Mining Energy-Aware Commits

Irineu Moura, Gustavo Pinto, Felipe Ebert and Fernando Castor
Federal University of Pernambuco
Informatics Center
{imlm2, ghlp, fe, castor}@cin.ufpe.br

***Abstract*—Over the last years, energy consumption has become a first-class citizen in software development practice. While energy-efficient solutions on lower-level layers of the software stack are well-established, there is convincing evidence that even better results can be achieved by encouraging practitioners to participate in the process. For instance, previous work has shown that using a newer version of a concurrent data structure can yield a 2.19x energy savings when compared to the old associative implementation [75]. Nonetheless, little is known about how much software engineers are employing energy-efficient solutions in their applications and what solutions they employ for improving energy-efficiency. In this paper we present a qualitative study of "energy-aware commits". Using Github as our primary data source, we perform a thorough analysis on an initial sample of 2,189 commits and carefully curate a set of 371 energy-aware commits spread over 317 real-world non-trivial applications. Our study reveals that software developers heavily rely on low-level energy management approaches, such as frequency scaling and multiple levels of idleness. Also, our findings suggest that ill-chosen energy saving techniques can impact the correctness of an application. Yet, we found what we call "energy-aware interfaces", which are means for clients (*e.g.,* developers or end-users) to save energy in their applications just by using a function, abstracting away the low-level implementation details.**

I. Introduction

Thanks to the diversification of modern computing platforms, battery-driven devices such as smartphones, tablets and unwired devices in general are now commonplace in our lives. However, such devices are energy-constrained, as they rely on limited battery power supply. Energy consumption directly affects the perception of users about their quality. For example, in a survey conducted with more than 3,500 respondents from 4 different countries [49], long-lasting battery life has been cited as the most desired feature in a new phone by 71% of the respondents. Likewise, recent research pointed out that battery usage is a key factor for evaluating and adopting mobile applications [86]. As a result, not only researchers [57], [88], [90] but also giants of the software industry [56], [84] have recently started to promote energy-efficient systems.

Traditionally, energy optimization research has focused at the hardware-level (*e.g.*, [52], [59]) and at the system-level (*e.g.*, [54], [78]). Arguably, the strategy of leaving the energy optimization issue to lower-level systems and architecture layers has been successful. The main advantage of this approach is that user applications can be seen as black-boxes, *i.e.*, no prior knowledge of the application source code or its behavior needs to be used to achieve better energy savings.

However, applications also impact energy consumption, although software itself does not consume energy. When one energy-inefficient piece of code is introduced in a system comprising hardware and software components, compilers and runtime systems cannot help much, since they are not aware of the semantics of the program. Unlike performance, where more efficient is always better, sometimes an application will purposefully consume more energy to provide its intended functionality, *e.g.,* by activating an energy-intensive device.

In contrast, energy consumption bottlenecks often stem from inappropriate design choices, as is often the case with performance bugs [61]. Finding and fixing these problems requires human insight. It is not always clear for programmers, however, what they can do from a software engineering perspective to save energy. There seem to be misconceptions and lack of appropriate tools [76].

Recent work [64], [63], [75], [77] has shown that there is ample opportunity to improve energy consumption at the application level. As an example, upgrading the `ConcurrentHashMap` class available in Java 7 to its improved successor, available in Java 8, can yield energy savings of 2.19x [75]. Nevertheless, developing an energy-efficient system is a non-trivial task. Even though researchers have made great strides in empirical studies aiming to understand the impact of different program characteristics in energy consumption (*e.g.,* [64], [77], [80], [89]), these studies do not cover a complete range of language constructs and implementation choices. Therefore, developers still do not fully understand how their source code modifications will impact energy consumption [76].

In spite of this grim scenario, many systems that are energy-efficient are being developed in practice. Starting from this premise, in this paper we focus on a timely but overlooked question:

- What solutions do software developers employ to save energy in practice?

To answer this question, we obtained data from GITHUB, the most popular source code hosting website in the software development world. At the time when this paper was written, it contained over 21.1 million software repositories[1] and more than 3.5 million contributors[2]. In addition, GITHUB is being used in recent software engineering studies [46], [51], [85].

In order to understand what are the energy-efficient solutions that software developers are employing, we started by searching at the commit messages, since programmers usually

[1]https://github.com/features
[2]https://github.com/blog/1470-five-years

express their intention through these messages [50]. Commit messages are also essential to program comprehension and software evolution, since they help developers understand and validate changes. Unfortunately, GITHUB does not provide a powerful infrastructure to search for commit messages, which led us to use a third-party mirror, GITHUBARCHIVE.

Using a set of energy consumption-related keywords defined elsewhere [76], we found more than 2,000 energy consumption-related commits. Through an automatic and double-checked manual analysis of these commits, and the associated source code, we identified a curated set of 371 "energy-aware commits". From this set, we found 290 "energy-saving commits", which had the explicit intention of saving energy. The group of energy-aware commits were performed by 265 open-source developers in 317 real world non-trivial open-source applications. We employ a thematic analysis [55] approach to examine the data and identify recurring topics in the commits. We highlight some of the findings of this study:

- The vast majority of the energy-saving commits (49.66% of them) target lower levels of the software stack (*e.g.,* Kernels and Drivers). Most of these commits describe solutions based on traditional approaches such as frequency scaling and exploring multiple levels of idleness. In contrast, we found only 47 commits (16.21% of them) that aim to improve energy consumption employing solutions such as using more efficient data structures or libraries. This suggests that there are opportunities to improve energy efficiency even further.
- We found that ill-chosen energy saving techniques can impact on the correctness of the application. 7 energy-aware commits warn about this.
- We identified 12 main themes regarding energy-saving commits, and we discuss 6 of them, namely: Frequency and voltage scaling, Use power efficient library/device, Disabling features or devices, Energy bug fix, and Low power idling. Interestingly, most of the commits (42.55% of them) in the Use power efficient library/device were employing the same library — the power efficient work queue.
- Developers are not always certain that their source code modifications will in fact save energy. 18 commit messages included "hesitating" words suggesting that the GITHUB contributors were not entirely sure about their effects, in terms of energy savings. We also identified 18 reverted commits.
- Even for systems where energy efficiency is critical, developers sometimes knowingly apply modifications that will have a negative impact on energy efficiency in order to balance trade-offs. We identified 42 commits documenting these situations. In contrast with the hesitating issue, this result suggests that some developers have a strong grasp over energy consumption-related solutions.
- Energy bugs are common. Nearly 11.03% of the 290 ones describe fixes to energy bugs, (*e.g.,* keeping the system in the wrong c-state or continuously updating a UI component when the screen is turned off). This motivates research on the identification and correction of these bugs [44], [73].
- We found two energy consumption experts. These experts performed 21 commits — 5.66% of the energy-aware commits.

II. METHODOLOGY

In this section we state the research questions of this study (Section II-A), the data collection procedure (Section II-B), and the qualitative research approach we employ (Section II-C).

A. Research Questions

As a first step towards our research goal, we designed the following research questions:

RQ1. What are the solutions that developers use to save energy in practice?

RQ2. What software quality attributes may be given precedence over energy consumption?

RQ3. How are energy-saving solutions distributed over the software stack?

RQ4. To what extent are software developers certain that their commits will save energy?

To answer **RQ1** we used a qualitative research methodology to analyze and group in themes each one of the energy-aware commits. For **RQ2** we investigated whether developers trade energy efficiency off by other quality attributes and, if so, what are these attributes. To answer **RQ3**, we analyzed the documentation of the projects to identify in which layer of the software stack (device driver, operating system, application, etc.) the project is located. To answer **RQ4** we first defined a set of "hesitating words" and then counted how many energy-aware commit messages are employing them. We then turned our attention to reverted commits and how they are used.

