Energy Efficiency: A New Concern for Application Software Developers

Gustavo Pinto
Federal Institute of Pará
gustavo.pinto@ifpa.edu.br

Fernando Castor
Federal University of Pernambuco
castor@cin.ufpe.br

ABSTRACT

Energy efficiency is a problem that must be addressed at all levels of the software stack. However, developing energy-efficient software is not an easy task. In this paper we argue that this is mostly due to two main problems: the lack of knowledge and the lack of tools. These problems prevent software developers from identifying, refactoring, fixing, and removing energy consumption hotspots. We review how current research in the area of software engineering is tackling these two problems. Furthermore, based on an investigation on the problems faced by energy-aware developers, we discuss avenues for future research in the area.

Keywords

Software Energy Consumption, Lack of Knowledge, Lack of Tools.

1. INTRODUCTION

The prevalence and ubiquity of mobile computing platforms such as smartphones, tablets, smartwatches, and smartglasses, changed the way people use and interact with software. In particular, these platforms share a common yet challenging requirement: they are battery-driven. As users interact with them, they tend to be less available, since even simple well-optimized operations (*e.g.*, texting a friend) consume energy. At the same time, wasteful, poorly-optimized software can deplete a device's battery much faster than necessary. Heavy resource usage has been shown to be one of the reasons leading to poor apps reviews in online app stores [19].

This concern, however, pertains not only to mobile platforms. Big players of the software industry are also reaching the same conclusion, as stated in one of the very few energy efficient software development guides: "*Even small inefficiencies in apps add up across the system, significantly affecting battery life, performance, responsiveness, and temperature*"[1]. Corporations that maintain data centers struggle with soaring energy costs. These costs can be attributed in part to overprovisioning with severs constantly operating under their maximum capacity (*e.g.,* America's data centers are wasting huge amount of energy [13]), and to the developers of the apps running on these data centers generally not taking energy into consideration [33].

Unfortunately, during the last decades, little attention has been placed on creating techniques, tools, and processes to empower software developers to better understand and use energy resources. As a consequence, software developers still lack textbooks, guidelines, courses, and tools to refer to when dealing with energy consumption issues [33, 42]. Moreover, most of the research that connects computing and energy efficiency has concentrated on the lower levels of the hardware and software stack. However, recent studies show that these lower level solutions do not capture the whole picture [2, 9, 22], when it comes to energy consumption. Although software systems do not consume energy themselves, they affect hardware utilization, leading to indirect energy consumption.

1.1 How is software related to energy consumption?

Energy consumption E is an accumulation of power dissipation P over time t, that is, $E = P \times t$. Power P is measured in watts, whereas energy E is measured in joules. As an example, if one operation takes 10 seconds to complete and dissipates 5 watts, it consumes 50 joules of energy. In particular, when taking about software energy consumption, one should pay attention to:

- a given software under execution,
- on a given hardware platform,
- on a given context,
- during a given time.

To understand the importance of a *hardware platform*, consider an application that uses network. Any commodity smartphone nowadays supports, at least, WiFi, 3G, and 4G. A recent study observed that 3G can consume about 1.7x more energy than WiFi, whereas 4G can consume about 1.3x

[1]https://developer.apple.com/library/content/documentation/Performance/Conceptual/power_efficiency_guidelines_osx/index.html#//apple_ref/doc/uid/TP40013929

DOI: 10.1145/1235

more energy than 3G, while performing the same task, on the same hardware platform [20].

Context also plays a key role, since the way software is built and used has a critical influence on energy consumption. For instance, software can also stress energy consumption on CPUs, when performing CPU-intensive computations [43], on DRAMs, when performing random accesses to data structures [31], on networks, when running several HTTP requests [25, 9], and on displays, when using lighter backgrounds [26, 29] or playing videos.

Finally, *time* plays a key role in this equation, A common misconception among developers is that reducing execution time also reduces energy consumption [42, 33], the t of the equation. However, chances are that this reduction in execution time might increase the number of CPU cycles (*e.g.,* using multi-core CPUs) and, therefore, the number of context switches. This, in turn, might increase the P of the equation, impacting the resulting energy consumption.

1.2 Software engineering meets energy consumption

While the strategy of leaving the energy consumption optimization problem to the lower-level layers has been successful, recent studies show that even better energy savings can be achieved by empowering and encouraging software developers to participate in the process [20, 31, 39, 9]. However, the application level, which is the focus of most mainstream software being developed these days, has been the target of few studies.

