Noname manuscript No.
(will be inserted by the editor)

Catalog of Energy Patterns for Mobile Applications

Luis Cruz · Rui Abreu

the date of receipt and acceptance should be inserted later

Abstract Software engineers make use of design patterns for reasons that range from performance to code comprehensibility. Several design patterns capturing the body of knowledge of best practices have been proposed in the past, namely creational, structural and behavioral patterns. However, with the advent of mobile devices, it becomes a necessity a catalog of design patterns for energy efficiency. In this work, we inspect commits, issues and pull requests of 1027 Android and 756 iOS apps to identify common practices when improving energy efficiency. This analysis yielded a catalog, available online, with 22 design patterns related to improving the energy efficiency of mobile apps. We argue that this catalog might be of relevance to other domains such as Cyber-Physical Systems and Internet of Things. As a side contribution, an analysis of the differences between Android and iOS devices shows that the Android community is more energy-aware.

Keywords Mobile applications; Energy Efficiency; Energy Patterns; Catalog; Open source software.

1 Introduction

The importance of providing developers with more knowledge on how they can modify mobile apps to improve energy efficiency has been reported in previous works (Li and Halfond, 2014; Robillard and Medvidovic, 2016). In particular, mobile apps often have energy requirements but developers are unaware that energy-specific design patterns do exist (Manotas et al., 2016). Moreover, developers have to support multiple platforms while providing a similar user experience (An et al., 2018).

Luis Cruz
INESC ID and University of Porto, Portugal
E-mail: luiscruz@fe.up.pt

Rui Abreu
INESC ID and IST, University of Lisbon, Portugal
E-mail: rui@computer.org

Design patterns have been formalized to provide general, reusable solutions to recurrent problems in software design. According to their main purpose, design patterns were originally categorized in *creational*, *structural*, and *behavioral* patterns (Vlissides et al., 1995). Further efforts have leveraged domain-specific catalogs of design patterns to meet non-functional requirements such as security (Yskout et al., 2015). Design patterns also play an important role in educating developers, since they tend to learn by looking at code examples or using boilerplate code following well defined solutions (Pham et al., 2015, 2013).

There is a number of practices from experienced developers that lie in the history of mobile app projects (Negara et al., 2014; Palomba et al., 2018). In this work, we collect the set of patterns that developers adopt to improve the energy efficiency of their apps. We analyze 1783 apps from Android (1027) and iOS (756) and compare practices of developers towards energy efficiency amongst the two most popular mobile platforms.

In particular, we aim to answer the following questions:

> **RQ1:** Which design patterns do mobile app developers adopt to improve energy efficiency?

We describe a catalog of 22 patterns that mobile app developers resort to when addressing energy efficiency. We document these patterns so that other developers learn more about energy best practices and reuse them in their projects.

> **RQ2:** How different are mobile app practices addressing energy efficiency across different platforms?

We found that Android developers have higher awareness towards energy efficiency improvement than iOS developers. We show the prevalence of each energy pattern in the two platforms and discuss potential causes.

This paper makes the following contributions:

- We propose a catalog of energy patterns with a detailed description and instructions for mobile app developers and designers. It is available online: `https://tqrg.github.io/energy-patterns`, and we welcome contributions from the community as pull request.
- We provide a dataset with 1563 commits, issues, and pull requests in which mobile app development practitioners address the energy efficiency of their apps. The dataset and collection tools are available online: `https://github.com/TQRG/energy-patterns`.
- We compare energy efficiency awareness in mobile app development in different platforms (viz. Android and iOS).

The remainder of this paper is organized as follows. Related work is discussed in Section 2. Section 3 outlines the methodology used to collect data and extract energy patterns in our study, followed by Section 4 describing the collection of energy patterns. Section 5 summarizes the collected data and discusses implications of the list of proposed patterns. Threats to the validity are discussed in Section 6. Finally, we draw our conclusions and point directions for future work in Section 7.

2 Related Work

Improving energy efficiency of mobile apps has gained the attention of the research community recently, which addressed the challenge in different ways: identifying energy bugs (Pathak et al., 2011a; Banerjee et al., 2014; Vekris et al., 2012), profiling energy consumption (Wilke et al., 2013; Liu et al., 2013; Hao et al., 2013; Behrouz et al., 2015; Pathak et al., 2011b; Zhang et al., 2010; Chowdhury et al., 2018a; Di Nucci et al., 2017; Hindle et al., 2014; Romansky et al., 2017), or understanding best coding practices for energy efficiency (Sahin et al., 2016; Cruz and Abreu, 2017, 2018; Linares-Vásquez et al., 2014; Pathak et al., 2012a).

In previous work, Moura et al. (2015) have mined 290 energy-saving software commits, identifying 12 categories of source code modification to improve energy usage (Moura et al., 2015): *Frequency and voltage scaling*, *Use power efficient library/device*, *Disabling features or devices*, *Energy bug fix*, *Low power idling*, *Timing out*, *Avoid polling*, *Pin management*, *Display and UI tuning*, *Avoid unnecessary work*, *Miscellaneous*, and *Outlier*. The programming languages used to implement the software systems used in this study were diverse: programming C (158 projects), Java (25 projects), Bourne Shell (17 projects), Arduino Sketch (15 projects), and C++ (12 projects). They found that roughly 50% of energy-saving commits target lower levels of the software stack (e.g., kernels and drivers), which is not a level of abstraction commonly considered during the design of mobile apps. Our work extends this approach to the ecosystem of mobile apps by compiling a set of coding practices that can be used by practitioners across mobile apps on different platforms. Thus, our dataset of apps also includes projects written in *Swift*, *Objective-C*, *Java*, *Kotlin*, and any other language used for mobile app development in iOS or Android. In addition, we detail these and other energy-saving categories with a context and guidelines to help developers decide on the most appropriate pattern. Moreover, we compare the prevalence of these patterns across different mobile platforms.

With a similar approach, Bao et al. (2016) have mined 468 power management commits to find coding practices in Android apps (Bao et al., 2016). Using a hybrid card sort approach, six different power management practices were identified: *Power Adaptation*, *Power Consumption Improvement*, *Power Usage Monitoring*, *Optimizing Wake Lock*, *Adding Wake Lock and Bug Fix & Code Refinement*. The study shows that power management activities are more prevalent in navigation apps. Conversely, our work focuses on energy-saving commits, pull requests, and issues. Using the same taxonomy, our work concentrates exclusively on coding practices for *Power Adaptation*, and *Power Consumption Improvement*. Moreover, rather than analyzing the prevalence of power management activities amongst different app categories, we emphasize on providing actionable findings for mobile app practitioners. Finally, we extend this work to the iOS mobile platform, which shares a big part of the mobile app market.

Previous work studied the views of mobile app developers on energy efficiency improvement by mining *StackOverflow*[1] posts (Pinto et al., 2014). It was found that developers make interesting questions about energy efficiency problems. However,

[1] *StackOverflow* is a collaborative Web platform for questions and answers on a wide range of topics in computer programming.

the answers provided on this topic are often flawed or vague. Our work analyses mobile app projects to collect recurrent solutions adopted by developers.

There is work that studied the impact of performance optimizations on the energy efficiency of mobile apps (Hecht et al., 2016; Cruz and Abreu, 2017; Linares-Vásquez et al., 2014; Sahin et al., 2016). However, only platform-specific optimizations were addressed — e.g., in early versions of Android using `get/set` methods internally was less energy efficient than accessing fields directly. In our work, we focus on patterns that can be applied regardless of the mobile platform being used.

Furthermore, related work studied the impact of high-level coding and design practices. E.g., the use of advertisement increases energy usage of mobile apps (Pathak et al., 2012a), bundling small HTTP requests can be used to enhance energy efficiency (Li and Halfond, 2014). We assess how mobile app developers are using these and other patterns to improve energy efficiency in real mobile apps. Measuring the effective impact of these optimizations on energy efficiency is out of the scope of our work.

Previous work has delivered a catalog of quality smell patterns for Android apps (Reimann et al., 2014). Our work differentiates by focusing on energy efficiency improvements and including iOS apps. Still, there is one pattern that is common in the two catalogs: Reimann et al.'s *Early Resource Binding* and our's *Open Only When Necessary*.

The impact of general purpose software design patterns on energy efficiency has been studied in previous work (Sahin et al., 2012). It was shown that design patterns may have different impacts on energy consumption. Related work has also evaluated the impact of different machine learning algorithms (McIntosh et al., 2018). The most efficient technique algorithm depends on properties such as the size of the dataset, and the number of data attributes. Our work differs as we exclusively focus on patterns that are applied to improve the energy efficiency of mobile apps.