B. Data Collection Procedure

In order to select our dataset, we followed a two-phase approach. Since programmers usually express the intention of a code modification through commit messages, we start by examining them. However, using the default GITHUB web-interface or its REST API, we can only obtain commit messages for a particular project. Given the limitation of the number of requests that a host can send to GITHUB within a 60 minute interval and the absence of a queryable global index, we would not be able to conduct a large-scale study. Instead, we chose to use GITHUBARCHIVE[3]. According to the website, it is "*a project to record the public* GITHUB *timeline, archive it, and make it easily accessible for further analysis*". The archive is updated every hour and its dataset is available since February, 2011. This project exposes its data through a web tool that enables querying and exporting query results. Using its interface, we were able to query the commit

[3]http://www.githubarchive.org

messages of any open-source project available on GITHUB. At the time when this paper was written, the archive contained 96,387,889 commits.

We performed a query to select commits that are most likely to be related to energy consumption. Thus, our query searched for commits messages that contains at least one of the following terms: *energy consum*, *energy efficien*, *energy sav*, *save energy*, *power consum*, *power efficien*, *power sav* and *save power*. The character "*" in each term works as a wildcard: the query will select commits with messages that match at least one of these terms, regardless of the beginning or the end of the message content. These terms were already used in a previous study [76]. Using only "energy" or "power" terms resulted in 112,900 commits, which would prevent us from conducting a mostly manual study.

This query resulted in a total of 2,189 commits. However, we observed that some of these commits existed only in the GITHUBARCHIVE database, but not in GITHUB itself. At least three reasons might explain this fact: (1) the project could have been renamed or removed, (2) the project owner may have left GITHUB, or (3) the branch where the commit was placed was removed. We also directly asked the GITHUB staff if there is another reason for "unreachable commits", and they answered that "*We don't periodically gc repositories, however we do run gc on repositories on request from an owner of the repository, which would be the reason for unreachable commits*". We automatically removed 599 commits that were not found on GITHUB. We also found several commits that shared the same SHA key and/or the same commit message. Similarly, we automatically removed 600 duplicated commits. This left us with 1589 commits.

Next, we manually analyzed each of the 1589 remaining commits. GITHUB helped us with a web interface that greatly increases the readability of a patch. Using this interface, programmers can easily map a commit to its source code modifications, observing which lines of code were added or removed. We used this interface to investigate the source code of a given commit in order to understand if it, in fact, was performed with the intent of saving energy. During this analysis, we observed what we call "soft-duplicated" commits, that is, commits that do not share the same SHA code, but the commit message and the source code modification are rather similar to the original one. We believe that this might happen because GIT enables easy copying of changes between branches. Using the *rebase* or the *cherry-pick* utilities, it is possible to bring changes from one branch to another. In such cases, new commits are created with new SHA hashes, but the contents of these commits, *i.e.*, message and changes, are identical or nearly identical to the original ones, depending on the state of the target branch and how the operation was performed. We removed 42 occurrences of commits with these characteristics.

This manual code review process was divided among the first three authors of the paper. Each commit was, at least, double-checked. Each author analyzed two thirds of the commit population. For instance, taking in consideration a population of 100 commits, the first author analyzed the commits between 1 and 66, while the second author analyzed the commits between 34-100, and finally the third author analyzed the commits in the ranges 1-33 and 67-100. In order to gather even more information about the projects, we also reviewed their description on the root dir or on the own webpage, if available. Since some code changes require an in-depth knowledge of the application domain, when we did not understand what a certain modification or feature is for, we also searched in mailing lists and in Q&A websites. This manual analysis took about 3 and a half months.

Next we grouped the commits using high-level codes. Codes aim to help us to discriminate commits in terms of the ones that are, and are not, of interest. These codes are not connected to the thematic analysis we have conducted (Section II-C). We coded a commit as "OK" when the commit message and the source code modification are clearly focused on reducing energy consumption. "OK-UNABLE-TO-CATEGORIZE" was the class of commits where the intention in the message is clear, but we were unable to match this intention with the source code modifications. A "TRADEOFF" code indicates that a developer is trading energy consumption off for another quality attribute. Code "TRADEOFF-UNABLE-TO-CATEGORIZE" refers to commits that modified the system in ways that increase energy consumption, but did not explain what was gained by doing so. The "RELATED" code refers to commits that add an energy saving option, so that programmers or end-users can use it to save energy. "REVERT" commits are commits that undo energy saving intending commits, most probably using the `revert` GIT utility. "DUPLICATE" is a duplicate commit, "FALSE-POSITIVE" is a commit that is not related to saving energy, for instance, when a programmer is working on a feature that calculates the energy consumption, and "BINARIES" is for binary code. A final code, "NOT-FOUND", is regarding when a project gives a 404 error. This was the only one set automatically. Two authors coded the same groups of commits. When these two authors did not agree with a code, a third author provided an additional perspective. After this manual extraction phase, a total of 371 energy-aware commits were selected. These commits fit in the "OK", "RELATED", and "TRADE-OFF" codes. These commits were performed between 03/12/2012 and 05/15/2014. Table I summarizes the codes and their occurrences.

C. Qualitative Research Approach

Once the data is collected, we extract reliable information using a qualitative approach named Thematic Analysis [55]. Thematic analysis is a common qualitative analysis method that emphasizes examining and recording themes within data. The application of this approach has six stages: familiarization with data, generating initial codes, searching for themes among codes, reviewing themes, defining and naming themes, and producing the final report. We explain briefly how we conducted each one in the remainder of this section. A more detailed explanation of the general approach is available elsewhere [55].

TABLE I
COMMITS PER CODE.

Code	# Commits
OK	290
OK-UNABLE-TO-CATEGORIZE	72
TRADEOFF	23
TRADEOFF-UNABLE-TO-CATEGORIZE	19
RELATED	58
REVERTED	18
DUPLICATE	600
SOFT-DUPLICATE	42
FALSE-POSITIVE	454
BINARIES	14
NOT-FOUND	599
TOTAL	2,189

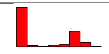
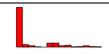
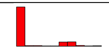
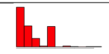
1) *Familiarization with data:* Here the first three authors analyzed the commit message and the source code modification for each commit. For commits that used specific language constructs or libraries, we have searched on internet forums and mailing lists in order to become familiar with them.
2) *Generating initial codes:* Each author gave a code for each analyzed commit. The code is an attempt to express the core of the modification. For instance, a commit that adds a new DVFS [59], a technique for dynamic scaling CPU frequency, algorithm can be coded as "DVFS". In this step, we also refined codes by combining and splitting potential codes.
3) *Searching for themes:* In this step, we already had a list of initial themes. Here, only the first author tried to combine coded data on appropriated themes. When in doubt, the other authors provided support to find broader patterns within data.
4) *Reviewing themes:* At this stage, we have a potential set of themes. We then searched for data that supports or refutes our theme. For instance, we updated the theme of a commit that was initially themed as "DVFS" to "Frequency and voltage scaling". In this commit, the programmer decreased the voltage of the display. This solution is related to voltage scaling but not to DVFS.
5) *Defining and naming themes:* Here we refined existing themes. At this time, most of the themes already had a name. However, we have renamed some of them to cover codes with small numbers of commits, otherwise we would have to discard them. We established 5 as a threshold for the minimum number of repetitions of a code required for it to be considered a theme.
6) *Producing the final report:* This process led to the elicitation of 12 main themes. Due to space constrains, we kept our focus to the 6 themes with the greatest number of occurrences.

