2016 IEEE/ACM 13th Working Conference on Mining Software Repositories

How Android App Developers Manage Power Consumption?

An Empirical Study by Mining Power Management Commits

Lingfeng Bao[1], David Lo[2], Xin Xia[1]*, Xinyu Wang[1], Cong Tian[3]
[1]College of Computer Science and Technology, Zhejiang University, China
[2]School of Information Systems, Singapore Management University, Singapore
[3]ICTT and ISN Lab, Xidian University, Xi'an, China
lingfengbao@zju.edu.cn, davidlo@smu.edu.sg, xxia@zju.edu.cn
wangxinyu@zju.edu.cn, ctian@mail.xidian.edu.cn

ABSTRACT

As Android platform becomes more and more popular, a large amount of Android applications have been developed. When developers design and implement Android applications, power consumption management is an important factor to consider since it affects the usability of the applications. Thus, it is important to help developers adopt proper strategies to manage power consumption. Interestingly, today, there is a large number of Android application repositories made publicly available in sites such as GitHub. These repositories can be mined to help crystalize common power management activities that developers do. These in turn can be used to help other developers to perform similar tasks to improve their own Android applications.

In this paper, we present an empirical study of power management commits in Android applications. Our study extends that of Moura *et al.* who perform an empirical study on energy aware commits; however they do not focus on Android applications and only a few of the commits that they study come from Android applications. Android applications are often different from other applications (e.g., those running on a server) due to the issue of limited battery life and the use of specialized APIs. As subjects of our empirical study, we obtain a list of open source Android applications from F-Droid and crawl their commits from Github. We get 468 power management commits after we filter the commits using a set of keywords and by performing manual analysis. These 468 power management commits are from 154 different Android applications and belong to 15 different application categories. Furthermore, we use open card sort to categorize these power management commits and we obtain 6 groups which correspond to different power management activities. Our study also reveals that for different kinds of Android application (e.g., *Games*, *Connectivity*, *Navigation*, *Internet*, *Phone & SMS*, *Time*, etc.), the dominant power management activities differ. For example, the percentage of power management commits belonging to *Power Adaptation* activity is larger for *Navigation* applications than those belonging to other categories.

*Corresponding author

MSR'16, May 14-15, 2016, Austin, TX, USA

DOI: http://dx.doi.org/10.1145/2901739.2901748

Keywords

Power Consumption, Power Management, Mining Software Repository, Empirical Study

1. INTRODUCTION

Mobile devices such as smartphone and tablet have become a commonplace in our daily lives. For such mobile devices, Android[1] is a very popular open source mobile platform and has dominated the smartphone market with a share of 82.8% in the second quarter of 2015 [2]. More and more Android applications are produced by thousands of developers. In the first quarter of 2016, there are about 1,900,000 apps in Google Play [1]. As the functionality of these Android applications becomes more and more powerful, their power consumption increases too. Battery usage has become a very important quality metric for Android apps; for example, in a survey conducted with more than 3,500 respondents from 4 different countries [3], long-lasting battery life has been cited as the most desired feature in a new phone by 71% of the respondents. Thus, power management optimization is an important goal for Android app developers.

In the recent years, many research groups have focused on the energy consumption or power management of mobile devices. Some researchers propose useful tools to help developers to gain insights into the energy usage patterns of their applications. These tools include cycle-accurate simulators [27, 4], power monitors [24], program analysis techniques [9, 10], and statistical-based measurement techniques [20]. Additionally, many energy-efficient techniques have been proposed and they work at the operating system level [39, 6, 37], VM-level [34] and application-level [40, 25, 8]. Although these research studies provide techniques that can help developers understand and optimize power con-

[1]www.android.com

sumption, they do not provide direct guidance on how to write power efficient mobile apps.

Interestingly, today, there is a large number of Android application repositories made publicly available. More and more open source Android applications also begin to pay attention on power management. Thus, we can possibly crystallize their power management practices to help other developers. In a previous study, Moura *et al.* performed an empirical study to investigate solutions that software developers employ to save energy [26]. They analyzed hundreds of energy consumption-related commits from GITHUB[2]. The commits analyzed in their study are from a wide spectrum of repositories and only a few comes from repositories of mobile software. They state that:"*they found 47 commits that are targeting mobile software*". Also, they mention that "*Although most of these commits are related to Android Kernels, we found several commits that focus on the application-level*". In this work, we want to complement their study by focusing on power management commits that focus on Android and the application level. We focus on the application level since this is what matters to most Android developers – much fewer developers contribute to Android kernels than the millions of Android applications. Our purpose is to understand what are the activities performed by developers in order to manage energy consumption in Android applications.

As subjects of our study, we analyze Android applications listed in F-Droid[3], which is an installable catalogue of free and open source Android applications. F-Droid contains some meta information for each Android app, such as source code website, category, license, etc. A majority of apps in F-Droid put their source code in GITHUB, and these apps are the focus of our study. We extract commits of these apps from GITHUB and filter those that are related to power management.

To obtain the power management commits we follow a semi-automated approach. First, we use four keywords: "power", "energy", "battery" and "wakelock" to identify a set of commits that are likely to be power management related. Two of these keywords "battery" and "wakelock" are not used in the study of Moura et al. [26]. Unique from other applications, Android applications mostly run on hardware that are dependent on battery. Android applications also use a specialized library and "wakelock" is a key power management API for the Android platform. We find about 1,500 commits using these four key words. Second, we manually check the 1,500 commits to remove irrelevant ones. We identify a total of 468 power management commits which are part of 154 different Android applications.

After we identify the power management commits, we perform an open card sort to categorize the commits into groups. Card sorting is widely used to generate a category tree or folksonomy. In an open card sort, participants create their own names for categories. By contrast, in a closed card sort, participants are provided with a predetermined set of category names. At the end of the open card sort, six groups emerge which correspond to different power management activities. Finally, we identify the dominant power management activities for different categories of Android applications (e.g., *Games*, *Connectivity*, *Navigation*, etc.).

[2]www.github.com
[3]https://f-droid.org/

The main findings of this study are the following:

- We identify 6 main power management activities, namely: *Power Adaptation*, *Power Consumption Improvement*, *Power Usage Monitoring*, *Optimizing Wake Lock*, *Adding Wake Lock* and *Bug Fix & Code Refinement*.

- We confirm that wake lock is a key factor in Android power management. We find many power management commits (i.e., 156 (33.33%)) that contain keywords "wakelock"; 66 (14.10%) of them are in the group *Optimizing Wake Lock* and 90 (19.23%) of them are in the group *Adding Wake Lock*.

- We find that power management commits appear in applications across 15 out of the 17 application categories in F-Droid. The remaining two categories (i.e., *Graphics* and *Sports & Health*) only contain a few applications.

- We also find that 12.04% of the crawled applications include power management commits. The percentage of crawled applications with power management commits is higher for categories *Phone & SMS* and *Connectivity*. For these categories, 32.26% and 22.03% of the applications that we crawl contain power management commits. The percentages are lowest for *Theming*, *Writing* and *Games* categories – they are 3.90%, 4.35% and 6.45% respectively.

- We find that different application categories have different dominant power management activities. For instance, the dominant power management activities for the *Navigation* category is *Power Adaptation*, while the dominant power management activities for the *Games* category is *Power Usage Monitoring*.