Most of the works described above focus on a single platform, notably Android. As recently reported by the Google Android VP Dave Burke, Google may actually have less than 66% global market share. Hence, we have also considered iOS in our study. Related work has extended the study of coding practices to iOS. In a study with 279 iOS apps and 1551 Android apps, no significant distinction was found in the prevalence of code smells between iOS and Android apps (Oliveira et al., 2017). A different work has studied error handling practices of Swift[2] (Cassee et al., 2018). Nearly half of the 2733 Swift projects did not exhibit any error handling code. Our work provides more enlightenment on practices of developers amongst the iOS ecosystem.

3 Methodology

We designed a methodology to extract energy patterns from existing mobile apps. In total, we collect a total of 1027 Android apps and 756 iOS apps, including apps designed for smartphones, tablets, wearables, or e-paper devices. Essential to our analysis, apps from both platforms are open source and have their git repositories

[2] Swift is the official programming language for iOS apps.

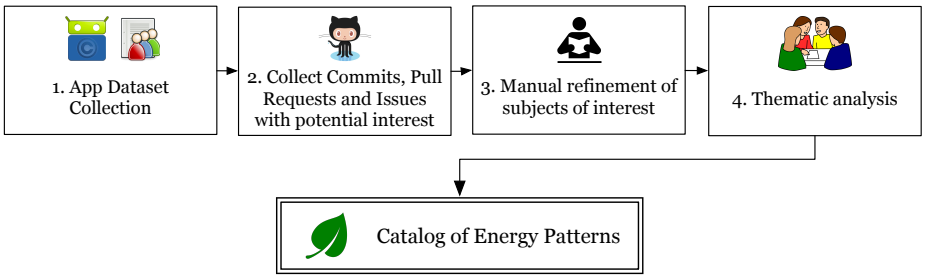

Fig. 1 Methodology used to extract energy patterns from mobile apps.

available on GitHub[3]. Our methodology is illustrated in Figure 1 and comprised the following four main tasks:

- App dataset collection
- Automatic gathering of subjects with potential interest (i.e., commits, issues, and pull requests)
- Manual refinement of subjects of interest
- Thematic analysis (infer energy patterns) according to the solution encountered to improve energy efficiency

3.1 App Dataset Collection

Multiple open source mobile app catalogs were combined to collect Android and iOS mobile apps. For Android, we resort to *F-Droid*, a catalog that lists 2800 free and open source Android apps[4]. There are open source apps that are not available in *F-droid* for not fulfilling free software requirements (e.g., Signal app[5]). Although these are just a minority of apps we argue that they can provide relevant input on energy efficiency practices. Thus, we included Android apps listed in community-curated collections of Android open source apps[6]. This resulted in 1027 apps — 1001 from *F-droid* and 26 from curated lists.

For iOS we use the *Collaborative List of Open-Source iOS Apps* [7], amounting to 829 apps listed with the help of a community of 195 collaborators. Given our constraint of having a publicly available GitHub repository, we ended up including 756 iOS apps in our study.

The apps used in this study are from a wide range of categories for Android and iOS, as depicted in Figure 2 and Figure 3, respectively. In addition, Table 1 shows the dispersion of apps in terms of popularity metrics: GitHub stars, GitHub forks, number of reviews, and rating. The number of reviews and the ratings are

[3] GitHub is a social coding platform with a git web interface.

[4] F-droid's website: `https://f-droid.org/` (Visited on January 10, 2019).

[5] Signal is an open source messaging app: `https://signal.org` (Visited on January 10, 2019).

[6] *Amazing open source Android apps* curated list available here: `https://github.com/Mybridge/amazing-android-apps` (Visited on January 10, 2019).

[7] The list is available here: `https://github.com/dkhamsing/open-source-ios-apps` (Visited on January 10, 2019)

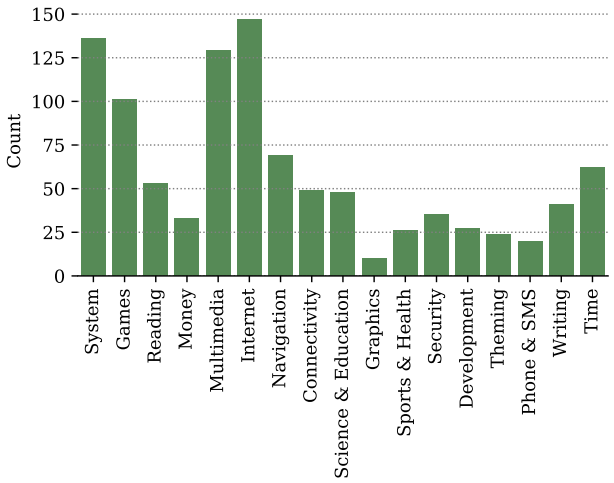

Fig. 2 Distribution of categories in Android apps.

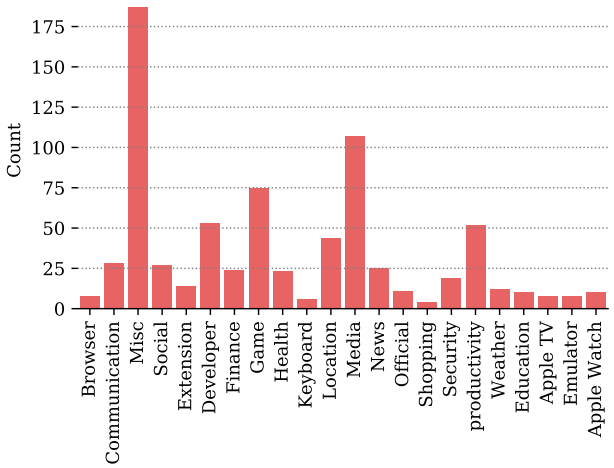

Fig. 3 Distribution of categories in iOS apps.

from the apps in the dataset that are published in the Google Play Store or the iOS App Store. Thus, only from a subset of the apps in this study: 64% of Android apps and 20% of iOS apps. GitHub stars go up to roughly 15K in both platforms, while GitHub forks up to 8K in Android and 5K in iOS. On average, the apps have a rating of 4 out of 5.

Table 1 Descriptive statistics of the popularity metrics of the Android and iOS apps in the dataset.

	Platform	Mean	Std	Min	25%	Median	75%	Max
Stars	Android	145	608	0	6	20	68	15159
	iOS	486	1272	0	18	71	329	15318
Forks	Android	75	302	0	4	14	46	7811
	iOS	118	354	0	7	20	71	4820
Number of Reviews*	Android	31855	529676	1	24	138	1087	13080790
	iOS	3241	12597	5	18	75	1038	115011
Rating*	Android	3.8	0.5	1.0	4.0	4.0	4.0	5.0
	iOS	4.1	0.7	2.0	3.6	4.0	4.5	5.0

*as in Google Play Store and iOS App Store.

3.2 Automatic gathering of commits, issues, and pull requests

In this step, we collect from all GitHub repositories in our dataset any commit, issue or pull request that potentially contains energy improvement practices. As done in previous work (Bao et al., 2016), any instance that mentions the words *energy*, *battery*, or *power* is selected. The following regular expression is used:

```
.*(energy|battery|power).*
```

The *GitHub API v3* was used to automatically collect data from public repositories. Note that we only include commits that were merged in the *default* branch of the projects. In total, we gathered 6028 entries that matched this regular expression.

3.3 Manual Refinement

We understand that the regular expression used in the automatic data collection yields many false positives. As an example, consider the following entries collected in the previous step:

- *"Adding a link to the app's device settings in iOS Settings.app would be great for **power** users."* (False positive found in the app *WordPress* for iOS[8]).
- *"(...) recently a lot of issues that the core team does not have the **energy** to implement themselves have been closed."* (False positive found in the app *Minetest* for Android[9]).
- *"One thing is really important on mobile devices, and that is **power** consumption"* (True positive found in the app *ChatSecure* for iOS[10]).

Although the first two examples match with the regular expression, they are not expected to deal with energy-related practices. On contrary, the last example

[8] The whole thread can be found here: `https://github.com/wordpress-mobile/WordPress-iOS/issues/6057` (Visited on January 10, 2019).

[9] The whole thread can be found here: `https://github.com/minetest/minetest/issues/6394` (Visited on January 10, 2019).

[10] The whole thread can be found here: `https://github.com/ChatSecure/ChatSecure-iOS/issues/31` (Visited on January 10, 2019).

is referring to the topic of power consumption. Thus, it is likely to provide useful insights on energy improvement practices.