III. STUDY RESULTS

We found a total of 371 energy-aware commits. This number represents 16.95% of the total number of commits we found in our initial query. These commits were performed in 317 different projects by 265 different GITHUB contributors. As regarding the GITHUB contributors, we found two energy consumption experts; these two GITHUB contributors have performed 21 energy-aware commits (10 commits each one) — that is, 5.66% of our population of energy-aware commits. Analyzing the commits of one these top GITHUB contributors, we observed that, even though they were performed by the same GITHUB contributor, they greatly differ between each other in terms of the intention used to save energy. For instance, they vary from (1) changes to tweak governor parameters [39], to (2) disabling a feature when the display is turned on [36] and to (3) directly reducing an LCD display voltage [29].

The project that received the most commits is android_kernel_motorola_omap4-common, with 9 energy-aware commits. This is a Kernel project based on Motorola 3.0.8 Android Kernel. The energy-saving intention here also varied greatly. For instance, some commits were performed with the intention to (1) select different energy-efficient governors [1], to (2) improve an existing governor implementation [10], and to (3) enable a power saving feature [6]. This shows that the same software application can benefit from different energy-aware optimizations. Other than this project, 5 other projects have between 5 to 3 energy-aware commits each. 19 projects have 2 energy-aware commits each, and the other ones have one energy-aware commit each. Table II shows the diversity of our target applications.

TABLE II
THE DIVERSITY OF OUR TARGET APPLICATIONS.

Metric	Mean	Median	Standard Dev.	Histogram
LoC	4,069,000	68,930	5,239,099	
Contributors	202.3	6.5	335.31	
Commits	68,250	475.5	126,104	
Age	4.50	3	3.72	

As we can see in the table, on average, our target applications have 4,069,193 lines of code (3rd quartile: 10,200,000, min: 15, max: 14,180,000), 5 different GITHUB contributors (3rd quartile: 267.8, min: 1, max: 1,320), 68,250 commits (3rd quartile: 36,440, min: 1, max: 495,800), and are 4.5 years old (3rd quartile: 4.5, min: 1, max: 22). Age is measured in years, considering the time when the first commit was placed until 02/10/2015. 13.72% of them were performed in applications with up to 1,000 lines of code, 19.86% of them in applications sizing from 1,001 to 10,000 lines of code, and the vast majority 66.43% of them were performed in applications with over 10,000 lines of code. C is the most used programming language (158 projects use it as the main language), followed by Java (25 projects), Bourne Shell (17 projects), Arduino Sketch (15 projects), and C++ (12 projects).

In the following subsections (Sections III-A to III-D), we answer the four research questions.

A. RQ1. What are the solutions that developers use to save energy in practice?

We now analyze the source code modifications of each one of the 290 energy-saving commits. We identified 12 categories of source code modification aiming at saving energy. Table III summarizes each one.

TABLE III
THE CATEGORIZATION OF THE APPROACHES DEVELOPERS USE TO SAVE ENERGY.

Theme	# Commits	%
Frequency and voltage scaling	47	16.21
Use power efficient library/device	47	16.21
Disabling features or devices	44	15.17
Energy bug fix	32	11.03
Low power idling	21	7.24
Timing out	15	5.17
Avoid polling	15	5.17
Pin management	10	3.45
Display and UI tuning	9	3.10
Avoid unnecessary work	5	1.72
Miscellaneous	36	12.41
Outlier	9	3.10

As Table III shows, the approaches that developers use in practice to save energy are quite diverse. Due to space limitations we choose to describe and provide discussions on just the top 6 most often used approaches.

Frequency and voltage scaling (47 occurrences) Most commits we analyzed fit within the frequency and voltage scaling theme. The key insight is that a lower frequency yields lower power consumption. Saving energy, however, is not the same of saving power, because a reduction in frequency may increase the execution time. The challenge here is to figure out when the reduction in frequency is significant enough to cause performance degradation, thus negatively impacting on energy saving. In our qualitative analysis, we observed that such manipulations can be static or dynamic.

- Static: The programmer hard-coded a new frequency/voltage value directly in the source code.
- Dynamic: The programmer is using a dynamic frequency/voltage technique, usually DVFS [74].

Although frequency and voltage scaling became a popular technique to make CPU processors more energy-efficient, we have identified several GITHUB contributors who focused on peripherals. For instance, a GITHUB contributor said that "*Reduce Wifi voltage for power savings. Should be beneficial for a wifi only device*" [30]. In such commit, the GITHUB contributor changed a single line of code, updating a variable from `.microvolts = 2000000` to `.microvolts = 1800000`. This commit used a static approach, that is, the GITHUB contributor hard-coded a new voltage value.

Solutions using the dynamic approach are greatly diverse. For instance, DVFS offers the chance to change the CPU frequency on the fly. DVFS algorithms, or cpufreq governors, dynamically decide what frequency should be used at a given time. We found several commits focused on using such DVFS features. They vary from (1) tuning existing governors [39] or (2) setting a different governor as default [1]. However, this approach hides important perils. As discussed in recent literature [68], [62], well-established Linux governors do not provide effective energy savings. In the worse case, governors can even increase energy consumption, instead of reducing it.

Use power efficient library/device (47 occurrences) The commits in this theme perform changes to use more efficient versions of certain libraries or services, as well as more energy-efficient devices. We also include in this theme commits that make use of power saving mode, usually a black-box technique that does not require knowledge about how the energy saving is achieved. Examples in this category are: (1) use power efficient work queue [38], (2) use a motion accelerometer instead of a general accelerometer [34], and (3) enable the Thermal framework to achieve energy savings [6]. Most of these power savings modes are offered by newer kernels (*e.g.*, the power efficient workqueue and the Thermal Framework), which greatly reduce the barrier for employing a power saving technique in low-level applications, since the programmer does not need to worry about low-level implementation details, which are abstracted away in those libraries.

Disabling features or devices (44 occurrences) This theme encompasses source code modifications aimed at (1) disabling application features or devices or (2) putting devices in low power mode/state. Some examples include: (1) disabling logging [43], (2) "*explicitly turn off SD card power after each data cycle*" [14], and (3) enabling audio codecs power saving [22]. In particular, the last example shows that the energy-saving modification is not always related to source code, since it can also come in configuration files. We discuss configuration files in more details in Section IV.

Energy bug fix (32 occurrences) This theme contains commits that fix "Energy Bugs". An energy bug is: "*an error in the system, either application, OS, hardware, firmware or external, that causes an unexpected amount of high energy consumption by the system as a whole*" [70]. We consider a commit as an Energy Bug Fix if the programmer clearly states at the commit message that it will fix an energy bug. Examples of energy bug fixes include (1) "*Hiding a UI dialog when the screen is off. Otherwise it causes continuous buffer updates to SurfaceTexture even when SCREEN is turned off (while In-Call) causing a spike in power consumption*" and (2) fixing a numerical overflow that causes wrong C-state to be selected [16]. Interestingly, however, we observed that energy bug fixes greatly differ in size. We have observed a commit that fixes a power consumption issue by inserting a single line of code [27]. This commit inserts a flag that enables the application to go to sleep mode. Otherwise, using the KEEP_SCREEN_ON flag, it would prevent the application from going to sleep. Unfortunately, most of the commits do not refer to an issue tracker. This prevented us from conducting an in-depth investigation of the energy bug life-cycle.