This lack of studies was observed in a recent paper [45], where the authors surveyed the last 10 years of top software engineering venues, and found only 20 research papers that have "power" or "energy" on their titles or abstracts. More interestingly, however, the authors observed that none of them were published before 2012. In 2012, 3 papers were published, whereas 6 papers were published in 2013 and 11 papers in 2014. That shows the emerging character of the field.

The need for studies that focus on the higher levels of the software stack is important from at least two important perspectives:

Software engineer's perspective. Battery usage is a key factor for adopting and evaluating mobile applications. Users of an energy-inefficient app might review it badly, encouraging other users to not use it. This can negatively impact the app's revenue.

End user's perspective. The last mile in energy efficiency comes from the choices of end-users. To make better choices, and further minimize energy consumption, end users should be aware of the different energy characteristics of software applications that serve for the same purpose.

This paper. This paper is a review of the most prominent software engineering approaches for writing, maintaining, and evolving energy-efficient software applications. We organize the contributions according to the Guide to the Software Engineering Body of Knowledge (SWEBOK) [1], a common practice in software engineering studies (*e.g.,* [36]). When conducing such review, we found that the literature does not cover well certain areas of the SWEBOK. For these cases, we share our visions of possible research avenues that energy-aware researchers can follow to reduce this gap.

The rest of the paper is organized as follows: Section 2 unveils the perceptions of mobile developers when dealing with energy consumption issues, scratching their problems and possible solutions. Section 3 acknowledges that most of the energy related problems, in fact, can be reduced in two main problems: the lack of knowledge and the lack of tools. Section 4 surveys recent literature to understand how software engineering researchers are tackling these two problems. Section 5 concludes this work.

2. A FORMATIVE STUDY

Energy consumption issues are now knocking on the door of application software developers. To shed light on this matter, similarly to Pang *et al.* [39], we conducted a survey with software developers to understand their perceptions about software energy consumption issues. Differently from this previous paper, which surveyed a wide range of software developers, our target population is more focused and consists of 62 software developers who have performed at least one commit to a mobile open-source application.

Among the respondents, 68.75% have more than 8 years of software development experience, 57.81% have more than 2 years of mobile development experience, and 77.41% have more than 2 years of open-source development experience. The majority of them (57.8%) are source code contributors or project owners (35.9%). More interestingly, 70.31% of the respondents agree that energy consumption could be an issue in their mobile applications. Also, 37 respondents have already faced energy-related problems, as a respondent said: "*We have a limited energy envelope for the whole system and we must make sure even our power hungry components don't cause the system to go beyond this limit*". Also, some respondents are aware that energy inefficiencies can impact on app popularity and, therefore, revenue: "*Users will leave bad reviews if you drain the battery*".

When asked if they found the root cause for the energy-related problems, 50% of the respondents did not answer. For those who answered, background activities, GPS, and unnecessary resource usage are among the most recurring answers. Interestingly, these problems were also observed in other studies [42, 33]. However, 31.81% of the respondents did not observe any significant improvement in energy consumption after applying their solutions. For those who observed an improvement, only 5 of them made use of specialized tools. The majority of them have the *perception* of an improvement, *e.g.*: "*The battery is lasting longer*", "*Less heat from device*", or "*I really do not measure before and after. It's just a perception*". When we asked where they find reliable information about what solutions can be used to save energy, 7 of them refer to the official documentation, 5 of them use StackOverflow, and 5 use other channels (blogs, youtube, open-source repositories). Unfortunately, the solutions described in such sources of documentation often are not supported by empirical evidence [42, 35]. To make the matter worse, two respondents rely on "Trial and error", which is far from being accurate.

Moreover, 67% of the respondents said that energy-related features are "important" or "very important" to have in well-known IDEs. Only 8 of the overall respondents have actually used software energy consumption tools. Respondents said that the most important energy-related features to have

in well-known IDEs are profiling tools (16 answers), varying from CPU, network, method, wakelocks, thread, and live profile. Indeed, one respondent synthesize that well-known IDEs, such as Android Studio, lack these features: "*Android Studio needs a good energy profiler to check the Android power consumption from all power consumers (radios, CPU, memory, storage, everything).*" These results not only corroborate with the findings of Pang *et al.* [39], but also reinforce that application-level energy management is in high demand among application software developers, although better support is urgently needed.

We also asked five leading researchers in the area of Software Energy Consumption what are, on their opinions, the most significant contributions and biggest open challenges in this area. All the researchers agreed that tool support is still lacking when it comes to energy measurement, reengineering, refactoring and other related activities. Even though there is a recent interest from IDE builders to provide an energy consumption perspective of the software systems under development[2], this finding suggests that there is much to do still.