To filter out unrelated entries, we resort to a manual analysis of each instance, comprising two iterations:

1. Check the line where a match with the regular expression was found. If the sentence does not mention anything related to energy consumption, the subject is discarded from the dataset.
2. Check the whole thread in which the mention was found. I.e., open the GitHub page where the commit, issue, or pull request is documented and analyze the context in which the energy topic is being discussed. This step removes cases in which contributors are discussing the topic of energy for contexts that are not related to energy efficiency improvement. E.g., cases were found in which developers were talking about the battery of their own laptops, or in which the app actually had a feature in which it displayed the status of the battery[11].

After this step, we ended up with a total of 1563 subjects: 332 commits, 1089 issues, and 142 pull requests.

3.4 Thematic Analysis

We resort to a methodology based on Thematic Analysis (Fereday and Muir-Cochrane, 2006) to derive design patterns from commits, issues and pull requests. Thematic Analysis is a widely-used qualitative data analysis method, that focuses on identifying patterned meaning in a dataset. Its hybrid process of deductive and inductive analysis has been successfully used in previous work to categorize software commits (Moura et al., 2015). We follow a similar approach by adopting a four-stage process:

Familiarization with data: We have carefully read the information provided in commits, issues or pull-requests, including comments and descriptions. Relevant advice and reasoning are collected and studied using online documentation.

Generating initial labels: For each commit, issue, and pull request, we describe the change in a generic short sentence — i.e., without resorting to specific properties of the app. This process was split into several iterations to discuss amongst both authors in order to refine the labels.

Reviewing themes: After having all subjects spread in different labels, we discuss and review them to find themes that should be merged or split. Some themes were discarded for not being present in a sufficient number of subjects. In particular, we filter out themes that did not occur in at least three different apps. In addition, we corroborate and legitimate coded themes, by finding evidence in the literature that supports or discards themes.

Defining and naming themes: In this stage, we make a structured description of each theme to provide a set of straight-forward guidelines that can be reused in the design of different mobile app projects. Each theme is now converted into an **Energy Pattern**. Each pattern includes a *name*, a brief *description*, a

[11] An example can be found here: `https://github.com/hrydgard/ppsspp/issues/7765` (Visited on January 10, 2019).

context or problem in which the pattern can be applied, and a *solution* with instructions on how to apply the pattern. The solutions provided are based on a combination of authors' experience, logical arguments, literature, mobile platform documentation, and the data itself.

In total 332 commits, 1089 issues, and 142 pull requests were analyzed using this approach. As a result, we identify and document 22 energy patterns that appear in 431 of the analyzed subjects.

3.5 Reproducibility-Oriented Summary

Based on previous guidelines for app store analyses (Martin et al., 2017), we describe ours as follows to help repeat the experimental procedure and reproduce the results:

App Stores used to gather collections of apps: We use apps available on *F-Droid*, and on the list *Collaborative List of Open-Source iOS Apps*.
Total number of apps used: The study comprises 1783 apps.
Breakdown of free/paid apps used in the study: Only non-paid apps are listed in our dataset.
Categories used: All categories were included in this study.
API usage: GitHub REST API v3[12].
Whether code was needed from apps: Source code was required to analyze code changes in commits.
Fraction of open source apps: Open source apps are used exclusively.
Static analysis techniques: No static analysis was performed. In some tasks, the code was analyzed manually by the authors.

All scripts and tools developed in this work are publicly available with an open source license: `https://github.com/TQRG/energy-patterns`.

4 Energy Patterns

In this section, we present the energy patterns collected in this study. As mentioned before, each energy pattern is described by the following entries: context, solution, and an example illustrating a practical usage of the pattern. All these patterns are also available online: `https://tqrg.github.io/energy-patterns`. The website also provides links to the occurrences in the apps, disclosing the discussion performed by developers and practical examples of the patterns in this catalog.

We summarize the occurrences of these patterns in Table 2, presenting their frequency in Android and iOS platforms along with an indication of their prevalence in related work and grey literature (as listed in the Appendix A).

[12] Documentation of GitHub REST API v3 available here: `https://developer.github.com/v3/` (Visited on January 10, 2019).

Table 2 Energy patterns' occurrences and related work.

Pattern	Android	iOS	Related Work	Grey Literature
Dark UI Colors	28	2	(Agolli et al., 2017; Linares-Vásquez et al., 2017; Li et al., 2014, 2015)	-
Dynamic Retry Delay	10	2	-	-
Avoid Extraneous Work	32	9	-	[1]
Race-to-idle	27	5	(Liu et al., 2016; Banerjee and Roychoudhury, 2016; Cruz and Abreu, 2017; Pathak et al., 2012b)	-
Open Only When Necessary	4	3	(Banerjee and Roychoudhury, 2016; Reimann et al., 2014)	-
Push over Poll	13	3	-	[2, 3]
Power Save Mode	24	5	-	[4]
Power Awareness	35	6	(Bao et al., 2016)	[4]
Reduce Size	3	0	(Boonkrong and Dinh, 2015)	[5]
WiFi over Cellular	13	2	(Metri et al., 2012)	[6, 7, 8]
Suppress Logs	7	1	(Chowdhury et al., 2018b)	-
Batch Operations	17	1	(Li and Halfond, 2014; Corral et al., 2015; Cai et al., 2015)	[9, 10, 11]
Cache	14	4	(Gottschalk et al., 2014)	[10]
Decrease Rate	27	10	-	-
User Knows Best	33	11	-	-
Inform Users	6	4	-	-
Enough Resolution	10	7	-	-
Sensor Fusion	12	3	(Shafer and Chang, 2010)	[12]
Kill Abnormal Tasks	11	1	-	-
No Screen Interaction	8	2	-	-
Avoid Extraneous Graphics and Animations	11	8	(Kim et al., 2016)	[1]
Manual Sync, On Demand	4	5	-	-

4.1 Dark UI Colors

Provide a dark UI color theme to save battery on devices with AMOLED[13] screens (Agolli et al., 2017; Linares-Vásquez et al., 2017; Li et al., 2014, 2015).

Context: Screen is one of the major sources of power consumption in mobile devices. Apps that require heavy usage of screen (e.g., reading apps) can have a substantial negative impact on battery life.

Solution: Provide a UI with dark background colors, as illustrated in Figure 4. This is particularly beneficial for mobile devices with AMOLED screens, which are more energy efficient when displaying dark colors. In some cases, it might be reasonable to allow users to choose between a light and a dark theme. The

[13] AMOLED is a display technology used in mobile devices and stands for Active Matrix Organic Light Emitting Diodes.

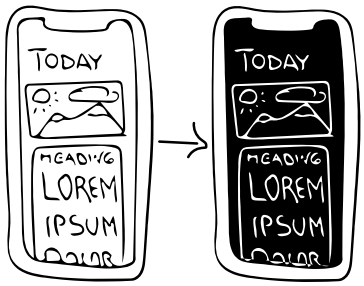

Fig. 4 UI themes with dark colors are more energy efficient.

dark theme can also be activated using a special trigger (e.g., when battery is running low).

Example: In a reading app, provide a theme with a dark background using light colors to display text. When compared to themes using light backgrounds, a dark background will have a higher number of dark pixels.

4.2 Dynamic Retry Delay

Whenever an attempt to access a resource fails, increase the time interval before retrying to access the same resource.

Context: Mobile apps that need to collect or send data from/to other resources (e.g., update information from a server). Commonly, when the resource is unavailable, the app will unnecessarily try to connect the resource for a number of times, leading to unnecessary power consumption.

Solution: Increase retry interval after each failed connection. It can be either a linear or exponential growth. Update interval can be reset upon a successful connection or a given change in the context (e.g., network status).

Example: Consider a mobile app that provides a news feed and the app is not able to reach the server to collect updates. Instead of continuously polling the server until the server is available, use the Fibonacci series[14] to increase the time between attempts.

4.3 Avoid Extraneous Work

Avoid performing tasks that are either not visible, do not have a direct impact on the user experience to the user or quickly become obsolete. This has been documented in the iOS online documentation[15].

[14] Fibonacci series is a sequence of numbers in which each number is the sum of the two preceding numbers (e.g., 1, 1, 2, 3, 5, 8, etc.).

[15] *Energy Efficiency Guide for iOS Apps – Avoid Extraneous Graphics and Animations* available here: `https://developer.apple.com/library/archive/documentation/Performance/Conceptual/EnergyGuide-iOS/AvoidExtraneousGraphicsAndAnimations.html` (Visited on January 10, 2019).

