a mobile phone).

To ensure the success of an application (e.g., whether it will be used, updated, or uninstalled [1], [2]), developers aim to maximize user experience quality, and, consequently, user satisfaction [3], [4], [5]. User satisfaction is mainly influenced by functional (*Does the application operate as the user expects?*) and non-functional (*How does the application perform?*) application characteristics [1], [2]. Examples of functional issues can include missing or buggy features (e.g., a game application that functions in a different way than presented in its description). An example of a non-functional characteristic is the energy consumption of an application. Regardless of an application's functionality, users will be dissatisfied if the application drains the battery of their mobile devices within minutes.

Finkelstein et al. [6] found that the success of mobile applications in terms of downloads is correlated to the rating that the application attracts. These ratings are recorded by App Stores (e.g., Google Play, Apple Store, BlackBerry World). In 2018, the number of total applications downloaded amounted to 194 billion,[2] with every user having a multitude of different applications installed on their phone [7], [8]. With such a high number of applications, 75% of mobile device usage is filled by mobile applications [5]. While several studies show the importance of fixing software bugs that hinder applications' smooth function [9], [10], [11], [12], non-functional performance characteristics have shown to have a strong impact on user satisfaction as well [1], [2], [3], [13], [14], [15], [16], [17], [18]. This impact can be seen in the user reviews of real-world mobile applications:

- *"This app is destroying my battery. I will have to uninstall it if there isn't a fix soon."* [13]
- *"It lags and doesn't respond to my touch which almost always causes me to run into stuff."* [19]
- *"Bring back the old version. Scrolling lags."* [3]
- *"Makes GPS stay on all the time. Kills my battery."* [3]
- *"Too much memory usage for a glorified web portal ad machine."* [18]

Furthermore, Banerjee and Roychoudhury [20] conducted a study on 170,000 user reviews, and showed that poor performance and energy consumption lead to application downvotes from users. Among all causes of downvotes, energy consumption caused the highest ratio of uninstallations.

Given the importance of non-functional performance characteristics on user satisfaction and the consequent success of mobile applications, as well as new optimization approaches that are developed each year, we provide a

• *Department of Computer Science, University College London*
E-mail: max.hort.19@ucl.ac.uk, m.kechagia@ucl.ac.uk, f.sarro@ucl.ac.uk, mark.harman@ucl.ac.uk

1. https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/

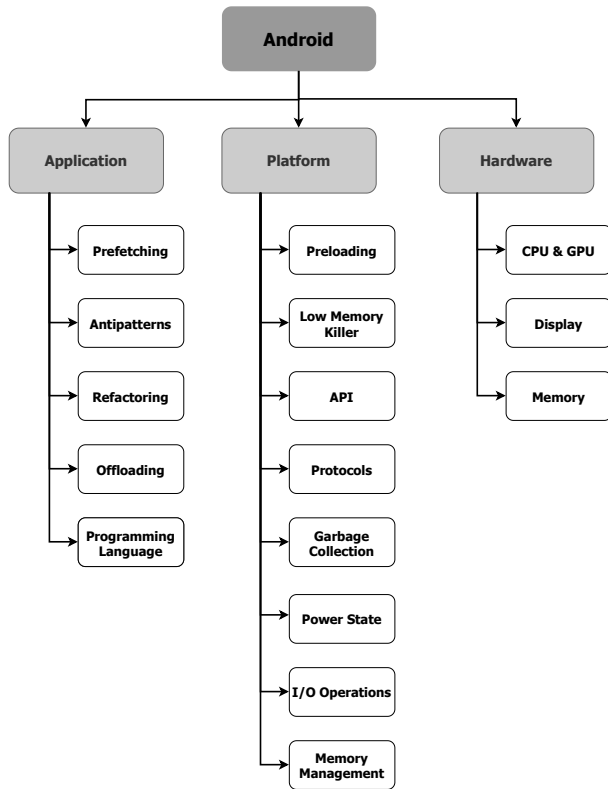2. https://www.statista.com/statistics/271644/worldwide-free-and-paid-mobile-app-store-downloads/

Fig. 1. Categorization of existing optimization approaches for non-functional characteristics of mobile applications.
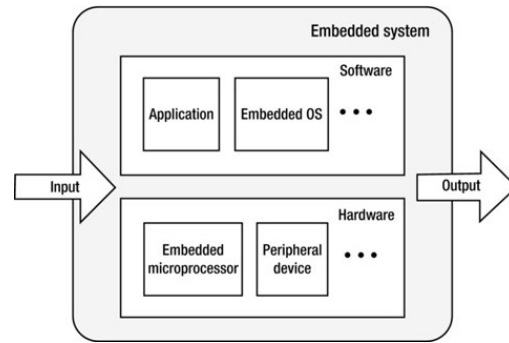

Fig. 2. Overview of mobile-device architecture [23].

with optimizing energy consumption. Moreover, we were able to detect relationships among the four non-functional performance characteristics (e.g., energy consumption can increase with an improved responsiveness of applications).

To the best of our knowledge, this is the first survey to investigate multiple non-functional performance characteristics and their relationships. To summarize, our work:

1) provides a comprehensive literature review of the state-of-the-art research on the optimization of non-functional characteristics for mobile applications;
2) provides a categorization of existing optimization approaches based on their level and type;
3) identifies challenges and opportunities for future research in this area.

We have made publicly available some additional resources [21] and an online version of the work reviewed in this survey, which we will keep up-to-date by accepting external contributions [22].

The rest of this paper is structured as follows. Section 2 presents an overview of mobile devices and mobile-application ecosystems. The search methodology is described in Section 3. Sections 4-7 describe research on non-functional performance optimization. These refer to: responsiveness (Section 4), launch time (Section 5), memory consumption (Section 6) and energy consumption (Section 7). A discussion of results considering all non-functional performance optimization characteristics is given in Section 8. Section 9 presents related work and Section 10 outlines threats to validity. Section 11 concludes this survey.

2 BACKGROUND

This section presents an overview of the context of this survey. Initially, we present key terms and definitions regarding the architecture of mobile devices. Then, we focus on the characteristics of the Android platform that we take into account in this work. Finally, we explain how the function and performance of mobile applications can affect users.

2.1 Mobile Devices

Mobile devices are embedded systems that consist of hardware and software components. Figure 2 illustrates a representative architecture of a mobile device.

The foundation of mobile devices is their hardware. The capacity of hardware components, such as physical memory,

comprehensive overview of existing approaches for non-functional performance optimization of mobile applications. This can be used by practitioners, developers, and researchers to search for approaches appropriate to their needs (e.g., "How can I reduce energy consumption only by changing application source code?" or "Can I improve responsiveness by applying changes to the device hardware?"). Furthermore, we provide information on dependencies among non-functional properties, which reside in mobile applications. We focus our review on the Android platform, since it is open-source software and has the highest market share among mobile platforms at the time of writing.[3]

Initially, we gathered and analyzed existing work to detect non-functional performance characteristics (Section 3). Based on this, we identify four non-functional characteristics, which describe user-perceived *performance* of mobile applications, and thereby their success, i.e.: *responsiveness*, *launch time*, *memory consumption*, and *energy consumption*. For each of these characteristics, we have categorized previous work based on the optimization level (e.g., optimization applied to application, platform, or hardware level) and proposed optimization type (e.g., prefetching, preloading, display), as shown in Figure 1.

We found that the majority of approaches to optimize responsiveness applied changes to the application's source code, while launch time was improvement by changes to the Android platform. Approaches that optimize memory apply changes to both the application and Android platform's source code. The majority of work was concerned

---

3. https://gs.statcounter.com/os-market-share/mobile/worldwide

processors, and battery, is constrained. Additionally, mobile devices come with a growing set of embedded sensors, including accelerometers, digital compasses, GPS, microphones, and cameras, which enable the emergence of personal, group and community-scale sensing applications [24]. However, the use of these sensors in applications requires a higher energy consumption [25].

The software of mobile devices comprises two basic layers: the mobile platform and the hosted mobile applications. The mobile platform (e.g., Android, iOS) consists of an embedded operating system (OS) that connects hardware with software components. It offers services such as memory management, networking, and power management. In mobile platforms, software libraries, which are used for the interaction with components, such as the database and media framework, are on the top of the OS. Mobile applications (e.g., calculator, photos, contacts) are either provided by a mobile framework (e.g., the Android platform) or third-party applications provided by online stores for mobile applications (e.g., Google Play Store, Apple iOS App Store, AppNokia, Samsung, BlackBerry World and Windows Phone Store).

2.2 Android

This survey focuses on the Android platform because it is currently the most used mobile platform[4] and open-source software, facilitating the analysis and evaluation of mobile systems. The following paragraphs present the main components of the Android platform.

Android is an embedded system based on the Linux OS. The Linux kernel links hardware and software components of a mobile device. It manages services such memory, processes, power, and networking access, and it offers drivers for flash memory, Bluetooth, WiFi, keyboard and audio. On top of the Linux kernel, lies the Android Runtime (ART), which is essential for running different applications. Each application runs as a separate process, having its own virtual machine instance.

The Android platform provides several methods and tools for improving the performance of mobile applications. For instance, memory is freed by the OS if the available memory on a device is low. To achieve that, Android uses the Low Memory Killer (LMK) to remove the Least Recently Used (LRU) cached application from the memory. Cached data is stored in the virtual memory, as long as memory is available [26]. In order to optimize the cache memory, and address problems such as duplicated pages in virtual memory, Kernel Same-page Merging (KSM) [27] and zRAM [28] are applied by Android [29]. Even though these methods optimize the cache memory, they consume power while they are being executed.

Furthermore, Android offers a variety of tools in its SDK to analyze system information and support application development.[5] Tools can be used for logging (LOGCAT), retrieving application and system information (APKANALYZER, DUMPSYS, SYSTRACE), as well as for simulations and debugging (ANDROID DEBUG BRIDGE, AVDMANAGER).

Finally, Android provides developers with selected performance measures (Android Vitals)[6] that use real user data, in case users have agreed on providing such information. If that happens, several metrics related to startup time, battery usage, and crash stack traces are recorded. Such metrics can assist developers to monitor memory and energy consumption, to identify synchronization issues, and to avoid application crashes [30].

2.3 User Experience

User experience and satisfaction are important factors that can ensure the success of mobile applications [31]. Application rating, or user satisfaction with an application, has been shown to correlate with the number of downloads [32]. After installing and using an application, users are able to make judgements regarding their satisfaction. Reviews regarding user satisfaction of mobile applications appear in App Stores and new users consider them in order to decide whether they will download an application or not.

To achieve a high level of user satisfaction, developers focus on improving both the functional and non-functional characteristics of mobile applications [1], [2], [3], [13], [15], [16]. Apart from fatal issues with functionality, such as application crashes [3], non-functional performance characteristics also shape users' perception [1]. Non-functional performance characteristics are the first characteristics to, potentially adversely, affect users [15] and can lead to application uninstallations [2]. In the following, we describe functional and non-functional characteristics of mobile applications that can affect user experience.

Functional characteristics describe whether an application is doing what it is supposed to do (i.e., its behavior). Frequent complaints about functional aspects of applications include freezes or crashes [1], functional errors, such as not getting push notifications, and the removal of features [3].

Non-functional characteristics determine how an application carries out (performs) its behavior. Even though it is difficult to measure and judge non-functional characteristics [33], they represent a vital part of user satisfaction for mobile applications. Related work analyzes a range of different non-functional characteristics regarding applications' performance [1], [2], [3], [13], [15], [16].

Different schemes exist to classify functional and non-functional characteristics of applications [34], [35], [36]. Among these, the FURPS model [36], [37] clearly distinguishes *performance* characteristics from other functional and non-functional characteristics, as follows:

- Functionality: feature set, capabilities, generality, security;
- Usability: human factors, aesthetics, consistency, documentation;
- Reliability: frequency/severity of failure, recoverability, predictability, accuracy, mean time to failure;
- Performance: speed, efficiency, resource consumption, throughput, response time;
- Supportability: testability, extensibility, adaptability, maintainability, compatibility, configurability, serviceability, installability, localizability, portability.

---

4. https://gs.statcounter.com/os-market-share/mobile/worldwide/#monthly-201909-201909-bar

5. https://developer.android.com/studio/command-line

6. https://developer.android.com/topic/performance/vitals

Among these characteristics, we are interested in the *Performance* category, which in the context of mobile applications, can be further refined into the following four non-functional performance characteristics:

1) *Responsiveness* captures the time required to update the frames of the graphical user interface after user interaction.
2) *Launch time* describes the time required to start an application. This can happen as a cold start, when the application is launched without cached data (e.g., after device boot). Another launch type refers to the hot/warm start of application, which occurs when an application activity is kept in memory for a faster launch [38].
3) *Memory consumption* describes the amount of occupied memory. Memory can be shared between multiple applications or stored separately [39]. On constrained systems, such as smartphones, memory is a critical resource [26], [29].
4) *Energy consumption* is associated with the battery life. Energy is consumed by various components, including CPU, LCD, GPS, audio and WiFi services [25], [40].

A detailed description of each of the four non-functional performance characteristics is given in sections 4-7 (responsiveness, launch time, memory consumption, energy consumption, respectively).

3 SURVEY METHODOLOGY

The purpose of this survey is to gather and categorize research work published in the mobile computing and software engineering literature that refers to the optimization of non-functional performance of Android applications.

As this is an emergent topic and there is limited related work on Android performance optimization techniques to perform a systematic literature review (according to the guidelines of Kitchenham [41]), we conduct a *comprehensive literature review*. In the following, we present our search methodology in detail, starting with a preliminary and venue search, followed by a repository search and snowballing.

3.1 Search Methodology

Our literature review on performance optimization includes publications that refer to optimization techniques on mobile applications and measurement of application performance.

3.1.1 Preliminary Search

Prior to systematically searching online repositories, we conducted a preliminary search. The goal of the preliminary search is to gain a deeper understanding of the field and assess whether there is a sufficient amount of publications that allows for subsequent analysis. Based on these results, we distinguish between four different non-functional performance characteristics: responsiveness, launch time, memory and energy consumption.

Other than Sadeghi et al. [42], who refined keywords during their search, we perform a preliminary search to guide our repository search. Additionally, we use the results of the preliminary search to define keywords (Table 2) and venues (listed in Section 3.1.2).

3.1.2 Repository Search

Proceeding the preliminary search, we conduct a search of six established online repositories (IEEE, ACM, ScienceDirect, Scopus, arXiv, and Google Scholar). We have gathered publications from 2008 to February 2020, since the first version of Android was released in 2008.

To ensure that we provide an exhaustive literature search, we manually examine relevant venues from the field of software engineering and mobile computing, which we encountered during the preliminary search. We search venues with at least five publications in our preliminary search.

- **Conferences:** ICSE, ASE, MSR, MobiSys, MobileHCI, MobileSoft, UbiComp, CHI, ESEC/FSE.
- **Journals:** IEEE TSE.

3.1.3 Selection

Table 2 lists keywords used to guide our repository search. Keywords are divided into five categories. Firstly, the keywords that belong to the *Platform* category ensure that the selected publications deal with mobile platforms, particularly Android. Furthermore, keywords that belong to the *Responsiveness*, *Launch time*, *Memory*, and *Energy* categories filter publications referring to non-functional performance characteristics. We restrict search results to publications that contain at least one platform keyword and one non-functional keyword in their title.

To ensure that the publications found during our search are relevant to the context of non-functional performance optimization of mobile applications, we consider the following inclusion criteria:

- The publication should refer to at least one of the non-functional performance characteristics investigated in this survey (e.g., responsiveness, memory, energy, and launch time), or to an approach that profiles at least one of the mentioned performance characteristics.
- The publication investigates the proposed methods on smartphones with an Android OS.

To assesses whether the publications satisfy our inclusion criteria, we manually examined every publication using the process adopted by Martin et al. [43], as follows:

1) **Title:** First, all those publications whose title clearly does not match our inclusion criteria are excluded;
2) **Abstract:** Second, the abstract of every remaining publications is checked. Publications whose abstract does not meet our inclusion criteria are excluded at this step;
3) **Body:** Publications that passed the previous two steps are then read in full, and excluded if their content does neither satisfy the inclusion criteria nor contribute to this survey.

Based on the above three-stage process and inclusion criteria, we iteratively reduce the amount of publications obtained from online repositories, until we end up with the set of publications investigated in the following sections (sections 4 to 7). This process is performed by two authors independently, the results are compared at each stage, and disagreements discussed until an agreement is reached.

TABLE 1
Results of the Repository Search. The number of papers retained at each stage of the search (e.g., Hits, Title, Abstract, Body) is given for each online repository (e.g., Google Scholar, IEEE, Scopus, ACM, Science Direct, arXiv) and non-functional performance characteristic (Responsiveness, Launch Time, Memory, Energy). Google Scholar is abbreviated with GS; Science Direct is abbreviated with SD. For example, searching for "responsiveness" in GS retrieves 835 publications, among those 31 have a relevant title, 24 of those have an abstract satisfying our inclusion criteria, and a total of 11 (of those 24) publications are included in our survey after reading them entirely.

		Responsiveness	Launch Time	Memory	Energy		Responsiveness	Launch Time	Memory	Energy
Hits	GS	835	90	392	1129	IEEE	180	14	47	487
Title		31	10	16	76		21	5	9	52
Abstract		24	9	11	58		17	4	9	37
Body		11	1	4	31		9	0	7	8
Hits	Scopus	146	6	57	269	ACM	73	21	23	152
Title		20	3	6	51		17	10	3	47
Abstract		15	3	6	44		16	8	2	44
Body		7	0	2	15		8	4	2	20
Hits	SD	71	13	97	62	arXiv	17	1	1	18
Title		2	2	0	21		3	0	0	10
Abstract		2	0	0	15		1	0	0	9
Body		0	0	0	1		2	0	0	2

TABLE 2
Keywords Used for the Repository Search.

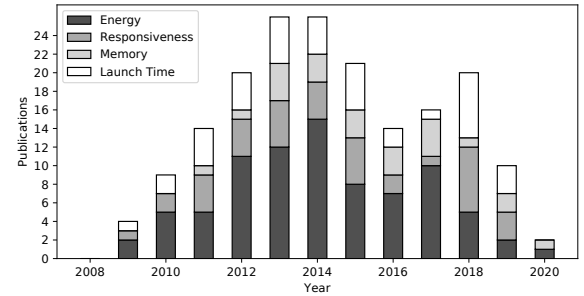| Category	Keywords
Platform	android, smartphone, app, apps
Responsiveness	responsiveness, performance
Launch time	launch, start
Memory	memory
Energy	energy, battery, power

Fig. 3. Number of publications on non-functional performance optimization per year.

3.1.4 Snowballing

After a collection of publications is obtained from the repository search, we proceed to inspect the related work of the publications selected in the previous search to gather cited publications using snowballing [44]. We apply one level of backwards snowballing.