3. THE ENERGY RELATED PROBLEMS

As observed in our formative study, software developers currently have to rely on Q&A websites, blog posts, or youtube videos when trying to optimize energy consumption, which are anecdotal, not supported by empirical evidence, or even incorrect [21, 33]. The consequence of the lack of appropriated textbooks, guidelines, and cookbooks for green software development is the *Lack of knowledge* on how to write, maintain, and evolve energy-efficient software applications. Furthermore, our respondents also mentioned that they believe that energy-related features are very important to have in well-known IDEs. In particular, energy profiling techniques can be very helpful. This lack of energy-related features incurs in the *Lack of tools* to find, refactor, and fix energy-inefficient code.

The lack of knowledge and the lack of tools to write energy-efficient software is also discussed in the recent literature. For instance, Pinto *et al.* [42] noticed that a common *misconception* is to confuse concepts such as "power" and "energy". Manotas *et al.* [33] observed that developers believe in *panaceas*, that is, solutions that are presented as universal but, in fact, only work in specific contexts. For instance, while one developer suggested "offloading computation to the cloud" as a way to improve energy consumption, another developer mentioned "decreased radio use increases battery life". As a result, developers should consider the underlying thresholds to take proper advantage of each solution. These are examples of lack of knowledge. To further complicate matters, optimizing performance does not always help to save energy [22, 23, 28, 43]. Thus, the extensive performance textbooks and guidelines are not always useful.

The aforementioned lack of knowledge is intrinsically connected the lack of tools. Moura *et al.* [35] observed that energy-aware developers often employ low-level solutions that sometimes result in hard-to-detect *correctness* problems. The following commit message provides an example of a correctness problem: "*Disable Auto Power Saving when resetting the modem. This can cause several bugs with serial communication*"[3]. High-level energy saving tools might be useful in mitigating this problem. In addition, Pang *et al.* [39] found that 88% of the respondents of their survey do not know what tool they can use to measure the energy consumption of their software. These are examples of lack of tools. Although software energy consumption tools do exist, they have yet-to-be-addressed limitations:

- They require an in-depth knowledge of low-level implementation details and programmers under time pressure have little chance to learn how to use them;
- They do not provide direct guidance on energy optimization, *i.e.*, bridging the gap between understanding where energy is consumed and understanding how the code can be modified in order to reduce energy consumption.

The next section discusses how current software engineering research is addressing these two key problems.

4. ENERGY RELATED SOLUTIONS

Since there is no single solution for conserving energy, we organize the contributions in terms of the topics of the SWEBOK [1], a common practice in software engineering studies (*e.g.,* [36]). Although energy consumption can be related to any software engineering topic, we chose to focus only on topics directly related to software coding, since (1) it is one of the main activities of software developers, and (2) it is the target of most of the recent research contributions. Therefore, we do not cover the following topics: software configuration management, software engineering management, software engineering process, and software requirements.

4.1 Software Tools & Methods

We organize our discussion of software engineering tools and methods in terms of enhancement methods, measurement tools, and static analysis tools.

Enhancement methods. These methods refer to energy saving techniques that developers can use, even though they have no prior knowledge of the application domain. For instance, software developers often leverage modern CPUs to dynamically change their operating frequencies, thus reducing power dissipation [35]. However, when applying this technique, software developers should use low-level system interfaces, which are error-prone and platform dependent. Notwithstanding, blindly downscaling CPU frequency might increase energy consumption while reducing performance [31, 17]. This is an important example of the *lack of tools*. To mitigate this problem, novel approaches are based on dynamic adaptation through an energy profiler module, energy policies, and energy adaptation APIs [46, 47]. The energy profiler module can recognize the system states and estimate the energy potentially demanded by an application.

Another example is method reallocation [10], which refers to the analysis of a software system considering all the levels of the stack (*e.g.,* kernel, library, and source code level), and reorganizing the classes and methods through the levels of the stack, in a way in which they can be placed in the level where the energy consumption is minimal. As a limitation, this technique can be utilized only if the operating system

[2]https://developer.apple.com/library/ios/documentation/Performance/Conceptual/EnergyGuide-iOS/MonitorEnergyWithXcode.html

[3]https://github.com/alobo/SerialGSM/commit/c616b950

and the software development environment allow application software developers to go through the different levels (*e.g.*, from source code level to kernel level). In a similar strategy, cloud offloading [20] is a technique in which heavy computations are sent to a remote computer; after the remote execution the result is sent back to the local machine. This approach aims to re-organize the implementation of the system at the source code level, thus saving energy by minimizing processing. Interestingly, when we asked if the respondents found any solution to overcome the energy-related problems, one of the respondents said that "*Offload intensive work to workers in the cloud.*" However, this technique is only effective if the savings can compensate the extra energy toll required to send a computation through a network. Therefore, trade-offs exist and, as we have discussed in Section 1.1, different components have different energy usage characteristics.