Context: Mobile apps have to perform a number of tasks simultaneously. There are cases in which the result of those tasks is not visible (e.g., the UI is presenting other pieces of information), or the result is not necessarily relevant to the user. This is particularly critical when apps go to the background. Since the data quickly becomes obsolete, the phone is using resources unnecessarily.
Solution: Select a concise set of data that should be presented to the user and enable/disable update and processing tasks depending on their effect on the data that is visible or valuable to the user.
Example: Consider a time series plot that displays real-time data. The plot needs to be constantly updated with the incoming stream of data — however, if the user scrolls up/down in the UI view making the plot hidden, the app should cease drawing operations related with the plot.

4.4 Race-to-idle

Release resources or services as soon as possible (such as wake locks, screen) (Liu et al., 2016; Banerjee and Roychoudhury, 2016; Cruz and Abreu, 2017; Pathak et al., 2012b).

Context: Mobile apps use a number of resources that can be manually closed after use. While active, these resources are ready to respond to requests from the app and require extra power consumption.
Solution: Make sure resources are inactive when they are not necessary by manually closing them.
Example: Implement handlers for events that are fired when the app goes to background, and release wake locks accordingly.

4.5 Open Only When Necessary

Open/start resources/services only when they are strictly necessary.

Context: Some resources require to be opened before use. It might be tempting to open the necessary resources at the beginning of some task (e.g., upon the creation of an activity). However, resources will be actively waiting for requests, and consequently consuming energy.
Solution: Open resources only when necessary. This also avoids activating resources that will never be used (Banerjee and Roychoudhury, 2016).
Example: In a mobile app for video calls, only start capturing video at the moment that it will be displayed to the user[16] .

4.6 Push over Poll

Use push notifications to receive updates from resources, instead of actively querying resources (i.e., polling).

[16] This is a real example that can be found here: `https://github.com/signalapp/Signal-Android/commit/cb9f225f5962d399f48b65d5f855e11f146cbbcb` (Visited on January 10, 2019).

Context: Mobile apps need to get updates from resources (e.g., from a server). One way of checking for updates is by periodically query those resources. However, this will lead to several requests that will return no update, leading to unnecessary energy consumption.
Solution: Use push notifications to get updates from resources. Note – this is a big challenge amongst FOSS apps since there is no good open source alternative for Firebase Cloud Messaging (former Google Cloud Messaging).
Example: In a messaging app, instead of actively check for new messages, the app can subscribe push notifications.

4.7 Power Save Mode

Provide an energy efficient mode in which user experience can drop for the sake of better energy usage.

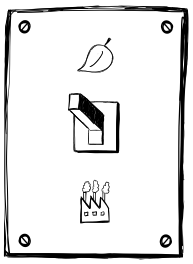

Fig. 5 Power Save Mode allows to run the app in two different modes: a fully-featured mode and an energy efficient mode.

Context: Whenever the device battery is running low, users want to avoid losing connectivity before they reach a power charging station. If the device shuts down, users might miss important calls or will not be able to do an important task. Still, apps might be running unimportant tasks that will reduce battery life in this critical context.
Solution: The app provides a power save mode in which it uses fewer resources while providing the minimum functionality that is indispensable to the user. It can be activated manually or upon some power events (e.g., when battery reaches a given level). User experience can drop for the sake of energy efficiency. Note, this is enforced in iOS for some use cases if the apps use the BackgroundSync APIs.
Example: Deactivate features, reduce update intervals, or deactivate animated effects in the UI.

4.8 Power Awareness

Have a different behavior when the device is connected/disconnected to a power station or has different battery levels.

Context: There are some features that are not strictly necessary to users although they improve user experience (e.g., UI animations). Moreover, there are operations that do not have high priority and do not need to execute immediately (e.g., backup data in the cloud).
Solution: Enable or disable tasks or features according to power status. Even when the device is connected to power, the battery might still be running low, it might be advisable to wait until a pre-defined battery level is reached (or the power save mode is deactivated).
Example: Delay intensive operations such as cloud syncing or image processing until the device is connected to a charger.

4.9 Reduce Size

When transmitting data, reduce its size as much as possible.

Context: Data transmission is a common operation in mobile apps. However, such operations are energy greedy and the time of transmission should be reduced as much as possible.
Solution: Exchange only what is strictly necessary, avoiding sending unnecessary data. Use data compression when possible.
Example: When performing HTTP requests, use gzip content encoding to compress data.

4.10 WiFi over Cellular

Delay or disable heavy data connections until the device is connected to a WiFi network.

Context: Data needs to be synchronized with a server but it is not urgent and can be postponed.
Solution: Data connections using cellular networks are usually more battery intensive than connections using WiFi (Metri et al., 2012). Low priority operations that require a data connection to exchange considerable amounts of data should be delayed until a WiFi connection is available.
Example: Consider a mobile app to organize photos that allows users to backup their photos in a cloud server. Use an API to check the availability of a WiFi connection and postpone cloud synchronizing in case it cannot be reached.

4.11 Suppress Logs

Avoid using intensive logging. Previous work has found that logging activity at rates above one message per second significantly reduces energy efficiency (Chowdhury et al., 2018b).

Context: Developers resort to logging in their mobile apps to ensure their correct behavior and simplify bug reporting. However, logging operations create overhead on energy consumption without creating value to the end user.

Solution: Avoid using intensive logging, keeping rates below one message per second.

Example: Disable logging when processing real-time data. If necessary enable only during debugging executions.

4.12 Batch Operations

Batch multiple operations, instead of putting the device into an active state many times.

Fig. 6 Illustration of the energy pattern Batch Operation. Energy usage can be reduced by combining the execution of different tasks.

Context: Executing operations separately leads to extraneous tail energy consumptions (Li and Halfond, 2014; Corral et al., 2015; Cai et al., 2015). As illustrated in Figure 6, executing a task often induces tail energy consumptions related with starting and stopping resources (e.g., starting a cellular connection).

Solution: Bundle multiple operations in a single one. By combining multiple tasks, tail energy consumptions can be optimized. Although background tasks can be expensive, very often they have flexible time constraints. I.e., a given task needs to be eventually executed, but it does not need to be executed in a specific time.

Example: Use Job Scheduling APIs (e.g., 'android.app.job.JobScheduler', 'Firebase JobDispatcher') that manage multiple background tasks occurring in a device. These APIs will guarantee that the device will exit sleep mode only when there is a reasonable amount of work to do or when a given task is urgent. It combines several multiple tasks to prevent the device from constantly exiting sleep mode (or doze mode). Other examples: execute low priority tasks only if another task is using the same required resources; try to collect location data when other apps are collecting it as well.

4.13 Cache

Avoid performing unnecessary operations by using cache mechanisms.

Context: Typically mobile apps present data to users that is collect from a remote server. However, it may happen that the same data is being collected from the server multiple times.

Solution: Implement caching mechanisms to temporarily store data from a server (Gottschalk et al., 2014). In addition, verify whether there is an update before downloading all data.

Example: Considering a social network app that shows other users' profiles. Instead of downloading basic information and profile pictures every time a given profile is opened, the app can use data that was locally stored from earlier visits.

4.14 Decrease Rate

Increase time between syncs/sensor reads as much as possible.

Context: Mobile apps have to periodically perform operations. If the time between two executions is small, the app will be executing operations more often. In some cases, even if operations are executed more often, it will not affect users' perception.

Solution: Increase the delay between operations to find the minimal interval that does not compromise user experience. This delay can be manually tuned by developers or defined by users. More sophisticated solutions can also use context (e.g., time of the day, history data, etc.) to infer the optimal update rate.

Example: Consider a news app that collects news from different sources, each one having its own thread. Some news sources might have new content only once a week, while others might be updated every hour. Instead of updating all threads at the same rate, use data from previous updates to infer the optimal update rate of these threads. Connect to the news source only if new updates are expected.

4.15 User Knows Best

Allow users to enable/disable certain features in order to save energy.

Context: Energy efficiency solutions often provide a tradeoff between features and power consumption. However, this tradeoff is different for different users — some users might be okay with fewer features but better energy efficiency, and vice versa.

Solution: Allow users to customize their preferences regarding energy critical features. Since this might be more intuitive for power users, mobile apps should provide optimal preferences by default for regular users.

Example: Consider a mail client for POP3 accounts as an example. In some cases, users are not expecting any urgent message and are okay with checking for new mail in no less than 10 minutes for the sake of energy efficiency. On the other hand, there are cases in which users are waiting for urgent messages and would like to check for messages every two minutes. Since there is no automatic mechanism to infer the optimal update interval, the best option is to allow users to define it.

4.16 Inform Users

Let the user know if the app is doing any battery intensive operation.