3.2 Selected Publications

Table 1 shows the results of the repository search. The amount of publications found during each step of the search is listed.[7]

In the following, we give the number of unique publications after each stage of the search procedure in addition to the number of newly added publications:

1) Preliminary search: 96
2) Repository search: 174 (+80)
3) Venue search: 180 (+4)
4) Snowballing: 252 (+72)
5) Author feedback: 297 (+45)

In addition to the discussed stages of the search procedure (1-4), we added 45 publications based on the feedback from the authors cited. Among all 294 publications, 156 unique publications optimize at least one non-functional performance characteristic. These 156 publications were published in 97 different venues. We further classify top publication venues (A, A* according the CORE ranking Portal),[8]

7. A collection of publications after checking *Abstract* is available in our on-line appendix [21]

8. http://www.core.edu.au/conference-portal. Additionally, we include MobiSys, classified as "B", due to its popularity on mobile systems.

regarding their category based on the ACM's Computing Classification System (CCS).[9] Among these, a majority of publications is obtained from Software Engineering (32%), and Computer Systems Organisation (29.33%) venues. The remaining publications are retrieved from Mobile Computing (20%), Networks (17.33%), and Security and Privacy (1.33%) venues. A full list of conferences and journals is available online [21].

The publication distribution over the entire search period is illustrated in Figure 3. Note, a publication can contribute to more than one subtotal if it explicitly optimizes more than one non-functional performance characteristic. During our search, we found ten publications that optimize more than one non-functional performance aspect [38], [45], [46], [47], [48], [49], [15], [50], [51], [52]. Among these, there is one publication that optimizes three characteristics (responsiveness, energy consumption, memory consumption) [45], while the others optimize two. Section 8.3 provides further details on the relationships between performance characteristics.

Based on our search results, we devise the categorization of approaches shown in Figure 1. These categories consist of approaches (e.g., offloading, code optimization) and elements of the Android platform (e.g., Low Memory Killer, API). In

9. https://dl.acm.org/ccs

the remainder of the survey, we discuss our search results in detail.

4 RESPONSIVENESS

Responsiveness refers to the ability of mobile applications to respond to user interactions fast and smoothly. An application is highly responsive when the time it takes to respond to user requests is minimal. A highly responsive application offers high user satisfaction as users prefer not to wait when interacting with an application. On the other hand, an application with poor responsiveness can have a negative impact on user perception, and on its success [53].

Specifically, Tolia et al. [54] argued that response times lower than 150ms do not negatively affect user satisfaction. In fact, delays that last almost one second do not significantly affect users, but start making them aware of these delays, whereas delays that last more than one second indeed make users "unhappy" [54]. Willocx et al. [55] stated that response times under 100ms appear as instantaneous to users. Furthermore, users would accept response times up to a few seconds, if delays were to occur rarely [55].

To detect and fix hot spots in mobile applications that may cause response-time delays, developers use several techniques, including profiling and optimization approaches. Section 4.1 presents methods and approaches for profiling, while Section 4.2 presents approaches to optimize responsiveness. Section 4.3 summarizes our findings.

4.1 Profiling

There are several profiling approaches that developers use to measure the responsiveness of applications and locate hot spots for improvement. Popular profiling techniques include the measurement of page loading time [56], the measurement of the overall frame time and the calculation of the number of delayed frames [51], as well as the estimation of Central Processing Unit (CPU) time [57]. Furthermore, responsiveness can be measured at different levels of the Android platform, considering the UI [58] and hardware components [59].

Several tools have been developed to measure the responsiveness of mobile applications. Specifically, Ravindranath et al. [60] introduced APPINSIGHT to detect critical paths in applications, which represent bottlenecks for user transactions. Hong et al. [61] proposed PERFPROBE, a profiling approach to diagnose hardware and software causes for slowdowns with runtime information. Kim et al. [62] conducted performance testing, using unit tests, at early development stages of the applications to identify response-time delays. Kang et al. [63], [64] presented a technique that analyzes application performance focusing on particular asynchronous executions. Wang and Rountev [65] introduced a novel approach that profiles responsiveness by tracking the usage of mobile resources such as bitmap or SQLITE databases. Kwon et al. [66] proposed MANTIS, a framework that predicts the execution time of an application while using particular inputs.

4.2 Optimization Approaches

For a high responsiveness of mobile applications, developers apply several categories of optimization approaches. In the

TABLE 3
Studies on Responsiveness Optimization.

Category	Authors [Ref]	Year	Venue
Offloading	Kemp et al. [67]	2010	MobiCASE
	Chun et al. [68]	2011	EuroSys
	Ra et al. [69]	2011	MobiSys
	Kosta et al. [70]	2012	INFOCOM
	Gordon et al. [71]	2012	OSDI
	Gordon et al. [72]	2015	MobiSys
	Das et al. [46]	2016	IACC
	Montella et al. [73]	2017	CCPE
	Chen and Hao [74]	2018	J-Sac
Antipatterns	Jin et al. [75]	2012	SIGPLAN
	Yang et al. [53]	2013	MOBS
	Nistor et al. [76]	2013	ICSE
	Liu et al. [13]	2014	ICSE
	Ongkosit and Takada [77]	2014	DeMobile
	Hecht et al. [78]	2015	ASE
	Habchi et al. [79]	2018	ASE
	Hecht et al. [51]	2016	MobileSoft
	Li et al. [80]	2019	SANER
Refactoring	Lin et al. [81]	2014	FSE
	Okur et al. [82]	2014	ICSE
	Lin et al. [83]	2015	ASE
	Lyu et al. [47]	2018	ISSTA
	Feng et al. [84]	2019	ICSTW
Prefetching	Higgins et al. [85]	2012	MobiSys
	Zhao et al.[86]	2018	ICSE
	Choi et al.[87]	2018	CoNEXT
	Malavolta et al.[88]	2019	ICSE-NIER
Programming languages	Batyuk et al. [89]	2009	MobileWare
	Lee and Jeon [90]	2010	ICCAS
	Lee and Lee [91]	2011	iCast
	Lin et al. [92]	2011	IBICA
	Saborido et al. [45]	2018	EMSE
CPU & GPU	Wang et al. [93]	2013	CGO
	Cheng et al. [94]	2013	IWSSIP
	Thongkaew et al. [95]	2015	JIP
I/O operations	Nguyen et al. [50]	2015	MobiSys
	Mao et al. [96]	2018	ITCSDI
Hardware components	Kim and Shin [97]	2015	ICUIMC

following, we describe techniques found in literature. Table 3 lists our findings.

Offloading refers to the transfer of heavy computational tasks to external computing units with less performance-related restrictions. This technique is popular in the development of mobile applications as mobile devices are embedded systems with restricted memory and CPU. However, offloading comes with an overhead while transferring the results of processes from an embedded system to external computing units and vice versa [72]. The first offloading implementation for Android applications was introduced by Kemp et al. [67] and refers to the CUCKOO framework. CUCKOO helps developers to easily implement offloading tasks in their applications. The offloading decision is made at runtime based on heuristics, context, and historic information. Other frameworks that support offloading include CLONECLOUD [68], THINKAIR [70], COMET [71], and TANGO [72]. Offloading has frequently been used for responsiveness improvements [46], [69], [73], [74].

Antipatterns are bad programming patterns, such as performance bugs, which deteriorate software quality and reduce application responsiveness that can negatively affect user experience [13], [51]. For this reason, several tools have been developed for the identification and removal of antipatterns. In particular, Liu et al. [13] introduced PERFCHECKER to automatically detect performance bugs in mobile applications. PERFCHECKER is built on top of the SOOT [98] Java optimization framework and analyzes applications at a bytecode level. PERFCHECKER applies static

code analysis to detect antipatterns. Other tools focus on detecting particular types of bugs related to application responsiveness [51], [76]. For instance, Nistor et al. [76] searched for repetitive computations in code loops, following the intuition that repetitive behavior is likely to be optimizable. To detect repetitive behavior, they created TODDLER, an automated oracle to analyze memory access patterns. Hecht et al. [51] used a static analysis tool called PAPRIKA [78] to detect three types of code smells (Internal Getter/Setter, Member Ignoring Method, and HashMap Usage). They investigated the removal of code smells in an empirical study, obtaining responsiveness improvements of up to 12.4%.

Furthermore, Inefficient Image Displaying (IID) can cause performance degradation (e.g., repeated and redundant image decoding). Li et al. [80] developed the static analysis tool TAPIR to detect IID issues, which can be strongly correlated with antipatterns.

Finally, responsiveness-related bugs can be detected using predefined rule sets (e.g., efficiency rules) [75], [77] and test amplification (insertion of artificial delays in application source code) [53].

Refactoring can be performed to utilize efficient programming practices. Lin et al. [81], [83] provided an analysis showing that even though applications include concurrent code, they often contain bugs or end up with executing the source code sequentially. For concurrent code execution, and higher responsiveness, the authors located and refactored long-running operations, using the two tools ASYNCHRONIZER and ASYNCDROID. Okur et al. [82] developed two tools to refactor asynchronous code in Windows Phone applications. Asynchronous code is converted by ASYNCIFIER and common misuses in asynchronous code are corrected by CORRECTOR. Their empirical study showed that developers accept the proposed changes to asynchronous code. Lyu et al. [47] applied static analysis to change ineffcient database operations that are placed in within a loop. Database operations called in loops can cause Repetitive Autocommit Transaction (RAT), which creates a new transaction in each iteration of the loop. Furthermore, Feng et al. [84] mined optimization patterns from GITHUB projects, considering performance-aware APIs, which can be manually injected into the source code of mobile applications and improve their performance.

Prefetching refers to a technique that caches data in advance, so that it can timely provide required data when it is needed. Higgins et al. [85] provided a library called Informed Mobile Prefetching (IMP) that assigns the task of determining when to prefetch data to the mobile system, rather than leaving the choice to developers. Developers solely specify which items could benefit from prefetching, while IMP determines whether and how prefetching is handled, based on responsiveness, battery lifetime, and mobile data usage. Zhao et al. [86] proposed a technique named PALOMA, that prefetches HTTP requests to reduce responsiveness latency. PALOMA uses string analysis to detect prefetchable content in the application source code. While users navigate in an application, PALOMA uses short pauses ("user think time") for prefetching. Choi et al. [87] identified resource dependencies with static analysis to automatically generate acceleration proxies for dynamic prefetching. Application binary files are analyzed to detect HTTP(s) messages, which

are later used for prefetching. As static analysis lacks certain information, missing information of HTTP(s) requests is added at runtime. Lastly, Malavolta et al. [88] proposed a technique called NAPPA that prefetches network requests based on user navigation patterns.

Programming languages can impact the processing speed of applications and therefore responsiveness. Android provides the Native Development Kit (NDK) that allows developers to write native C/C++ code. Native instructions are directly executed by the CPU and, therefore, they provide a better performance over non-native ones [93], [99]. Several empirical studies compared the performance of programming languages, and found that native C code reduces the running time of the same algorithms written in Dalvik Java code [92], [90], [91], [89]. Furthermore, efficient implementation choices, such as which map variant to use (e.g., HashMap, ArrayMap, and SparseArray) can improve responsiveness [45].

CPU and Graphics Processing Unit (**GPU**) adaptations can accelerate the execution of time-consuming programming tasks and increase application responsiveness. Wang et al. [93] proposed ACCELDROID to accelerate the execution of bytecode on the HW/SW co-designed processor of Android. Therefore, instead of translating bytecode twice, this is only translated once. Cheng et al. [94] provided guidelines to map applications to the Android platform (e.g., whether to use CPU or GPU and how many cores are used). This mapping is platform as well as task-dependent. An optimal performance choice can avoid performance degradation. Additionally, Thongkaew et al. [95] developed architectural hardware extensions that can fetch and decode Dalvik bytecode directly.

I/O operations have an impact on responsiveness and can enable optimization [100]. For instance, Nguyen et al. [50] proposed an approach that adapts the prioritization of read and write operations for avoiding slowdowns. Mao et al. [96] introduced a trace collection tool to identify redundant I/O requests in mobile applications and eliminate them to reduce response times. As redundancy is minimally shared among applications, they performed an application-aware optimization.

Hardware components, such as the use of embedded Multimedia Cards (eMMC), can be investigated for responsiveness improvements. Kim and Shin [97] studied whether additional features of eMMCs are utilized by Android smartphones, and reduced the I/O latency.

4.3 Summary

Responsiveness is a non-functional performance characteristic concerned with the time an applications needs to respond to user requests. In practice, this is either measured in time (ms) [56], [57] or in frames [51]. Several tools have been proposed to measure responsiveness and detect responsiveness issues [60], [61], [63], [64].

Since responsiveness measures the duration required to complete computations, a naive approach to improve response times is to move the computations from the smartphone to devices with less restrictions. This approach is called "offloading" and requires additional infrastructure (e.g., external servers for computation). If such an infrastructure

is not available, other approaches can be followed, which in majority are applicable to a source-code level (23/38).

Furthermore, changes that have been applied to mobile applications' source code include the removal of bad programming patterns (antipatterns), and the usage of good programming practices (e.g., concurrency with the help of refactoring [81], [83], [82], and prefetching of content [85], [86], [87], [88]). Other than changing source code after responsiveness issues have been detected, a carefully considered choice of the right programming language can lead to improvements (e.g., native C/C++ is faster when executed on CPUs [89], [90], [91], [92]). Lastly, changes to the hardware have achieved responsiveness improvements. This can directly happen at CPU or GPU level [93], [94], which can carry out computations, or on other hardware components (e.g., memory [97]).

5 LAUNCH TIME

During the launch of a mobile application, operations and data are loaded to make the application available to the user. Therefore, launch time is the first performance characteristic of a mobile application that users have the opportunity to notice. Launch time directly influences user experience and satisfaction. Nagata et al. [101], Song et al. [38] and Kim et al. [15] defined the launch time of an application as the required time until user input is accepted. Yan et al. [102] described the Total Launch Time (TLT) of an application as the needed time until the entire content, including asynchronously loaded content, can be displayed to the user.

Furthermore, Song et al. [38] found that cold start time (when an application is started from scratch) has a significant impact on the application launching experience of users. Developers can address this issue by analyzing their applications' source code to identify and fix bottlenecks that possibly increase launch time. The following sections discuss profiling methods (Section 5.1) and approaches to optimize launch time and cold start issues (Section 5.2). A summary is given in Section 5.3.

5.1 Profiling

Profiling approaches have been proposed for locating issues in mobile applications that may increase the application launch time. Using monitoring functions in the source code of Android applications, in the Android platform, and in third-party libraries used by Android applications, developers can pinpoint performance issues causing launch-time delays [101], [103]. Also, developers can profile the usage of system resources to pinpoint the application launch completion [104]. Additionally, Nguyen et al. [50] studied launch delays of an application as the time taken in kernel mode and the time spent waiting for disk network operations.

To understand how launch time can affect user behavior, Song et al. [38] investigated logs from application usages. Other approaches for assessing the launch behavior of applications include monitoring of: handling of I/O requests [105], system memory usage [15], restart ratio of applications (the number of cold starts over all application launches) [38] and user satisfaction with regards to launch-time delays [106].

TABLE 4
Studies on Launch Time Optimization.

Category	Authors [Ref]	Year	Venue
Preloading	Yan et al. [102]	2012	MobiSys
	Nagata et al. [101]	2013	CANDAR
	Parate et al. [107]	2013	UbiComp
	Tang et al. [108]	2013	SIGAPP
	Chung et al. [48]	2013	TECS
	Song et al. [38]	2014	TECS
	Lee et al. [49]	2017	J-SAC
	Baumann and Santini [109]	2017	IMWUT
	Martins et al. [16]	2018	ICMLT
Low Memory Killer	Chung et al. [48]	2013	TECS
	Prodduturi and Phatak [110]	2013	IIT
	Song et al. [38]	2014	TECS
	Baik and Huh [111]	2014	ICSE
	Kim et al. [15]	2015	IEEE Micro
	Vimal and Trivedi [112]	2015	RAICS
	Singh et al. [113]	2016	IOTA
	Kim et al. [104]	2016	TECS
	Lee et al. [49]	2017	J-SAC
	Li et al. [114]	2017	IWCMC
Memory	Joo et al. [105]	2011	FAST
	Kim et al. [15]	2015	IEEE Micro
I/O operations	Nguyen et al. [50]	2015	MobiSys

5.2 Optimization Approaches

In order to reduce launch time, developers apply optimization techniques. Table 4 summarizes publications found in literature, which are described in the following.

Preloading of application data prevents cold starts and thereby the overall user waiting time. Frequently, the usage of applications follows patterns [38], and enables the prediction and preloading of the next application to be used. Context information, such as the time of day or the location can be taken into account to improve predictions [16], [49], [115] and preload applications.

The FALCON approach by Yan et al. [102], which refers to an OS extension, preloads applications and application-specific content based on the context (e.g., location) and usage patterns. For this purpose, spatial and temporal features are designed based on an extensive analysis. Usage patterns include the use of weather applications in the morning, or playing games at home.

Additionally, application predictions determine the applications to be launched next [38], [48], [108], [109] and when they are going to be used [107]. In particular, Nagata et al. [101] analyzed the relationship of application launch time regarding the number of preloaded classes. For this, they manually selected a number of preloaded classes, and showed for one application that the launch time is reduced when the number of preloaded classes is high. The prediction of the next application to be used has been also investigated for restructuring user interfaces [116], [117], [118].

The **Low Memory Killer** (LMK), which removes the LRU application from memory, may not lead to optimal results [15], [113], because users do not always rely on recently used applications. To address this issue, several studies introduce techniques that determine which data should be removed from memory. Specifically, Song et al. [38] and Li et al. [114] devised models to detect patterns in application usages based on application cold start times. Decisions for the LMK are based on usage patterns to prioritize data of applications that are likely to be launched.

Instead of removing the LRU application from mem-

ory, choices can be made based on application cold start times [15], the required storage size [104], and the importance of an application to a user [110], [112], [113]. Furthermore, Baik and Huh [111] analyzed usage patterns and determined a threshold on how many processes to keep in memory before freeing them. If this limit is fixed to a high value, more applications can be kept in memory, leading to fewer restarts.

Memory can be adapted to suit application launches better. Accordingly, Joo et al. [105] proposed the use of SSDs instead of HHDs to speedup application launch time. This approach was not designed for mobile devices; however, mobile devices use NAND flash memory as secondary storage carrying almost identical performance characteristics as SSDs. One could therefore apply this approach to mobile devices as well. Furthermore, Kim et al. [15] proposed the use of Non-Volatile Memory (NVM) to store frequently used applications and shared libraries among applications. Shared data is stored on Phase-Change Memory (PCM). Therefore, less data needs to be loaded when launching applications.

I/O operations impact the application launch time, as their speed can be seen as a performance bottleneck during application launch [15]. In fact, Nguyen et al. [50] analyzed the impact of read and write operations on launch time, as mobile devices wait for I/O operations to complete. As application launches are dominated by read operations (five times as many read operations as write operations [50]) this can have a high impact on overall application launch time. A prioritization of read and write operations avoids slowdowns and reduces launch time.

5.3 Summary

Launch time describes the time required until a user input is received [15], [38], [101] or the entire application content is displayed [102], after an application has been started by the user. The application launch completion has been profiled according to the system's resource-management usage [104].

Due to the high negative impact that cold starts can have on user satisfaction [38], the majority of launch time optimization methods (15/18) prevent cold starts, and reduce application launch time. On one hand, preloading of application data can be applied to spend loading times before the application launch, and reduce the actual launch itself. For this purpose, predictions are used to determine applications that are likely to be used next, based on usage patterns [16], [38], [49], [102], [115]. On the other hand, changes to memory management (LMK) have achieved similar results. In contrast to preloading, which loads desired application data, changes to the LMK to keep important data in memory for a longer time. Both of these approaches require access to the Android OS to implement the required adaptations.