Low power idling (21 occurrences) General rule of thumb here is that the program, or a particular thread, will go to suspend/sleep/power saving/low power. It also covers increasing low power idle times. The rationale behind this theme comes from the idea that sending the device to an idle state might save energy. For instance, one of the solutions [42] is described in Figure 1. After performing its work on every iteration of the loop the device will be put in sleep mode until an external event wakes it up.

```
for (;;) {
  if (!Usb::start_of_frame())
    continue;
  Matrix::Scan();
  Tick();
  Usb::Tick();
  Usb::SendReport();
}
```

$\Downarrow$

```
for (;;) {
  if (Usb::start_of_frame()) {
    Matrix::Scan();
    Tick();
    Usb::SendReport();
  }
  sleep_mode();
}
```

Fig. 1. An example of low power idling

According to a recent study, idle states can be seen as complementary to other energy savings techniques, and can save up to 25% of energy consumption [62]. However, there are at least two important concerns that the programmer should consider when using this approach. First, putting threads in an idle state may require polling to wake them up. Depending on the polling frequency, it might be a double-edged sword, increasing energy consumption behind the scenes. Second, the frequency with which a thread is sent to an idle state, and then waken up, is also important. If this frequency is too high, it might increase the energy consumption due to several changes of states.

Timing out (15 occurrences) This theme covers commits that add, decrease or improve timeouts to stop a computation as an attempt to decrease the number of times that the computation is performed. Usually, the computation is wrapped in a loop, which is continuously running until a threshold or a halt event is reached. For instance, Figure 2 shows an example of such an approach [13]. The author of such commit removed the condition `if (scanTimes%100 == 0)` which guarantees that the application must run at least 100 times before getting the opportunity to stop after the 3 minutes timeout. The removal of the condition makes it possible for the computation to be stopped earlier, saving energy by not performing unnecessary work.

```
public void run() {
  int scanTimes = 0;
  while (isScanning){
    scanTimes++;
    if (scanTimes%100 == 0){
      //stop after 3 min
      if (currentTime-startTime>1000*3*60){
        isScanning=false;
        break;
      }
      doWork();
}}}
```

$\Downarrow$

```
public void run() {
  while (isScanning){
    //stop after 3 min
    if (currentTime-startTime>1000*3*60){
      isScanning=false;
      break;
    }
    doWork();
}}
```

Fig. 2. An example of timing out.

B. *RQ2. What software quality attributes may be given precedence over energy consumption?*

We have observed cases where GITHUB contributors are trading software energy consumption for other software attribute. We found 42 commits in this regard. However, for 19 of them, we did not recognize what software quality attribute was being considered. This happens mainly because the commit message was poorly elaborated and we did not infer any clear intention from it. Examples of such hard to understand commit messages are (1) "*no power saving in X*" [24], (2) "*Remove Wifi power save off patch* " [31], and (3) "*turns off screen power saving*" [41]. More importantly, however, we observed that for *all* aforementioned cases, the GITHUB contributor is disabling/removing the power saving mode provided by a third-party device. Since these third-party features are usually shipped as black-box functions, the programmer is not able to understand the underlying techniques being used behind the scenes and, with no other choice, she trades this black-box power saving mode for another software attribute. This suggests that such third-party power saving libraries cannot be seen as silver bullets.

For the 23 remaining ones, we observed that the most traded software attributes are the following: "Correctness", "Responsiveness", "No actual energy saving", "Performance", and "Miscellaneous". We describe each one of them below.

Correctness (7 occurrences) One of the most interesting observations from this group of commits is that ill-chosen energy saving techniques can impact on the correctness of a software, as stated by different GITHUB contributors, for instance (1) "*the power saving delay has the ability to corrupt serial transmissions. Changing these to regular delays fixes this*" [33], (2) "*USB autosuspend might be causing some trouble with various smartboards and WLAN accesspoints*" [32] where "autosuspend" is a power saving mode, and (3) "*Disable Auto Power Saving when resetting the modem. This can cause several bugs with serial communication*" [8]. Likewise the aforementioned unable to categorize ones, the root cause for *all* correctness problems were due to the use of power saving mode provided by third-party devices. In particular, two GITHUB contributors have acknowledged the same correctness problem: *power saving mode can corrupt serial transmission.*

Responsiveness (6 occurrences) Responsiveness is an important software attribute in general, and for smartphones in particular. In this regard, 2 out of the 6 occurrences are related to touch events, for instance: "*For better Ux responsiveness ondemand sampling rate needs to be 20ms. But, a 20ms sampling rate increases power consumption. So, boost the sampling rate to 20ms on every touch event for 2.5 ms and later reset to default rate*" [5]. Other two occurrences are related to wifi usage, for instance: "*Wi-Fi should be in active state during the entire DHCP process, and shouldn't go to IEEE 802.11 power save mode. If the framework requests scan during the DHCP process, the Wi-Fi chip has to start scanning on channels different from the current one, and going to power save mode is a prerequisite for scan. The result directly impacts user experience: DHCP process takes longer, and even can fail*" [28].

Performance (3 occurrences) Performance is an energy consumption close relative. As expected, programmers trade energy consumption for better performance, for instance: "*The lowest power level does not correspond to significant power savings as the whole system doesn't drop to SVS voltage. Performace is improved without this level for serialized test cases*" [23].

No actual energy saving (3 occurrences) Some GITHUB contributors did not observe any effective energy saving with the employed solution. In these commits, the GITHUB contributor is removing the supposed energy-efficient code. For instance, "*Enable LPA playback for music streams only. There are no significant power savings for using LPA with other modes like ringtone*"[17], or "*It is unlikely this gives any power savings and only add some ugly code*"[25]. Since we did not find any mention to energy consumption profiling tools in the commit messages, we believe that this perception of lack of energy saving is based on developers own wisdom. However, reasoning about the energy behavior of an application is not a straight-forward task [76].

Miscellaneous (3 occurrences) In this category we grouped all remaining commits. They vary from improve memory usage [3], power saving not needed [9], and accuracy [26].

C. RQ3. How are energy-saving solutions distributed over the software stack?

We have observed that the energy-saving commits are spread over the entire software stack. We used the same software layer definition provided by Stallings [83], which encompasses Operating System, Libraries/Utilities and Applications. Operating System includes Kernels, Embedded Kernels, Drivers and Firmwares. Libraries/Utilities include scripts (general purpose scripts, building scripts and compile scripts) and embedded libraries. Application includes embedded applications, desktop application, and mobile applications. To determine the conforming software layer for each commit we used as references: (1) the project description on GITHUB, (2) the files names, extensions and directory structure and (3) the changed source code itself and related documentation. The project description generally provides an important hint on the conforming layer as it describes the purpose of the software. Files names, extensions and folder structure may also give information about the stack level, *e.g.*, a driver related commit [15] may change files inside a folder named "driver" or a commit to an Arduino application may change files ending with an ".ino" extension [20]. In the same manner, the code itself provides contextual information about the stack level, for example, changes to code conforming to the same level may follow a given pattern or have certain similarities [11], [40], [12]. Table IV shows the number of energy-saving commits per software level.

TABLE IV
COMMITS PER APPLICATION LEVEL.

Level	# Commits
Application	93
Libraries/Utilities	53
Operating System	144

As we can see, most of the energy-saving commits are targeting the Operating System level. In particular, Kernels received the greatest number of commits (70 commits), followed by drivers (53 commits). Application software received 93 commits. However, 48 of these commits are related to embedded software — *e.g.*,21 commits are targeting Arduino applications. The remaining ones, 45 of them, are targeting traditional desktop and mobile applications. This low number of commits is unfortunate because there is convincing evidence that even better energy savings can be achieved by encouraging application developers to produce energy-aware software solutions [75], [77], [63]. Application developers ought to seize more energy saving opportunities.