Context: There are specific use cases in mobile apps that can be energy greedy. On the other hand, some features might be dropping user experience in order to improve energy efficiency. If users do not know what to expect from the mobile app, they might think it is not behaving correctly.
Solution: Let users know about battery intensive operations or energy management features. Properly flag this information in the user interface (e.g., alerts).
Example: Alert users when a power saving mode is active, or alert when a battery intensive operation is about to be executed.

4.17 Enough Resolution

Collect or provide high accuracy data only when strictly necessary.

Context: When collecting or displaying data, it is tempting to use high resolutions. The problem of using data with high resolution is that its collection and manipulation require more resources (e.g., memory, processing capacity, etc.). As a consequence, energy consumption increases unnecessarily.
Solution: For every use case, find the optimal resolution value that is required to provide the intended user experience.
Example: Consider a running app that is able to record running sessions. While the user is running, the app presents the current overall distance in real-time. While calculating the most accurate value of the total distance would provide the correct information, it would require precise real-time processing of GPS or accelerometer sensors, which can be energy greedy. Instead, a lightweight method could be used to estimate this information with lower but reasonable accuracy. At the end of the session, the accurate results would still be processed, but without real-time constraints.

4.18 Sensor Fusion

Use data from low power sensors to infer whether new data needs to be collected from high power sensors

Context: Mobile apps provide features that require reading data or executing operations in different sensors or components. Such operations can be energy greedy, causing high power consumption. Thus, they should be called as fewer times as possible.
Solution: Use complementary data from low power sensors to assure whether a given energy-greedy operation needs to be executed.
Example: Use the accelerometer to infer whether the user has changed location. In the case that the user is in the same location, data from GPS does not need to be updated.

4.19 Kill Abnormal Tasks

Provide means of interrupting energy greedy operations (e.g., using timeouts, or users input).

Context: Mobile apps might feature operations that can be unexpectedly energy greedy (e.g., taking a long time to execute).
Solution: Provide a reasonable timeout for energy greedy tasks or wake locks. Alternatively, provide an intuitive way of interrupting those tasks.
Example: In a mobile app that features an alarm clock, set a reasonable timeout for the duration of the alarm. In case the user is not able to turn it off it will not drain the battery.

4.20 No Screen Interaction

Whenever possible allow interaction without using the display.

Context: There are apps that require a continuous usage of the screen. However, there are use cases in which the screen can be replaced by less power intensive alternatives.
Solution: Allow users to interact with the app using alternative interfaces (e.g., audio).
Example: In a navigation app, there are use cases in which users might be only using audio instructions and do not need the screen to be on all the time. This pattern is commonly adopted by audio players that use the earphone buttons to play/pause or skip songs.

4.21 Avoid Extraneous Graphics and Animations

Graphics and animations are really important to improve the user experience. However, they can also be battery intensive — use them with moderation (Kim et al., 2016). This is also a recommendation in the official documentation for iOS developers[17]

Context: Mobile apps often feature impressive graphics and animations. However, they need to be properly tuned in order to prevent battery drain of users' devices. This is particularly critical in e-paper devices.
Solution: Study the importance of graphics and animations to the user experience. The improvement in user experience may not be sufficient to cover the overhead on energy consumption. Avoid using graphics animations or high-quality graphics. Resort to low frame rates for animations when possible.
Example: For example, a high frame rate may make sense during game play, but a lower frame rate may be sufficient for a menu screen. Use a high frame rate only when the user experience calls for it.

[17] *Energy Efficiency Guide for iOS Apps – Avoid Extraneous Graphics and Animations* available here: `https://developer.apple.com/library/archive/documentation/Performance/Conceptual/EnergyGuide-iOS/AvoidExtraneousGraphicsAndAnimations.html` (Visited on January 10, 2019).

4.22 Manual Sync, On Demand

Perform tasks exclusively when requested by the user.

Context: Some tasks can be energy intensive, but not strictly necessary for some use cases of the app.
Solution: Provide a mechanism in the UI (e.g., button) that allows users to trigger energy intensive tasks.
Example: In a beacon monitoring app, there are occasions in which the user does not need to keep track of her/his beacons. Allow the user to start and stop monitoring manually.

5 Data Summary and Discussion

In this section, we report and discuss findings regarding the presence of energy patterns in the studied mobile applications. In particular, we study differences between Android and iOS platforms and how often energy patterns co-occur within the same app.

5.1 Energy Patterns: Android vs. iOS

Next, we assess whether energy efficiency is addressed in a different way in Android and iOS environments.

5.1.1 Energy Efficiency Changes Per mobile app

From the 1783 apps used in this study, we have found 332 (19%) with at least one commit, issue, or pull request related to energy efficiency. In Android we have found 256 out of the 1021 apps (25%), while in iOS we have found 76 out of 756 apps (10%). Congruent results are observed in the extraction of energy patterns: we were able to extract energy patterns in 133 Android apps (13%) and 28 iOS apps (4%). In general, Android developers put more effort into improving the energy efficiency of their mobile apps than iOS developers.

However, research is necessary to explain why this is the case. First, our data comprises only development activities that consciously address energy efficiency. We understand that there are many factors that can affect these results: power management mechanisms implemented by the system, documentation of the frameworks, differences in targeted users, differences in targeted devices, etc. For instance, the iOS platform provides APIs with more strict power management rules, and developers are enforced to use energy best practices beforehand even though they were not addressing energy efficiency per se.

Developers even express their concern on not having their apps accepted in the iOS App store for certain practices such as having tasks periodically running in background[18]. Although in this case the main goal is having the app accepted in the app store, energy efficiency is addressed indirectly. On the contrary, Google

[18] An example of developers dealing with the strict policy of the iOS app store: `https://github.com/owncloud/ios/issues/13` (Visited on January 10, 2019).

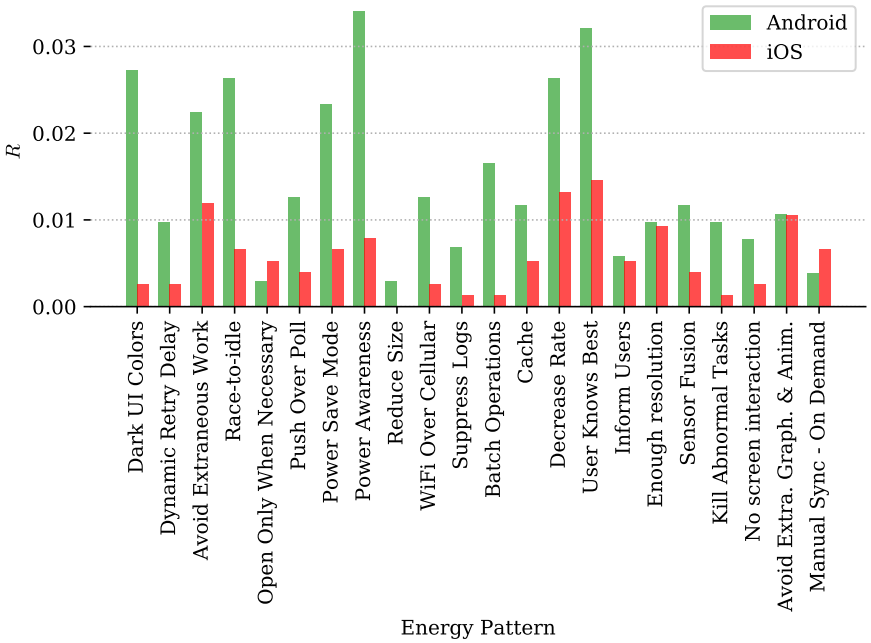

Fig. 7 Comparison of the usage of energy patterns between Android and iOS mobile apps.

Play store, the official store for Android apps, is known to have less strict policies (Cuadrado and Dueñas, 2012).

5.1.2 Prevalence of Patterns in Android and iOS

We compare the prevalence of each energy pattern in the two platforms, Android and iOS. Since our app dataset comprises a different number of apps for the two platforms, we use the ratio of the number of occurrences N_p of a given pattern p divided by the total number of apps (M_X) studied for a given platform X:

$$R(p, X) = \frac{N_p}{M_X}, \; X = \{\text{iOS}, \text{Android}\} \tag{1}$$

Figure 7 shows the values of R for every pattern and platform. In general, energy patterns are more prevalent in Android rather than iOS apps. Only two energy patterns were more frequent in iOS: *Open Only When Necessary* and *Manual Sync, On Demand*. In addition, *Inform Users*, *Enough Resolution*, and *Avoid Extraneous Graphics And Animations* had a similar ratio of occurrences in both platforms. The remaining 18 patterns were notably more frequent in Android apps.