Paper organization.
The remainder of this paper is organized as follows. Section 2 describes the methodology that we follow for this empirical study. Section 3 presents our empirical study results. Section 4 discusses the implications of our study and the threats to validity. Section 5 briefly reviews the related work. Section 6 draws the conclusions and presents future work.

2. CASE STUDY SETUP

In this section, we describe the details of our approach which contains two phases: a data collection phase and a data analysis phase. Figure 1 presents the overall framework of our aproach. In the data collection phase, we crawl meta data from thousands of open source Android applications in F-Droid[3]. By using these meta data, we then mine the commits from GITHUB. In the data analysis phase, we obtain the commits that are related to power management in Android by an automatic filtering process and a manual checking step. At last, we analyze these commits by computing some statistics and performing a qualitative analysis. Our data is available for researchers wishing to replicate our study at https://github.com/baolingfeng/AndroidPowerCommits.

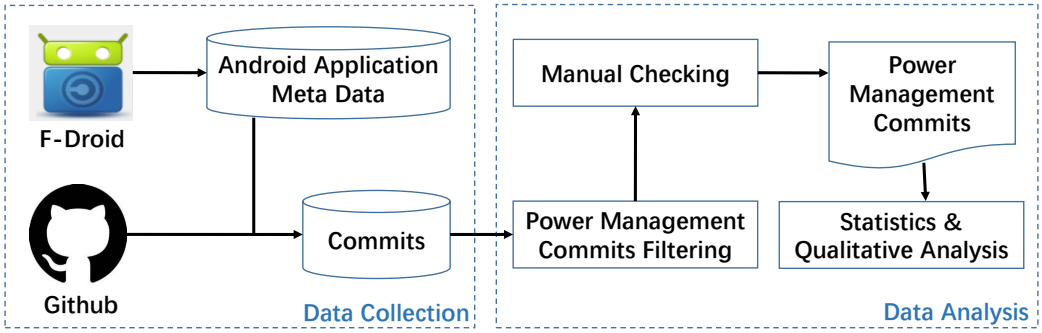
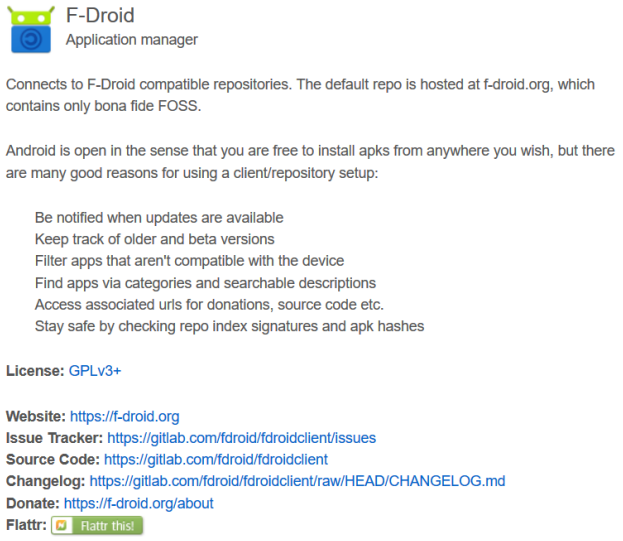


Figure 1: The Overall Framework of Our Proposed Approach

2.1 Data Collection

As data for this empirical study, we use mobile applications from F-Droid which is a catalogue of free and open source applications for the Android platform. In total, we find 1,993 apps on F-Droid[4]. Each mobile application available on F-Droid has a corresponding wiki page which provides details about the app. For us, wiki pages are very important since they provide us the meta data of the apps, such as description, license, web site, source code, issue tracker, category, etc. For example, Figure 2 presents a partial screenshot of the wiki page of F-Droid Application Manager. Table 1 presents the 17 categories of applications that are present on F-Droid. Notice one app can be assigned to one or more categories. For example, DroidShow which is a TV Series/TV show browser and tracker is assigned two categories – *Multimedia* and *Internet*.


F-Droid
Application manager

Connects to F-Droid compatible repositories. The default repo is hosted at f-droid.org, which contains only bona fide FOSS.

Android is open in the sense that you are free to install apks from anywhere you wish, but there are many good reasons for using a client/repository setup:

Be notified when updates are available
Keep track of older and beta versions
Filter apps that aren't compatible with the device
Find apps via categories and searchable descriptions
Access associated urls for donations, source code etc.
Stay safe by checking repo index signatures and apk hashes

License: GPLv3+

Website: https://f-droid.org
Issue Tracker: https://gitlab.com/fdroid/fdroidclient/issues
Source Code: https://gitlab.com/fdroid/fdroidclient
Changelog: https://gitlab.com/fdroid/fdroidclient/raw/HEAD/CHANGELOG.md
Donate: https://f-droid.org/about
Flattr: Flattr this!

Figure 2: A Partial Screenshot of F-Droid Application Manager Wiki Page

These Android applications are hosted in different platforms, such as GITHUB, BITBUCKET, GOOGLE CODE, etc. In this paper, we only consider applications whose source code is hosted on GITHUB. GITHUB provides very convenient REST APIs. We find that there are 1,208 applications on F-Droid whose source code is hosted on GITHUB. We also compare our collected applications with the data set of Krutz *et al.* [16] which contains 1,179 applications in total. There are 65 applications that are in their data set but not in our collected data. We manually check the data and find that the difference is caused by F-Droid recently deleted some applications. Since we could still get these applications in GITHUB, we also add these 65 applications into our dataset. Finally, we get 1,273 Android applications in total.

After obtaining this set of applications, we obtain all their commits from GITHUB using the GITHUB source code link in the meta data. At last, we get 671,443 commits from all of the 1,273 open source Android applications.

[4]https://f-droid.org/wiki/page/Repository_Maintenance

Table 1: The Categories of Android Applications on F-Droid

Application Categories	
Connectivity	Development
Games	Graphics
Internet	Money
Multimedia	Navigation
Phone & SMS	Reading
Science & Education	Security
Sports & Health	System
Theming	Time
Writing	

2.2 Data Analysis

For the collected commits, we first filter power management commits by using several regular expressions automatically, then we check the filtered commits manually. At last, we do some quantitative and qualitative analysis on power management commits.

2.2.1 *Power Management Commit Filtering*

Previous studies on power/energy saving [26, 30] automatically select commits that are most likely to be related to energy consumption by using the following regular expressions: *energy consum*, *energy efficien*, *energy sav*, *save energy*, *power consum*, *power efficien*, *power sa*, and *save power*. The character '*' in each regular expression corresponds to a wildcard. Unfortunately, for Android applications, we find that using these regular expressions would cause us to miss many relevant commits. For example, one relevant commit message says: "*added accelerometer tracking and GPS tweaking to save battery power*". Such commit

Table 2: The Different Wake Lock Flags for Android Power Management

Flag Value	CPU	Screen	Keyboard
PARTIAL_WAKE_LOCK	On	Off	Off
SCREEN_DIM_WAKE_LOCK	On	Dim	Off
SCREEN_BRIGHT_WAKE_LOCK	On	Bright	Off
FULL_WAKE_LOCK	On	Bright	Bright

would be omitted since it does not match any of the regular expressions. To prevent such case, we generalizes the above regular expressions to "*power*" and "*energy*". The use of more general regular expressions make it necessary for us to perform a manual step to exclude irrelevant commits (see Section 2.2.2). We get 510 and 59 commits whose messages contain the keywords "power" and "energy" respectively.