D. RQ4. To what extent are software developers certain that their commits will save energy?

The lack of tools for developing energy-aware applications is well-known (*e.g,* [68], [76]). However, our data suggest that developers are actively performing energy-aware commits. In this research question we analyze how certain software developers are that a modification will in fact save energy. To do so, we investigate if a set of "hesitating" words are used in the commit messages. This set encompass "seem", "might", "doubt", "could", "hope", "attempt", "supposed", "guess", and "likely". We then queried our commit database looking for these hesitating words, and we found a total 18 energy-aware commits. Some examples of such energy-aware hesitating commits are the followings: (1)"*Slowed down SensorProbe sampling frequency in hopes of reducing power consumption*" [37], and (2) "*Including some power savings modes. I doubt they amount to much, but I'm not using any of them, so why not*" [19]. This finding suggests that software developers may not be always confident when writing energy-efficient applications. One might argue that the hesitation happens only when a GITHUB contributor is performing a non-trivial task with several code changes. However, we found cases where a GITHUB contributor is performing a single variable change, but she is not sure if the change will reduce positively energy consumption, for instance: "*This patch disables [a feature] and advertises only SWSUP mode which seems to improve performance and reduce power consumed in AV record use-case*" [21].

Second, we observed that 18 commits were reverted. Revert is a GIT feature that undoes a given commit so that the branch points again to the prior commit and discards the last one, even though the reverted commit was already pushed to the main repository. Interestingly, most of the reverted commits, 8 of them, were using the *power efficient work queue*. According to the documentation, "*Workqueues can be performance or power oriented. For performance we may want to keep them running on a single cpu, so that it remains cache hot. For power we can give scheduler the liberty to choose target cpu for running work handler*"[4]. Although we cannot be sure why such commits were reverted, this result at least suggests that the decision of when to use a power-efficient library should be made with care; the trade-off between performance and energy consumption is not always clear [77], [66], [57].

IV. FURTHER ANALYSIS

In this section, we provide additional discussion on the data presented in the previous sections.

Mobile Applications. Thanks to the rapid proliferation of mobile phones, tablets, and unwired devices in general, energy-efficiency is becoming a key software design consideration where the energy consumption is closely related to battery lifetime. We found 47 commits that are targeting mobile software. Although most of these commits are related to Android Kernels, we found several commits that focus on the application-level.

Energy Efficient Software Product Lines. It is well-known that the Linux Kernel is developed as a software product line [82]. One important challenge with software product lines is to create the simplest working product, with no unnecessary features. This is important because redundant code can introduce inefficiencies that, when accumulated, can slow down the performance of an application [87]. This is also an important concern with the Linux Kernel. We have found 13 commits where the author is changing the Linux configuration files in order to create a more energy-efficient kernel.

[4]http://lwn.net/Articles/548281/

Check-Then-Act Oriented Programming. We found a total of 27 commits that follow a check-then-act idiom. In such commits, the programmer first verifies a given system property is available and, if so, acts. For instance, the programmer checks if the GPU is active, prior to collecting a sample. Such programming style is useful to avoid unnecessary work. We have found this programming style in different contexts, such as: (1) "*Disable accelerometer sensor while in-call and screen UI is off*" [7], and (2) "*charger: suspend when not animating*". Yet, such check-then-act style can be useful to bulk different energy saving techniques, for instance, "*turns mobile data and autosync on and off automatically [...] while the phone is inactive, turns Wifi and Bluetooth off when disconnected for a period of time, and reduces screen brightness and timeout when battery levels get low*" [18].

Energy-Aware Interfaces. We identified 58 commits that were performed with the intention to create what we call "energy-aware interfaces". An energy-aware interface provides means for clients (*e.g.,* developers or end-users) to save energy in their applications just by using a function, abstracting away the low-level implementation details. These interfaces are of particular importance to end-users because, without them, end-users would not be able to access such low-level implementation details, *e.g.,* because they work on the kernel mode. Take as an example project android_packages_apps_Settings, which is an Android application. In its energy-aware commit [35] it adds a feature on the application's menu that allows end-users to save energy by putting the WiFi in a power saving mode — an *end-user centric* approach. On the other hand, project i9105Sammy adds a new DVFS governor designed for latency-sensitive workloads. According to the commit message [2], this governor "*attempts to reduce the latency of clock increases so that the system is more responsive to interactive workloads in loweset steady-state but to reduce power consumption in middle operation level up will be done in step by step to prohibit system from going to max operation level*", a *programmer-centric* approach.

Energy-Aware Source Code Review. On GITHUB, when a commit is pushed to a remote repository, other GITHUB contributors can provide comments to the commit, so that the original author can improve the existing commit based on the comments provided in order to match with the team expectations. In fact, we found 5 energy-aware discussions in our dataset. These discussions are mostly related to the behavior of the application on a particular scenario, for instance, "*what does happen if the screen turns on due to an incoming call?*" [4]. This low number of discussions found does not suggest, however, that developers do not have interest in discussing energy consumption issues. According to a recent study, only few developers contribute to a broader range of design discussions in a project [47].

V. Implications

Developers. The results of our study provide some assistance to software developers. First, by showing that software energy-efficiency is important and they cannot ignore it (Section III). Second, our results encourage software developers to make more energy-aware commits. In particular, we observed that, even though application programmers can yield better energy savings [75], [63], application software received about half of the kernel/operating system commits (**RQ3**). Third, with this study, developers can learn from mistakes made by their peers (**RQ2**).

Researchers. Researchers can also benefit from our results. We have observed that voltage and frequency scaling are among the most often employed approaches to save energy (**RQ1**). However, such techniques require an in-depth knowledge of low-level implementations details, besides being *error-prone* and *omission-prone*. Since developers believe in the effectiveness of these techniques, researchers can propose novel, high-level and ease to use techniques in order to reduce the burden of using low-level ones (*e.g.,* [45], [78]). Also, this study provided evidence that third-party power saving techniques cannot be seen as a silver bullet and, at the worst case, can even corrupt a software (**RQ2**). Researchers can help in this direction by providing an in-depth investigation of the advantages and the disadvantages of general purpose power saving techniques.

Library Designers. Our analysis showed that programmers rely on energy-efficient libraries, such as the *power-efficient work queue* (**RQ1**). We believe that energy-efficient libraries can play an important role in reducing the burden of writing energy-efficient applications. Library designers can, at least, improve their documentation to mention how their libraries behave, from an energy consumption standpoint. In particular, such documentation should explain in which scenarios the library can, or cannot, be used, thus avoiding potential correctness problems (**RQ2**). At best, library designers can create energy-efficient versions of their libraries.

Tool Vendors. Developers are in need of appropriate tools to measure/identify/refactor energy consumption hotspots [76]. Energy consumption tools do exist [57], [66], [68], but most of them do not provide direct guidance on energy optimization, *i.e.*, bridging the gap between understanding where energy is consumed and how the code can be modified in order to reduce energy consumption. Tool vendors can play a role by introducing novel tools for profiling energy consumption. With appropriate tools, developers can be more certain about how their modifications will impact energy consumption (**RQ4**), and thus may hesitate less. Tool vendors can also play an important role in empowering application developers, for instance, supporting refactorings for energy-efficiency, thus decreasing significantly the barrier for placing an energy-aware commit (**RQ1**).

Lecturers in Computer Science. The results of this study can be helpful for lecturers and students. First, the diversity of projects showed in Section III can raise the discussion that energy consumption issues are not only important for data centers and high-performance computing in general, but for simple applications as well. Also, our themes can be used in programming-related courses to illustrate what are the real-life solutions proposed by software developers when writing energy-efficient software applications (**RQ1**). Lecturers can incentivize students to provide additional solutions to these problems, and potential problems behind them.