Measurement tools. Some measurement tools include methods that use data collected from different system interfaces to assess the energy consumption at the application level. One example is the Running Average Power Limit (RAPL). This module enables architectures monitor energy consumption and store it in Machine-Specific Registers (MSRs[4]). Several energy-consumption studies are based on this module (e.g., [27, 31, 44]). With such techniques, it is possible to profile a system and perform an analysis about, for instance, what are the system calls that have a major contribution to power dissipation [10, 31]. System calls, in particular, are being actively used for predicting and estimating energy consumption of a software system [3, 2, 8].

Other tools leverage energy models. This strategy utilizes a model developed by physically measuring the energy consumption of a device [15, 20, 23]. Energy models have a higher level of confidence only when approximating the energy consumption on the hardware based on which the model was created. Other hardware architectures can only consider the model as a rough estimation.

Although there are already some software tools for energy measurement (*e.g.,* [23, 15]), such tools have well-known drawbacks. First, energy measurement tools may pay an additional overhead on energy consumption, mostly due to the sampling mechanism. Data acquisition (*i.e.,* sampling) is the result of the process of acquiring information from the surrounding environment, processing the data, and sending them to another collection point to be consumed. Therefore, sampling techniques might impact energy consumption. This poses a challenge, since recent studies provided evidence that a high sampling rate is necessary to obtain reliable information [48]. Even though recent efforts have mitigated this problem [31], software-based approaches are often regarded as less rigorous than hardware-based ones.

Second, hardware- and software-based approaches often do not provide the granularity level that application software developers are interested in [42, 33]. For instance, there is no tool support to measure energy consumption per thread per system module. It is difficult to link the energy measurements across the running threads with fine-grained events that happen during program execution, such as method calls. To make matters worse, the tail energy – *i.e.,* a component that keeps a high power state long beyond the end of its utilization [23] – should be taken into consideration, even in the presence of context switches. As a result, there is a mismatch between the noise introduced by coarse-grained measurements and the tiny energy impact of methods calls. Still, in our survey, 11 respondents mentioned that measurement tools are among the most important energy-related features to have available in well-known IDEs.

Static Analysis tools. One of the main challenges of software energy consumption research is to bring analysis to the static level. Currently, software energy consumption instrumentation can only be conducted at runtime. This approach has several limitations, such as sophisticated (and expensive) hardware equipments [43] or applicability only to specific hardware configurations [31]. This fact has the potential of limiting the usability of software energy consumption tools.

Although there are few studies in this direction (*e.g.,* a static analysis technique for estimating the energy consumption of embedded programs [30]), these tools (1) often combine static analysis with dynamic analysis techniques (*e.g.,* [23]), which makes them hardware-dependent, and (2) do not exhibit maturity, nor the breadth of scope necessary for use in real software development. One of the main challenges for deriving static analysis tools for energy consumption is the need for a body of knowledge on how language constructs and design decisions impact energy consumption. Due to the emerging character of the field [45], we believe that new empirical energy consumption studies will be conducted in the following years, which in turn will help researchers to create such static analysis tools.

4.2 Software Maintenance

We organize our discussion of software maintenance in terms of refactoring, reengineering, and visualization.

Refactoring. Refactoring tools can take advantage of cutting-edge research and incorporate such knowledge into refactoring engines. However, as a researcher respondent said, "*There is a lot of work showing how different programming styles, techniques, structures influence the consumption, but there is still no real cataloging [..] based on these concrete software practices*". Although researchers have been speculating on this subject during the last years [12], to the best of our knowledge, there is only one study that deals with the problem of introducing novel refactoring tools for improving the energy efficiency of a software system [5]. In this study, the authors present a set of energy-efficiency guidelines that are specifically tailored for Android apps, such as location updates and resource leaks. When applied, the authors observed improvements of up to 29% of the overall energy consumption.