An increased speed of memory operations (e.g., usage of SSDs over HHDs [105]), shared libraries among applications [15], and I/O prioritization of read over write operations [50] have been also applied to reduce launch time.

6 MEMORY

Memory is a critical resource for embedded systems [26], [29], such as mobile devices. Specifically, in Android devices, data can be loaded either by the Android platform

(to be shared across multiple running applications) or by each application, separately. Application data is stored in separate heaps, per application [39]. The main memory is typically shared between the CPU and GPU [52]. Therefore, a considerable amount of the main memory is occupied by graphic processing operations [26]. Kim et al. [15] classified applications in two categories, based on their memory consumption, *stable* and *unstable*. The memory consumption of *stable* applications increases within the first ten seconds of the applications' launch, and it stabilizes afterwards. The memory consumption of *unstable* applications increases steadily, and it does not stabilize.

The following sections present approaches that are used to measure the consumed memory of mobile applications (Section 6.1), optimize application memory consumption (Section 6.2). At the end, we provide a summary (Section 6.3).

6.1 Profiling

Different tools and approaches have been proposed to measure the memory usage of applications. For instance, memory consumed by mobile applications can be measured by: kernel memory footprints [119], garbage collection calls [51], physical memory dumps, and logging information [29]. Vimal and Trivedi [112] used the Dalvik Debug Monitor Server (DDMS) to analyze memory footprints of Android components and measure memory consumption.[10] Tools such as ANDROSCOPE by Cho et al. [59] have been also used to analyze the performance (including memory) of all the layers of the Android platform. Furthermore, ANDROBENCH [120] and ANDROSTEP [121] are benchmark tools that assess the storage performance of Android devices by analyzing logs from read and write I/O operations.

6.2 Optimization Approaches

To reduce the memory usage of mobile applications developers use different categories of optimization approaches. The following paragraphs summarize the relevant approaches found in literature. Table 5 lists the representative studies.

Antipattern coding practices can be used to identify code that is likely to lead to memory leaks. Memory leaks occur when applications constantly request memory while running [122], or when unused objects are being kept in memory longer than required [123].

Hecht et al. [51] showed in an empirical study that memory consumption can be reduced by correcting code smells. In particular, memory can be improved in terms of memory usage and number of garbage collection calls. Shahriar et al. [123] developed memory leak patterns for Android applications and used fuzz testing to emulate and detect memory leaks. A total of three fuzzing types (application, resource, and API) are used in their experiments, which discovered crashes due to memory leaks in real-world applications. Furthermore, memory leaks can be identified by analyzing memory dumps [52], [139], the activity lifecycle [125], source code patterns [124] or memory execution information by applying process control block hooking [122].

---

10. DDMS is deprecated and was removed from Android Studio 3.2. Android offers other tools to carry out the functions of DDMS https://developer.android.com/studio/profile/monitor

TABLE 5
Studies on Memory Optimization.

Category	Authors [Ref]	Year	Venue
Antipatterns	Park and Choi [122]	2012	IJCA
	Shahriar et al. [123]	2014	HASE
	Hecht et al. [51]	2016	MobileSoft
	Santhanakrishnan et al. [124]	2016	i-Society
	Tasneem et al. [52]	2019	IJCA
	Amalfitano et al. [125]	2020	IEEE Access
Garbage collection	He et al. [39]	2011	IFIP
	Gerlitz et al. [126]	2013	JTRES
	Lim et al. [127]	2013	ICCE
	Mori et al. [128]	2017	GCCE
	Tasneem et al. [52]	2019	IJCA
Deduplication	Kim et al. [129]	2014	ICSE
	Lee et al. [29]	2015	APSys
Memory management	Kim et al. [130]	2013	IEEE TCE
	Jeong et al. [131]	2013	USENIX ATC
	Zhong et al. [132]	2014	EMSOFT
	Kim et al. [133]	2015	ISMM
	Nguyen et al. [134]	2016	WiMob
	Kim and Bahn. [135]	2017	PMC
	Kim et al. [136]	2017	TECS
	Kim and Bahn. [137]	2019	IEEE Access
GPU	Kwon et al. [26]	2015	EMSOFT
Programming languages	Escobar De La Torre and Cheon [138]	2017	UTEP
	Saborido et al. [45]	2018	EMSE

Garbage collection is used in Android to manage memory and identify unused objects that can be removed [52]. Since version 2.2., Android uses a stop-the-world (STW) garbage collector [39], [126]. This stops other operations to free the memory and resume them afterwards, resulting in pauses that can negatively effect user experience [39].

Different garbage collector designs have been evaluated for improvements: reference counting garbage collection [52], [126], concurrent garbage collection [39] and generational garbage collection [39], [128]. Lim et al. [127] proposed a memory partitioning scheme, which partitions available memory into two nodes (for critical and uncritical applications). If one node runs out of memory, only the memory of this node is freed.

Deduplication is a technique to remove redundant pages from memory. While duplicated memory reduces available memory for other applications, Android is prone to have page-level duplication in memory [29]. Lee et al. [29] developed a system (MEMSCOPE) to analyze memory duplication in Android OS. MEMSCOPE identifies memory segments that contain duplicated memory pages. One of the disadvantages of deduplication is the additional computation needed to detect and merge redundant pages. Therefore, Kim et al. [129] proposed a computationally efficient deduplication scheme, considering background applications that do not update memory contents and need to be scanned only once.

Memory management changes can be applied to achieve further improvements in memory usage by mobile applications. For example, *swapping* is a technique that reclaims memory by writing inactive memory pages to secondary storage (e.g., eMMC). Kim et al. [136] proposed a swapping scheme (Application-Aware Swapping) that considers OS processes in the swapping decision. For example, swapping an application to secondary memory is not useful if the LMK is about to remove it from memory, freeing the used memory pages. Other approaches utilized NVM for swapping [132], [137].

Journaling in Android applies a write-twice behavior, to ensure reliability, which reduces system performance by additional write operations. Kim et al. [130] proposed an architecture to reduce storage accesses for journaling. They

use non-volatile memory for this purpose. Among others, Jeong et al. [131] eliminated the journaling of unnecessary metadata. Nguyen et al. [134] proposed iRAM, a system that cleans low-priority processes to maintain a high level of free memory. Kim and Bahn [135] evicted write-only-once data from the buffer cache to improve the utilization of cache space. Kim et al. [133] proposed an approach to group memory pages with the same lifetime to alleviate fragmentation of I/O buffers.

GPU buffers have been analyzed by Kwon et al. [26], who introduced a compressing scheme. Once an application goes to the background, its GPU buffers are treated as *inactive* and compressed. If the application is launched in the foreground, GPU buffers are decompressed.

Programming languages influence the choice of language constructs that further impact storage requirements. Escobar De La Torre and Cheon [138] analyzed the impact of the Java language constructs on the allocated memory. For instance, for−each loops require more memory than equivalent code snippets using regular loops. Removing those constructs (iterators, for−each loops, lambda expressions and the Stream API) reduces memory requirements [138]. Saborido et al. [45] showed that map implementations consume different amounts of memory. Specifically, ArrayMap uses less memory than HashMap.

6.3 Summary

Memory describes the occupation of device memory by applications, and is critical for resource-constraint systems [26], [29]. For Android applications, data is either shared between multiple applications, or loaded separately by each application.

Memory has been measured according to kernel memory footprints [119], garbage collection calls [51], physical memory dumps, and logging information [29]. Memory can be analyzed by tools provided by the Android OS [112], and external tools provided by researchers [59], [120], [121].

Memory consumption has been reduced by removing code smells from application source code [51]. In particular, memory leaks (e.g., constantly requesting memory [122], or keeping unused objects in memory [123]) have a negative impact on memory consumption.

Memory consumption has been further improved by changes in the Android OS. For example, garbage collection, which is used to free memory in Android, can use different strategies for freeing memory [110]. Another approach is the removal of redundant data from memory (deduplication) [29], [129]. Improvements can furthermore be achieved by changes in swapping [132], [136], [137] and journaling strategies [130], [131].

7 ENERGY

Embedded systems include several components that consume battery. CPU, LCD, GPS, audio and WiFi services are power-intensive components [25], [40]. Due to the limitation in battery size and stored energy [140], [141], [142], reducing energy consumption is gaining more and more relevancy [143]. In general, optimizing energy consumption depends on individual usage [7], [144].

The following sections outline methods and tools to profile energy consumption (Section 7.1) and reduce the energy consumption of mobile applications (Section 7.2). A summary is given in Section 7.3.

7.1 Profiling

Several measurements and prediction approaches have been used to profile energy consumption on mobile devices. Hoque et al. [145] discussed two ways to measure energy consumption: with external instruments and self-metering. This section gives an overview of respective profiling techniques.

A common approach to determine energy consumption is to investigate hardware components. Zhang et al. [25] measured power consumption using battery voltage sensors and knowledge of battery discharge behavior. Additionally, fuel gauge chips [146] and Battery Monitoring Unit [147] can be used to measure energy consumption. Other approaches use physical power meters to measure energy consumption [99], [148], [149], [150], [151]. Morales et al. [143] used a digital oscilloscope for high frequency energy measurements. Ferrari et al. [152] designed a Portable Open Source Energy Monitor (POEM) to measure energy consumption of applications at a control flow level. Bokhari et al. [153] built energy models based on CPU utilization and lines of code, as external meters can be expensive and not easy for developers to set these meters up.

Other than measuring energy consumption with physical devices, several studies make energy consumption estimates. Energy consumption estimates can be performed based on hardware utilization and system-calls [154], Android kernel monitoring [155], pixel information [156], [157], user behavior [7], data transmission-flow characteristics [158], code level [159], [160] and source-code line level [161].

Jabbarvand et al. [162] proposed COBWEB, a search-based technique to generate test suites for energy testing. These tests are able to execute energy-greedy parts of the code. The computational cost of such a testing technique can be reduced by test-suite minimization [163]. Mittal et al. [164] presented an emulation tool WATTSON to estimate energy consumption during application development. CPU time has been used as a proxy for energy consumption. However, it is not as accurate as other techniques, because voltage is scaled dynamically and multiple hardware components are used [161]. One should consider that errors during the measurement and estimation of energy consumption, as noise, can be introduced by various hardware components, such as a rising temperature of the battery [165]. This impacts the number of samples required for ensuring statistical significance when comparing the energy consumption of applications [166]. Validation approaches should consider the level of noise to compare solutions fairly [167].

7.2 Optimization Approaches

Several techniques have been applied to reduce the energy consumption of mobile applications, which are discussed in the following and summarized in Table 6.

Offloading, e.g., transferring computationally expensive tasks to external devices, can be used to reduce energy

TABLE 6
Studies on Energy Optimization.

Category	Authors [Ref]	Year	Venue
Offloading	Cuervo et al. [149]	2010	MobiSys
	Saarinen et al. [168]	2012	SIGCOMM
	Ding et al. [169]	2013	SECON
	Saarinen et al. [170]	2013	Mobicom
	Khairy et al. [171]	2013	IWCMC
	Kwon and Tilewich [172]	2013	ICSME
	Corral et al. [173]	2014	MobiWis
	Bolla et al. [174]	2014	NGMAST
	Qian and Andresen [175]	2015	IJNDC
	Das et al. [46]	2016	IACC
Prefetching	Balasubramanian et al. [176]	2009	SIGCOMM
	Chen et al. [177]	2013	SOSP
	Mohan et al. [178]	2013	EuroSys
	Yang and Cao [179]	2017	IEEE TCM
	Dutta and Vandermeer [180]	2017	TWEB
Antipatterns	Pathak et al. [181]	2011	HotNets
	Zhang et al. [141]	2012	CODES
	Pathak et al. [182]	2012	MobiSys
	Banerjee et al. [148]	2014	FSE
	Liu et al. [183]	2014	TSE
	Jabbarvand and Malek [184]	2017	FSE
Refactoring	Pathak et al. [185]	2012	Eurosys
	Anwer et al. [186]	2014	MobileSoft
	Alam et al. [187]	2014	DATE
	Li et al. [188]	2014	ICSE
	Linares et al. [156]	2015	FSE
	Bruce et al. [189]	2015	GECCO
	Cito et al. [146]	2016	ASE
	Banerjee and Roychoudhury [190]	2016	MobileSoft
	Cruz et al. [191]	2017	MobileSoft
	Banerjee et al. [192]	2017	TSE
	Morales et al. [143]	2017	TSE
	Cruz and Abreu [193]	2017	FSE
	Bokhari et al. [165]	2017	GECCO
	Cruz and Abreu [194]	2018	CIbSE
	Lyu et al. [47]	2018	ISSTA
Power states	Pyles et al. [150]	2011	UbiComp
	Kim et al. [195]	2012	ICOIN
	Ding et al. [196]	2013	SIGMETRICS
	Metri et al. [142]	2014	UbiComp
	Bokhari and Wagner [197]	2016	GECCO
	Rao et al. [198]	2017	HPCA
Displays	Dong et al. [199]	2009	DAC
	Anand et al. [200]	2011	MobiSys
	Lin et al. [201]	2012	TC
	Lin et al. [202]	2014	DAC
	Chen et al. [203]	2014	Computers & graphics
	Huang et al. [204]	2014	ISLPED
	Chen et al. [205]	2014	HotPower
	Nixon et al. [206]	2014	HotPower
	Li et al. [188]	2014	ICSE
	He et al. [207]	2015	Mobicom
	Lin et al. [208]	2017	ISLPED
	Lee et al. [209]	2018	ITMCCJ
	Chang et al. [210]	2019	DAC
	Lin et al. [211]	2019	DAC
CPU	Nagata et al. [99]	2012	UIC
	Bezerra et al. [212]	2013	PM2HW2N
	Chang et al. [213]	2013	TECS
	Tseng et al. [214]	2014	DAC
	Hsiu et al. [215]	2016	TECS
	Li and Mishra [216]	2016	J PARALLEL DISTR COM
	Muhuri et al. [217]	2019	IEEE Trans. Fuzzy Syst.
	Han and Lee [218]	2020	IEEE Access
APIs	Paek et al. [219]	2010	MobiSys
	Zhuang et al. [220]	2010	MobiSys
	Chon et al. [221]	2011	SenSys
	Oshin et al. [222]	2012	TrustCom
	Zhang et al. [223]	2013	IEEE Sensors
	Linares et al. [224]	2014	MSR
Protocols	Ra et al. [225]	2010	MobiSys
	Nurminen [226]	2010	CCNC
	Pyles et al. [227]	2012	UbiComp
	Lee et al. [228]	2012	IEEE Transactions
	Cheng and Hsiu [229]	2013	INFOCOM
	Siekkinen et al. [230]	2013	MoVid
	Li et al. [231]	2016	ICSE
System strategies	Chen et al. [232]	2015	Mobicom
	Martins et al. [233]	2015	ATC
Memory management	Duan et al. [40]	2011	IGCC
	Nguyen et al. [234]	2013	UbiComp
	Hussein et al. [235]	2015	Systor
	Zhong et al. [236]	2015	ITCSDI
Programming languages	Nagata et al. [99]	2012	UIC
	Saborido et al. [45]	2018	EMSE

consumption [149]. Cuervo et al. [149] developed MAUI, a system that supports automatic and developer-specified code offload. MAUI determines which method to execute remotely based on the current state of the device at runtime. Offloading decisions can be motivated by device status [175], execution times [171], network conditions [46], [168], [169], [170] or developer decisions [172]. Bolla et al. [174] proposed the concept of Application State Proxy (ASP) to offload entire applications. ASP transfers internet-based applications to other network devices, when they are kept in the background. As long as no new events occur (e.g., messages), applications

are kept in the proxy, which reduces the resource load on the smartphone. Corral et al. [173] applied offloading to matrix multiplication and image processing tasks to reduce energy consumption.

Prefetching, e.g., the caching of data transmissions and advertisements in advance, can be used to reduce energy consumption. Balasubramanian et al. [176] distinguished applications in delay-tolerant and applications that can benefit from prefetching, to decide which networking technology (3G, GSM, WiFi) to use. Mohan et al. [178] and Chen et al. [177] prefetched multiple ads to reduce energy consumption induced by downloads. Dutta and Vandermeer [180] achieved energy reductions with caching of up to 45%, even with small cache sizes (e.g., 250MB). Yang and Cao [179] formalized the prefetching for energy reductions as an optimization problem. Two approaches (greedy and discrete) are investigated to minimize energy consumption with regard to the network condition (LTE).

Antipatterns can be defects such as energy bugs that will likely drain energy. Pathak et al. [181] defined energy bugs as errors that cause the system to unexpectedly consume energy. Banerjee et al. [148] categorized energy inefficiencies into two categories: energy hotspots and energy bugs. Energy hotspots cause high battery consumption even though the hardware utilization is low. Energy bugs prevent the idle state of smartphones causing undesired battery consumption without user activity. Pathak et al. [181] categorized energy bugs caused by hardware (faulty battery, hardware damage) and software (OS, configurations, applications) and proposed a framework to detect the causes of energy bugs. Pathak et al. [182] focused on detecting a particular type of energy bug (no-sleep bug) via static analysis. A no-sleep bug occurs when application components are being kept active when a smartphone is in an idle state, without the necessity of being kept active. Banerjee et al. [148] created a framework that automatically generates tests to detect energy bugs. Each test contains a sequence of user interactions that are aimed at revealing energy bugs. As system calls are a primary source for energy bugs, a directed search is used to generate test cases containing system calls. Zhang et al. [141] developed ADEL (Automatic Detector of Energy Leaks), to identify energy leaks caused by network operations. Liu et al. [183] created GREENDROID, a tool that extends Java PathFinder (JPF) to automatically detect energy problems and report actionable information to combat these problems. Jabbarvand and Malek [184] proposed µDROID, a mutation testing framework, that can be used to detect energy inefficiencies. This framework uses 50 different mutation operators and the similarity of power traces between original application and mutants is used as the test oracle. The detection and removal of energy bugs is not simple, as high energy consumption in applications is not necessarily a sign for wasted computations [141].

Refactoring, for example, by using energy-efficient algorithms, can be used to reduce energy consumption.

Pathak et al. [185] manually restructured the source code of applications to make efficient use of high power states of components. They observed that applications consume I/O energy in distinct lumps. Bundling these lumps can reduce energy consumption. Similarly, Alam et al. [187] optimized the placement of wakelock calls. Lyu et al. [47] refactored

database operations to avoid inefficiencies and reduce energy consumption. Another approach is to change the choice of colors used in an application, as the power consumption of displays is effected by the displayed color [237]. This goes as far that some applications consume double the energy as they would do if colors were optimized for energy consumption [157]. Li et al. [188] proposed an approach to automatically change the colors used in web applications. Linares et al. [156], [238] used multi-objective optimization to reduce the energy consumption of GUIs, while offering visually similar colors to the original design. Bruce et al. [189] applied Genetic Improvement (GI) to find a more energy efficient version of applications. Mutation operations were applied to the source code of a Boolean satisfiability solver, to reduce energy consumption as a measure of fitness. Bokhari et al. [165] applied approximate computing on Rebound,[11] a Java Physics library, to achieve a trade-off between accuracy and energy consumption.

Another approach to automatically refactor applications is to follow energy efficiency guidelines [190], [191], [194]. Cito et al. [146] adapted application binaries to adjust the frequency of network requests to advertisements and analytics based on the battery status. Anwer et al. [186] adapted permissions and corresponding source code of applications based on user requirements, which can for example prevent the unconscious sending of an SMS.