VI. Threats to Validity

Internal validity. First, we collected the commits using a GITHUB third-party mirror, GITHUBARCHIVE. Then, we cannot be sure whether we are collecting all commits available on GITHUB. This is, however, the most fair approach we found, since GITHUB itself does not provide an interface for querying commit messages among different software repositories. Second, our approach does not cover the whole spectrum of energy-related commit messages. To make the matters worse, commit messages are often vague, meaningless, or have typo errors. We mitigate this problem by searching with wildcards. Thus, we can query terms regardless their position (in the beginning, middle or end) in the commit message. Wildcards also allow us to query part of the term (*e.g.,* "consum*", instead of "consumption"), which can cover abbreviations, typos or similar words. Third, commits categorization is *human-prone*. For instance, a commit might be seen as "OK" for one, but "RELATED" for another one. We mitigate this risk by, at least, doubling-check each one of the analyzed commits. When the authors did not agree with each other, a third author was invited to the discussion and provided additional comments. Fourth, we only analyzed commit messages written in English. However, English is the *de facto* language used to communicate in software development. Finally, since we are not the commit authors, some findings might appear as a over/under-generalization.

External validity. First, our results only apply to software developers who performed energy-aware commits on GITHUB. It does not cover software developers in other source code hosting websites. Second, our results are limited by our selection of commit messages. Such commits were performed in wide spectrum of non-trivial applications, ranging from operating systems, kernels, and mobile apps. These applications were developed by different teams with 1 up to 1,320 contributors, using different programming languages, from a large and varied community. Third, there are other possible energy-related source code manipulations beyond the scope of this paper. With our methodology, we expected that similar analysis can be conducted by others when they became relevant. Fourth, this paper does not address the problem of understanding whether these source code modifications actually save energy consumption. Although we provide some discussions based on the source code modifications, it is well known that energy consumption is heavily application-dependent [77]. A definitive answer would require a in-depth runtime investigation of each target system. Finally, we have analyzed live and under evolution systems. During our analyzes, we have observed that some commits that were initially found became not-found. As a means for replicability, we have download and stored all commits analyzed in this study in a database. Along with it, we make available a fine-grained report[5]. We encourage others to replicate our study.

VII. Related Work

Most of the existing software empirical research has focused on the trade-off of comparing individual characteristics of an application and energy consumption. These characteristics vary from data structures [53], [60], [69], VM services [48], cloud offloading [63], code obfuscation [81], and design patterns [79], [64]. Such research serves as a guideline for future energy-aware application programmers.

The mobile arena is also an important topic of research. Hindle [58] investigated the relationship between software changes (several versions of the Mozilla Firefox app) and power consumption. The author observed that intentional performance optimization introduced a steady reduction in power consumption. Pathak *et al.* [71] categorized energy bugs through analyzing the posts from 4 online forums. They produced a comprehensive taxonomy ranging from battery problems, SIM card problems, OS configuration problems, to no-sleep bugs. Pathak *et al.* [72] presented an investigation aiming to understand the root causes for energy consumption problems in mobile applications. Linares-Vasquez *et al.* [67] investigated Android API usage patterns that can potentially consume high energy consumption. The authors observed that while some anomalous energy consumption are unavoidable, some can be avoided by using certain categories of Android APIs and patterns. Similarly, Li *et al.* [65] presented the a large scale study on the energy-efficiency of mobile applications. Among the findings, we describe two of interest: (1) a few set of APIs used in applications dominate non-idle energy consumption and (2) an HTTP request is the most energy consuming operation of the network.

However, to the best of our knowledge, there is only one work that deals with the topic of understanding what are the solutions proposed by software developers in order to improve software energy consumption [76]. In this work, Pinto *et al.* [76] surveyed StackOverflow, a programmer-centric website, considering a developer-oriented view of energy-aware *software development*. This work provides useful insights such as the most common energy consumption related problems, and the solutions to them. However, this work focused on what developers *believe*. In contrast, this work provides a set of solutions that developers actually *employ* in the hope to save software energy consumption.

VIII. Conclusion

In this paper we analyzed what are the solutions created by application programmers to save software energy consumption in practice. Starting from a set of more than 2,000 commits performed at several kind of open-source projects in Github, we manually analyzed 371 of them. We conducted an in-depth investigation in the source modified by such commits, we categorized them in 12 main themes and discuss 6 of them.

In future work we plan to significantly expand the scope of the paper study. In particular, we plan to investigate whether the approaches identified here are consistent across multiple mobile platform and operating systems. We also plan to conduct energy consumption experiments in order to verify if the identified approaches actually save energy, and if so which is the degree of saving that can be achieved in each solution. Also, we plan to contact the commit authors to better understanding the intention behind the commit, and thus mitigate the over/under generalization problem.

IX. Acknowledgments

We would like to thank the anonymous reviewers for their helpful comments, and the members of the SPG group[6] for the fruitful discussions. Gustavo is supported by CAPES/Brazil, Felipe is supported by FACEPE (IBPG-0791-1.03/13), and Fernando is supported by CNPq/Brazil (304755/2014-1, 487549/2012-0 and 477139/2013-2), FACEPE/Brazil (APQ-0839-1.03/14) and INES (CNPq 573964/2008-4, FACEPE APQ-1037-1.03/08, and FACEPE APQ-0388-1.03/14). Any opinions expressed here are from the authors and do not necessarily reflect the views of the sponsors.

References

[1] Adaptivex set to default governor. https://github.com/RAZR-K-Devs/android_kernel_motorola_omap4-common/commit/4b3f259eb0f97fab24ac3b18667a67c18c33476b. Accessed: 2015-03-19.

[2] Added adaptive cpu governor. https://github.com/k2wl/i9105Sammy/commit/e4a7fb4237b65cec9ca909c218331a41814cabac. Accessed: 2015-03-19.

[3] Added more comfortable ui to the attiny24 lamps. need more flash! https://github.com/madworm/ATtiny_projects/commit/412bbd3a53c990bbbc2d65c9d189b55e4a7c73b1. Accessed: 2015-03-19.

[4] Allows switching to power saving mode. https://github.com/k2wl/i9105Sammy/commit/e4a7fb4237b65cec9ca909c218331a41814cabac. Accessed: 2015-03-19.

[5] cpufreq: boost the sampling rate on touch event. https://github.com/CyanogenMod/android_kernel_htc_msm8960/commit/694447e. Accessed: 2015-03-19.

[6] defconfig: Enable hdmi_toggle, enable thermal_sys module, thermal_fra. https://github.com/RAZR-K-Devs/android_kernel_motorola_omap4-common/commit/3f2329812930888d37c55dc616f6a8356ff95852. Accessed: 2015-03-19.

[7] Disable accelerometer sensor while in-call and screen ui is off. https://github.com/burstlam/android_packages_apps_Phone/commit/1e509fe6d08f66d7fab47e01c632a6bd181afb34. Accessed: 2015-03-19.

[8] Disable auto power saving. https://github.com/alobo/SerialGSM/commit/c616b950bd144e9e4c32c337d6429d059ef12b94. Accessed: 2015-03-19.

[9] Disable wifi power saving. https://github.com/J1nx-Hackable-Gadgets/buildroot-linux-kernel-m3/commit/46fa2eea1f1c86331e4ef9e53224c103bef95533. Accessed: 2015-03-19.

[10] drivers: cpufreq: ktoonservative: tune for being more balanced and sa. https://github.com/RAZR-K-Devs/android_kernel_motorola_omap4-common/commit/e25e040cfe71ae99bb6d66dec37d4ad118da920d. Accessed: 2015-03-19.

[5]http://bit.ly/energy-aware-mining

[6]http://twiki.cin.ufpe.br/twiki/bin/view/SPG

[11] enable power save. https://github.com/samm-git/device_alcatel_OT993D/commit/d108dccc376b62e79ab5800ceac6a3d5a6acb07e. Accessed: 2015-03-19.