This lack of contributions is not related to a lack of opportunities, though. As mentioned before, there are several opportunities for application software developers to save energy by refactoring existing systems [45]. As two examples, Pinto *et al.* [44] observed that just updating from `Hashtable` to `ConcurrentHashMap` in a Java program can yield a 3.5x energy saving. In particular, this transformation yields a 1.4x and a 9.2x energy saving in CPU and DRAM, respectively. As another example, Pathak *et al.* [40] observed that I/O operations consume more energy partly because of the tail energy phenomenon. According to the authors, this tail energy leak can be mitigated by bundling I/O operations together. These results have a clear implication: Tools to

[4]https://01.org/msr-tools/overview

aid developers in quickly refactoring programs can be useful if energy is important.

Reengineering. Differently from Refactoring tools, which are more localized, reengineering efforts can be broader in scope and have a systemwide impact on the structure of an application. As mentioned, method reallocation [10] and method offloading [20] are two common strategies to implement reenginering energy-aware methods. This is corroborated by the findings of Othman *et al.*, which found that up to 20% energy savings can be achieved by uploading tasks from mobile devices to fixed servers [38]. Using a different strategy, Manotas *et al.* [34] proposed SEEDS, a general decision-making framework for optimizing software energy consumption. The SEEDS framework can identify energy-inefficient uses of Java collections, and automate the process of selecting more efficient ones. More recently, search-based software engineering approaches were used to reengineer a software system in order to minimize energy usage [6], yielding an energy reduction of up to 25%. These approaches mitigate the problem of *lack of tools*.

Visualization. Visualization techniques are useful to support the understanding of software systems in order to discover and analyze their anomalies. Li *et al.* [23] proposed a technique that overlays energy consumption information with application's source code. This technique colors different amount of energy consumed in a given line of code — blue lines describe low energy consumption whereas red lines indicate high energy consumption. This visualization technique is fine-grained and works at the source code level. On the other hand, the study of Couto *et al.* [11] focuses on a coarser granularity: It identifies the energy consumption per method, and aggregates this energy in terms of classes, packages, and the whole software system. The result is presented in a sunburst diagram, which allows developers to easily and quickly identify the most energy inefficient parts of the code. These studies combine art and technology as a way to mitigate energy consumption. With a better understanding of the whole program energy behavior, such visualization techniques can be useful to mitigate both *lack of knowledge* and *lack of tools*.

4.3 Software Design & Construction

Researchers have been studying different strategies for designing and constructing energy-efficient software [14, 22, 26, 28, 40]. These studies focus on understanding how a particular programming practice or design implementation might impact on energy consumption. To gain further confidence in the results, these studies often analyze dozens (*e.g.,* [17]), or even hundreds (*e.g.,* [22]), of software applications, and they mitigate the lack of knowledge by providing high-level guidelines for designing energy-efficient software. We organize our discussions of software design & construction in terms of mobile, network, data structures, and parallel programming techniques.

Mobile development. Linares-Vasquez *et al.* [28] investigated API calls that might cause high energy consumption. For example, they observed that the method `Activity.findViewById`, which is commonly used, is one of the most energy-consuming among the Android APIs. Similarly, Malik *et al.* [32] found that the `BroadcastReceiver` and the `Location` APIs are the most often discussed among Android energy questions on StackOverflow. Furthermore, since the display is one of the smartphone' most energy-intensive components [7], Li *et al.* [26] discussed how to improve energy efficiency by favoring darker colors instead of lighter ones for smartphones with OLED displays. Using a search-based multi-objective approach, Linares-Vasquez *et al.* [29] automatically optimized energy consumption and contrast, while using consistent colors with respect to the original color palette.

Network usage. Li *et al.* [22] analyzed more than 400 real-world Android apps, and found that an HTTP request is the most energy-consuming operation of the network. In a follow up study, the same authors observed that bulking HTTP requests is a good practice for energy saving [25]. Also regarding HTTP usage, Chowdhury *et al.* [9] observed that HTTP/2 is more energy efficient than its predecessor, HTTP/1.1, for networks with higher Round Trip time (RTTs). Since most mobile apps use network [22], we expect more contributions on this direction. Besides of bulking requests, researchers can evaluate the benefits of, for instance, reducing transactions, compressing data, and appropriately handling errors to energy conservation.