The two platforms differ in the patterns that have the highest number of occurrences. In Android, the most frequent patterns were *Power Awareness*, *User Knows Best*, *Dark UI Colors*, and *Race-to-idle*. In iOS, the most frequent patterns were *Avoid Extraneous Work*, *Decrease Rate*, *User Knows Best*, and *Avoid Extraneous Graphics And Animations*. These patterns are being mentioned in the official

iOS documentation for developers, reinforcing the importance of documentation to help developers build energy efficient software (Manotas et al., 2016; Sousa et al., 2018).

The pattern *Dark UI Color* is considerably more popular amongst Android apps than in iOS apps. This is an expected observation: only recently, iOS devices started to feature AMOLED displays that reduce energy consumption when using dark colors[19].

5.2 Co-occurrence of Patterns

We analyze which patterns tend to appear together within the same mobile app. We resort to the chord diagram in Figure 8. Each pattern is connected to another pattern if found in the same app. The thicker the edge the more frequent the pair of patterns co-occur.

To improve the interpretability of the chord diagram we have filtered out cases in which two patterns co-occur less than 5 times. An interactive version of the diagram containing all data is available in the online catalog: `https://tqrg.github.io/energy-patterns`.

The chord diagram of Figure 8 reveals the following relationships between patterns:

- *Dark UI Colors* and *Race-to-idle* are the most prevalent patterns (28 occurences), followed by *Avoid Extraneous Work* (27 occurrences), *User Knows Best* (25 occurences), and *Decrease Rate* (24 occurences).
- *User Knows Best* is a dominant pattern. It co-occurs with *Power Save Mode*, *Power Awareness*, *Decrease Rate*, *Avoid Extraneous Work*, and *Race To Idle*.
- *Avoid Extraneous Work* is also a dominant pattern. It appears in apps that also use *Race-to-Idle*, *Push Over Poll*, *Decrease Rate*, and *User Knows Best*.
- *Power Save Mode*, *Power Awareness*, and *User Knows Best* are typically used together.

5.3 Implications

This catalog wraps up the techniques used by developers of open source mobile apps to address energy efficiency. It helps developers understand how to design energy efficient mobile apps by looking at solutions from other projects. Most techniques are spread out in the literature, making energy efficiency a problem that requires specialized developers. We mitigate this barrier by providing common approaches to solve typical problems in the energy efficiency of mobile apps.

Although we create energy patterns as a platform to share knowledge from experienced developers, it can also serve as the base to future work on automated tools to improve energy efficiency. Patterns such as *Enough resolution* may be challenging, requiring developers to have a deep understanding of the devices they are targeting. Tools or APIs aiding to address this issue would significantly decrease the efforts to adopt this pattern. This is already the case for patterns such as *Batch Operations* that are featured in the platform's API: Android provides the

[19] Until mid 2018, iPhone X was the only iOS smartphone with an AMOLED screen.

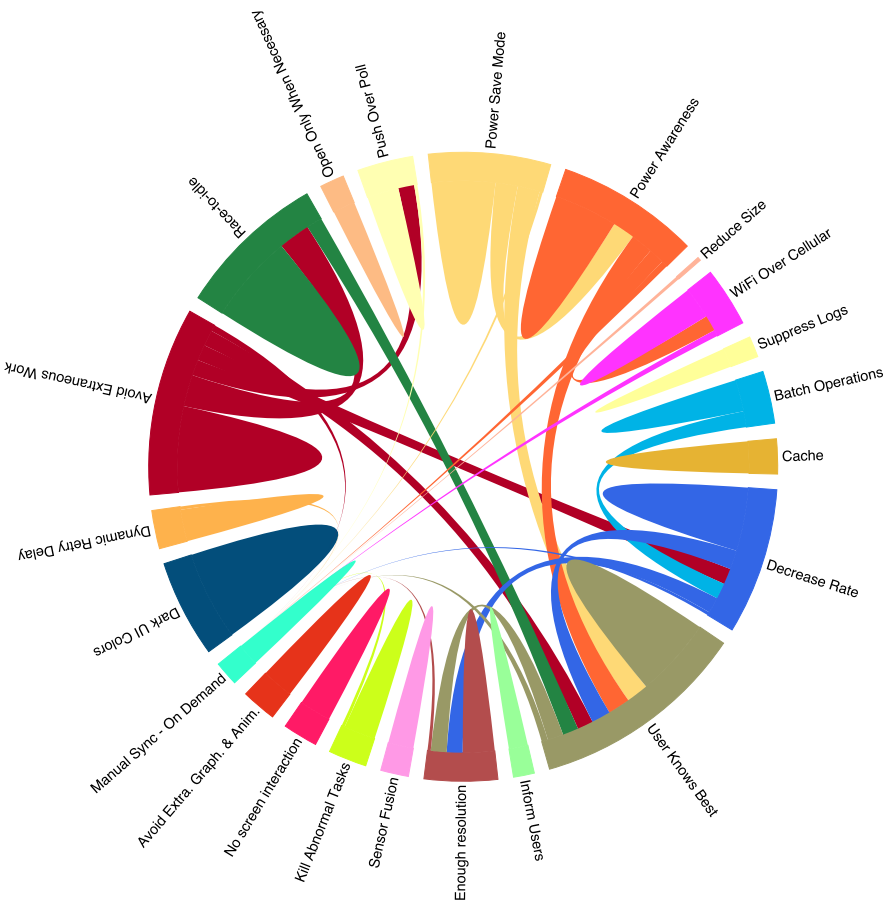

Fig. 8 Co-occurrence of energy patterns in the same mobile application.

`JobScheduler` that schedules the jobs to execute at the most efficient times. We understand that similar approaches should be leveraged to other patterns. For instance, *Power Save Mode* is a common pattern that often requires re-implementing existing features.

We also find some remarks of interest to the research community. While some techniques have been widely studied in previous work (e.g., *Dark UI Colors* and *Race-to-idle*), many remain unnoticed. Patterns such as *Dynamic Retry Delay* or *Kill Abnormal Tasks* can be argued based on logical arguments. However, there is no empirical study that has evaluated the cost and benefit of applying these patterns.

This catalog can also help educators include the topic of energy efficiency in Mobile Software Engineering courses. Students can reach to this catalog to seek more information on energy patterns and find examples of their application in real Android and iOS apps.

6 Threats to validity

In this section, we discuss potential systematic errors in our work and to what extent our results can generalize.

6.1 Internal validity

We select all commits, issues and pull requests that contain the words *energy*, *battery*, or *power*. While we understand that this covers the large majority of mentions of improvements in energy usage, some mentions may have been missed. There are commits that may have been created to improve energy efficiency, but the message may not mention it explicitly. Moreover, we only use mentions written in English, which is the most common language amongst developer communities.

In our methodology, we adopt a manual filtering of collected commits, issues, and pull requests. While we understand that this was the safest approach to avoid missing important mentions, human error can be expected from this process. False positives may have been left in the dataset — we argue that during the thematic analysis these subjects are easily discarded. False negatives (i.e., subjects of interest that were filtered out) may also occur due to misinterpretation of subjects by authors during the selection. However, we expect this to comprise a negligible number of cases.

Only commits merged with the *default* branch of projects were considered. Energy improvements that were implemented in different branches of the project were not considered in our study. Although some interesting patterns may lie in some of these branches we cannot guarantee their quality: changes that have not been merged are still lacking the validation from the development team.

In addition, we only collect energy patterns that were used by mobile app development practitioners and are available on open source projects. There might be other patterns that improve energy efficiency but that are not being used by the community. Those patterns are out of the scope of this study and are left for future work. Moreover, patterns that also improve other properties besides energy usage might occur in the projects in this study.

We solely list cases that occur with the main goal of improving energy efficiency. In this study, we only address energy improvements that are clearly described in the description of the respective commit, issue, or pull request. However, the classification of developers' intent in code changes is a non-trivial open problem (Pascarella et al., 2018). Thus, less obvious energy improvements were not studied. In addition, this catalog is based on developers common approaches to address energy efficiency. We do not measure the magnitude of potential gains of these patterns work. An empirical study would help to assess these gains, as done in previous work (Carette et al., 2017; Palomba et al., 2019).

Finally, we have discarded less significant patterns by filtering out patterns with less than three occurrences. We understand that some of them may hide interesting energy efficiency strategies. However, no literature was found to support these patterns, and assessing their impact is out of the scope of this paper.

6.2 External validity

The data analyzed in this work is typically private for commercial apps. Thus, we focus exclusively on open source mobile apps. Development of commercial apps is usually driven by different goals and budgets. Hence, energy usage may be targeted in a different way in these apps. However, energy patterns collected in this study can be applied in any mobile app project regardless of its license.