[12] enable power saving after boot. https://github.com/CyanogenMod/android_device_sony_montblanc-common/commit/bd032afd36ab0bd45c105239550d0d97d46bcf33. Accessed: 2015-03-19.

[13] Energy saving added, scan start, stop responsiveness added. https://github.com/Bagception/MiniMeLibrary/commit/5b9882167377b32c69c8eabc35f58a1c58b78ec6. Accessed: 2015-03-19.

[14] Explicitly turn off sd card power after each data cycle. https://github.com/rickshory/AVRGreenlogger/commit/bbd6f5e60fcf77192d2eb49ed0b6129a019af5d3. Accessed: 2015-03-19.

[15] fix power consumption issue caused by ill-defined power state at startup. https://github.com/kugel-/rockbox/commit/73732f406ebd3e5b85a70c8f7ff60fd26144551a. Accessed: 2015-03-19.

[16] Fixed wrapping timers at 4.294 seconds. https://github.com/desalesouche/kernel_huawei_u8220/commit/13549799684cbf29200fc183529bcf9de9a33622. Accessed: 2015-03-19.

[17] frameworks/base: Enable lpa for music stream only. https://github.com/CyanogenMod/android_frameworks_base/commit/283df8ca2e87c89ddb9952957319191281eba818. Accessed: 2015-03-19.

[18] Import of power saving/management project. https://github.com/adein/Tasker/commit/ef6669ee93562d4f14b44fca1b38f7871029caf1. Accessed: 2015-03-19.

[19] Including some power savings modes. https://github.com/pfriedel/FiveMilSpire/commit/4c3652781307a60e6e2bbc5bce0ca5c53ad95c11. Accessed: 2015-03-19.

[20] Interrupt and power saving. https://github.com/barney-parker/Arduino-Countdown-Timer/commit/9178594298dbf216a241c7dbfd5ecd2889588133. Accessed: 2015-03-19.

[21] [ivahd] disable hwsup mode and save power. https://github.com/ch33kybutt/kernel_cmplus_tuna/commit/342dd821bed948b15c40de7276094e6e011bd379. Accessed: 2015-03-19.

[22] Module options for power saving in ac97 and hda audio codecs. https://github.com/intelfx/configs/commit/5d42182ea36188e58ad8f1e0c9104b92c6f65bc0. Accessed: 2015-03-19.

[23] msm: kgsl: Remove lowest power level. https://github.com/Team-Blackout/Blackout-Monarudo/commit/ed0c174. Accessed: 2015-03-19.

[24] no power saving in x. https://github.com/lmio/liox/commit/c12e2c5bdcca28a06b5c33245ddf5f93bb528969. Accessed: 2015-03-19.

[25] ondemandplus governor: remove adaptive timer_rate logic. https://github.com/burstlam/leanKernel/commit/4bfcac6cd2be96ef489aabee96301efe0789e111. Accessed: 2015-03-19.

[26] Only goes into power save if has lock. https://github.com/mattbrejza/kraken/commit/4118a9a5570d3d144e9f9b06f22a1776760859d2. Accessed: 2015-03-19.

[27] Prevent lockscreen album art from activating flag_keep_screen_on. https://github.com/KitKatPurity/platform_frameworks_base/commit/9f3d5cf. Accessed: 2015-03-19.

[28] Prevent scanning during dhcp process. https://github.com/CyanogenMod/android_frameworks_base/commit/c70ebda. Accessed: 2015-03-19.

[29] Reduce voltage to lcd screen to save power. https://github.com/ryrzy/LG-D802-G2-_Android_KK_D802_v20b/commit/cebcdec40218cd3faa51a0e9a5a76082e417c77e. Accessed: 2015-03-19.

[30] Reduce wifi voltage for power savings. https://github.com/Metallice/android_kernel_samsung_espresso10/commit/34f4738019a9e04c882ffa05bed1b40b81017c51. Accessed: 2015-03-19.

[31] Remove wifi power save off patch. https://github.com/M66B/cm10-fxp-extended/commit/756592b0c655a06f48a75ec10ae3f18f532281c0. Accessed: 2015-03-19.

[32] Rest: disable usb power save for fatclients and thinclients. https://github.com/opinsys/puavo-users/commit/1bfb72030a7265bfe18b9a623b2bf1218834d4fd. Accessed: 2015-03-19.

[33] Semi working - sensors read out, but bad radio. https://github.com/Teslafly/emonTH/commit/659afda362c0e0a23dc36d6f56fa0aab83812c91. Accessed: 2015-03-19.

[34] Sensor: Enable the motion accelerometer for screen orientation change. https://github.com/PSX-PureSpeed/android_frameworks_base/commit/b37764a0f6268bd74166117a2603e76547a4a31a. Accessed: 2015-03-19.

[35] Settings: Wireless power saver. https://github.com/OmniKang/android_packages_apps_Settings/commit/06922d4ef1ea6b7a09c1ba3086cbea17. Accessed: 2015-03-19.

[36] Shutdown s2w on screen on, and activate on off. https://github.com/dorimanx/initramfs3/commit/b423b7c7a8afecbeb95a0441acc8e8d3fdc45211. Accessed: 2015-03-19.

[37] Slowed down sensorprobe sampling frequency in hopes of reducing power consumption. https://github.com/OpenSensing/funf-v4/commit/b109d4bd71df191c414eab61ef57e00265f2afdb. Accessed: 2015-03-19.

[38] Test: Wcd9310: Use power efficient workqueue. https://github.com/yseras/SGS4-3.13/commit/7618d681. Accessed: 2015-03-19.

[39] Tuned abyssplug for more power save! https://github.com/dorimanx/Dorimanx-SG2-I9100-Kernel/commit/fe6fc634b61f68306a8929b464e7d6ce1ff24a90. Accessed: 2015-03-19.

[40] Tuned govs to more agressive power save. https://github.com/dorimanx/initramfs3/commit/a8d84e287a9c27828577d940f66d6a1245706157. Accessed: 2015-03-19.

[41] Turns off screen power saving. https://github.com/lomogoto/ConfigFiles/commit/ea6bf6ea591e2c94e5094dbcf90f94e5. Accessed: 2015-03-19.

[42] Updated to be more energy efficient. https://github.com/Ferroin/mspbinclk/commit/c561c0db151d2988fdfce2f3482cc4846cdf45b9. Accessed: 2015-03-19.

[43] Use new control value to stol logcat from writing logs, and save power! https://github.com/dorimanx/initramfs3/commit/086132af909d46bb9dac3c713e5db0. Accessed: 2015-03-19.

[44] 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 2014, pages 588–598, 2014.

[45] T. W. Bartenstein and Y. D. Liu. Green streams for data-intensive software. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 532–541, 2013.

[46] A. Begel, J. Bosch, and M.-A. Storey. Social networking meets software development: Perspectives from github, msdn, stack exchange, and topcoder. *IEEE Softw.*, 30(1):52–66, Jan. 2013.

[47] J. a. Brunet, G. C. Murphy, R. Terra, J. Figueiredo, and D. Serey. Do developers discuss design? In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 340–343, 2014.

[48] T. Cao, S. M. Blackburn, T. Gao, and K. S. McKinley. The yin and yang of power and performance for asymmetric hardware and managed software. In *Proceedings of the 39th Annual International Symposium on Computer Architecture*, ISCA '12, pages 225–236, 2012.

[49] Cat Phones. New research reveals mobile users want phones to have a longer than average battery life. http://bit.ly/1Eccqr3, November 2013. [Online; accessed 14-Feb-2015].