Data Structures. The energy behavior of different data structures, one of the building blocks of computer programming, have been extensively studied in the last few years [44, 34, 14, 27]. Hasan and colleagues [14] investigated data structures grouped with three interfaces (List, Set, and Map). Among the findings, they found that the position where an element is inserted in a list can greatly impact energy consumption. Pinto *et al.* [44] studied the same group of interfaces, but focused on thread-safe data structures. They also observed that using a newer version of a thread-safe data structure can yield a 2.19x energy savings when compared to the old associative implementation. Lima *et al.* [27] studied the energy consumption of data structures in concurrent functional programs. Although they found that there is no clear universal winner, in certain circumstances, choosing one data sharing primitive (MVar) over another (TMVar) can yield 60% energy savings.

Parallel Programming. Parallel programming techniques have also been the subject of several studies. Pinto *et al.* [43] observed that a high-level, work-stealing parallel framework is more energy-friendly when performing fine-grained CPU-intensive computations than a thread-based implementation. Still, Ribic and Liu proposed a set of runtime systems for improving the energy efficiency of fine-grained CPU-intensive computations [46, 47]. To better leverage the energy saving achieved with these studies, we believe they can be integrated with well-known runtime systems, such as the Java Virtual Machine (JVM). If so, the whole chain of programming languages, software systems, and end-users that rely on the JVM can benefit from the findings of these studies.

Although these studies provide a comprehensive set of findings with practical and timely implications and can be useful to mitigate the problem of lack of knowledge, they are far from covering the whole spectrum of programming language constructs and libraries.

4.4 Software Quality & Testing

Here we organize our discussions in terms of software testing and software debugging techniques.

Software Testing. Although there are several studies aimed at characterizing energy bugs (*e.g.,* [41]), there are relatively few studies that propose new energy-aware testing

techniques [24, 16, 18]. Ding and colleagues [24] presented an energy-efficient testing suite minimization technique that can be used to perform post-deployment testing on embedded systems. Results suggest that the approach can promote a reduction of over 95% of the energy consumed by the original test suite. Similarly, Jabbarvand *et al.* [16] present another test suite minimization approach, but focusing on Android apps. The authors reported a reduction of, on average, 84%, while maintaining the effectiveness for revealing bugs. Kan [18] proposes a similar approach: use DVFS to scale frequency down when running the test suites. Although some researchers argued that DVFS techniques can lead to increased energy consumption and performance loss [31], the authors showed that important energy savings can be achieved. Banerjee *et al.* [4] proposed a technique that generates test inputs that are likely to capture energy bugs. This technique focuses on creating tests that use I/O components, which are one of the primary sources of energy consumption in a smartphone [7, 40].

Followed by these promising initial results, we believe that new testing techniques will be evaluated in terms of energy consumption. At best, energy testing will become a research area. Several possible areas of interest can be envisioned. One of them is what we call "green assertions", that is, the possibility to define an energy budget where the test case asserts whether the computation satisfies that budget. The test fails if the energy consumed is greater than the suggested budget. For instance, the code snippet `double maxEnergy = 200; assertTrue(render(), expected, maxEnergy);` defines that the `render()` method should consume, at most, 200 Joules. This technique can be further improved to cover additional hardware characteristics, for instance, asserting whether the computation consumes 100 Joules due to network communication or 50 Joules due to the CPU.

Software Debugging. Practitioners commonly use debugging tools to catch bugs in program formulation. However, debugging an energy-inefficient piece of code is more challenging than traditional debugging because such inefficiencies depend on the running contextual sensitive scenario, such as the state of the hardware device. In this regard, Banerjee and colleagues [5] propose a framework for debugging energy consumption related *field failures* in mobile apps. The authors found that tool support could localize energy bugs in a short amount of time, even for non-trivial Android apps. The authors observed energy savings of up to 29% after patching the energy bug. Pathak *et al.* [40] propose *eprof*, a fine-grained profiling energy consumption technique for applications running on smartphones. Similar to the work of Banerjee and colleagues [4], Pathak *et al.* focus on understanding and monitoring system calls that are related to I/O operations. As a results, they found that most of the energy consumed in free apps are related to third-party advertisement modules (which can be responsible for up to 75% of the overall energy consumed by an app). Using a collaborative black-box approach, Oliner *et al* [37] propose a method for diagnosing anomalies, estimating their severity, and identifing the device features that lead to the anomaly. Using feedback received by the proposed tool, end users improved their battery life by 21%.

We believe that debugging tools will have the capability of inspecting the energy consumption of fine-granularity program constructs during runtime, as well as their common ability to identify which value was attributed to a given variable. Debugging tools can go further and highlight the CPU-intensive lines of code, or the memory-intensive methods, in a way that developers can refactor them in an energy-savvy manner. Novel energy-related testing and debugging tools can mitigate the lack of tools.