We also observe that the keyword "battery" is indicative of power management commits in Android applications. For instance, we find a commit which says: "*Fix battery drain*". So we also use the regular expressions "*battery*" and "*batteries*" to filter the commits in our collected data and get an additional 582 commits.

Android developers use specific APIs to manage power consumption. After looking up the Android API document, we find the API *PowerManagement.WakeLock* plays a key role in Android power management. A wake lock can be used to indicate that an application requires a mobile device to stay on because the application needs to use system resources. The wake lock can be used to keep the status of CPU, screen and keyboard on or off. The Android Power Management API document describes the different wake lock flags available that change the power state of the device. Table 2 presents the different wake lock flags for Android power management[5].

Thus, the appropriate use of wake lock could affect the Android device's power or battery. When we are checking the commits filtered by the keywords "power", "energy" and "battery", we find some commits that are related to wake lock use. For instance, one commit contains both the keywords battery and wakelock, it says "*Lower wakelock time to hopefully improve on active battery usage*". Another example commit message does not contain keyword "wakelock", but we find in this commit the developer sets wake lock to be null (see Figure 3) and claims that the *power consumption has been fixed*. Based on these findings, we decide to also use the regular expressions "*wakelock*" and "*wake lock*" to filter the collected commits. Using these two regular expressions, we get an additional 331 commits.

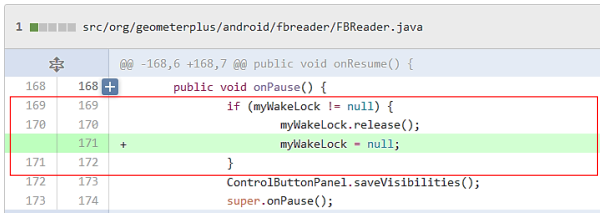


Figure 3: An Example Commit that Contains Wakelock Related Code

[5]http://developer.android.com/reference/android/os/PowerManager.html

Summary: We use four keywords: "power", "energy", "battery" and "wakelock" to filter the commits automatically and get 510, 59, 582, and 331 commits respectively. There are some overlaps among the four sets of commits since some commit messages contain multiple keywords, such as *save battery power*. For such cases, we only consider the commit once.

2.2.2 Manual Checking

We manually check each of the filtered commits through the interface provided by GITHUB which greatly increases the readability of a patch. By using this interface, we can easily map a commit to its source code modifications, observing which lines of code were added, modified, or removed. We use this interface to investigate the source code corresponding to a given commit in order to check whether it is relevant to Android power management.

There are many commits that are unrelated to Android power management since the filtering regular expressions that we use is not very strict. These kinds of commits are often easily recognizable by human; for example, there are some commits with the word "PowerPC" which is a RISC instruction set architecture in the application "dolphin-emu/dolphin". These commits match the regular expression "*power*" which we use, and we exclude them after a manual checking step. Another example is a commit whose message is: "*Fixed voltage and battery level (were negative)*". This commit is a bug fix on battery level display, and is not related to Android power management. There are many other kinds of commits which are unrelated to Android power management; due to the page limitation, we do not list all of them.

The manual check process is performed by the first author and another two graduate students in Zhejiang University. We divide all filtered commits into three parts according to the keywords we use: "power" and "energy", "battery", "wakelock". Each person analyzes two of the three parts, and thus each commit is at least checked by two people. For each commit, if the two people's opinions differ, then the third person will be involved and everyone would discuss it together and decide whether it is related to Android power management.

At the end of this manual check process, we get 468 commits that are related to Android power management.

2.2.3 Quantitative and Qualitative Analysis

We collect the statistics of Android applications that contain at least one power management commit. The collected statistics include their lines of code (LOC), number of commits, number of contributors, and their age. For each application category, we also compute the number of applications under that category, the number of applications under that category that contain at least one power management commit, and the number of commits of applications in that category that are related to power management.

We also perform a qualitative analysis to group the power management commits into power management activity groups. To do so, we follow an open card sort approach [36]. Our card sort process consists of two phases: In the *preparation* phase, we create one card for each commit message. In the *execution* phase, cards are sorted into meaningful groups with a descriptive title. Our card sort was open, meaning we had no predefined groups; instead,

we let the groups emerge and evolve during the sorting process. By contrast, a closed card sort has predefined groups, which is typically used when the themes are known in advance. The first author and the other two graduate students of Zheiang University jointly sorted the card.

3. CASE STUDY RESULTS

In this section, we first present the statistics of the collected power management commits. We then consider the following two research questions:

RQ1: How developers manage power consumption in Android applications?

RQ2: What type of Android application are more concerned about power management?

3.1 Statistics of Power Management Commits

We find a total of 468 Android power management commits. This number represents 32.12% of the total number of commits we found in our initial query with four keywords "power", "energy", "battery" and "wakelock". These commits are performed in 154 different Android applications which belong to 15 different categories. Notice there are 17 categories in F-Droid, and only two categories *Graphics* and *Sports & Health* are not involved in our collected commits. This is due to the low number of apps belonging to these two categories, i.e., the two categories only contain 11 and 13 apps respectively. Figure 4 presents a box plots showing the distribution of the lines of code (LOC), number of commits, number of contributors, and age (in years) of the 154 apps. Age is calculated as the interval between the first commit and the last commit that we collect. In the figure, the symbol × represents the mean value.

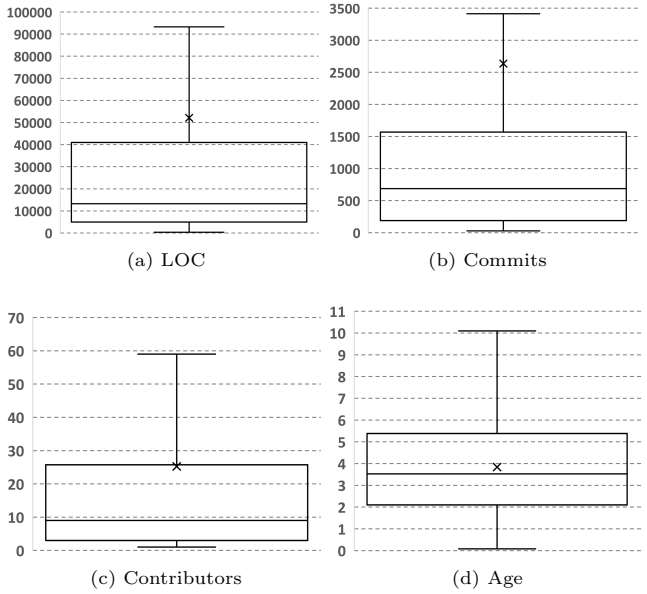


Figure 4: Distributions of the Lines of Code, Number of Commits, Number of Contributors, and Age of the Applications.

The 154 Android applications have 51,347±201,789 (mean±standard) lines of code (3rd quartile: 40,665, min: 317, max: 2,137,398), 25±42 different contributors (3rd quartile: 25, min: 1, max: 284), 2,594 commits (3rd quartile: 1568, min: 29, max: 82,692) and are 3.8±2.4 years old (3rd quartile: 5.4, min: 0.1, max: 14.25). The size of 3.18% of the applications are at most 1,000 lines of code, 38.85% of them are from 1,001 to 10,000 lines of code and more than half (i.e., 57.86%) of them are over 10,000 lines of code.