Morales et al. [143] showed that there is a correlation between anti-patterns and energy consumption of mobile applications, and proposed the use of multi-objective search to find a set of refactoring sequences able to simultaneously improve code design quality (including the removal of energy smells) and reduce energy consumption. Banerjee et al. [192] performed an automatic repair of energy bugs with static and dynamic analysis. Cruz and Abreu [193] manually fixed antipatterns based on Android performance-based guidelines.

Power states determine the operating modes of hardware components, which require different amount of energy [154]. Power state transitions can be initiated by hardware components, but are usually performed by the OS [40]. As idle power consumption accounts for approximately 50% of the total energy consumption in a smartphone, it is suggested that using different power modes to shut down components is useful to reduce energy consumption [140]. Metri et al. [142] developed BATTERYEXTENDER, a tool that enables users to reconfigure device resources to reduce battery consumption. For this purpose, battery consumption of components is predicted with little computational overhead, by using energy profiling. Users are able to pick a period of time for which they want to reduce energy consumption and then choose which components to put in an idle power state to save energy. Bokhari and Wagner [197] proposed a framework to optimize default settings of smartphone components to reduce energy consumption. This problem is formulated as an optimization problem, to minimize energy consumption by changing settings of components based on user behavior. Rao et al. [198] dynamically selected system configurations (CPU frequency and memory bandwidth) that reduce energy consumption while maintaining a user-

---

11. https://github.com/facebookarchive/rebound

specified level of responsiveness. Ding et al. [196] determined power modes based on wireless signal strength, as a poor signal strength drains energy. Based on this, network traffic can be delayed under poor signal strength and continued when the network strength improves. Kim et al. [195] limited data transmissions to smartphones (e.g., text, image, and video) based on battery status. By minimizing the amount of transferred data in Social Networking Services, energy consumption can be reduced. For instance, Pyles et al. [150] switched WiFi to a low power or sleep mode during periods where it is not being used.

Displays are under constant energy consumption while mobile devices are used. Dong et al. [199] were the first to study the transformation of GUI colors of OLED displays to reduce energy consumption. Their automatic transformation can be applied on GUI elements (structured) or on pixel information (unstructured). Anand et al. [200] adjusted the brightness of screens to reduce the backlight level of the display. Lin et al. [201] reduced backlight energy consumption for mobile streaming applications, while other work dimmed areas of the screen [188], [203], [205], [208]. Other approaches include the adaptation of pixels [202], reduction of frame refreshes [204], [206], [209], [210], pixel density [206], [207] and resolution [211].

CPU clock frequency impacts energy consumption [140]. Nagata et al. [99] proposed a method that adjusts CPU clock frequency based on application requirements. Hsiu et al. [215] allocated computing resources based on the sensitivity of different applications. Application sensitivity states can be HIGH (interactive), MEDIUM (foreground) or LOW (background). Tseng et al. [214] adapted the allocation of CPU resources to applications based on their delay-sensitivity. Further approaches adjust the frequency and voltage of devices (Dynamic Voltage and Frequency Scaling) [212], [213], CPU frequency [218] or the number of cores [216]. Muhuri et al. [217] considered linguistic feedback from users to adapt CPU frequency accordingly. They proposed the approach Per-C for Personalized Power Management Approach (Per-C PPMA), which collects user feedback about their degree of satisfaction when using an application. This can be applied to not only reduce energy consumption, but also to improve user satisfaction.

APIs impact energy consumption, as Li et al. [239] showed that 91.4% of applications consume more than 60% of their energy with APIs. Linares et al. [224] analyzed usage patterns of "energy-greedy" APIs and give recipes to reduce energy consumption. To support their quantitative and qualitative exploration of API usage pattern, they mined thousands of method calls and API usage patterns. Among those, there are usage patterns that have an unavoidable, high energy consumption, and others which can be improved. An example for energy-greedy APIs is GPS. Paek et al. [219] adapted the rate of GPS, and only turns on GPS, when the current location estimate is uncertain. Turning on GPS indoors is also avoided. Other approaches reduce the sampling rate of GPS [220], [221], [222], [223].

Protocols can be used by mobile devices to optimize the energy consumption of networking technologies. Ra et al. [225] designed an algorithm to optimize the energy-delay trade-off of delay-tolerant applications that can benefit from low-energy WiFi connections. Energy can be reduced

if mobile traffic is delayed to a situation where WiFi is available [228]. Pyles et al. [227] saved energy by prioritizing WiFi traffic based on application priority. Li et al. [231] bundled HTTP requests to reduce energy consumption. Cheng and Hsiu [229] considered signal strength to reduce energy consumption when fetching location-based information. Nurminen [226] showed that parallel TCP downloading can be used to reduce energy consumption. Siekkinen et al. [230] reduced energy consumption of streaming applications by shaping LTE traffic into bursts. Hoque et al. [240] surveyed other approaches for optimizing the energy efficiency of streaming.

System strategies can be used to manage background processes. Martins et al. [233] introduced TAMER, an OS mechanism that allows rate-limiting of background processes to reduce energy consumption. TAMER imposes on events and signals that cause background applications to wakeup and thereby consume a higher amount of energy. Among others, TAMER can limit the frequency of notifications an application sends while it runs in the background. Chen et al. [232] avoided running applications in the background when they are not beneficial for user experience.

Memory management strategies can be used to change or increase memory to cope with higher requirements of applications. However, a larger main memory size leads to higher energy consumption [40], [132]. Energy reductions can be achieved by using non-volatile memory [236], Phase Change Memory [40] or adaptations to the garbage collection [235] and scheduling algorithms [234].

Programming Languages impact the responsiveness and energy consumption of applications. Nagata et al. [99] compared applications developed in different programming languages (Java, JNI and C) and showed that the energy consumption for JNI and C is smaller than for Java. A programming language construct that impacts energy consumption refers to maps (e.g., using `HashMap` over `ArrayMap` can reduce energy consumption by 16% [45]).

7.3 Summary

Energy consumption is a crucial characteristic of embedded systems since these devices have a limited battery size. Energy is consumed by applications (often by multiple applications at the same time), which use several components (e.g., GPS, audio, WiFi, display) [25], [40].

To profile energy consumption, two techniques have been pursued. On one hand, energy has been measured with either internal or external instruments [145]. On the other hand, energy consumption has been empirically estimated. For this purpose, various indicators have been investigated [154], [156], [161]. However, when profiling energy consumption, one should consider noise, which impacts the validity of measurements [165], [166], [167].

Many approaches have been proposed to optimize energy consumption. The majority of these approaches address the application source code (38/85). Changes to source code included the removal of bad coding practices (antipatterns) [181], [182], [192], or refactoring to include best practices [190], [191], [194]. Source code adaptation approaches can also focus on particular elements of devices, such as the display. For instance, energy consumption of displays has
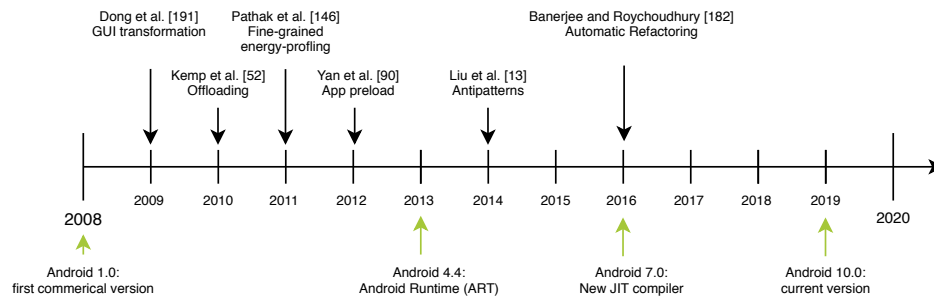
Fig. 4. Timeline of key ideas in non-functional performance optimization and changes to the Android OS.

been reduced by changing colors used in applications [156], [237], [238]. Adaptions have been also applied to displays directly. Thereby, display energy consumption has been reduced by changing brightness, colors and dimming [188], [199], [200], [203], [205], [208].

Similar to responsiveness optimization, energy inefficient computations have been offloaded to external devices with fewer computational restrictions [149]. Adapting the CPU clock frequency, to adjust computational speed, have also reduced energy consumption [99]. An adjustment of components can be performed with different power states (e.g., setting components in an idle state when they are not required).

8 DISCUSSION

This section provides an overview and discussion of the optimization approaches presented in previous sections (sections 4 to 7). In the following, we present a timeline of important methods and techniques proposed for advancing mobile applications' performance. Additionally, we discuss current challenges and opportunities in this field.

8.1 Timeline

Important changes to the Android Platform and publications concerned with optimization of non-functional performance characteristics are shown in the timeline in Figure 4. The first commercial Android version was introduced in 2008. In 2009, Dong et al. [199] studied the transformation of the GUI to reduce the energy consumption of mobile device displays.

In 2010, Kemp et al. [67] were the first that proposed an offloading framework for Android applications. This framework can be used to offload computationally expensive parts of mobile applications and improve both responsiveness and energy consumption.

In 2011, Pathak et al. [154] extended energy profiling of hardware utilization based on system-calls to provide fine-grained energy estimates. In 2012, Yan et al. [102] proposed an OS extension that preloads applications and application-specific content to improve the launch time of applications. In 2013, Android designers applied a big change to the Android platform, introducing ART over the previously used DVM. With this change, optimizations such as Ahead-Of-Time (AOT) compilation are applied by the Android platform to improve application performance.

In 2014, Liu et al. [13] characterized several types of performance bugs, which addressed different non-functional

characteristics of Android applications and caused excessive resource consumption of memory and battery. Examples of performance bugs include GUI lagging, energy leaks, and memory bloat. Based on the knowledge of bugs and antipatterns or guidelines, Banerjee and Roychoudhury [190] proposed the automated refactoring of application source code in 2016.

In 2016, Android 7.0-7.1 introduced a new JIT Compiler. This allowed faster application installations and it reduced the size of compiled code. The 10th major Android version was published in 2019.

8.2 Optimization Approaches per Android Layer

Reading the related work, we observed that approaches are applied to different layers of mobile devices. The distribution of these approaches differs between the four non-functional performance characteristics we investigated. In Figure 5, we organize optimization approaches based on layers (application, platform, hardware) to non-functional performance characteristics (i.e., responsiveness, launch time, memory and energy).

The majority of approaches to optimize responsiveness are application-based techniques and apply changes to the source code (e.g., antipatterns, refactoring or prefetching). Hardware and I/O optimization approaches are concerned with increasing the speed of reads and writes, faster fetching of bytecode as well as correct usages of CPU and GPU.

Launch time has not been improved itself by changes to applications, but, in majority, by changes in the Android platform. In particular, the preloading of applications [16], [38], [48], [49], [101], [102], [107], [108], [109], [241] and better choices for the LMK [15], [38], [48], [49], [104], [111], [112], [113], [114], to prevent cold starts from happening, are investigated frequently. The overall research direction is concerned with reducing the amount of cold starts rather than reducing the launch time itself.

Memory is mostly optimized by removing antipatterns [51], [52], [122], [124], [125] and applying different strategies for the garbage collection [39], [52], [96], [110], [126], [127], [128].

A great deal of research work optimizes energy consumption and almost every optimization category has been addressed for energy consumption. Unlike the optimization of responsiveness, launch time and memory usage, a large portion of approaches apply changes to the hardware, especially for screens. The power mode of components plays
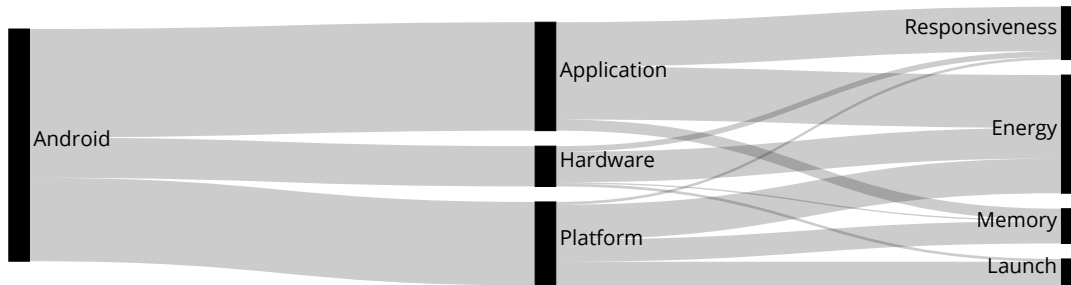
Fig. 5. Flow diagram of optimization approaches. Optimization approaches are matched with the layer of the embedded system (e.g., hardware, platform, and application). These layers are matched with corresponding, optimized non-functional performance characteristics (e.g., responsiveness, launch time, memory, and energy consumption). The width of the flows represent the number of publications with regards to a relationship.

an important role as well, which is governed by the Android OS. Furthermore, API usage has a huge impact on energy consumption. In particular, the use of GPS. Alike responsiveness, changes to application source code contribute to energy savings. Approaches like offloading optimize both, responsiveness and energy consumption.

8.3 Relationship of Optimization Approaches

A major trade-off that can be seen is between energy consumption and responsiveness [198], [216], [242], which should therefore be evaluated in union. One example for the relationship of responsiveness and energy consumption can be seen in the CPU clock frequency, as reducing it leads to less energy consumption, but lowers responsiveness and vice versa [99].

Considering application launch, hot starts are preferable over cold starts, not only to reduce the launch time but also to reduce energy consumption [26], [38], [243].

The choice of the memory system and storage technique used impacts energy consumption [40], [244]. One aspect of this is the size of memory, as a larger main memory size leads to higher energy consumption [40], [132]. Regardless, larger memory permits the preloading and prefetching of more application data, which can improve launch time, responsiveness and/or energy consumption. Evidence for potential improvement comes from the observation that the number of applications cached directly impacts the amount of blocked memory [111]. Memory leaks [243] or a full memory [15] limit the number of preloaded applications. Different memory management strategies for the LMK [49], [48] have shown to improve application launch.

While there is a relationship among non-functional characteristics, several optimization approaches improve more than one characteristic at the same time, which we will present in the following.

Gordon et al. [72], Chun et al. [68], Chen and Hao [74], Kemp et al. [67] and Kosta et al. [70] performed offloading to improve responsiveness, and as a result reduced energy consumption. Khairy et al. [171], and Qian and Andresen [175] proposed offloading systems aimed at reducing energy consumption, that reduced execution time as well. In order to achieve improvements with offloading, the saving has

to exceed the additional cost imposed by the offloading process [245].

Removing antipatterns can reduce the number of garbage collection calls as well [51], whereas memory leaks reduce available memory and responsiveness [125].

Nguyen et al. [50] improved both responsiveness and launch time by optimizing I/O read and write operations. Kim et al. [133] reduced CPU usage and energy consumption with a region-based physical memory management scheme. Hecht et al. [51] improved memory as well as user interface performance by correcting code smells. The approach of Han et al. [218] on CPU scheduling could not only reduce energy consumption but also application launch time. Hsiu et al. [215] reduced application response time by scheduling computing resources for energy reduction. Saborido et al. [45] showed that the choice of different map implementations impacts memory, energy and responsiveness. Lyu et al. [47] refactored database operations and thereby improved responsiveness and energy consumption. By reducing buffer cache, Kim and Bahn [135] reduced energy consumption at the same time.

Optimization approaches for responsiveness, launch time and memory can induce additional energy costs, if they extend the OS or require additional computations [29], [85], [95], [102], [107], [109], [129]. Overhead is not only produced by optimization approaches, but also by profiling and testing [60], [142], [160], [246], as well as by displaying ads [247] to generate revenue. In addition to causing overhead, non-functional improvements may come with a trade-off with regards to functional characteristics, such as the accuracy of algorithms [151].

8.4 Challenges

Reflecting on the different optimization approaches and the relationship between non-functional performance characteristics, we have identified challenges and opportunities for future work. We begin by outlining three challenges that developers face when optimizing non-functional performance characteristics, followed by an overview of future work targeted by the surveyed publications, and by opportunities we detected.

Cross-characteristic dependencies. Section 8.3 shows that a challenge that developers face while optimizing

applications' performance refers to the handling of the dependencies among different performance characteristics. This means that while improving one characteristic, they may decrease the performance of another [40], [99], [132]. For example, optimization approaches that extend the OS or invoke additional computation cause additional energy consumption [29], [85], [95], [102], [107], [109], [129]. Therefore, there should be a balance that developers need to achieve among performance characteristics. While we pointed out works that achieved improvements in more than one performance characteristic (Section 8.3), it would be interesting if adverse relationships and trade-offs between non-functional performance characteristics receive more attention. One example for this is the energy consumption and responsiveness trade-off when adjusting CPU clock frequency [99].

Testing cost. There are different definitions of the four non-functional characteristics to determine how performance is measured (e.g., responsiveness measured in ms [56], [57] or frames [51], or different definitions of application launch completion [101], [102]). Another problem arises with noisy measurements (e.g., due to hardware components [165]), as we have seen in Section 7. Therefore, testing should consider variance in measurements to ensure statistical significance [166], [167].

An attempt to reduce the cost of testing is the usage of emulators [123], [164] or prediction of non-functional performance without executing an application. Prediction of performance has been applied for responsiveness [57], [66] and energy consumption [7], [154], [155], [156], [157], [158], [159], [160], [161].

User satisfaction. While improvements in non-functional characteristics without performance deterioration in other characteristics, can always be seen as something positive, quantifying the impact of improvements on user satisfaction remains challenging.

Two of the surveyed publications attempted to tackle this issues. For example, Muhuri et al. [217] collected linguistic feedback about user satisfaction (e.g., ranging from "very low" to "extremely high") to examine the impact of performance on user satisfaction. This information has been used to adapt the CPU frequency. Zhao et al. [106] stated that collecting user feedback can be costly and inconvenient for developers. To overcome this issue, they mapped a user-perceived satisfaction score about launch times to the actual launch time delay, which is easily measurable. Such an incorporation of user satisfaction in the optimization procedure, could prove to be an interesting consideration for future work on other performance characteristics.

In addition to these challenges, the reviewed publications distinguished several fields of future work, including:

- Improvement and extension of prediction methods (e.g., for offloading, prefetching, and preloading) [38], [66], [67], [86], [115], [116], [132], [221];
- Investigation of antipatterns [13], [51], [58], [123], [224];
- Automation of the optimization process [51], [148]
- Extension of testing (e.g., usage of more devices and applications) [51], [183], [207];
- Improvement of measurements (profiling) [132], [189].

In particular, extensions of prediction methods include the consideration of additional information, such as the context (e.g., location, time) [38], behavioral patterns [86] and information about remote resources for offloading (e.g., processor speed, available memory) [67]. Future work on antipatterns is concerned with investigating a broader range of antipatterns [51], [58] and discovering new antipatterns or categories [13], [123], [224]. Automation could be applied to time-consuming tasks, such as finding causes for energy wastage [148] or the correction of antipatterns [148].

Based on our results, including Figure 5, we identified further gaps in the literature. For once, changes to applications are not investigated for launch time improvements. It could be investigated whether antipatterns, that exist for responsiveness, memory and energy usage, exist for application launch as well.

Automatic refactoring has been applied for both responsiveness and energy, individually. It could therefore be interesting to apply refactoring in a multi-objective setting, to optimize both. Lastly, changes to the platform have scarcely been used to improve responsiveness. There could be the potential to apply ideas from other non-functional characteristics, such as APIs for energy consumption.