5. CONCLUSIONS

Energy consumption is a ubiquitous problem and the years to come will require developers to be even more aware of it. However, developers currently do not fully understand how to write, maintain, and evolve energy-efficient software systems. In this study we suggest that this is primarily due to two problems: the *lack of knowledge* and the *lack of tools*. With these problems in mind, this paper reviews most of the recent energy-related contributions in the software engineering community. We discuss how software energy consumption research is evolving to mitigate these two problems and, when appropriate, we highlight key research gaps that need better attention.

6. REFERENCES

[1] A. Abran, P. Bourque, R. Dupuis, and J. W. Moore, editors. *Guide to the Software Engineering Body of Knowledge - SWEBOK*. 2001.

[2] K. Aggarwal, A. Hindle, and E. Stroulia. GreenAdvisor: A tool for analyzing the impact of software evolution on energy consumption. In *ICSME*, pages 311–320, Sept 2015.

[3] K. Aggarwal, C. Zhang, J. C. Campbell, A. Hindle, and E. Stroulia. The power of system call traces: Predicting the software energy consumption impact of changes. In *CASCON*, pages 219–233, 2014.

[4] A. Banerjee, L. K. Chong, S. Chattopadhyay, and A. Roychoudhury. Detecting energy bugs and hotspots in mobile apps. In *ESEC/FSE*, pages 588–598, 2014.

[5] A. Banerjee and A. Roychoudhury. Automated re-factoring of android apps to enhance energy-efficiency. In *MOBILESoft*, pages 139–150, 2016.

[6] B. R. Bruce, J. Petke, and M. Harman. Reducing energy consumption using genetic improvement. In *GECCO*, pages 1327–1334, 2015.

[7] A. Carroll and G. Heiser. An analysis of power consumption in a smartphone. In *USENIX*, 2010.

[8] S. A. Chowdhury and A. Hindle. Greenoracle: estimating software energy consumption with energy measurement corpora. In *MSR*, pages 49–60, 2016.

[9] S. A. Chowdhury, V. Sapra, and A. Hindle. Client-side energy efficiency of HTTP/2 for web and mobile app developers. In *SANER*, pages 529–540, 2016.

[10] L. Corral, A. B. Georgiev, A. Sillitti, and G. Succi. Method reallocation to reduce energy consumption: an implementation in android OS. In *SAC*, pages 1213–1218, 2014.

[11] M. Couto, T. Carção, J. Cunha, J. P. Fernandes, and J. Saraiva. Detecting anomalous energy consumption in android applications. In *SBLP*, pages 77–91, 2014.

[12] S. Fraser, E. Murphy-Hill, W. Wild, J. Yoder, and B. Q. Zhu. Going green with refactoring: Sustaining the "worldwide virtual machine". In *OOPSLA*, pages 171–174, 2011.

[13] E. Gelenbe and Y. Caseau. The impact of information technology on energy consumption and carbon emissions. *Ubiquity*, 2015(June):1:1–1:15, June 2015.

[14] S. Hasan, Z. King, M. Hafiz, M. Sayagh, B. Adams, and A. Hindle. Energy profiles of java collections classes. In *ICSE*, pages 225–236, 2016.

[15] 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 *MSR*, pages 12–21, 2014.

[16] R. Jabbarvand, A. Sadeghi, H. Bagheri, and S. Malek. Energy-aware test-suite minimization for android apps. In *ISSTA*, pages 425–436, 2016.

[17] M. Kambadur and M. A. Kim. An experimental survey of energy management across the stack. In *OOPSLA*, pages 329–344, 2014.

[18] E. Y. Y. Kan. Energy efficiency in testing and regression testing – a comparison of dvfs techniques. In *Proceedings of the 2013 13th International Conference on Quality Software*, QSIC '13, pages 280–283, 2013.

[19] H. Khalid, E. Shihab, M. Nagappan, and A. E. Hassan. What do mobile app users complain about? *IEEE Software*, 32(3):70–77, 2015.

[20] Y. Kwon and E. Tilevich. Reducing the energy consumption of mobile applications behind the scenes. In *ICSM*, pages 170–179, 2013.

[21] 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.

[22] D. Li, S. Hao, J. Gui, and W. G. J. Halfond. An empirical study of the energy consumption of android applications. In *ICSME*, pages 121–130, 2014.