[50] L. F. Cortes-Coy, M. L. Vásquez, J. Aponte, and D. Poshyvanyk. On automatically generating commit messages via summarization of source code changes. In *14th IEEE International Working Conference on Source Code Analysis and Manipulation, SCAM 2014, Victoria, BC, Canada, September 28-29, 2014*, pages 275–284, 2014.

[51] L. Dabbish, C. Stuart, J. Tsay, and J. Herbsleb. Social coding in github: Transparency and collaboration in an open software repository. In *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work*, CSCW '12, pages 1277–1286, 2012.

[52] H. David, E. Gorbatov, U. R. Hanebutte, R. Khanna, and C. Le. Rapl: Memory power estimation and capping. In *Proceedings of the 16th ACM/IEEE International Symposium on Low Power Electronics and Design*, ISLPED '10, pages 189–194, 2010.

[53] E. G. Daylight, T. Fermentel, C. Ykman-Couvreur, and F. Catthoor. 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*, WOSP '02, pages 134–141, 2002.

[54] K. I. Farkas, J. Flinn, G. Back, D. Grunwald, and J. M. Anderson. Quantifying the energy consumption of a pocket computer and a java virtual machine. In *Proceedings of the 2000 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '00, pages 252–263, 2000.

[55] 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*, 5, 2006.

[56] Google. Eficciency: How can other do it – data centers. http://www.google.com/about/datacenters/efficiency/external/, 2015. [Online; accessed 21-Jan-2015].

[57] S. Hao, D. Li, W. G. J. Halfond, and R. Govindan. Estimating mobile application energy consumption using program analysis. In *35th International Conference on Software Engineering, ICSE '13, San Francisco, CA, USA, May 18-26, 2013*, pages 92–101, 2013.

[58] A. Hindle. Green mining: A methodology of relating software change to power consumption. In *Mining Software Repositories (MSR), 2012 9th IEEE Working Conference on*, pages 78–87, June 2012.

[59] M. Horowitz, T. Indermaur, and R. Gonzalez. Low-power digital design. In *Low Power Electronics, 1994. IEEE Symposium*, 1994.

[60] N. Hunt, P. S. Sandhu, and L. Ceze. Characterizing the performance and energy efficiency of lock-free data structures. In *Proceedings of the 2011 15th Workshop on Interaction Between Compilers and Computer Architectures*, INTERACT '11, pages 63–70, 2011.

[61] G. Jin, L. Song, X. Shi, J. Scherpelz, and S. Lu. Understanding and detecting real-world performance bugs. In *ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 77–88, 2012.

[62] M. Kambadur and M. A. Kim. An experimental survey of energy management across the stack. In *Proceedings of the 2014 ACM International Conference on Object Oriented Programming Systems Languages and Applications, OOPSLA 2014, part of SPLASH 2014, Portland, OR, USA, October 20-24, 2014*, pages 329–344, 2014.

[63] Y. Kwon and E. Tilevich. Reducing the energy consumption of mobile applications behind the scenes. In *2013 IEEE International Conference on Software Maintenance, Eindhoven, The Netherlands, September 22-28, 2013*, pages 170–179, 2013.

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

[65] D. Li, S. Hao, J. Gui, and W. G. J. Halfond. An empirical study of the energy consumption of android applications. In *30th IEEE International Conference on Software Maintenance and Evolution, Victoria, BC, Canada, September 29 - October 3, 2014*, pages 121–130, 2014.

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

[67] 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*, MSR 2014, pages 2–11, 2014.

[68] K. Liu, G. Pinto, and D. Liu. Data-oriented characterization of application-level energy optimization. In *Proceedings of the 18th International Conference on Fundamental Approaches to Software Engineering*, FASE'15, 2015.

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

[70] 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*, HotNets-X, pages 5:1–5:6, 2011.

[71] 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*, HotNets-X, pages 5:1–5:6, 2011.

[72] 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*, EuroSys '12, pages 29–42, 2012.

[73] 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*, MobiSys '12, pages 267–280, 2012.

[74] T. Pering, T. D. Burd, and R. W. Brodersen. The simulation and evaluation of dynamic voltage scaling algorithms. In *Proceedings of the 1998 International Symposium on Low Power Electronics and Design, 1998, Monterey, California, USA, August 10-12, 1998*, pages 76–81, 1998.

[75] G. Pinto and F. Castor. Characterizing the energy efficiency of java's thread-safe collections in a multicore environment. In *Proceedings of the SPLASH'2014 workshop on Software Engineering for Parallel Systems (SEPS)*, SEPS '14, 2014.

[76] 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*, MSR 2014, pages 22–31, 2014.

[77] G. Pinto, F. Castor, and Y. D. Liu. Understanding energy behaviors of thread management constructs. In *Proceedings of the 2014 ACM International Conference on Object Oriented Programming Systems Languages and Applications*, OOPSLA '14, pages 345–360, 2014.

[78] H. Ribic and Y. D. Liu. Energy-efficient work-stealing language runtimes. In *Architectural Support for Programming Languages and Operating Systems, ASPLOS '14, Salt Lake City, UT, USA, March 1-5, 2014*, pages 513–528, 2014.

[79] C. Sahin, F. Cayci, I. L. M. Gutiérrez, J. Clause, F. E. Kiamilev, L. L. Pollock, and K. Winbladh. Initial explorations on design pattern energy usage. In *First International Workshop on Green and Sustainable Software, GREENS 2012, Zurich, Switzerland, June 3, 2012*, pages 55–61, 2012.

[80] C. Sahin, L. L. Pollock, and J. Clause. How do code refactorings affect energy usage? In *2014 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM '14, Torino, Italy, September 18-19, 2014*, page 36, 2014.

[81] C. Sahin, P. Tornquist, R. Mckenna, Z. Pearson, and J. Clause. How does code obfuscation impact energy usage? In *30th IEEE International Conference on Software Maintenance and Evolution, Victoria, BC, Canada, September 29 - October 3, 2014*, pages 131–140, 2014.

[82] J. Sincero, H. Schirmeier, W. Schröder-Preikschat, and O. Spinczyk. Is the linux kernel a software product line? In *Proc. SPLC Workshop on Open Source Software and Product Lines*, 2007.

[83] W. Stallings. *Operating Systems - Internals and Design Principles (7th ed.)*. Pitman, 2011.

[84] TheGuardian. Facebook 'unfriends' coal and 'likes' clean power. http://www.theguardian.com/environment/2011/dec/15/facebook-coal-clean-power-energy-greenpeace, 2015. [Online; accessed 21-Jan-2015].

[85] F. Thung, T. F. Bissyande, D. Lo, and L. Jiang. Network structure of social coding in github. In *Proceedings of the 2013 17th European Conference on Software Maintenance and Reengineering*, CSMR '13, pages 323–326, 2013.

[86] C. Wilke, S. Richly, S. Gotz, C. Piechnick, and U. Assmann. 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*, pages 134–141, Aug 2013.

[87] G. Xu, M. Arnold, N. Mitchell, A. Rountev, and G. Sevitsky. Go with the flow: Profiling copies to find runtime bloat. In *Proceedings of the 2009 ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '09, pages 419–430, 2009.

[88] C. Zhang, A. Hindle, and D. M. Germán. The impact of user choice on energy consumption. *IEEE Software*, 31(3):69–75, 2014.

[89] Y. Zhang, G. Huang, X. Liu, W. Zhang, H. Mei, and S. Yang. Refactoring android java code for on-demand computation offloading. In *Proceedings of the ACM International Conference on Object Oriented Programming Systems Languages and Applications*, OOPSLA '12, pages 233–248, 2012.

[90] Z. Zhuang, K.-H. Kim, and J. P. Singh. Improving energy efficiency of location sensing on smartphones. In *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*, MobiSys '10, pages 315–330, 2010.