9 RELATED WORK

In the following, we give an overview of literature related to the non-functional performance optimization of Android applications. At first, we look into optimization approaches for software engineering. We furthermore describe the developer and user perspective on mobile-application optimization.

9.1 Optimization in Software Engineering

This survey outlined optimization approaches for mobile applications. In the following, we present studies and insights on optimization with regards to software engineering.

During the development and optimization of software, it is important to consider software requirements [248], which can contain functional, non-functional, business and user requirements. Different techniques for prioritizing requirements [249], [250] can be applied to determine the importance of non-functional over functional characteristics.

Nonetheless, the performance of software is difficult to be measured, as it is pervasive and affected by various different aspects (e.g., the platform used) [251]. Software Performance Engineering (SPE) is an approach to measure and improve system performance [251]. SPE subsumes software engineering activities that are applied to meet performance requirements and achieve improvements. Profiling tools can be used to measure performance (e.g., for energy consumption [252], [253], GPUs [254], responsiveness [255], and memory [256]).

Building upon performance profilers, optimization and improvement techniques can be applied to software. Petke et al. [33] conducted a survey on genetic improvement of software. They mention improvements for non-functional characteristics for energy and memory consumption as well as functional improvements including repairs and addition of new functionalities.

Following, we outline representative publications on software optimization. Mao and Humphrey [257] analyzed

long, unexpected startup times of Virtual Machines on the cloud. Kaminaga [258] discussed techniques to achieve a faster startup time for embedded Linux systems. Van Emden and Moonen [259] analyzed code smells in Java source code. Baer and Chen [260] proposed a hardware scheme for preloading data. Sahin et al. [261] analyzed the impact of code refactoring on the energy consumption of 197 applications. Chen et al. [262] used flash drives in mobile computers for caching and prefetching data to save energy. Chen et al. [263] reduced writes to memory by using deduplication. Hauswirth and Chilimbi [264] detected memory leaks with low computational overhead. Padmanabhan and Mogul [265] investigated latency reductions by prefetching web documents.

9.2 Developer Perspective

In the following, we outline the developer perspective on optimization and development of mobile applications, including challenges during the development process. While certain characteristics of smartphones and PCs are similar [105], developers encounter differences to conventional software development as mobile applications are smaller than traditional software [266].

Developers use tools to support the development of applications [267], profiling, and debugging [268]. Static analysis can be used to support developers to find bugs and inspect code [269]. Other tools perform security assessment, automated test case generation and detection of non-functional issues such as energy consumption [270], [271]. While fixing non-functional performance bugs, developers need to consider the threat of introducing functional bugs [272] and hindering code maintainability [273]. In this context, Linares et al. [274] suggested that developers rarely implement micro-optimizations (e.g., changes at statement level).

When developing an application, developers need to decide which and how many platforms to use (e.g., Android, iOS, Windows OS). Note that each platform faces non-functional issues alike (e.g., antipatterns can be found in iOS [275], [276]). This impacts the development effort, as multiple codebases need to be maintained. Development of applications for multiple platforms can be supported by cross-platform tools (CPTs) [277]. By doing so, developers compromise between user experience and the ability to publish an application on multiple platforms. Willocx et al. [55] found that CPTs lead to an increased launch time of applications.

Furthermore, devices vary in their available memory, CPU, and display size, which has implications on application performance [278]. Therefore, developers need to test their applications on multiple devices.

Further changes and added functionality to the Android OS can be imposed by phone vendors [279]. This leads to varying behavior across different smartphone types [280]. Khalid et al. [281] analyzed the number of different phones that use applications in order to help developers to decide how many devices to use when testing applications, as the rating varies for different phone types. Often, applications run on more than hundred different phone types, implying a huge computational effort if all devices would be tested for.

They found that around a third of the devices account for 80% of the reviews and thereby usage, which can be used to prioritize which devices to use during testing. Lu et al. [282] prioritized devices to test applications based on the amount of user activity rather than the number of devices.

In order to analyze the performance of applications, the Google Play Store provides developers with pre-launch reports after an application is published. Tests are carried out on different devices and for up to five languages [283]. Another tool that developers can use to judge the quality and performance of applications are Android Vitals. Statistics including battery consumption, and crashes are collected from real users and reported to developers [30]. Comparisons with regard to non-functional performance characteristics, such as energy consumption, can also be measured with applications of the same category [284].

Frequently, developers use user reviews and test applications manually to detect performance issues and bugs [268]. Developers can change applications based on user reviews. Those reviews contain among others information about performance, bugs, problems or new features [3], [31]. There is a huge number of reviews that are written for applications, where some of them contain relevant information for developers [285]. For this purpose, Chen et al. [19] developed a framework to filter informative reviews by applying text mining and ranking methods. Furthermore, reviews can be analyzed for trends [286], [287] and emerging issues [288].

Even though developer identities do not often impact the choice of applications (only 11% of users choose an application based on who developed it [2]), they significantly impact the quality and success of applications [1]. Other factors that are correlated with the rating of applications include the apk size, minimum required SDK, and number of images on the application description page [289].

9.3 User Perspective

In the context of mobile applications, user shows a high degree of individualism [290]. Therefore, not every optimization approach can be applied in a general fashion. This is why understanding user behavior and differences among users and user groups is important when improving performance. Ultimately it is the user who decides the changes to an application result in a higher level of satisfaction.

The user-perceived quality of an application is not only determined by the application itself, but also by the device and attributes of its components [291]. Aspects that lead to the most negative complaints in reviews are related to privacy, hidden costs and features of the application [3].

Users' behavior varies in terms of number of interactions, amount of data received, interaction length and number of applications used [7]. Some users even show addictive behavior [292]. Approaches that aim to improve user experience (e.g., by reducing energy consumption or improving the responsiveness), should therefore be adaptive to user behavior. Making matters more difficult to predict, usage patterns can change within a few days [117].

Additionally, user behavior across countries shows significant differences as well [2]. Users of different countries prioritize other aspects of applications and show variations in rating applications as well as writing reviews [2]. For

example, users in China are more likely to rate an application, while users in Brazil are more likely to abandon a slow or buggy application. While there exist differences among Android users based on countries, a study on application launch performed by Morrison et al. [293] showed that similarities between Android and iOS exist.

Application usage can be seen as a sequence, with multiple applications used consecutively in a short period of time, whereas 68.2% of sequences only contain a single application [115]. Application usage varies based on the context, such as location [294] and/or time [115], [295].

10 THREATS TO VALIDITY

In this section, we discuss potential threats to our survey based on internal and external validity.

Internal validity refers to problems of our methods that could threat the validity of results and claims made in this survey [9]. A potential threat to internal validity is the completeness of the reviewed literature. Ideally, every relevant publication is included after the search process; however, the risk of missing a publication cannot be eliminated. A relevant publication can be missed if the respective data source containing the publication is not properly searched. Another cause for missing a publication is that corresponding keywords are not included in our searching criteria. To address both causes, we performed a preliminary search to gather relevant keywords and venues to guide the literature search. Moreover, two authors independently carried out the filtering process and their results were cross-checked, in order to ensure reliability and reduce researchers' bias. A different threat to internal validity refers to the precision of our results (e.g., the inclusion of irrelevant publications). To mitigate the impact of irrelevant publications, we check the title, abstract, and body of the publications examined as outlined in Section 3.1.3. Furthermore, there is a risk of drawing incorrect conclusions or claims. For this purpose, every stage of the search and analysis of results (e.g., repository search and categorization of approaches) has been performed by one author and cross-checked by another.

External validity describes the generalizability of our results outside of the given scope [296]. A potential, external threat is that the chosen non-functional performance characteristics are insufficient to describe optimization techniques for embedded systems. Through our preliminary search and analysis of related work (Section 9) we found that the selected four non-functional characteristics (responsiveness, launch time, memory and energy consumption) are representative. Another threat is the applicability of our study to other mobile platforms (e.g., iOS). While there are differences between iOS and Android, the general organization and functionality within embedded systems remain the same. We therefore argue that our categorization can be applied to mobile platforms, other than Android.

11 CONCLUSIONS

In this paper, we have provided an overview of the existing research work on non-functional performance optimization for Android applications published between 2008 and 2020. Our survey presents optimization approaches for non-functional performance characteristics (e.g., responsiveness,

launch time, memory, and energy). It also shows relationships among these characteristics, and identifies research gaps for potential future works. We hope that this survey will help researchers and developers to have a holistic perception on optimization approaches for mobile devices, the impact of these approaches, and the significance of different performance characteristics.

ACKNOWLEDGMENTS

REFERENCES

[1] V. Inukollu, D. Keshamoni, T. Kang, and M. Inukollu, "Factors influencing quality of mobile apps: Role of mobile app development life cycle," *International Journal of Software Engineering Applications*, vol. 5, 10 2014.

[2] S. L. Lim, P. J. Bentley, N. Kanakam, F. Ishikawa, and S. Honiden, "Investigating country differences in mobile app user behavior and challenges for software engineering," *IEEE Transactions on Software Engineering*, vol. 41, no. 1, pp. 40–64, 2014.

[3] H. Khalid, E. Shihab, M. Nagappan, and A. E. Hassan, "What do mobile app users complain about?" *IEEE Software*, vol. 32, no. 3, pp. 70–77, 2014.

[4] A. AlSubaihin, F. Sarro, S. Black, L. Capra, and M. Harman, "App store effects on software engineering practices," *IEEE Transactions on Software Engineering*, pp. 1–1, 2019.

[5] A. Mazuera-Rozo, C. Trubiani, M. Linares-Vásquez, and G. Bavota, "Investigating types and survivability of performance bugs in mobile apps," *Empirical Software Engineering*, pp. 1–43, 2020.

[6] A. Finkelstein, M. Harman, Y. Jia, W. Martin, F. Sarro, and Y. Zhang, "Investigating the relationship between price, rating, and popularity in the blackberry world app store," *Information and Software Technology*, vol. 87, pp. 119–139, 2017.

[7] H. Falaki, R. Mahajan, S. Kandula, D. Lymberopoulos, R. Govindan, and D. Estrin, "Diversity in smartphone usage," in *Proceedings of the 8th international conference on Mobile systems, applications, and services*. ACM, 2010, pp. 179–194.

[8] P. Welke, I. Andone, K. Blaszkiewicz, and A. Markowetz, "Differentiating smartphone users by app usage," in *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing*. ACM, 2016, pp. 519–523.

[9] M. Kechagia, D. Mitropoulos, and D. Spinellis, "Charting the api minefield using software telemetry data," *Empirical Softw. Engg.*, vol. 20, no. 6, pp. 1785–1830, 12 2015. [Online]. Available: http://dx.doi.org/10.1007/s10664-014-9343-7

[10] C. Hu and I. Neamtiu, "Automating gui testing for android applications," in *Proceedings of the 6th International Workshop on Automation of Software Test*, 2011, pp. 77–83.

[11] P. Bhattacharya, L. Ulanova, I. Neamtiu, and S. C. Koduru, "An empirical analysis of bug reports and bug fixing in open source android apps," in *2013 17th European Conference on Software Maintenance and Reengineering*. IEEE, 2013, pp. 133–143.

[12] W. J. Martin, F. Sarro, and M. Harman, "Causal impact analysis for app releases in google play," in *Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2016, Seattle, WA, USA, November 13-18, 2016*, T. Zimmermann, J. Cleland-Huang, and Z. Su, Eds. ACM, 2016, pp. 435–446.

[13] Y. Liu, C. Xu, and S.-C. Cheung, "Characterizing and detecting performance bugs for smartphone applications," in *Proceedings of the 36th International Conference on Software Engineering*. ACM, 2014, pp. 1013–1024.

[14] F. Ferrucci, C. Gravino, P. Salza, and F. Sarro, "Investigating functional and code size measures for mobile applications: A replicated study," in *International Conference on Product-Focused Software Process Improvement*. Springer, 2015, pp. 271–287.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2021.3071193, IEEE Transactions on Software Engineering

19

[15] H. Kim, H. Lim, D. Manatunga, H. Kim, and G.-H. Park, "Accelerating application start-up with nonvolatile memory in android systems," *IEEE Micro*, vol. 35, no. 1, pp. 15–25, 2015.

[16] A. L. N. Martins, C. A. Duarte, and J. Jeong, "Improving application launch performance in smartphones using recurrent neural network," in *Proceedings of the 2018 International Conference on Machine Learning Technologies*. ACM, 2018, pp. 58–62.

[17] C. Gao, J. Zeng, F. Sarro, M. R. Lyu, and I. King, "Exploring the effects of ad schemes on the performance cost of mobile phones," in *Proceedings of the 1st International Workshop on Advances in Mobile App Analysis, A-Mobile@ASE 2018, Montpellier, France, September 4, 2018*, L. Li, G. Meng, J. Klein, and S. Malek, Eds. ACM, 2018, pp. 13–18.

[18] C. Gao, J. Zeng, F. Sarro, D. Lo, M. R. Lyu, and I. King, "Do users care about ad's performance costs? exploring the effects of the performance costs of in-app ads on user experience," *Information and Software Technology*, 2020.

[19] N. Chen, J. Lin, S. C. Hoi, X. Xiao, and B. Zhang, "Ar-miner: mining informative reviews for developers from mobile app marketplace," in *Proceedings of the 36th International Conference on Software Engineering*. ACM, 2014, pp. 767–778.

[20] A. Banerjee and A. Roychoudhury, "Future of mobile software for smartphones and drones: Energy and performance," in *2017 IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*, 2017, pp. 1–12.

[21] M. Hort, M. Kechagia, F. Sarro, and M. Harman, "On-line appendix for the paper "A Survey of Performance Optimization for Mobile Applications"." [Online]. Available: https://solar.cs.ucl.ac.uk/os/appoptimization.html

[22] ——, "Living survey for the paper "A Survey of Performance Optimization for Mobile Applications"." [Online]. Available: https://solar.cs.ucl.ac.uk/appoptimization.github.io/

[23] R. Cohen and T. Wang, *Overview of Embedded Application Development for Intel Architecture*. Berkeley, CA: Apress, 2014, pp. 1–18. [Online]. Available: https://doi.org/10.1007/978-1-4842-0100-8_1

[24] N. D. Lane, E. Miluzzo, H. Lu, D. Peebles, T. Choudhury, and A. T. Campbell, "A survey of mobile phone sensing," *Comm. Mag.*, vol. 48, no. 9, pp. 140–150, 9 2010.

[25] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, and L. Yang, "Accurate online power estimation and automatic battery behavior based power model generation for smartphones," in *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*. ACM, 2010, pp. 105–114.

[26] S. Kwon, S.-H. Kim, J.-S. Kim, and J. Jeong, "Managing gpu buffers for caching more apps in mobile systems," in *Proceedings of the 12th International Conference on Embedded Software*. IEEE Press, 2015, pp. 207–216.

[27] A. Arcangeli, I. Eidus, and C. Wright, "Increasing memory density by using ksm," in *Proceedings of the linux symposium*. Citeseer, 2009, pp. 19–28.

[28] N. Gupta, "Compcache: in-memory compressed swapping," *Retrieved at*, p. 6, 2009.

[29] B. Lee, S. M. Kim, E. Park, and D. Han, "Memscope: analyzing memory duplication on android systems," in *Proceedings of the 6th Asia-Pacific Workshop on Systems*. ACM, 2015, p. 19.

[30] J. Harty and M. Müller, "Better android apps using android vitals," in *Proceedings of the 3rd ACM SIGSOFT International Workshop on App Market Analytics*. ACM, 2019, pp. 26–32.

[31] A. Di Sorbo, S. Panichella, C. V. Alexandru, J. Shimagaki, C. A. Visaggio, G. Canfora, and H. C. Gall, "What would users change in my app? summarizing app reviews for recommending software changes," in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, 2016, pp. 499–510.

[32] M. Harman, Y. Jia, and Y. Zhang, "App store mining and analysis: MSR for app stores," in *Proceedings of the 9th IEEE Working Conference on Mining Software Repositories*. IEEE Press, 2012, pp. 108–111.

[33] J. Petke, S. O. Haraldsson, M. Harman, W. B. Langdon, D. R. White, and J. R. Woodward, "Genetic improvement of software: a comprehensive survey," *IEEE Transactions on Evolutionary Computation*, vol. 22, no. 3, pp. 415–432, 2017.

[34] B. W. Boehm, *Characteristics of software quality*. North-Holland, 1978, vol. 1.

[35] G.-C. Roman, "A taxonomy of current issues in requirements engineering," *Computer*, no. 4, pp. 14–23, 1985.

[36] R. B. Grady and D. L. Caswell, *Software metrics: establishing a company-wide program*. Prentice-Hall, Inc., 1987.

[37] L. Chung and J. C. S. do Prado Leite, "On non-functional requirements in software engineering," in *Conceptual modeling: Foundations and applications*. Springer, 2009, pp. 363–379.

[38] W. Song, Y. Kim, H. Kim, J. Lim, and J. Kim, "Personalized optimization for android smartphones," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 13, no. 2s, p. 60, 2014.

[39] Y. He, C. Yang, and X.-F. Li, "Improve google android user experience with regional garbage collection," in *IFIP International Conference on Network and Parallel Computing*. Springer, 2011, pp. 350–365.

[40] R. Duan, M. Bi, and C. Gniady, "Exploring memory energy optimizations in smartphones," in *2011 International Green Computing Conference and Workshops*. IEEE, 2011, pp. 1–8.

[41] B. Kitchenham, "Procedures for performing systematic reviews," *Keele, UK, Keele University*, vol. 33, no. 2004, pp. 1–26, 2004.

[42] A. Sadeghi, H. Bagheri, J. Garcia, and S. Malek, "A taxonomy and qualitative comparison of program analysis techniques for security assessment of android software," *IEEE Transactions on Software Engineering*, vol. 43, no. 6, pp. 492–530, 2016.

[43] W. Martin, F. Sarro, Y. Jia, Y. Zhang, and M. Harman, "A survey of app store analysis for software engineering," *IEEE transactions on software engineering*, vol. 43, no. 9, pp. 817–847, 2016.

[44] C. Wohlin, "Guidelines for snowballing in systematic literature studies and a replication in software engineering," in *Proceedings of the 18th international conference on evaluation and assessment in software engineering*, 2014, pp. 1–10.

[45] R. Saborido, R. Morales, F. Khomh, Y.-G. Guéhéneuc, and G. Antoniol, "Getting the most from map data structures in android," *Empirical Software Engineering*, vol. 23, no. 5, pp. 2829–2864, 2018.

[46] P. K. Das, S. Shome, and A. K. Sarkar, "Apps: Accelerating performance and power saving in smartphones using code offload," in *2016 IEEE 6th International Conference on Advanced Computing (IACC)*. IEEE, 2016, pp. 759–765.

[47] Y. Lyu, D. Li, and W. G. Halfond, "Remove rats from your code: automated optimization of resource inefficient database writes for mobile applications," in *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2018, pp. 310–321.

[48] Y.-F. Chung, Y.-T. Lo, and C.-T. King, "Enhancing user experiences by exploiting energy and launch delay trade-off of mobile multimedia applications," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 12, no. 1s, pp. 1–19, 2013.

[49] J. Lee, K. Lee, E. Jeong, J. Jo, and N. B. Shroff, "Cas: Context-aware background application scheduling in interactive mobile systems," *IEEE Journal on Selected Areas in Communications*, vol. 35, no. 5, pp. 1013–1029, 2017.