We only analyze iOS and Android mobile apps. While this comprises most of the mobile apps in the market, other mobile platforms also have their share (Windows Phone, BlackBerry OS, Firefox OS, Ubuntu Touch, etc.). We discarded coding practices for energy efficiency that we understood being specific to the platform in which they were implemented (e.g., using certain API methods). Unless the mobile operative system or the hardware device deals with the contexts identified in our work under the hood, we expect these patterns to be useful in other platforms.

7 Conclusion

We analyzed commits, issues and pull requests from 1021 Android apps and 756 iOS apps to identify design practices to improve the energy efficiency of mobile apps. As an outcome, this work delivers a catalog of 22 design patterns based on 1563 energy-related changes of mobile apps. To the best of our knowledge, this is the first time energy practices for mobile apps are studied at a large scale in both iOS and Android. This catalog will help mobile app designers and developers make educated decisions when building (energy efficient) apps, regardless of the target platform.

We leverage a dataset with changes that address energy efficiency in mobile apps for Android and iOS. In addition, we list the occurrence of energy patterns in this dataset. The dataset is publicly available and we welcome contributions from the community as pull request. An interesting remark from this dataset is that it comprises only 19% of the total number of mobile apps that we analyze. This means that only a minority of mobile apps have had changes to improve energy efficiency. Moreover, results show that efforts to improve energy efficiency are more common within Android apps (25%) than iOS apps (10%). Further research should investigate the root causes of this observation.

As future work, it would be interesting to implement these patterns in an automated refactoring tool. Similar tools have been delivered in previous work on quality smells (Palomba et al., 2017; Cruz and Abreu, 2018). Moreover, we would like to study these patterns in the context of Cyber-Physical Systems and Internet of Things applications, in which power consumption has been identified as a challenge to be addressed (White et al., 2010; Palattella et al., 2016). Finally, we plan to continuously extend the catalog with a broader set of energy patterns. We welcome contributions from the mobile app development community as pull requests in our online repository.

Acknowledgment

This work is financed by the ERDF — European Regional Development Fund through the Operational Program for Competitiveness and Internationalization - COMPETE 2020 Program and by National Funds through the Portuguese funding agency, FCT - Fundação para a Ciência e a Tecnologia with reference UID/CEC/50021/2019 and within projects GreenLab (POCI-01-0145-FEDER-016718), and FaultLockerRef (PTDC/CCI-COM/29300/2017). Luis Cruz is sponsored by an FCT scholarship grant number PD/BD/52237/2013. Furthermore, we would like to thank Sofia Reis for the illustrations of energy patterns.

Appendix A Grey Literature

[1] Apple Developer Documentation Archive. Energy Efficiency Guide for iOS Apps – Avoid Extraneous Graphics and Animations. URL: https://developer.apple.com/library/archive/documentation/Performance/Conceptual/EnergyGuide-iOS/AvoidExtraneousGraphicsAndAnimations.html

[2] Daniel Gultsch. The State of Mobile XPP in 2016. URL: https://gultsch.de/xmpp_2016.html

[3] Alternative Push Notification Transport. URL: https://github.com/matrix-org/GSoC/blob/master/IDEAS.md#alternative-push-notification-transport

[4] Apple Developer Documention Archive. Energy Efficiency Guide for iOS Apps – React to Low Power Mode on iPhones. URL: https://developer.apple.com/library/archive/documentation/Performance/Conceptual/EnergyGuide-iOS/LowPowerMode.html

[5] Stackoverflow. Is gzip compression useful for mobile devices?. URL: https://stackoverflow.com/questions/3065920/is-gzip-compression-useful-for-mobile-devices

[6] Mitch Bartlett. Does Wi-Fi Consume More Battery Power Than 3G or 4G/LTE?. URL: https://www.technipages.com/does-wi-fi-consume-more-battery-power-than-3g-or-4glte

[7] Android SDK Documentation. Modifying your Download Patterns Based on the Connectivity Type. URL: https://developer.android.com/training/efficient-downloads/connectivity_patterns

[8] Apple Developer Documentation Archive. Energy Efficiency Guide for iOS Apps – Energy and Networking. URL: https://developer.apple.com/library/archive/documentation/Performance/Conceptual/EnergyGuide-iOS/EnergyandNetworking.html

[9] Android SDK Documentation. Optimizing for Doze and App Standby. URL: https://developer.android.com/training/monitoring-device-state/doze-standby

[10] Android SDK Documentation. Android Developer Guides — Optimizing for Battery Life. URL: https://developer.android.com/topic/performance/power/

[11] Android Developers Youtube Channel. DevBytes: Efficient Data Transfers - Batching, Bundling, and SyncAdapters. URL: https://www.youtube.com/watch?v=5onKZcJyJwI

[12] Apple's Core Location Documentation. CLLocationManager. URL: `https://developer.apple.com/documentation/corelocation/cllocationmanager`

References

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

M. P. Robillard and N. Medvidovic, "Disseminating architectural knowledge on open-source projects: A case study of the book" architecture of open-source applications"," in *Software Engineering (ICSE), 2016 IEEE/ACM 38th International Conference on*. IEEE, 2016, pp. 476–487.

I. Manotas, C. Bird, R. Zhang, D. Shepherd, C. Jaspan, C. Sadowski, L. Pollock, and J. Clause, "An empirical study of practitioners' perspectives on green software engineering," in *Software Engineering (ICSE), 2016 IEEE/ACM 38th International Conference on*. IEEE, 2016, pp. 237–248.

K. An, N. Meng, and E. Tilevich, "Automatic inference of java-to-swift translation rules for porting mobile applications," in *Proceedings of the 5th International Conference on Mobile Software Engineering and Systems*, ser. MOBILESoft '18. New York, NY, USA: ACM, 2018, pp. 180–190. [Online]. Available: http://doi.acm.org/10.1145/3197231.3197240

J. Vlissides, R. Helm, R. Johnson, and E. Gamma, "Design patterns: Elements of reusable object-oriented software," *Addison-Wesley*, 1995.

K. Yskout, R. Scandariato, and W. Joosen, "Do security patterns really help designers?" in *Software Engineering (ICSE), 2015 IEEE/ACM 37th IEEE International Conference on*, vol. 1. IEEE, 2015, pp. 292–302.

R. Pham, Y. Stoliar, and K. Schneider, "Automatically recommending test code examples to inexperienced developers," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering (FSE)*. ACM, 2015, pp. 890–893.

R. Pham, L. Singer, O. Liskin, F. Figueira Filho, and K. Schneider, "Creating a shared understanding of testing culture on a social coding site," in *Proceedings of the 2013 International Conference on Software Engineering (ICSE)*. IEEE Press, 2013, pp. 112–121.

S. Negara, M. Codoban, D. Dig, and R. E. Johnson, "Mining fine-grained code changes to detect unknown change patterns," in *Proceedings of the 36th International Conference on Software Engineering (ICSE)*. ACM, 2014, pp. 803–813.

F. Palomba, D. A. Tamburri, A. Serebrenik, A. Zaidman, F. A. Fontana, and R. Oliveto, "How do community smells influence code smells?" in *Proceedings of the 40th International Conference on Software Engineering Companion*. ACM, 2018, pp. 240–241.

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.

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 (FSE)*. ACM, 2014, pp. 588–598.

P. Vekris, R. Jhala, S. Lerner, and Y. Agarwal, "Towards verifying android apps for the absence of no-sleep energy bugs." in *HotPower*, 2012.

C. Wilke, S. Götz, and S. Richly, "Jouleunit: a generic framework for software energy profiling and testing," in *Proceedings of the 2013 workshop on Green in/by software engineering*. ACM, 2013, pp. 9–14.

Y. Liu, C. Xu, and S.-C. Cheung, "Where has my battery gone? finding sensor related energy black holes in smartphone applications," in *Pervasive Computing and Communications (PerCom), 2013 IEEE International Conference on*. IEEE, 2013, pp. 2–10.

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

R. J. Behrouz, A. Sadeghi, J. Garcia, S. Malek, and P. Ammann, "Ecodroid: An approach for energy-based ranking of android apps," in *Green and Sustainable Software (GREENS), 2015 IEEE/ACM 4th International Workshop on*. IEEE, 2015, pp. 8–14.

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.

L. Zhang, B. Tiwana, R. P. Dick, Z. Qian, Z. M. Mao, Z. Wang, and L. Yang, "Accurate online power estimation and automatic battery behavior based power model generation for smartphones," in *Hardware/Software Codesign and System Synthesis (CODES+ ISSS), 2010 IEEE/ACM/IFIP International Conference on*. IEEE, 2010, pp. 105–114.