Table 3 shows the top-10 Android applications which have the largest number of power management commits. From the table, we notice that *Zanshinmu/Wifi-Fixer*, which is a *Connectivity* application, has the highest number of power management commits (i.e., 25 commits). We reanalyze these commits and find that the purposes of these power management commits vary. For example, some commits are performed to conserve battery by reducing network scan frequency, or optimize wake lock usage. Some commits add wake lock to ensure that some functionalities work correctly. This shows that the same Android application often requires developers to perform different power management activities. We can find a similar conclusion by analyzing other applications in the table. Their commits were also performed for different purposes related to power management. Note that these 10 applications belong to 6 different categories, i.e., *Connectivity*, *Navigation*, *Development*, *Games*, *System*, and *Phone & SMS*.

Table 3: Top-10 Android Applications with Largest Number of Power Management Commits

Application Name (Owner/Repo)	Category	#Power Management Commits
Zanshinmu/Wifi-Fixer	Connectivity	25
mozilla/MozStumbler	Navigation	23
Mobiperf/MobiPerf	Development	20
scummvm/scummvm	Games	12
Grarak/KernelAdiutor	Development	11
yuriykulikov/AlarmClock	System	10
dolphin-emu/dolphin	Games	10
jonatkins/ingress-intel-total-conversion	Games	9
servalproject/batphone	Phone & SMS	9
Yakoo63/gtalksms	Phone & SMS	9

3.2 RQ1: How developers manage power consumption in Android applications?

To answer this research question, we cluster the collected power management commits into different groups. We perform an open card sort [36] to create the groups. First we print each commit message on a card, we then discuss the commit message and iteratively sort them into groups. Our card sort identify six categories, as shown in Table 4. In the following paragraphs, we discuss the detail of each category.

Table 4: The Different Categories of Selected Commits

Category	# Commits
Power Adaptation	51
Power Consumption Improvement	64
Power Usage Monitoring	84
Optimizing Wake Lock Usage	66
Adding Wake Lock	90
Power Management Bug Fix or Code Refinement	113

3.2.1 Power Adaptation

For mobile devices, their batteries could be under different conditions, such as power-save mode, low power mode or high power mode. Some Android applications would adjust their power management strategies to ensure that their functionalities work optimally as the battery condition changes. Different power adaptation actions are performed in this

group of commits according to different battery conditions. We list the messages of some representative commits for different battery conditions below:

Power-save mode

- *Make CircleView respect power-save mode + code clean-up* **(1)**
- *adding new battery saving gps handling mode* **(2)**
- *Only disable WiFi (battery saving mode) when JAWS switched it automatically on* **(3)**
- *Merge pull request #1270 from garvankeeley/motion-detection. Battery-saving mode (by pausing GPS and scanners) on unchanging location.* **(4)**

Low power or under certain power threshold

- *ActiveMode: Option to disable on low battery. Not tested, but should be fine* **(5)**
- *Send turn off broadcast when battery is low* **(6)**
- *Stop scanning on low battery* **(7)**
- *Auto-off on low battery only if the device is not plugged-in Fixed compilation with standard ARM version* **(8)**
- *shutdown at certain battery level* **(9)**

High power or battery is charging

- *Add ability to only do autodl when charging. Defaults to allow autodl on battery to preserve existing behavior* **(10)**
- *activate artist and album search by default on higher power phones* **(11)**
- *Added Wifi on when power connected rule* **(12)**
- *added "turn wifi on when power connected" and "don't turn wifi on when in airplane mode" feature* **(13)**

From the above examples, we notice battery condition has an important effect on Android power management, and developers need to consider adapting the functionalities of the applications in different battery conditions. For example, if the power is lower than a threshold or if the mobile device is in a power-save mode, some features such as GPS, Wifi, animation, etc., may be disabled to save power (see commit **1-9**). And if the mobile devices are in high-power or are charging, Android applications can enable more functionalities to improve user experience, since the battery usage is not a big issue then (see commit **10-13**).

3.2.2 Power Consumption Improvement

The commits in power consumption improvement group perform changes to optimize the power consumption by using different methods, such as disabling certain application features, optimizing some functions or models, using more power efficient libraries, etc. Some commits in this group clearly write the reason why the improvement is made and/or how to modify the code to optimize power consumption. Some representative examples are listed below:

- *Pebble: try to shut up datalog, which might cause battery drain on the watch* **(14)**
- *Testing new fix for massive battery usage caused by GPS not being disabled during application pause.* **(15)**
- *added accelerometer tracking and GPS tweaking to save battery power - WIP* **(16)**
- *Reduce battery drain caused by insainly high value of widget update period. The period is now set to 1 hour but we still wake up the device to perform the update. We might consider the use of an alarm to avoid waking up the device when it is asleep, or at least let the user configure the update period himself.* **(17)**
- *Revert "Add event content observer". Content observer is called multiple times without any apparent reason which leads to high battery usage.* **(18)**
- *user more battery efficient heartbeat* **(19)**
- *added multicore power saving* **(20)**
- *Improve CPU resources when ignoring blocked apps. Use improved iptables rules to ignore logging for blocked apps. Instead of logging all apps and having to parse each and every log message to determine if a message belongs to a blocked app, we can now simply add an iptables rule to not provide any log messages for blocked apps! This means we no longer have to waste precious CPU/battery resources parsing messages that we don't care about.* **(21)**

From the above examples, we notice that various abnormal battery usage problems are fixed in different ways, e.g., disabling a functionality such as GPS (see commit **15, 16**) and logging (see commit **14, 21**), using more power efficient libraries or functionalities (see commit **16, 19**), reducing the update frequency or number of function calls (see commit **17, 18**), and supporting multicore power saving (see commit **20**).

Some commits just claim that they solve the power consumption problems but do not say how they solve the problems, such as "*Fix battery consumption*" or "*Fixed battery drain*". But through looking into the commit patch using GITHUB interface, we also know what is the power consumption problem and how it is fixed. For example, Figure 5 is the screenshot of the commit patch with message: "*Fixed battery drain*" viewed from GITHUB interface. From this patch, we could know that in order to save battery, the developer makes one thread of this application to stop waiting if the waiting time exceeds 10 seconds.

```
1433 +          long time = System.currentTimeMillis();
1434            while (!mOpened) {
     -          } // Delay the thread until the fragment has finished opening
1435 +              if (System.currentTimeMillis() - time > 10000) {
1436 +                  Log.w(TAG, "Task running for over 10 seconds, something is wrong");
1437 +                  cancel(true);
1438 +                  break;
1439 +              }
1440 +          }
```

Figure 5: Screenshot of An Example Patch Viewed Using the GITHUB Interface

3.2.3 Power Usage Monitoring

In the power usage monitoring group, the Android developers usually add some UIs or configurations to inform users the status of battery usage. These commits do not change the application behavior directly but inform users the status of battery usage to help users manage the power consumption by themselves. The following are some of the representative example commits in this group:

- *Display battery label in RED when battery is LOW* **(22)**

- *Use CircleChart for battery level* **(23)**
- *Added battery level from/to to screen (with pref)* **(24)**
- *footer: battery value monitoring is added. myBatteryLevel is added to ZLApplication. FBReader battery monitor is added.* **(25)**
- *show system battery usage* **(26)**
- *Display current energy for whole portal* **(27)**
- *Added battery level indicator when device is not charging* **(28)**
- *New dna binary with instrumentation logging mode & more battery monitoring.* **(29)**

3.2.4 Optimizing Wake Lock Usage

The commits in this group try to optimize wake lock usage. Wake lock is used to help developers control the CPU, screen and keyboard, and its usage may affect the power consumption. Inappropriate wake lock usage will cause power to be wasted unnecessarily. For example, in commit **36**, the developer tried to use wake lock instead of a handler but this change causes battery drain, so they had to revert back to previous code. The following are some of the representative example commits in this group:

- *Cleaned up wakelock. Cleaned up dialog constants. Added JOU to the database* **(30)**
- *Release WakeLock when it isn't needed. Release WakeLock in Presto's MediaPlayer when it is not needed. That way, it is going to behave just like Android's MediaPlayer.* **(31)**
- *eliminated wake locks for network check when wifi is disabled* **(32)**
- *changed wakelock timing* **(33)**
- *changed to use PARTIAL_WAKE_LOCK* **(34)**
- *Try again to only use partial wakelock* **(35)**
- *TokencodeBackend: Revert back to Handler in favor of AlarmService. AlarmService wound up holding wakelocks and draining the battery. Use Handler instead, and add checks to make sure we don't try to update the UI when the display is off.* **(36)**
- *Lower wakelock time to hopefully improve on active battery usage* **(37)**

These examples give us some hints on how to use wake lock appropriately: 1) Release wake lock after using it (see commit **30**). 2) Try lower wake lock time if possible (see commit **33, 37**). 3) Use PARTIAL_WAKE_LOCK if possible since it only requires to keep CPU on and keep screen and keyboard off (see commit **34, 35**). 4) Use wake lock correctly (see commit **30, 32, 36**).

3.2.5 Adding Wake Lock

The commits in this group are performed to keep the apps running well by adding wake locks. Any usage of wake lock has impact on battery power. But it is often necessary to ensure the normal working of applications. Comparing with the previous commit group "Optimizing Wake Lock Usage" which tries to lower power consumption, the commits in this group will increase power consumption. For example, commit **41** tries to keep network connection on using a wake lock. More examples are shown below.

- *Use a wake lock during syncing. Ensure that the device remains on until the sync is complete.* **(38)**
- *Added a WakeLock to keep the device awake until scrobbling is completed. Changed the scrobble waiting time to 1 minute.* **(39)**
- *Switching to startService instead of alarm manager. Using a partial wake lock in the service to prevent CPU from stopping.* **(40)**
- *Create and release a wakelock every 25 minutes, to avoid having the network disconnect after 30 minutes* **(41)**

3.2.6 Power Management Bug Fix or Code Refinement

The purpose of commits in power management bug fix or code refinement group is to fix Android power management bug, or refine and refactor related code. If a code fragment is related to Android power management, and a bug or a code refinement happens in the code fragment, we consider this kind of commit to belong to this group. Below are some of the representative examples in this group.

- *Fix crash when notification received on API < 7. -Caused by calling powerMan.isScreenOn() which was introduced in API 7.* **(42)**
- *added a few features to power manager* **(43)**
- *Fixed the remaining known power bugs; cleaned code a bit- $2 \wedge 3 \wedge 2$ is 64 instead of 512; not a bug but not ideal either* **(44)**
- *Bugfix: Leaked wake lock via scan code path* **(45)**

Some bugs are caused by using inconsistent version of APIs. (see commit **42**). In some commits, developers refine the power management code; for example in commit **43**, we find that the developer refined the code about battery status update. We also observe that there are many commits that contain keyword "wakelock" in this group (see commit **45**).

3.3 RQ2: What types of Android applications are more concerned about power management?

Although power consumption is a very important aspect for Android applications, for certain applications or certain categories of applications, developers often need to focus on other aspects such as functionality, user experience, performance, etc. and pay little even no attention on power management. In this work, we are interested to find the types of Android applications (out of the 17 that appear in F-Droid – see Table 1) whose developers are more concerned about power management.

Table 5 shows the percentage of applications for each application category that contain some power management commits. The first column is the application category, the 2nd column is the number of applications we crawled from F-Droid whose source code is in Github, the 3rd column is the number of applications with power management commits, and the last column is the percentage of applications with power management commits over the total number of crawled applications for each category.

Two out of the 17 categories, i.e., *Graphics* and *Sport & Health*, contain no applications with power management

commits. This is maybe because there are only few applications for these two categories – 11 and 13 respectively (see the last two rows in the Table 5). Notice that since some Android applications can belong to multiple categories, the total number of applications (across all categories) shown in this table is larger than the number of applications in our data set.

Table 5: Percentage of Applications with Power Management Commits ($\frac{\#AppsWithPM}{\#Apps}$) For Each Application Category

Category	#Apps	#AppsWithPM	Percentage
Connectivity	59	13	22.03%
Development	74	10	13.51%
Games	124	8	6.45%
Internet	187	27	14.44%
Money	24	3	12.50%
Multimedia	172	27	15.70%
Navigation	95	14	14.74%
Phone & SMS	31	10	32.26%
Reading	76	7	9.21%
Science & Education	91	8	8.79%
Security	56	7	12.50%
System	175	19	10.86%
Theming	77	3	3.90%
Time	75	6	8.00%
Writing	46	2	4.35%
Graphics	11	0	0.00%
Sport & Health	13	0	0.00%
All	**1362**	**164**	**12.04%**

On average, power management commits appear in the repositories of 12.04% Android applications in our dataset. This suggests that many Android apps pay more attention on other aspects such as user experience, performance, etc. However, this percentage is significantly larger and smaller for some categories of apps.

Among the 17 categories in Table 5, the percentages for categories *Phone & SMS* and *Connectivity* are much higher than other categories, i.e., 32.26% and 22.03% respectively. Android apps that belong to *Phone & SMS* category help manage call and message data of users and they usually run in the background. One power management commit of an app named *GTalkSMS* which allows users to control phone and send/receive SMS has the following message: "*set keepAlive interval to 15 min to save energy*". Many apps like *GTalkSMS* that run in the background are doing so for good reason, e.g. syncing, providing location data or otherwise doing what they were designed to do, etc. However, it is also necessary to keep these kinds of apps energy efficient at the same time. The apps in the category *Connectivity* help users manage connections between their mobile device and other devices over Bluetooth, NFC, Wi-Fi P2P, USB, and SIP, etc. The connectivity function can consume much power. Thus, these apps are often careful with power consumption.

Excluding *Graphics* and *Sport & Health* categories which have very few applications, *Theming*, *Writing* and *Games* have very low percentages of applications with power management commits, i.e., 3.90%, 4.35% and 6.45%, respectively. This suggests that developers of apps in these categories focus more on other concerns beyond power consumption. For instance, for game apps, user experience is often the most important thing. Often these game apps consume much power to support good user experience.

Figure 6 shows the number of power management commits for each category. From this figure, we could see that categories *Connectivity*, *Internet* and *System* have the most numbers of power management commits, while the categories *Money*, *Theming* and *Writing* have the least number of power management commits.