[50] D. T. Nguyen, G. Zhou, G. Xing, X. Qi, Z. Hao, G. Peng, and Q. Yang, "Reducing smartphone application delay through read/write isolation," in *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*, 2015, pp. 287–300.

[51] G. Hecht, N. Moha, and R. Rouvoy, "An empirical study of the performance impacts of android code smells," in *Proceedings of the International Conference on Mobile Software Engineering and Systems*. ACM, 2016, pp. 59–69.

[52] K. Tasneem, A. Siddiqui, and A. Liaquat, "Android memory optimization," *International Journal of Computer Applications*, vol. 975, pp. 36–43, 2019.

[53] S. Yang, D. Yan, and A. Rountev, "Testing for poor responsiveness in android applications," in *2013 1st international workshop on the engineering of mobile-enabled systems (MOBS)*. IEEE, 2013, pp. 1–6.

[54] N. Tolia, D. G. Andersen, and M. Satyanarayanan, "Quantifying interactive user experience on thin clients," *Computer*, vol. 39, no. 3, pp. 46–52, 2006.

[55] M. Willocx, J. Vossaert, and V. Naessens, "Comparing performance parameters of mobile app development strategies," in *2016 IEEE/ACM International Conference on Mobile Software Engineering and Systems (MOBILESoft)*. IEEE, 2016, pp. 38–47.

[56] J. Huang, Q. Xu, B. Tiwana, Z. M. Mao, M. Zhang, and P. Bahl, "Anatomizing application performance differences on smartphones," in *Proceedings of the 8th international conference on Mobile systems, applications, and services*. ACM, 2010, pp. 165–178.

[57] J. Zhang, X. Wang, and Y. Chen, "Android app performance detection framework based on dynamic analysis of function call

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2021.3071193, IEEE Transactions on Software Engineering

20

graphs," in *Proceedings of the 2019 The World Symposium on Software Engineering*, 2019, pp. 1–5.

[58] W. Zhao, Z. Ding, M. Xia, and Z. Qi, "Systematically testing and diagnosing responsiveness for android apps," in *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, pp. 449–453.

[59] M. Cho, H. J. Lee, M. Kim, and S. W. Kim, "Androscope: An insightful performance analyzer for all software layers of the android-based systems," *ETRI Journal*, vol. 35, no. 2, pp. 259–269, 2013.

[60] L. Ravindranath, J. Padhye, S. Agarwal, R. Mahajan, I. Obermiller, and S. Shayandeh, "Appinsight: mobile app performance monitoring in the wild," in *Presented as part of the 10th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 12)*, 2012, pp. 107–120.

[61] D. K. Hong, A. Nikravesh, Z. M. Mao, M. Ketkar, and M. Kishinevsky, "Perfprobe: a systematic, cross-layer performance diagnosis framework for mobile platforms," in *2019 IEEE/ACM 6th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*. IEEE, 2019, pp. 50–61.

[62] H. Kim, B. Choi, and W. E. Wong, "Performance testing of mobile applications at the unit test level," in *2009 Third IEEE International Conference on Secure Software Integration and Reliability Improvement*. IEEE, 2009, pp. 171–180.

[63] Y. Kang, Y. Zhou, H. Xu, and M. R. Lyu, "Persisdroid: Android performance diagnosis via anatomizing asynchronous executions," *arXiv preprint arXiv:1512.07950*, 2015.

[64] ——, "Diagdroid: Android performance diagnosis via anatomizing asynchronous executions," in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2016, pp. 410–421.

[65] Y. Wang and A. Rountev, "Profiling the responsiveness of android applications via automated resource amplification," in *2016 IEEE/ACM International Conference on Mobile Software Engineering and Systems (MOBILESoft)*. IEEE, 2016, pp. 48–58.

[66] Y. Kwon, S. Lee, H. Yi, D. Kwon, S. Yang, B-G. Chun, L. Huang, P. Maniatis, M. Naik, and Y. Paek, "Mantis: Automatic performance prediction for smartphone applications," in *Proceedings of the 2013 USENIX Conference on Annual Technical Conference*, ser. USENIX ATC'13, 2013, pp. 297–308.

[67] R. Kemp, N. Palmer, T. Kielmann, and H. Bal, "Cuckoo: a computation offloading framework for smartphones," in *International Conference on Mobile Computing, Applications, and Services*. Springer, 2010, pp. 59–79.

[68] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: elastic execution between mobile device and cloud," in *Proceedings of the sixth conference on Computer systems*, 2011, pp. 301–314.

[69] M.-R. Ra, A. Sheth, L. Mummert, P. Pillai, D. Wetherall, and R. Govindan, "Odessa: enabling interactive perception applications on mobile devices," in *Proceedings of the 9th international conference on Mobile systems, applications, and services*. ACM, 2011, pp. 43–56.

[70] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, "Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading," in *2012 proceedings IEEE Infocom*. IEEE, 2012, pp. 945–953.

[71] M. S. Gordon, D. A. Jamshidi, S. Mahlke, Z. M. Mao, and X. Chen, "{COMET}: Code offload by migrating execution transparently," in *Presented as part of the 10th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 12)*, 2012, pp. 93–106.

[72] M. S. Gordon, D. K. Hong, P. M. Chen, J. Flinn, S. Mahlke, and Z. M. Mao, "Accelerating mobile applications through flip-flop replication," in *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*. ACM, 2015, pp. 137–150.

[73] R. Montella, S. Kosta, D. Oro, J. Vera, C. Fernández, C. Palmieri, D. Di Luccio, G. Giunta, M. Lapegna, and G. Laccetti, "Accelerating linux and android applications on low-power devices through remote gpgpu offloading," *Concurrency and Computation: Practice and Experience*, vol. 29, no. 24, p. e4286, 2017.

[74] M. Chen and Y. Hao, "Task offloading for mobile edge computing in software defined ultra-dense network," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 3, pp. 587–597, 2018.

[75] G. Jin, L. Song, X. Shi, J. Scherpelz, and S. Lu, "Understanding and detecting real-world performance bugs," *ACM SIGPLAN Notices*, vol. 47, no. 6, pp. 77–88, 2012.

[76] A. Nistor, L. Song, D. Marinov, and S. Lu, "Toddler: Detecting performance problems via similar memory-access patterns," in *Proceedings of the 2013 International Conference on Software Engineering*. IEEE Press, 2013, pp. 562–571.

[77] T. Ongkosit and S. Takada, "Responsiveness analysis tool for android application," in *Proceedings of the 2nd International Workshop on Software Development Lifecycle for Mobile*, 2014, pp. 1–4.

[78] G. Hecht, O. Benomar, R. Rouvoy, N. Moha, and L. Duchien, "Tracking the software quality of android applications along their evolution (t)," in *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2015, pp. 236–247.

[79] S. Habchi, X. Blanc, and R. Rouvoy, "On adopting linters to deal with performance concerns in android apps," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, 2018, pp. 6–16.

[80] W. Li, Y. Jiang, C. Xu, Y. Liu, X. Ma, and J. Lü, "Characterizing and detecting inefficient image displaying issues in android apps," in *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2019, pp. 355–365.

[81] Y. Lin, C. Radoi, and D. Dig, "Retrofitting concurrency for android applications through refactoring," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, 2014, pp. 341–352.

[82] S. Okur, D. L. Hartveld, D. Dig, and A. v. Deursen, "A study and toolkit for asynchronous programming in c#," in *Proceedings of the 36th International Conference on Software Engineering*, 2014, pp. 1117–1127.

[83] Y. Lin, S. Okur, and D. Dig, "Study and refactoring of android asynchronous programming (t)," in *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2015, pp. 224–235.

[84] R. Feng, G. Meng, X. Xie, T. Su, Y. Liu, and S.-W. Lin, "Learning performance optimization from code changes for android apps," in *2019 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. IEEE, 2019, pp. 285–290.

[85] B. D. Higgins, J. Flinn, T. J. Giuli, B. Noble, C. Peplin, and D. Watson, "Informed mobile prefetching," in *Proceedings of the 10th international conference on Mobile systems, applications, and services*. ACM, 2012, pp. 155–168.

[86] Y. Zhao, M. S. Laser, Y. Lyu, and N. Medvidovic, "Leveraging program analysis to reduce user-perceived latency in mobile applications," in *Proceedings of the 40th International Conference on Software Engineering*, 2018, pp. 176–186.

[87] B. Choi, J. Kim, D. Cho, S. Kim, and D. Han, "Appx: an automated app acceleration framework for low latency mobile app," in *Proceedings of the 14th International Conference on emerging Networking EXperiments and Technologies*, 2018, pp. 27–40.

[88] I. Malavolta, F. Nocera, P. Lago, and M. Mongiello, "Navigation-aware and personalized prefetching of network requests in android apps," in *2019 IEEE/ACM 41st International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*. IEEE, 2019, pp. 17–20.

[89] L. Batyuk, A.-D. Schmidt, H.-G. Schmidt, A. Camtepe, and S. Albayrak, "Developing and benchmarking native linux applications on android," in *International Conference on Mobile Wireless Middleware, Operating Systems, and Applications*. Springer, 2009, pp. 381–392.

[90] S. Lee and J. W. Jeon, "Evaluating performance of android platform using native c for embedded systems," in *ICCAS 2010*. IEEE, 2010, pp. 1160–1163.

[91] J. K. Lee and J. Y. Lee, "Android programming techniques for improving performance," in *2011 3rd International Conference on Awareness Science and Technology (iCAST)*. IEEE, 2011, pp. 386–389.

[92] C.-M. Lin, J.-H. Lin, C.-R. Dow, and C.-M. Wen, "Benchmark dalvik and native code for android system," in *2011 Second International Conference on Innovations in Bio-inspired Computing and Applications*. IEEE, 2011, pp. 320–323.

[93] C. Wang, M. Cintra, and Y. Wu, "Acceldroid: Co-designed acceleration of android bytecode," in *Proceedings of the 2013 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*. IEEE Computer Society, 2013, pp. 1–10.

[94] K.-T. T. Cheng, X. Yang, and Y.-C. Wang, "Performance optimization of vision apps on mobile application processor," in *2013 20th International Conference on Systems, Signals and Image Processing (IWSSIP)*. IEEE, 2013, pp. 187–191.

[95] S. Thongkaew, T. Isshiki, D. Li, and H. Kunieda, "Dalvik bytecode

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2021.3071193, IEEE Transactions on Software Engineering

21

acceleration using fetch/decode hardware extension," *Journal of Information Processing*, vol. 23, no. 2, pp. 118–130, 2015.

[96] B. Mao, J. Zhou, S. Wu, H. Jiang, X. Chen, and W. Yang, "Improving flash memory performance and reliability for smartphones with i/o deduplication," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 6, pp. 1017–1027, 2018.

[97] H. Kim and D. Shin, "Optimizing storage performance of android smartphone," in *Proceedings of the 7th International Conference on Ubiquitous Information Management and Communication*, 2013, pp. 1–7.

[98] R. Vallée-Rai, P. Co, E. Gagnon, L. Hendren, P. Lam, and V. Sundaresan, "Soot: A java bytecode optimization framework," in *CASCON First Decade High Impact Papers*, 2010, pp. 214–224.

[99] K. Nagata, S. Yamaguchi, and H. Ogawa, "A power saving method with consideration of performance in android terminals," in *2012 9th International Conference on Ubiquitous Intelligence and Computing and 9th International Conference on Autonomic and Trusted Computing*. IEEE, 2012, pp. 578–585.

[100] S. Georgiou, M. Kechagia, P. Louridas, and D. Spinellis, "What are your programming language's energy-delay implications?" in *Proceedings of the 15th International Conference on Mining Software Repositories*, ser. MSR '18. New York, NY, USA: Association for Computing Machinery, 2018, pp. 303—-313.

[101] K. Nagata, Y. Nakamura, S. Nomura, and S. Yamaguchi, "Measuring and improving application launching performance on android devices," in *2013 First International Symposium on Computing and Networking*. IEEE, 2013, pp. 636–638.

[102] T. Yan, D. Chu, D. Ganesan, A. Kansal, and J. Liu, "Fast app launching for mobile devices using predictive user context," in *Proceedings of the 10th international conference on Mobile systems, applications, and services*. ACM, 2012, pp. 113–126.

[103] K. Nagata and S. Yamaguchi, "An android application launch analyzing system," in *2012 8th International Conference on Computing Technology and Information Management (NCM and ICNIT)*, vol. 1. IEEE, 2012, pp. 76–81.

[104] S.-H. Kim, J. Jeong, J.-S. Kim, and S. Maeng, "Smartlmk: A memory reclamation scheme for improving user-perceived app launch time," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 15, no. 3, pp. 1–25, 2016.

[105] Y. Joo, J. Ryu, S. Park, and K. G. Shin, "Fast: Quick application launch on solid-state drives." in *FAST*, 2011, pp. 259–272.

[106] Z. Zhao, M. Zhou, and X. Shen, "Satscore: Uncovering and avoiding a principled pitfall in responsiveness measurements of app launches," in *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, 2014, pp. 21–32.

[107] A. Parate, M. Böhmer, D. Chu, D. Ganesan, and B. M. Marlin, "Practical prediction and prefetch for faster access to applications on mobile phones," in *Proceedings of the 2013 ACM international joint conference on Pervasive and ubiquitous computing*. ACM, 2013, pp. 275–284.

[108] L.-Y. Tang, P.-C. Hsiu, J.-L. Huang, and M.-S. Chen, "ilauncher: an intelligent launcher for mobile apps based on individual usage patterns," in *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, 2013, pp. 505–512.

[109] P. Baumann and S. Santini, "Every byte counts: Selective prefetching for mobile applications," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 1, no. 2, pp. 1–29, 2017.

[110] R. Prodduturi and D. B. Phatak, "Effective handling of low memory scenarios in android using logs," *Indian Institute of Technology*, 2013.

[111] K. Baik and J. Huh, "Balanced memory management for smartphones based on adaptive background app management," in *The 18th IEEE International Symposium on Consumer Electronics (ISCE 2014)*. IEEE, 2014, pp. 1–2.

[112] K. Vimal and A. Trivedi, "A memory management scheme for enhancing performance of applications on android," in *2015 IEEE Recent Advances in Intelligent Computational Systems (RAICS)*. IEEE, 2015, pp. 162–166.

[113] A. Singh, A. V. Agrawal, and A. Kanukotla, "A method to improve application launch performance in android devices," in *2016 International Conference on Internet of Things and Applications (IOTA)*. IEEE, 2016, pp. 112–115.

[114] C. Li, J. Bao, and H. Wang, "Optimizing low memory killers for mobile devices using reinforcement learning," in *2017 13th Inter-*

national Wireless Communications and Mobile Computing Conference (IWCMC). IEEE, 2017, pp. 2169–2174.

[115] M. Böhmer, B. Hecht, J. Schöning, A. Krüger, and G. Bauer, "Falling asleep with angry birds, facebook and kindle: a large scale study on mobile application usage," in *Proceedings of the 13th international conference on Human computer interaction with mobile devices and services*. ACM, 2011, pp. 47–56.

[116] R. Baeza-Yates, D. Jiang, F. Silvestri, and B. Harrison, "Predicting the next app that you are going to use," in *Proceedings of the eighth ACM international conference on web search and data mining*. ACM, 2015, pp. 285–294.

[117] C. Shin, J.-H. Hong, and A. K. Dey, "Understanding and prediction of mobile application usage for smart phones," in *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*. ACM, 2012, pp. 173–182.

[118] X. Zou, W. Zhang, S. Li, and G. Pan, "Prophet: What app you wish to use next," in *Proceedings of the 2013 ACM conference on Pervasive and ubiquitous computing adjunct publication*, 2013, pp. 167–170.

[119] J. Sylve, A. Case, L. Marziale, and G. G. Richard, "Acquisition and analysis of volatile memory from android devices," *Digital Investigation*, vol. 8, no. 3-4, pp. 175–184, 2012.

[120] J.-M. Kim and J.-S. Kim, "Androbench: Benchmarking the storage performance of android-based mobile devices," in *Frontiers in Computer Education*. Springer, 2012, pp. 667–674.

[121] S. Jeong, K. Lee, J. Hwang, S. Lee, and Y. Won, "Androstep: Android storage performance analysis tool," *Software Engineering 2013-Workshopband*, 2013.

[122] J. Park and B. Choi, "Automated memory leakage detection in android based systems," *International Journal of Control and Automation*, vol. 5, no. 2, pp. 35–42, 2012.

[123] H. Shahriar, S. North, and E. Mawangi, "Testing of memory leak in android applications," in *2014 IEEE 15th International Symposium on High-Assurance Systems Engineering*. IEEE, 2014, pp. 176–183.

[124] G. Santhanakrishnan, C. Cargile, and A. Olmsted, "Memory leak detection in android applications based on code patterns," in *2016 International Conference on Information Society (i-Society)*. IEEE, 2016, pp. 133–134.

[125] D. Amalfitano, V. Riccio, P. Tramontana, and A. R. Fasolino, "Do memories haunt you? an automated black box testing approach for detecting memory leaks in android apps," *IEEE Access*, vol. 8, pp. 12 217–12 231, 2020.

[126] T. Gerlitz, I. Kalkov, J. F. Schommer, D. Franke, and S. Kowalewski, "Non-blocking garbage collection for real-time android," in *Proceedings of the 11th International Workshop on Java Technologies for Real-time and Embedded Systems*. ACM, 2013, pp. 108–117.

[127] G. Lim, C. Min, and Y. I. Eom, "Enhancing application performance by memory partitioning in android platforms," in *2013 IEEE International Conference on Consumer Electronics (ICCE)*. IEEE, 2013, pp. 649–650.

[128] R. Mori, S. Yamaguchi, and M. Oguchi, "Memory consumption saving by optimization of promotion condition of generational gc in android," in *2017 IEEE 6th Global Conference on Consumer Electronics (GCCE)*. IEEE, 2017, pp. 1–2.

[129] S.-h. Kim, J. Jeong, and J. Lee, "Efficient memory deduplicati on for mobile smart devices," in *2014 IEEE International Conference on Consumer Electronics (ICCE)*. IEEE, 2014, pp. 25–26.

[130] D. Kim, E. Lee, S. Ahn, and H. Bahn, "Improving the storage performance of smartphones through journaling in non-volatile memory," *IEEE Transactions on Consumer Electronics*, vol. 59, no. 3, pp. 556–561, 2013.

[131] S. Jeong, K. Lee, S. Lee, S. Son, and Y. Won, "I/o stack optimization for smartphones," in *Presented as part of the 2013 USENIX Annual Technical Conference (USENIX ATC 13)*, 2013, pp. 309–320.

[132] K. Zhong, T. Wang, X. Zhu, L. Long, D. Liu, W. Liu, Z. Shao, and E. H.-M. Sha, "Building high-performance smartphones via non-volatile memory: The swap approach," in *2014 International Conference on Embedded Software (EMSOFT)*. IEEE, 2014, pp. 1–10.

[133] S.-H. Kim, S. Kwon, J.-S. Kim, and J. Jeong, "Controlling physical memory fragmentation in mobile systems," in *Proceedings of the 2015 International Symposium on Memory Management*, ser. ISMM '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 1–14.