S. Chowdhury, S. Borle, S. Romansky, and A. Hindle, "Greenscaler: training software energy models with automatic test generation," *Empirical Software Engineering*, pp. 1–44, 2018.

D. Di Nucci, F. Palomba, A. Prota, A. Panichella, A. Zaidman, and A. De Lucia, "Software-based energy profiling of android apps: Simple, efficient and reliable?" in *Software Analysis, Evolution and Reengineering (SANER), 2017 IEEE 24th International Conference on*. IEEE, 2017, pp. 103–114.

A. Hindle, A. Wilson, K. Rasmussen, E. J. Barlow, J. C. Campbell, and S. Romansky, "Greenminer: A hardware based mining software repositories software energy consumption framework," in *Proceedings of the 11th Working Conference on Mining Software Repositories*. ACM, 2014, pp. 12–21.

S. Romansky, N. C. Borle, S. Chowdhury, A. Hindle, and R. Greiner, "Deep green: modelling time-series of software energy consumption," in *Software Maintenance and Evolution (ICSME), 2017 IEEE International Conference on*. IEEE, 2017, pp. 273–283.

C. Sahin, L. Pollock, and J. Clause, "From benchmarks to real apps: Exploring the energy impacts of performance-directed changes," *Journal of Systems and Software*, vol. 117, pp. 307–316, 2016.

L. Cruz and R. Abreu, "Performance-based guidelines for energy efficient mobile applications," in *IEEE/ACM International Conference on Mobile Software Engineering and Systems, MobileSoft 2017*, 2017, pp. 46–57.

——, "Using automatic refactoring to improve energy efficiency of android apps," in *CIbSE XXI Ibero-American Conference on Software Engineering*, 2018.

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.

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*. ACM, 2012, pp. 29–42.

I. Moura, G. Pinto, F. Ebert, and F. Castor, "Mining energy-aware commits," in *Proceedings of the 12th Working Conference on Mining Software Repositories*. IEEE Press, 2015, pp. 56–67.

L. Bao, D. Lo, X. Xia, X. Wang, and C. Tian, "How android app developers manage power consumption?: An empirical study by mining power management commits," in *Proceedings of the 13th International Conference on Mining Software Repositories*. ACM, 2016, pp. 37–48.

G. Pinto, F. Castor, and Y. D. Liu, "Mining questions about software energy consumption," in *Proceedings of the 11th Working Conference on Mining Software Repositories*. ACM, 2014, pp. 22–31.

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

J. Reimann, M. Brylski, and U. Aßmann, "A tool-supported quality smell catalogue for android developers," in *Proc. of the conference Modellierung 2014 in the Workshop Modellbasierte und modellgetriebene Softwaremodernisierung–MMSM*, vol. 2014, 2014.

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

A. McIntosh, S. Hassan, and A. Hindle, "What can android mobile app developers do about the energy consumption of machine learning?" *Empirical Software Engineering*, pp. 1–40, 2018.

W. Oliveira, R. Oliveira, and F. Castor, "A study on the energy consumption of android app development approaches," in *Mining Software Repositories (MSR), 2017 IEEE/ACM 14th International Conference on*. IEEE, 2017, pp. 42–52.

N. Cassee, G. Pinto, F. Castor, and A. Serebrenik, "How swift developers handle errors," in *15th International Conference on Mining Software Repositories (MSR 2018)*, 2018.

J. Fereday and E. Muir-Cochrane, "Demonstrating rigor using thematic analysis: A hybrid approach of inductive and deductive coding and theme development," *International journal of qualitative methods*, vol. 5, no. 1, pp. 80–92, 2006.

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, 2017.

T. Agolli, L. Pollock, and J. Clause, "Investigating decreasing energy usage in mobile apps via indistinguishable color changes," in *Mobile Software Engineering and Systems (MOBILESoft), 2017 IEEE/ACM 4th International Conference on*. IEEE, 2017, pp. 30–34.

M. Linares-Vásquez, C. Bernal-Cárdenas, G. Bavota, R. Oliveto, M. Di Penta, and D. Poshyvanyk, "Gemma: multi-objective optimization of energy consumption

of guis in android apps," in *Proceedings of the 39th International Conference on Software Engineering Companion*. IEEE Press, 2017, pp. 11–14.

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 (ICSE)*. ACM, 2014, pp. 527–538.

——, "Nyx: A display energy optimizer for mobile web apps," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering (FSE)*. ACM, 2015, pp. 958–961.

Y. Liu, C. Xu, S.-C. Cheung, and V. Terragni, "Understanding and detecting wake lock misuses for android applications," in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE)*. ACM, 2016, pp. 396–409.

A. Banerjee and A. Roychoudhury, "Automated re-factoring of android apps to enhance energy-efficiency," in *Proceedings of the International Workshop on Mobile Software Engineering and Systems*. ACM, 2016, pp. 139–150.

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*. ACM, 2012, pp. 267–280.

S. Boonkrong and P. C. Dinh, "Reducing battery consumption of data polling and pushing techniques on android using gzip," in *Information Technology and Electrical Engineering (ICITEE), 2015 7th International Conference on*. IEEE, 2015, pp. 565–570.

G. Metri, A. Agrawal, R. Peri, and W. Shi, "What is eating up battery life on my smartphone: A case study," in *Energy Aware Computing, 2012 International Conference on*. IEEE, 2012, pp. 1–6.

S. Chowdhury, S. Di Nardo, A. Hindle, and Z. M. J. Jiang, "An exploratory study on assessing the energy impact of logging on android applications," *Empirical Software Engineering*, vol. 23, no. 3, pp. 1422–1456, 2018.

L. Corral, A. B. Georgiev, A. Janes, and S. Kofler, "Energy-aware performance evaluation of android custom kernels," in *Green and Sustainable Software (GREENS), 2015 IEEE/ACM 4th International Workshop on*. IEEE, 2015, pp. 1–7.

H. Cai, Y. Zhang, Z. Jin, X. Liu, and G. Huang, "Delaydroid: Reducing tail-time energy by refactoring android apps," in *Proceedings of the 7th Asia-Pacific Symposium on Internetware*. ACM, 2015, pp. 1–10.

M. Gottschalk, J. Jelschen, and A. Winter, "Saving energy on mobile devices by refactoring." in *EnviroInfo*, 2014, pp. 437–444.

I. Shafer and M. L. Chang, "Movement detection for power-efficient smartphone wlan localization," in *Proceedings of the 13th ACM international conference on Modeling, analysis, and simulation of wireless and mobile systems*. ACM, 2010, pp. 81–90.

D. Kim, N. Jung, Y. Chon, and H. Cha, "Content-centric energy management of mobile displays," *IEEE Transactions on Mobile Computing*, vol. 15, no. 8, pp. 1925–1938, 2016.

F. Cuadrado and J. C. Dueñas, "Mobile application stores: success factors, existing approaches, and future developments," *IEEE Communications Magazine*, vol. 50, no. 11, pp. 160–167, 2012.

L. Sousa, A. Oliveira, W. Oizumi, S. Barbosa, A. Garcia, J. Lee, M. Kalinowski, R. de Mello, B. Fonseca, R. Oliveira *et al.*, "Identifying design problems in the source code: a grounded theory," in *Proceedings of the 40th International Conference on Software Engineering (ICSE)*. ACM, 2018, pp. 921–931.

L. Pascarella, F.-X. Geiger, F. Palomba, D. Di Nucci, I. Malavolta, and A. Bacchelli, "Self-reported activities of android developers," in *5th IEEE/ACM International Conference on Mobile Software Engineering and Systems, New York, NY*, 2018.

A. Carette, M. A. A. Younes, G. Hecht, N. Moha, and R. Rouvoy, "Investigating the energy impact of android smells," in *Software Analysis, Evolution and Reengineering (SANER), 2017 IEEE 24th International Conference on*. IEEE, 2017, pp. 115–126.

F. Palomba, D. Di Nucci, A. Panichella, A. Zaidman, and A. De Lucia, "On the impact of code smells on the energy consumption of mobile applications," *Information and Software Technology*, vol. 105, pp. 43–55, 2019.

——, "Lightweight detection of android-specific code smells: The adoctor project," in *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2017, pp. 487–491.

J. White, S. Clarke, C. Groba, B. Dougherty, C. Thompson, and D. C. Schmidt, "R&d challenges and solutions for mobile cyber-physical applications and supporting internet services," *Journal of internet services and applications*, vol. 1, no. 1, pp. 45–56, 2010.

M. R. Palattella, M. Dohler, A. Grieco, G. Rizzo, J. Torsner, T. Engel, and L. Ladid, "Internet of things in the 5g era: Enablers, architecture, and business models," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 3, pp. 510–527, 2016.