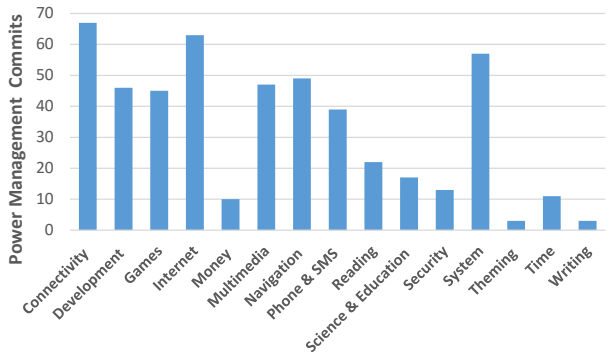


Figure 6: The Number of Power Management Commits for Each Category

For each of the six power management activities described in RQ1 (Section 3.2), we also calculate the number and percentage of power management commits for each application category. Table 6 shows the result where percentages which are larger than 25% are highlighted in bold. We ignore the percentages for application categories *Money*, *Theming* and *Writing* since the number of commits in these categories are very small (≤ 10).

From this table, we can see for many application categories, the percentage of power management commits belonging to *Bug Fix & Code Refinement* activities is the largest (24.19% on average). This shows that writing power management code is not trivial and developers need to be careful lest they introduce bugs.

The percentage of power management commits belonging to *Power Adaptation* activity is larger for application category *Navigation* than other categories (i.e., 26.53%). We find that an application named *MozStumbler* in category *Navigation* contributes 10 out of the 13 power management commits belonging to this activity. *MozStumbler* is a data gathering application for Mozilla Location Service. It requires GPS and has a scanner mode, and both consume a lot of energy. Thus, it chooses to disable these two functionalities when in power save mode or low power. For example, one of its power management commits says: "*Battery-saving mode (by pausing GPS and scanners) on unchanging location*".

The percentage of commits belonging to *Power Consumption Improvement* activity is larger than others for the application categories *Internet* and *Time* (26.98% and 27.27% respectively). The applications in these two categories often optimize power consumption by reducing update frequency or function calls. For example, the commit **18** described in Section 3.2 belongs to a *Time* application named *PoliteDroid* which activates silent mode during events recorded in user's calendar.

Some categories of applications, i.e., *Games*, *Navigation* and *Phone & SMS*, made more *Power Usage Monitoring* commits than other power management commits (i.e., 42.22%, 30.61% and 38.46% of the total number of commits respectively). Often users know that some applications (e.g., *Games* and *Navigation* applications) would consume a lot of power, but they still want to launch these applications. For

Table 6: Numbers and Percentages of Power Management Commits Belonging to Different Activities for Each Application Category

	#Power Adaptation	#Power Consumption Improvement	#Power Usage Monitoring	#Optimizing Wake Lock	#Adding Wake Lock	#Bug Fix & Code Refinement
Connectivity	12 (17.91%)	8 (11.94%)	9 (13.43%)	12 (17.91%)	9 (13.43%)	**17 (25.37%)**
Development	2 (4.35%)	4 (8.70%)	6 (13.04%)	5 (10.87%)	9 (19.57%)	**20 (43.48%)**
Games	4 (8.89%)	6 (13.33%)	**19 (42.22%)**	3 (6.67%)	1 (2.22%)	**12 (26.67%)**
Internet	1 (1.59%)	**17 (26.98%)**	1 (1.59%)	12 (19.05%)	**19 (30.16%)**	13 (20.63%)
Money	5 (50.00%)	0 (0%)	1 (10.00%)	1 (10.00%)	2 (20.00%)	1 (10.00%)
Multimedia	3 (6.38%)	6 (12.77%)	1 (2.13%)	10 (21.28%)	**16 (34.04%)**	11 (23.40%)
Navigation	**13 (26.53%)**	7 (14.29%)	**15 (30.61%)**	1 (2.04%)	2 (4.08%)	9 (18.37%)
Phone & SMS	2 (5.13%)	4 (10.26%)	**15 (38.46%)**	5 (12.82%)	**11 (28.21%)**	2 (5.13%)
Reading	2 (9.09%)	1 (4.55%)	2 (9.09%)	**6 (27.27%)**	5 (22.73%)	**6 (27.27%)**
Science & Education	0 (0%)	2 (11.76%)	4 (23.53%)	2 (11.76%)	**6 (35.29%)**	3 (17.65%)
Security	1 (7.69%)	3 (23.08%)	0 (0%)	2 (15.38%)	2 (15.38%)	**5 (38.46%)**
System	6 (10.53%)	8 (14.04%)	9 (15.79%)	8 (14.04%)	9 (15.79%)	**17 (29.82%)**
Theming	0 (0%)	1 (33.33%)	2 (66.67%)	0 (0%)	0 (0%)	0 (0%)
Time	2 (18.18%)	**3 (27.27%)**	1 (9.09%)	1 (9.09%)	2 (18.18%)	2 (18.18%)
Writing	0 (0%)	1 (33.33%)	0 (0%)	0 (0%)	1 (33.33%)	1 (33.33%)
All	53 (10.77%)	71 (14.43%)	85 (17.28%)	68 (13.82%)	94 (19.11%)	119 (24.19%)

such cases, it is better to show users the battery usage of the applications or notify low battery situations.

Wake locks are often required to ensure that some functionalities remain workable and to improve user experience. There are many commits that belong to *Optimizing Wake Lock* activity across all application categories, especially *Reading*. The percentages of commits belonging to *Adding Wake Lock* activity are high for four application categories, i.e., *Internet*, *Multimedia*, *Phone & SMS* and *Science & Education*. They are 30.16%, 34.04%, 28.21% and 35.29% respectively.

4. DISCUSSION

4.1 Implications

Our study shows that power consumption management is important when developing many Android applications. Many developers care about power consumption and made commits which improve power management. We identify six groups of power management commits which correspond to different power management activities. We also find the dominant power management activities performed by apps in different groups. We make the following recommendations to Android developers related to power management:

- Extra attention must be given to resources or features that consume much energy. These include resources such as GPS, wifi, etc. and features such as animation, frequent updates, etc. Many power management commits in our dataset control the usage of such resources and features.

- It is a good practice to consider the battery condition (e.g., power is low, battery is charging) and manage power consumption accordingly. Developers need to balance application usability and battery life.

- Showing power usage is a frequently used alternative strategy which delegates power management decision to end users. This is useful for applications that always consume much power, such as game apps.

- Wakelock needs to be used with care. It can help save power if it is used appropriately. However, misuse of wakelock will cause power consumption problems.

- Power management needs to be designed with care and not an after-thought. Our study shows that many bugs affect power management code.

- When performing power management activities, the category of the Android app is a very important factor to be considered.

4.2 Threats to Validity

Internal validity. We perform a semi-automated process to identify power management commits. The process may miss some power management commits that do not contain either "power", "energy", "battery", or "wakelock" in their messages. Still, compared with previous studies [30, 26], we have included more keywords, i.e., "battery" and "wakelock", which are closely related to Android power management. Furthermore, the keywords we use are less strict than those used in prior studies [30, 26], e.g., *power* versus *power consum* or *power sav*. In this way, we miss less relevant commits, but need to make extra effort to manually filter unrelated commits. Note that we may wrongly include or exclude a commit as a power management commit in the manual analysis process. To reduce this threat to internal validity, three people perform this manual analysis, and each commit is analyzed by at least two people, and any conflict is resolved by everyone.