[134] D. T. Nguyen, H. Zhao, G. Zhou, G. Peng, and G. Xing, "iram: Sensing memory needs of my smartphone," in *2016 IEEE 12th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*. IEEE, 2016, pp. 1–10.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2021.3071193, IEEE Transactions on Software Engineering

22

[135] D. Kim and H. Bahn, "Exploiting write-only-once characteristics of file data in smartphone buffer cache management," *Pervasive and Mobile Computing*, vol. 40, pp. 528–540, 2017.

[136] S.-H. Kim, J. Jeong, and J.-S. Kim, "Application-aware swapping for mobile systems," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 16, no. 5s, pp. 1–19, 2017.

[137] J. Kim and H. Bahn, "Analysis of smartphone i/o characteristics—toward efficient swap in a smartphone," *IEEE Access*, vol. 7, pp. 129 930–129 941, 2019.

[138] A. Escobar De La Torre and Y. Cheon, "Impacts of java language features on the memory performances of android apps," Departmental Technical Reports (CS), Tech. Rep., 2017.

[139] M. Jun, L. Sheng, Y. Shengtao, T. Xianping, and L. Jian, "Leakdaf: An automated tool for detecting leaked activities and fragments of android applications," in *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)*, vol. 1. IEEE, 2017, pp. 23–32.

[140] A. Carroll, G. Heiser *et al.*, "An analysis of power consumption in a smartphone." in *USENIX annual technical conference*, vol. 14. Boston, MA, 2010, pp. 21–21.

[141] L. Zhang, M. S. Gordon, R. P. Dick, Z. M. Mao, P. Dinda, and L. Yang, "Adel: An automatic detector of energy leaks for smartphone applications," in *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*. ACM, 2012, pp. 363–372.

[142] G. Metri, W. Shi, M. Brockmeyer, and A. Agrawal, "Batteryextender: an adaptive user-guided tool for power management of mobile devices," in *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*. ACM, 2014, pp. 33–43.

[143] R. Morales, R. Saborido, F. Khomh, F. Chicano, and G. Antoniol, "Earmo: an energy-aware refactoring approach for mobile apps," *IEEE Transactions on Software Engineering*, vol. 44, no. 12, pp. 1176–1206, 2017.

[144] A. Shye, B. Scholbrock, and G. Memik, "Into the wild: studying real user activity patterns to guide power optimizations for mobile architectures," in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 2009, pp. 168–178.

[145] M. A. Hoque, M. Siekkinen, K. N. Khan, Y. Xiao, and S. Tarkoma, "Modeling, profiling, and debugging the energy consumption of mobile devices," *ACM Computing Surveys (CSUR)*, vol. 48, no. 3, pp. 1–40, 2015.

[146] J. Cito, J. Rubin, P. Stanley-Marbell, and M. Rinard, "Battery-aware transformations in mobile applications," in *2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2016, pp. 702–707.

[147] W. Jung, C. Kang, C. Yoon, D. Kim, and H. Cha, "Devscope: a nonintrusive and online power analysis tool for smartphone hardware components," in *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, 2012, pp. 353–362.

[148] 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*. ACM, 2014, pp. 588–598.

[149] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "Maui: making smartphones last longer with code offload," in *Proceedings of the 8th international conference on Mobile systems, applications, and services*. ACM, 2010, pp. 49–62.

[150] A. J. Pyles, Z. Ren, G. Zhou, and X. Liu, "Sifi: exploiting voip silence for wifi energy savings insmart phones," in *Proceedings of the 13th international conference on Ubiquitous computing*, 2011, pp. 325–334.

[151] B. R. Bruce, J. Petke, M. Harman, and E. T. Barr, "Approximate oracles and synergy in software energy search spaces," *IEEE Transactions on Software Engineering*, vol. 45, no. 11, pp. 1150–1169, 2018.

[152] A. Ferrari, D. Gallucci, D. Puccinelli, and S. Giordano, "Detecting energy leaks in android app with poem," in *2015 IEEE International Conference on Pervasive Computing and Communication Workshops (PerCom Workshops)*. IEEE, 2015, pp. 421–426.

[153] M. A. Bokhari, B. Alexander, and M. Wagner, "In-vivo and offline optimisation of energy use in the presence of small energy signals: A case study on a popular android library," in *Proceedings of the 15th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*, 2018, pp. 207–215.

[154] A. Pathak, Y. C. Hu, M. Zhang, P. Bahl, and Y.-M. Wang, "Fine-grained power modeling for smartphones using system call tracing," in *Proceedings of the sixth conference on Computer systems*. ACM, 2011, pp. 153–168.

[155] C. Yoon, D. Kim, W. Jung, C. Kang, and H. Cha, "Appscope: Application energy metering framework for android smartphone using kernel activity monitoring," in *Presented as part of the 2012 USENIX Annual Technical Conference (USENIX ATC 12)*, 2012, pp. 387–400.

[156] M. Linares-Vásquez, G. Bavota, C. E. B. Cárdenas, R. Oliveto, M. Di Penta, and D. Poshyvanyk, "Optimizing energy consumption of guis in android apps: a multi-objective approach," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, 2015, pp. 143–154.

[157] M. Wan, Y. Jin, D. Li, and W. G. Halfond, "Detecting display energy hotspots in android apps," in *2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 2015, pp. 1–10.

[158] Y. Xiao, P. Savolainen, A. Karppanen, M. Siekkinen, and A. Ylä-Jääski, "Practical power modeling of data transmission over 802.11 g for wireless applications," in *Proceedings of the 1st International Conference on Energy-efficient Computing and Networking*, 2010, pp. 75–84.

[159] S. Hao, D. Li, W. G. Halfond, and R. Govindan, "Estimating android applications' cpu energy usage via bytecode profiling," in *2012 First international workshop on green and sustainable software (GREENS)*. IEEE, 2012, pp. 1–7.

[160] ——, "Estimating mobile application energy consumption using program analysis," in *Proceedings of the 2013 International Conference on Software Engineering*. IEEE Press, 2013, pp. 92–101.

[161] D. Li, S. Hao, W. G. Halfond, and R. Govindan, "Calculating source line level energy information for android applications," in *Proceedings of the 2013 International Symposium on Software Testing and Analysis*. ACM, 2013, pp. 78–89.

[162] R. Jabbarvand, J.-W. Lin, and S. Malek, "Search-based energy testing of android," in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 2019, pp. 1119–1130.

[163] R. Jabbarvand, A. Sadeghi, H. Bagheri, and S. Malek, "Energy-aware test-suite minimization for android apps," in *Proceedings of the 25th International Symposium on Software Testing and Analysis*, 2016, pp. 425–436.

[164] R. Mittal, A. Kansal, and R. Chandra, "Empowering developers to estimate app energy consumption," in *Proceedings of the 18th annual international conference on Mobile computing and networking*. ACM, 2012, pp. 317–328.

[165] M. A. Bokhari, B. R. Bruce, B. Alexander, and M. Wagner, "Deep parameter optimisation on android smartphones for energy minimisation: a tale of woe and a proof-of-concept," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. ACM, 2017, pp. 1501–1508.

[166] M. A. Bokhari, L. Weng, M. Wagner, and B. Alexander, "Mind the gap–a distributed framework for enabling energy optimisation on modern smart-phones in the presence of noise, drift, and statistical insignificance," in *2019 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2019, pp. 1330–1337.

[167] M. A. Bokhari, B. Alexander, and M. Wagner, "Towards rigorous validation of energy optimisation experiments," *arXiv preprint arXiv:2004.04500*, 2020.

[168] A. Saarinen, M. Siekkinen, Y. Xiao, J. K. Nurminen, M. Kemppainen, and P. Hui, "Smartdiet: offloading popular apps to save energy," *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 4, pp. 297–298, 2012.

[169] A. Y. Ding, B. Han, Y. Xiao, P. Hui, A. Srinivasan, M. Kojo, and S. Tarkoma, "Enabling energy-aware collaborative mobile data offloading for smartphones," in *2013 IEEE International Conference on Sensing, Communications and Networking (SECON)*. IEEE, 2013, pp. 487–495.

[170] A. Saarinen, M. Siekkinen, Y. Xiao, J. K. Nurminen, M. Kemppainen, and P. Hui, "Can offloading save energy for popular apps?" in *Proceedings of the seventh ACM international workshop on Mobility in the evolving internet architecture*, 2012, pp. 3–10.

[171] A. Khairy, H. H. Ammar, and R. Bahgat, "Smartphone energizer: Extending smartphone's battery life with smart offloading," in *2013 9th International Wireless Communications and Mobile Computing Conference (IWCMC)*. IEEE, 2013, pp. 329–336.

[172] Y.-W. Kwon and E. Tilevich, "Reducing the energy consumption of

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2021.3071193, IEEE Transactions on Software Engineering

23

mobile applications behind the scenes," in *2013 IEEE International Conference on Software Maintenance*. IEEE, 2013, pp. 170–179.

[173] L. Corral, A. B. Georgiev, A. Sillitti, G. Succi, and T. Vachkov, "Analysis of offloading as an approach for energy-aware applications on android os: A case study on image processing," in *International Conference on Mobile Web and Information Systems*. Springer, 2014, pp. 29–40.

[174] R. Bolla, R. Khan, X. Parra, and M. Repetto, "Improving smartphones battery life by reducing energy waste of background applications," in *2014 Eighth International Conference on Next Generation Mobile Apps, Services and Technologies*. IEEE, 2014, pp. 123–130.

[175] H. Qian and D. Andresen, "Jade: Reducing energy consumption of android app," *International Journal of Networked and Distributed Computing*, vol. 3, no. 3, pp. 150–158, 2015.

[176] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani, "Energy consumption in mobile phones: a measurement study and implications for network applications," in *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement*. ACM, 2009, pp. 280–293.

[177] X. Chen, A. Jindal, and Y. C. Hu, "How much energy can we save from prefetching ads?: energy drain analysis of top 100 apps," in *Proceedings of the Workshop on Power-Aware Computing and Systems*. ACM, 2013, p. 3.

[178] P. Mohan, S. Nath, and O. Riva, "Prefetching mobile ads: Can advertising systems afford it?" in *Proceedings of the 8th ACM European Conference on Computer Systems*, 2013, pp. 267–280.

[179] Y. Yang and G. Cao, "Prefetch-based energy optimization on smartphones," *IEEE Transactions on Wireless Communications*, vol. 17, no. 1, pp. 693–706, 2017.

[180] K. Dutta and D. Vandermeer, "Caching to reduce mobile app energy consumption," *ACM Transactions on the Web (TWEB)*, vol. 12, no. 1, pp. 1–30, 2017.

[181] A. Pathak, Y. C. Hu, and M. Zhang, "Bootstrapping energy debugging on smartphones: a first look at energy bugs in mobile devices," in *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*. ACM, 2011, p. 5.

[182] A. Pathak, A. Jindal, Y. C. Hu, and S. P. Midkiff, "What is keeping my phone awake? characterizing and detecting no-sleep energy bugs in smartphone apps," in *Proceedings of the 10th international conference on Mobile systems, applications, and services*, 2012, pp. 267–280.

[183] Y. Liu, C. Xu, S.-C. Cheung, and J. Lü, "Greendroid: Automated diagnosis of energy inefficiency for smartphone applications," *IEEE Transactions on Software Engineering*, vol. 40, no. 9, pp. 911–940, 2014.

[184] R. Jabbarvand and S. Malek, "μdroid: an energy-aware mutation testing framework for android," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, 2017, pp. 208–219.

[185] A. Pathak, Y. C. Hu, and M. Zhang, "Where is the energy spent inside my app? fine grained energy accounting on smartphones with eprof," in *Proceedings of the 7th ACM european conference on Computer Systems*, 2012, pp. 29–42.

[186] S. Anwer, A. Aggarwal, R. Purandare, and V. Naik, "Chiromancer: A tool for boosting android application performance," in *Proceedings of the 1st International Conference on Mobile Software Engineering and Systems*, 2014, pp. 62–65.

[187] F. Alam, P. R. Panda, N. Tripathi, N. Sharma, and S. Narayan, "Energy optimization in android applications through wakelock placement," in *2014 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2014, pp. 1–4.

[188] D. Li, A. H. Tran, and W. G. Halfond, "Making web applications more energy efficient for oled smartphones," in *Proceedings of the 36th International Conference on Software Engineering*. ACM, 2014, pp. 527–538.

[189] B. R. Bruce, J. Petke, and M. Harman, "Reducing energy consumption using genetic improvement," in *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*. ACM, 2015, pp. 1327–1334.

[190] A. Banerjee and A. Roychoudhury, "Automated re-factoring of android apps to enhance energy-efficiency," in *2016 IEEE/ACM International Conference on Mobile Software Engineering and Systems (MOBILESoft)*. IEEE, 2016, pp. 139–150.

[191] L. Cruz, R. Abreu, and J.-N. Rouvignac, "Leafactor: Improving energy efficiency of android apps via automatic refactoring," in *2017 IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*. IEEE, 2017, pp. 205–206.

[192] 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*, vol. 44, no. 5, pp. 470–490, 2017.

[193] L. Cruz and R. Abreu, "Performance-based guidelines for energy efficient mobile applications," in *2017 IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*. IEEE, 2017, pp. 46–57.

[194] ——, "Using Automatic Refactoring to Improve Energy Efficiency of Android Apps," in *CIbSE XXI Ibero-American Conference on Software Engineering*, 2018.

[195] M. W. Kim, D. G. Yun, J. M. Lee, and S. G. Choi, "Battery life time extension method using selective data reception on smartphone," in *The International Conference on Information Network 2012*. IEEE, 2012, pp. 468–471.

[196] N. Ding, D. Wagner, X. Chen, A. Pathak, Y. C. Hu, and A. Rice, "Characterizing and modeling the impact of wireless signal strength on smartphone battery drain," *ACM SIGMETRICS Performance Evaluation Review*, vol. 41, no. 1, pp. 29–40, 2013.

[197] M. Bokhari and M. Wagner, "Optimising energy consumption heuristically on android mobile phones," in *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion*. ACM, 2016, pp. 1139–1140.

[198] K. Rao, J. Wang, S. Yalamanchili, Y. Wardi, and Y. Handong, "Application-specific performance-aware energy optimization on android mobile devices," in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2017, pp. 169–180.

[199] M. Dong, Y.-S. K. Choi, and L. Zhong, "Power modeling of graphical user interfaces on oled displays," in *Proceedings of the 46th Annual Design Automation Conference*. ACM, 2009, pp. 652–657.

[200] B. Anand, K. Thirugnanam, J. Sebastian, P. G. Kannan, A. L. Ananda, M. C. Chan, and R. K. Balan, "Adaptive display power management for mobile games," in *Proceedings of the 9th international conference on Mobile systems, applications, and services*, 2011, pp. 57–70.

[201] C.-H. Lin, P.-C. Hsiu, and C.-K. Hsieh, "Dynamic backlight scaling optimization: A cloud-based energy-saving service for mobile streaming applications," *IEEE Transactions on Computers*, vol. 63, no. 2, pp. 335–348, 2012.

[202] C.-H. Lin, C.-K. Kang, and P.-C. Hsiu, "Catch your attention: Quality-retaining power saving on mobile oled displays," in *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 2014, pp. 1–6.

[203] H. Chen, J. Wang, W. Chen, H. Qu, and W. Chen, "An image-space energy-saving visualization scheme for oled displays," *Computers & graphics*, vol. 38, pp. 61–68, 2014.

[204] Y. Huang, M. Chen, L. Zhang, S. Xiao, J. Zhao, and Z. Wei, "Intelligent frame refresh for energy-aware display subsystems in mobile devices," in *Proceedings of the 2014 international symposium on Low power electronics and design*, 2014, pp. 369–374.

[205] X. Chen, K. W. Nixon, H. Zhou, Y. Liu, and Y. Chen, "Finger-shadow: An {OLED} power optimization based on smartphone touch interactions," in *6th Workshop on Power-Aware Computing and Systems (HotPower 14)*, 2014.

[206] K. W. Nixon, X. Chen, H. Zhou, Y. Liu, and Y. Chen, "Mobile {GPU} power consumption reduction via dynamic resolution and frame rate scaling," in *6th Workshop on Power-Aware Computing and Systems (HotPower 14)*, 2014.

[207] S. He, Y. Liu, and H. Zhou, "Optimizing smartphone power consumption through dynamic resolution scaling," in *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*, 2015, pp. 27–39.

[208] H.-Y. Lin, P.-C. Hsiu, and T.-W. Kuo, "Shiftmask: Dynamic oled power shifting based on visual acuity for interactive mobile applications," in *2017 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*. IEEE, 2017, pp. 1–6.

[209] G. Lee, S. Lee, G. Kim, Y. Choi, R. Ha, and H. Cha, "Improving energy efficiency of android devices by preventing redundant frame generation," *IEEE Transactions on Mobile Computing*, vol. 18, no. 4, pp. 871–884, 2018.

[210] Y.-C. Chang, W.-M. Chen, P.-C. Hsiu, Y.-Y. Lin, and T.-W. Kuo, "Lsim: Ultra lightweight similarity measurement for mobile graphics applications," in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019, pp. 1–6.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2021.3071193, IEEE Transactions on Software Engineering

24

[211] H.-Y. Lin, C.-C. Hung, P.-C. Hsiu, and T.-W. Kuo, "Duet: an oled & gpu co-management scheme for dynamic resolution adaptation," in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*. IEEE, 2018, pp. 1–6.

[212] P. T. Bezerra, L. A. Araujo, G. B. Ribeiro, A. C. d. S. B. Neto, A. G. Silva-Filho, C. A. Siebra, F. QB da Silva, A. L. Santos, A. Mascaro, and P. H. Costa, "Dynamic frequency scaling on android platforms for energy consumption reduction," in *Proceedings of the 8th ACM workshop on Performance monitoring and measurement of heterogeneous wireless and wired networks*, 2013, pp. 189–196.

[213] Y.-M. Chang, P.-C. Hsiu, Y.-H. Chang, and C.-W. Chang, "A resource-driven dvfs scheme for smart handheld devices," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 13, no. 3, pp. 1–22, 2013.

[214] P.-H. Tseng, P.-C. Hsiu, C.-C. Pan, and T.-W. Kuo, "User-centric energy-efficient scheduling on multi-core mobile devices," in *Proceedings of the 51st Annual Design Automation Conference*, 2014, pp. 1–6.

[215] P.-C. Hsiu, P.-H. Tseng, W.-M. Chen, C.-C. Pan, and T.-W. Kuo, "User-centric scheduling and governing on mobile devices with big. little processors," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 15, no. 1, pp. 1–22, 2016.

[216] S. Li and S. Mishra, "Optimizing power consumption in multicore smartphones," *Journal of Parallel and Distributed Computing*, vol. 95, pp. 124–137, 2016.

[217] P. K. Muhuri, P. K. Gupta, and J. M. Mendel, "Person footprint of uncertainty-based cww model for power optimization in handheld devices," *IEEE Transactions on Fuzzy Systems*, vol. 28, no. 3, pp. 558–568, 2019.

[218] J. Han and S. Lee, "Performance improvement of linux cpu scheduler using policy gradient reinforcement learning for android smartphones," *IEEE Access*, vol. 8, pp. 11 031–11 045, 2020.

[219] J. Paek, J. Kim, and R. Govindan, "Energy-efficient rate-adaptive gps-based positioning for smartphones," in *Proceedings of the 8th international conference on Mobile systems, applications, and services*, 2010, pp. 299–314.