[23] D. Li, S. Hao, W. G. J. Halfond, and R. Govindan. Calculating source line level energy information for android applications. In *ISSTA*, pages 78–89, 2013.

[24] D. Li, Y. Jin, C. Sahin, J. Clause, and W. G. J. Halfond. Integrated energy-directed test suite optimization. In *ISSTA*, pages 339–350, 2014.

[25] D. Li, Y. Lyu, J. Gui, and W. G. J. Halfond. Automated energy optimization of http requests for mobile applications. In *ICSE*, pages 249–260, 2016.

[26] D. Li, A. H. Tran, and W. G. J. Halfond. Making web applications more energy efficient for oled smartphones. In *ICSE*, pages 527–538, 2014.

[27] L. G. Lima, F. Soares-Neto, P. Lieuthier, F. Castor, G. Melfe, and J. P. Fernandes. Haskell in green land: Analyzing the energy behavior of a purely functional language. In *SANER*, pages 517–528, 2016.

[28] 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 *MSR*, pages 2–11, 2014.

[29] M. Linares-Vásquez, G. Bavota, C. E. B. Cárdenas, R. Oliveto, M. Di Penta, and D. Poshyvanyk. Optimizing energy consumption of GUIs in android apps: A multi-objective approach. In *ESEC/FSE*, pages 143–154, 2015.

[30] U. Liqat, S. Kerrison, A. Serrano, K. Georgiou, P. López-García, N. Grech, M. V. Hermenegildo, and K. Eder. Energy consumption analysis of programs based on XMOS isa-level models. In *23rd International Symposium on Logic-Based Program Synthesis and Transformation LOPSTR*, pages 72–90, 2013.

[31] K. Liu, G. Pinto, and Y. D. Liu. Data-oriented characterization of application-level energy optimization. In *FASE*, 2015.

[32] H. Malik, P. Zhao, and M. W. Godfrey. Going green: An exploratory analysis of energy-related questions. In *MSR*, pages 418–421, 2015.

[33] 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 *ICSE*, pages 237–248, 2016.

[34] I. Manotas, L. Pollock, and J. Clause. Seeds: A software engineer's energy-optimization decision support framework. In *ICSE*, pages 503–514, 2014.

[35] I. Moura, G. Pinto, F. Ebert, and F. Castor. Mining energy-aware commits. In *MSR*, pages 56–67, 2015.

[36] E. Murphy-Hill, T. Zimmermann, and N. Nagappan. Cowboys, ankle sprains, and keepers of quality: How is video game development different from software development? In *ICSE*, pages 1–11, 2014.

[37] A. J. Oliner, A. P. Iyer, I. Stoica, E. Lagerspetz, and S. Tarkoma. Carat: Collaborative energy diagnosis for mobile devices. In *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems*, SenSys '13, pages 10:1–10:14, 2013.

[38] M. Othman and S. Hailes. Power conservation strategy for mobile computers using load sharing. *SIGMOBILE Mob. Comput. Commun. Rev.*, 2(1):44–51, Jan. 1998.

[39] C. Pang, A. Hindle, B. Adams, and A. E. Hassan. What do programmers know about software energy consumption? *IEEE Software*, 33(3):83–89, 2016.

[40] 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 *EuroSys*, pages 29–42, 2012.

[41] 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 *MobiSys*, pages 267–280, 2012.

[42] G. Pinto, F. Castor, and Y. D. Liu. Mining questions about software energy consumption. In *MSR*, pages 22–31, 2014.

[43] G. Pinto, F. Castor, and Y. D. Liu. Understanding energy behaviors of thread management constructs. In *OOPSLA*, pages 345–360, 2014.

[44] G. Pinto, K. Liu, F. Castor, and Y. D. Liu. A comprehensive study on the energy efficiency of java thread-safe collections. In *ICSME*, 2016.

[45] G. Pinto, F. Soares-Neto, and F. Castor. Refactoring for energy efficiency: A reflection on the state of the art. In *GREENS*, 2015.

[46] H. Ribic and Y. D. Liu. Energy-efficient work-stealing language runtimes. In *ASPLOS*, pages 513–528, 2014.

[47] H. Ribic and Y. D. Liu. Aequitas: Coordinated energy management across parallel applications. In *Proceedings of the 2016 International Conference on Supercomputing*, ICS '16, pages 4:1–4:12, 2016.

[48] R. Saborido, V. Arnaoudova, G. Beltrame, F. Khomh, and G. Antoniol. On the impact of sampling frequency on software energy measurements. *PeerJ PrePrints*, 3:e1219, 2015.