The open card sort process relies on human judgement and it is possible that we make wrong generalizations. To reduce this threat to internal validity, three people performed this process. Similar open card sort processes were performed in many studies in the literature [12, 22, 33, 35, 41].

External validity. We only analyze 1,273 open source Android applications from F-Droid whose source code is in GITHUB. Out of these applications, we only investigate a total of 468 power management commits from 154 different applications. It is possible that our findings do not generalize to all Android applications. Still, we are the first to analyze hundreds of power management commits from more than a hundred Android applications by investigating the version control systems of more than a thousand Android applications. In the future, we plan to expand our study to include even more applications and power management commits. Finally, the commit messages we analyzed in this work are all written in English.

5. RELATED WORK

The energy consumption problem has been investigated by many studies. Most of them focus on the trade-off of individual characteristics of an application and energy consumption. These characteristics vary from data structures [15, 7, 23], VM services [5], cloud offloading [17], code obfuscation [32], design patterns [31, 18], and static OO metrics [13, 14]. Such studies give software developers some assistance to develop energy efficient applications. However, the applications in these studies are in wide spectrum, ranging from operating systems, kernels and mobile applications but not focused on mobile applications, so some of their findings might not be generalizable for power consumption management of applications running on the Android platform. Android applications have unique properties (e.g., high reliance of libraries) and use unique libraries (e.g., wake lock API) and thus require special treatment. Thus, in this paper, different from the above studies, we focus on Android applications and perform an empirical study on their power consumption management commits.

Our study is inspired by the studies of Pinto *et al.* [30] and Moura *et al.* [26]. Both of their studies want to investigate what are the solutions proposed by software developers in order to improve software energy consumption. Pinto *et al.* [30] use StackOverflow which is a developer oriented Q&A website as their primary data source. They analyze many questions and answers that were related to energy efficiency in StackOverflow and get some interesting findings which provide useful insights such as the most common energy consumption related problems, and the solutions to them. Moura *et al.* [26] use Github as their primary data source. Notice the commits in Github represent the actual solutions that developers employ in practice, and their study also provides some useful findings about power consumption in practice, such as developers are not always certain that their source code modifications will affect energy consumption. Similar to the studies mentioned in the previous paragraph, these studies analyze various kinds of applications and are not focused on Android applications. Moura *et al.* stated that "'*they found 47 commits that are targeting mobile software*", and "*Although most of these commits are related to Android Kernels, we found several commits that focus on the application-level*". Different from these studies we focus on Android applications and analyze hundreds commits from 147 Android apps. Indeed, if we only use the regular expressions used in the previous studies by Moura et al. and Pinto et al. (i.e., *energy consum*, *energy efficien*, *energy sav*, *save energy*, *power consum*, *power efficien*, *power sa*, and *save power*) only few commits are filtered (i.e., 21 commits). Thus in this paper, we use regular expressions *power* and *energy* instead, and add other two additional keywords "battery" and "wakelock". Then, we manually check these retrieved commit messages.

Pathak *et al.* [28] crawl 4 online forums and bug trackers relating to mobile platforms and their applications, searching for energy-efficiency issues. They analyze posts relating to energy problems and derive a taxonomy for faulty implementations and hardware errors they classified as different types of energy bugs. They also present an investigation aiming to understand the root causes for energy consumption problems in mobile applications [29]. Most of energy bugs in their studies are in hardware level, such as battery problems, SIM card problems and OS configuration problems. On the other hand, our study focuses on Android power consumption in application level.

Heikkinen *et al.* [11] study energy problems of mobile handsets through a questionnaire-based study. Their findings reveal that users are interested in energy consumption statistics of their mobile devices as well as in information on how they could optimize their devices. Wilke *et al.* [38] perform a study using apps of Google Play as their data source. Their study focus on post-programming stages of software lifecyle which is different from our study. Their work is particularly interesting in correlating energy consumption problems with the user ratings to the apps, the pricing of apps, and different natures of apps. Different from the above studies, in this paper, we study the power management activities that developers perform on Android applications by mining their commit logs.

Some researchers such as Li *et al.* [19] and Linares-Vasquez *et al.* [21] analyze power consumption of Android applications by analyzing their source code and APIs. Their studies focus on what kinds of APIs and source code usage are more energy-greedy or energy-efficient. There are several interesting findings, such as HTTP request is the most energy consuming operation of the network. In this work, we analyze commits rather than code; different from analyzing raw code, we have short textual description that can provide semantics of changes made in a commit. Our goal is to find power management activities that developers perform on Android applications which are different from the goals of the works by Li *et al.* and Linares-Vasquez *et al.*

6. CONCLUSION AND FUTURE WORK

In this paper, we conduct an empirical study on power management commits of Android apps. Starting from a set of 1,273 Android apps in F-Droid who host their source code in Github, we get 468 power management commits through automatic filtering and manual analysis. We identify 6 different kinds of power management commits using an open card sort process: *Power Adaptation*, *Power Consumption Improvement*, *Power Usage Monitoring*, *Optimizing Wake Lock*, *Adding Wake Lock* and *Bug Fix & Code Refinement*. These correspond to six common power management activities that developers perform. We find that the dominant power management activities differ for different application categories. For example, the percentage of commits belonging to *Power Consumption Improvement activity* is larger than others for the application categories *Internet* and *Time*, while *Navigation* apps make more *Power Adaptation* commits.

In future work, we plan to survey Android developers to understand their view of power consumption management. We also plan to develop automated techniques that can help developers detect, locate, and fix power consumption problems. Furthermore, we want to compare the energy commits to other classes of commits and integrate issue tracker information to analysis power consumption commits.

Acknowledgment

This research was supported by NSFC Program (No.61572426) and National Key Technology R&D Program of the Ministry of Science and Technology of China under grant 2015BAH17F01. We also would like to thank Qiao Huang and Qingye Wang for their help in the manual checking of commits and card sort process.

7. REFERENCES

[1] Google play. https://en.wikipedia.org/wiki/Google_Play.

[2] Smartphone market share. http://www.idc.com/prodserv/smartphone-os-market-share.jspl.

[3] New research reveals mobile users want phones to have a longer than average battery life, November 2013.

[4] Brooks, D., Tiwari, V., and Martonosi, M. Wattch: A framework for architectural-level power analysis and optimizations. In *Proceedings of the 27th Annual International Symposium on Computer Architecture* (2000), pp. 83–94.

[5] Cao, T., Blackburn, S. M., Gao, T., and McKinley, K. S. The yin and yang of power and performance for asymmetric hardware and managed software. In *ACM SIGARCH Computer Architecture News* (2012), vol. 40, IEEE Computer Society, pp. 225–236.

[6] Chun, B.-G., Ihm, S., Maniatis, P., Naik, M., and Patti, A. Clonecloud: elastic execution between mobile device and cloud. In *Proceedings of the Sixth Conference on Computer Systems* (2011), ACM, pp. 301–314.

[7] Daylight, E. G., Fermentel, T., Ykman-Couvreur, C., and Catthoor, F. Incorporating energy efficient data structures into modular software implementations for internet-based embedded systems. In *Proceedings of the 3rd International Workshop on Software and Performance* (2002), ACM, pp. 134–141.