[220] Z. Zhuang, K.-H. Kim, and J. P. Singh, "Improving energy efficiency of location sensing on smartphones," in *Proceedings of the 8th international conference on Mobile systems, applications, and services*, 2010, pp. 315–330.

[221] Y. Chon, E. Talipov, H. Shin, and H. Cha, "Mobility prediction-based smartphone energy optimization for everyday location monitoring," in *Proceedings of the 9th ACM conference on embedded networked sensor systems*, 2011, pp. 82–95.

[222] T. O. Oshin, S. Poslad, and A. Ma, "Improving the energy-efficiency of gps based location sensing smartphone applications," in *2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications*. IEEE, 2012, pp. 1698–1705.

[223] L. Zhang, J. Liu, H. Jiang, and Y. Guan, "Senstrack: Energy-efficient location tracking with smartphone sensors," *IEEE sensors journal*, vol. 13, no. 10, pp. 3775–3784, 2013.

[224] M. Linares-Vásquez, G. Bavota, C. Bernal-Cárdenas, R. Oliveto, M. Di Penta, and D. Poshyvanyk, "Mining energy-greedy api usage patterns in android apps: an empirical study," in *Proceedings of the 11th Working Conference on Mining Software Repositories*. ACM, 2014, pp. 2–11.

[225] M.-R. Ra, J. Paek, A. B. Sharma, R. Govindan, M. H. Krieger, and M. J. Neely, "Energy-delay tradeoffs in smartphone applications," in *Proceedings of the 8th international conference on Mobile systems, applications, and services*, 2010, pp. 255–270.

[226] J. K. Nurminen, "Parallel connections and their effect on the battery consumption of a mobile phone," in *2010 7th IEEE Consumer Communications and Networking Conference*. IEEE, 2010, pp. 1–5.

[227] A. J. Pyles, X. Qi, G. Zhou, M. Keally, and X. Liu, "Sapsm: Smart adaptive 802.11 psm for smartphones," in *Proceedings of the 2012 ACM conference on ubiquitous computing*, 2012, pp. 11–20.

[228] K. Lee, J. Lee, Y. Yi, I. Rhee, and S. Chong, "Mobile data offloading: How much can wifi deliver?" *IEEE/ACM Transactions on networking*, vol. 21, no. 2, pp. 536–550, 2012.

[229] C.-C. Cheng and P.-C. Hsiu, "Extend your journey: Introducing signal strength into location-based applications," in *2013 Proceedings IEEE INFOCOM*. IEEE, 2013, pp. 2742–2750.

[230] M. Siekkinen, M. A. Hoque, J. K. Nurminen, and M. Aalto, "Streaming over 3g and lte: How to save smartphone energy in radio access network-friendly way," in *Proceedings of the 5th Workshop on Mobile Video*, 2013, pp. 13–18.

[231] D. Li, Y. Lyu, J. Gui, and W. G. Halfond, "Automated energy optimization of http requests for mobile applications," in *Proceedings of the 38th international conference on software engineering*. ACM, 2016, pp. 249–260.

[232] X. Chen, A. Jindal, N. Ding, Y. C. Hu, M. Gupta, and R. Vannithamby, "Smartphone background activities in the wild: Origin, energy drain, and optimization," in *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*, 2015, pp. 40–52.

[233] M. Martins, J. Cappos, and R. Fonseca, "Selectively taming background android apps to improve battery lifetime," in *2015 {USENIX} Annual Technical Conference ({USENIX} {ATC} 15)*, 2015, pp. 563–575.

[234] D. T. Nguyen, G. Zhou, X. Qi, G. Peng, J. Zhao, T. Nguyen, and D. Le, "Storage-aware smartphone energy savings," in *Proceedings of the 2013 ACM international joint conference on Pervasive and ubiquitous computing*, 2013, pp. 677–686.

[235] A. Hussein, M. Payer, A. Hosking, and C. A. Vick, "Impact of gc design on power and performance for android," in *Proceedings of the 8th ACM International Systems and Storage Conference*, 2015, pp. 1–12.

[236] K. Zhong, D. Liu, L. Liang, X. Zhu, L. Long, Y. Wang, and E. H.-M. Sha, "Energy-efficient in-memory paging for smartphones," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 10, pp. 1577–1590, 2015.

[237] X. Chen, Y. Chen, Z. Ma, and F. C. Fernandes, "How is energy consumed in smartphone display applications?" in *Proceedings of the 14th Workshop on Mobile Computing Systems and Applications*, 2013, pp. 1–6.

[238] M. Linares-Vásquez, G. Bavota, C. Bernal-Cárdenas, M. D. Penta, R. Oliveto, and D. Poshyvanyk, "Multi-objective optimization of energy consumption of guis in android apps," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 27, no. 3, pp. 1–47, 2018.

[239] D. Li, S. Hao, J. Gui, and W. G. Halfond, "An empirical study of the energy consumption of android applications," in *2014 IEEE International Conference on Software Maintenance and Evolution*. IEEE, 2014, pp. 121–130.

[240] M. A. Hoque, M. Siekkinen, and J. K. Nurminen, "Energy efficient multimedia streaming to mobile devices—a survey," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 1, pp. 579–597, 2012.

[241] S. K. Gudla, J. K. Sahoo, A. Singh, J. Bose, and N. Ahamed, "A systematic framework to optimize launch times of web apps," in *Proceedings of the 26th International Conference on World Wide Web Companion*. International World Wide Web Conferences Steering Committee, 2017, pp. 785–786.

[242] L. Corral, A. B. Georgiev, A. Sillitti, and G. Succi, "Can execution time describe accurately the energy consumption of mobile apps? an experiment in android," in *Proceedings of the 3rd International Workshop on Green and Sustainable Software*, 2014, pp. 31–37.

[243] M. Xia, W. He, X. Liu, and J. Liu, "Why application errors drain battery easily? a study of memory leaks in smartphone apps," in *Proceedings of the Workshop on Power-Aware Computing and Systems*, 2013, pp. 1–5.

[244] D. T. Nguyen, "Evaluating impact of storage on smartphone energy efficiency," in *Proceedings of the 2013 ACM conference on Pervasive and ubiquitous computing adjunct publication*. ACM, 2013, pp. 319–324.

[245] A. P. Miettinen and J. K. Nurminen, "Energy efficiency of mobile clients in cloud computing." *HotCloud*, vol. 10, no. 4-4, p. 19, 2010.

[246] L. Cruz and R. Abreu, "On the energy footprint of mobile testing frameworks," *IEEE Transactions on Software Engineering*, 2019.

[247] R. Saborido, F. Khomh, G. Antoniol, and Y.-G. Guéhéneuc, "Comprehension of ads-supported and paid android applications: are they different?" in *2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC)*. IEEE, 2017, pp. 143–153.

[248] K. Wiegers and J. Beatty, *Software requirements*. Pearson Education, 2013.

[249] J. Karlsson, "Software requirements prioritizing," in *Proceedings of the Second International Conference on Requirements Engineering*. IEEE, 1996, pp. 110–116.

[250] J. Karlsson, C. Wohlin, and B. Regnell, "An evaluation of methods for prioritizing software requirements," *Information and software technology*, vol. 39, no. 14-15, pp. 939–947, 1998.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2021.3071193, IEEE Transactions on Software Engineering

25

[251] M. Woodside, G. Franks, and D. C. Petriu, "The future of software performance engineering," in *Future of Software Engineering (FOSE'07)*. IEEE, 2007, pp. 171–187.

[252] S. Schubert, D. Kostic, W. Zwaenepoel, and K. G. Shin, "Profiling software for energy consumption," in *2012 IEEE International Conference on Green Computing and Communications*. IEEE, 2012, pp. 515–522.

[253] A. Sinha and A. P. Chandrakasan, "Jouletrack: a web based tool for software energy profiling," in *Proceedings of the 38th annual Design Automation Conference*, 2001, pp. 220–225.

[254] M. Stephenson, S. K. S. Hari, Y. Lee, E. Ebrahimi, D. R. Johnson, D. Nellans, M. O'Connor, and S. W. Keckler, "Flexible software profiling of gpu architectures," in *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2015, pp. 185–197.

[255] X. Zhao, K. Rodrigues, Y. Luo, D. Yuan, and M. Stumm, "Non-intrusive performance profiling for entire software stacks based on the flow reconstruction principle," in *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, 2016, pp. 603–618.

[256] G. Xu and A. Rountev, "Precise memory leak detection for java software using container profiling," in *Proceedings of the 30th international conference on Software engineering*, 2008, pp. 151–160.

[257] M. Mao and M. Humphrey, "A performance study on the vm startup time in the cloud," in *2012 IEEE Fifth International Conference on Cloud Computing*. IEEE, 2012, pp. 423–430.

[258] H. Kaminaga, "Improving linux startup time using software resume (and other techniques)," in *Linux Symposium*, 2006, p. 17.

[259] E. Van Emden and L. Moonen, "Java quality assurance by detecting code smells," in *Ninth Working Conference on Reverse Engineering, 2002. Proceedings.* IEEE, 2002, pp. 97–106.

[260] J.-L. Baer and T.-F. Chen, "An effective on-chip preloading scheme to reduce data access penalty," in *Proceedings of the 1991 ACM/IEEE conference on Supercomputing*, 1991, pp. 176–186.

[261] C. Sahin, L. Pollock, and J. Clause, "How do code refactorings affect energy usage?" in *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, 2014, pp. 1–10.

[262] F. Chen, S. Jiang, and X. Zhang, "Smartsaver: turning flash drive into a disk energy saver for mobile computers," in *Proceedings of the 2006 international symposium on Low power electronics and design*, 2006, pp. 412–417.

[263] F. Chen, T. Luo, and X. Zhang, "Caftl: A content-aware flash translation layer enhancing the lifespan of flash memory based solid state drives." in *FAST*, vol. 11, 2011, pp. 77–90.

[264] M. Hauswirth and T. M. Chilimbi, "Low-overhead memory leak detection using adaptive statistical profiling," in *Acm SIGPLAN notices*, vol. 39, no. 11. ACM, 2004, pp. 156–164.

[265] V. N. Padmanabhan and J. C. Mogul, "Using predictive prefetching to improve world wide web latency," *ACM SIGCOMM Computer Communication Review*, vol. 26, no. 3, pp. 22–36, 1996.

[266] R. Minelli and M. Lanza, "Software analytics for mobile applications–insights & lessons learned," in *2013 17th European Conference on Software Maintenance and Reengineering*. IEEE, 2013, pp. 144–153.

[267] O. J. Romero and S. A. Akoju, "Adroitness: An android-based middleware for fast development of high-performance apps," *arXiv preprint arXiv:1906.02061*, 2019.

[268] M. Linares-Vasquez, C. Vendome, Q. Luo, and D. Poshyvanyk, "How developers detect and fix performance bottlenecks in android apps," in *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2015, pp. 352–361.

[269] B. Johnson, Y. Song, E. Murphy-Hill, and R. Bowdidge, "Why don't software developers use static analysis tools to find bugs?" in *Proceedings of the 2013 International Conference on Software Engineering*. IEEE Press, 2013, pp. 672–681.

[270] L. Li, T. F. Bissyandé, M. Papadakis, S. Rasthofer, A. Bartel, D. Octeau, J. Klein, and L. Traon, "Static analysis of android apps: A systematic literature review," *Information and Software Technology*, vol. 88, pp. 67–95, 2017.

[271] M. Christakis and C. Bird, "What developers want and need from program analysis: an empirical study," in *2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2016, pp. 332–343.

[272] A. Nistor, T. Jiang, and L. Tan, "Discovering, reporting, and fixing performance bugs," in *Proceedings of the 10th Working Conference on Mining Software Repositories*. IEEE Press, 2013, pp. 237–246.

[273] L. Cruz, R. Abreu, J. Grundy, L. Li, and X. Xia, "Do energy-oriented changes hinder maintainability?" in *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2019, pp. 29–40.

[274] M. Linares-Vásquez, C. Vendome, M. Tufano, and D. Poshyvanyk, "How developers micro-optimize android apps," *Journal of Systems and Software*, vol. 130, pp. 1–23, 2017.

[275] L. Cruz and R. Abreu, "Catalog of energy patterns for mobile applications," *Empirical Software Engineering*, vol. 24, no. 4, pp. 2209–2235, 2019.

[276] S. S. Afjehei, T.-H. P. Chen, and N. Tsantalis, "iperfdetector: characterizing and detecting performance anti-patterns in ios applications," *Empirical Software Engineering*, vol. 24, no. 6, pp. 3484–3513, 2019.

[277] I. Dalmasso, S. K. Datta, C. Bonnet, and N. Nikaein, "Survey, comparison and evaluation of cross platform mobile application development tools," in *2013 9th International Wireless Communications and Mobile Computing Conference (IWCMC)*. IEEE, 2013, pp. 323–328.

[278] M. E. Joorabchi, A. Mesbah, and P. Kruchten, "Real challenges in mobile app development," in *2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. IEEE, 2013, pp. 15–24.

[279] M. Mahmoudi and S. Nadi, "The android update problem: An empirical study," in *Proceedings of the 15th International Conference on Mining Software Repositories*. ACM, 2018, pp. 220–230.

[280] D. Han, C. Zhang, X. Fan, A. Hindle, K. Wong, and E. Stroulia, "Understanding android fragmentation with topic analysis of vendor-specific bugs," in *2012 19th Working Conference on Reverse Engineering*. IEEE, 2012, pp. 83–92.

[281] H. Khalid, M. Nagappan, E. Shihab, and A. E. Hassan, "Prioritizing the devices to test your app on: A case study of android game apps," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, 2014, pp. 610–620.

[282] X. Lu, X. Liu, H. Li, T. Xie, Q. Mei, D. Hao, G. Huang, and F. Feng, "Prada: prioritizing android devices for apps by mining large-scale usage data," in *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*. IEEE, 2016, pp. 3–13.

[283] J. Harty, "Google play console: insightful development using android vitals and pre-launch reports," in *2019 IEEE/ACM 6th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*. IEEE, 2019, pp. 62–65.

[284] R. Saborido, G. Beltrame, F. Khomh, E. Alba, and G. Antoniol, "Optimizing user experience in choosing android applications," in *2016 IEEE 23Rd international conference on software analysis, evolution, and reengineering (SANER)*, vol. 1. IEEE, 2016, pp. 438–448.

[285] E. Guzman and W. Maalej, "How do users like this feature? a fine grained sentiment analysis of app reviews," in *2014 IEEE 22nd international requirements engineering conference (RE)*. IEEE, 2014, pp. 153–162.

[286] B. Fu, J. Lin, L. Li, C. Faloutsos, J. Hong, and N. Sadeh, "Why people hate your app: Making sense of user feedback in a mobile app store," in *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2013, pp. 1276–1284.

[287] P. M. Vu, T. T. Nguyen, H. V. Pham, and T. T. Nguyen, "Mining user opinions in mobile app reviews: A keyword-based approach (t)," in *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2015, pp. 749–759.

[288] C. Gao, J. Zeng, M. R. Lyu, and I. King, "Online app review analysis for identifying emerging issues," in *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*. IEEE, 2018, pp. 48–58.

[289] Y. Tian, M. Nagappan, D. Lo, and A. E. Hassan, "What are the characteristics of high-rated apps? a case study on free android applications," in *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2015, pp. 301–310.

[290] C. Shepard, A. Rahmati, C. Tossell, L. Zhong, and P. Kortum, "Livelab: measuring wireless networks and smartphone users in the field," *ACM SIGMETRICS Performance Evaluation Review*, vol. 38, no. 3, pp. 15–20, 2011.

[291] E. Noei, M. D. Syer, Y. Zou, A. E. Hassan, and I. Keivanloo, "A study of the relation of mobile device attributes with the user-perceived quality of android apps," *Empirical Software Engineering*, vol. 22, no. 6, pp. 3088–3116, 2017.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2021.3071193, IEEE Transactions on Software Engineering

26

[292] U. Lee, J. Lee, M. Ko, C. Lee, Y. Kim, S. Yang, K. Yatani, G. Gweon, K.-M. Chung, and J. Song, "Hooked on smartphones: an exploratory study on smartphone overuse among college students," in *Proceedings of the 32nd annual ACM conference on Human factors in computing systems.* ACM, 2014, pp. 2327–2336.

[293] A. Morrison, X. Xiong, M. Higgs, M. Bell, and M. Chalmers, "A large-scale study of iphone app launch behaviour," in *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems.* ACM, 2018, p. 344.

[294] D. Yu, Y. Li, F. Xu, P. Zhang, and V. Kostakos, "Smartphone app usage prediction using points of interest," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 1, no. 4, pp. 1–21, 2018.

[295] J. Huangfu, J. Cao, and C. Liu, "A context-aware usage prediction approach for smartphone applications," in *Asia-Pacific Services Computing Conference.* Springer, 2015, pp. 3–16.

[296] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in software engineering.* Springer Science & Business Media, 2012.

[297] M. Allamanis, E. T. Barr, P. Devanbu, and C. Sutton, "A survey of machine learning for big code and naturalness," *ACM Computing Surveys (CSUR)*, vol. 51, no. 4, p. 81, 2018.

Mark Harman works full time at Facebook London as a Research Scientist in a team focussing in AI for scalable software engineering. He also holds a part-time professorship at UCL. Previously, Mark was the manager of the Facebook team that deployed Sapienz to test mobile apps, which grew out of Majicke, a start up co-founded by Mark and acquired by Facebook in 2017. In his more purely scientific work, Mark co-founded the field Search Based Software Engineering (SBSE), and is also known for scientific research on source code analysis, software testing, app store analysis and empirical software engineering. He received the IEEE Harlan Mills Award and the ACM Outstanding Research Award in 2019 for this work. In addition to Facebook itself, Mark's scientific work is also supported by the European Research Council (ERC), with an advanced fellowship grant, and has also been regularly and generously supported by the UK Engineering and Physical Sciences Research Council (EPSRC), with regular grants, a platform and a programme grant. Web page:http://www0.cs.ucl.ac.uk/staff/M.Harman/

Max Hort is a PhD student in Software Engineering at University College London, under the supervision of Prof. Federica Sarro and Prof. Mark Harman. His research research interests lie on software fairness, non-functional optimization of software, and search-based software engineering. Web page: http://www0.cs.ucl.ac.uk/staff/mbhort/

Maria Kechagia is a Research Fellow at University College London. Previously, she was a postdoctoral researcher at the Delft University of Technology. Before that, she obtained a PhD degree from the Athens University of Economics and Business and a MSc degree from Imperial College London. Her research interests include program analysis, software testing, automated program repair, and software analytics. Web page: http://www0.cs.ucl.ac.uk/staff/M.Kechagia/

Federica Sarro is a Professor of Software Engineering at University College London. Her research covers Predictive Analytics for Software Engineering (SE), Empirical SE and Search-Based SE, with a focus on software effort estimation, software sizing, software testing, and mobile app store analysis. On these topics, she has published several articles in prestigious international venues including ICSE, FSE, TSE, TOSEM, and has received several international awards, including the FSE'19 ACM Distinguished Paper Award and the ACM SIGEVO HUMIES GECCO'16 Award. She has also been invited to serve on several programme, organisation and steering committees, and editorial boards of well-renowned venues, such as ICSE, FSE, ACM TOSEM, IEEE TSE, IEEE TEVC. Web page: http://www0.cs.ucl.ac.uk/staff/F.Sarro/