[8] Gu, X., Nahrstedt, K., Messer, A., Greenberg, I., and Milojicic, D. Adaptive offloading for pervasive computing. *Pervasive Computing, IEEE 3*, 3 (2004), 66–73.

[9] Hao, S., Li, D., Halfond, W. G., and Govindan, R. Estimating android applications' cpu energy usage via bytecode profiling. In *Proceedings of the First International Workshop on Green and Sustainable Software* (2012), IEEE Press, pp. 1–7.

[10] Hao, S., Li, D., Halfond, W. G., and Govindan, R. Estimating mobile application energy consumption using program analysis. In *Proceedings of the 35th International Conference on Software Engineering (ICSE)* (2013), IEEE, pp. 92–101.

[11] Heikkinen, M. V., Nurminen, J. K., Smura, T., and Hämmäinen, H. Energy efficiency of mobile handsets: Measuring user attitudes and behavior. *Telematics and Informatics 29*, 4 (2012), 387–399.

[12] Hemmati, H., Nadi, S., Baysal, O., Kononenko, O., Wang, W., Holmes, R., and Godfrey, M. W. The msr cookbook: Mining a decade of research. In *Proceedings of the 10th Working Conference on Mining Software Repositories (MSR)* (2013), IEEE, pp. 343–352.

[13] Hindle, A. Green mining: investigating power consumption across versions. In *Proceedings of 34th International Conference on Software Engineering (ICSE)* (2012), IEEE, pp. 1301–1304.

[14] Hindle, A. Green mining: a methodology of relating software change and configuration to power consumption. *Empirical Software Engineering 20*, 2 (2015), 374–409.

[15] Hunt, N., Sandhu, P. S., and Ceze, L. Characterizing the performance and energy efficiency of lock-free data structures. In *Proceedings of the 15th Workshop on Interaction between Compilers and Computer Architectures (INTERACT)* (2011), IEEE, pp. 63–70.

[16] Krutz, D. E., Mirakhorli, M., Malachowsky, S. A., Ruiz, A., Peterson, J., Filipski, A., and Smith, J. A dataset of open-source android applications. In *Proceedings of the 12th Working Conference on Mining Software Repositories (MSR)* (2015), pp. 522–525.

[17] Kwon, Y.-W., and Tilevich, E. Reducing the energy consumption of mobile applications behind the scenes. In *Proceedings of the 29th International Conference on Software Maintenance (ICSM)* (2013), IEEE, pp. 170–179.

[18] Li, D., and Halfond, W. G. An investigation into energy-saving programming practices for android smartphone app development. In *Proceedings of the 3rd International Workshop on Green and Sustainable Software* (2014), ACM, pp. 46–53.

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

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

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

[22] Lo, D., Nagappan, N., and Zimmermann, T. How practitioners perceive the relevance of software engineering research. In *Proceedings of the 10th Joint Meeting on Foundations of Software Engineering (FSE)* (2015), ACM, pp. 415–425.

[23] Manotas, I., Pollock, L., and Clause, J. Seeds: a software engineer's energy-optimization decision support framework. In *Proceedings of the 36th International Conference on Software Engineering (ICSE)* (2014), ACM, pp. 503–514.

[24] McIntire, D., Ho, K., Yip, B., Singh, A., Wu, W., and Kaiser, W. J. The low power energy aware processing (leap) embedded networked sensor system. In *Proceedings of the 5th International Conference on Information Processing in Sensor Networks* (2006), ACM, pp. 449–457.

[25] Messer, A., Greenberg, I., Bernadat, P., Milojicic, D., Chen, D., Giuli, T. J., and Gu, X. Towards a distributed platform for resource-constrained devices. In *Proceedings of the 22nd International Conference on Distributed Computing Systems* (2002), IEEE, pp. 43–51.

[26] Moura, I., Pinto, G., Ebert, F., and Castor, F. Mining energy-aware commits. In *Proceedings of the*

12th Working Conference on Mining Software Repositories (MSR) (2015), pp. 56–67.

[27] Mudge, T., Austin, T., and Grunwald, D. The reference manual for the sim-panalyzer version 2.0.

[28] Pathak, A., Hu, Y. C., and Zhang, M. 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* (2011), ACM, p. 5.

[29] Pathak, A., Hu, Y. C., and Zhang, M. 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), ACM, pp. 29–42.

[30] Pinto, G., Castor, F., and Liu, Y. D. Mining questions about software energy consumption. In *Proceedings of the 11th Working Conference on Mining Software Repositories (MSR)* (2014), pp. 22–31.

[31] Sahin, C., Cayci, F., Gutiérrez, I. L. M., Clause, J., Kiamilev, F., Pollock, L., and Winbladh, K. Initial explorations on design pattern energy usage. In *Proceedings of First International Workshop on Green and Sustainable Software (GREENS)* (2012), IEEE, pp. 55–61.

[32] Sahin, C., Tornquist, P., McKenna, R., Pearson, Z., and Clause, J. How does code obfuscation impact energy usage? In *Proceedings of IEEE International Conference on Software Maintenance and Evolution (ICSME)* (2014), IEEE, pp. 131–140.

[33] Sartoli, S., and Namin, A. S. Poster: Reasoning based on imperfect context data in adaptive security. In *Proceedings of the 37th IEEE International Conference on Software Engineering (ICSE)* (2015), vol. 2, IEEE, pp. 835–836.

[34] Satyanarayanan, M., Bahl, P., Caceres, R., and Davies, N. The case for vm-based cloudlets in mobile computing. *Pervasive Computing, IEEE 8*, 4 (2009), 14–23.

[35] Siegmund, J., Siegmund, N., and Apel, S. Views on internal and external validity in empirical software engineering. In *Proceedings of the 37th International Conference on Software Engineering (ICSE)* (2015).

[36] Spencer, D. *Card sorting: Designing usable categories*. Rosenfeld Media, 2009.

[37] Weissel, A., Beutel, B., and Bellosa, F. Cooperative i/o: A novel i/o semantics for energy-aware applications. *ACM SIGOPS Operating Systems Review 36*, SI (2002), 117–129.

[38] Wilke, C., Richly, S., Gotz, S., Piechnick, C., and Assmann, U. Energy consumption and efficiency in mobile applications: A user feedback study. In *Green Computing and Communications (GreenCom), 2013 IEEE and Internet of Things (iThings/CPSCom), IEEE International Conference on and IEEE Cyber, Physical and Social Computing* (2013), IEEE, pp. 134–141.

[39] Yuan, W., and Nahrstedt, K. Energy-efficient soft real-time cpu scheduling for mobile multimedia systems. *ACM SIGOPS Operating Systems Review 37*, 5 (2003), 149–163.

[40] Zhang, Y., Huang, G., Liu, X., Zhang, W., Mei, H., and Yang, S. Refactoring android java code for on-demand computation offloading. In *ACM SIGPLAN Notices* (2012), vol. 47, ACM, pp. 233–248.

[41] Zimmermann, T., Nagappan, N., Guo, P. J., and Murphy, B. Characterizing and predicting which bugs get reopened. In *Proceedings of 34th International Conference on Software Engineering (ICSE)* (2012), IEEE, pp. 1074–1083.