Green Software Engineering: The Curse of Methodology

Abram Hindle
Department of Computing Science
University of Alberta
Edmonton, Alberta, Canada
abram.hindle@ualberta.ca

***Abstract*—Computer Science often seems distant from its natural science cousins, especially software engineering which feels closer to sociology and psychology than to physics. Physical measurements are often rare in software engineering, except in a few niches. One such important niche is that of software energy consumption, green mining, green IT, and sustainable computing, which all fall under the umbrella of green software engineering.**

With the physical measurement of energy consumption comes all of the limitations of measurement and experimentation that exist in the natural sciences and engineering. Issues abound, from attribution of energy use, isolation of components, to replicable experiments. These get further complicated by cloud computing whereby systems are virtualized and attribution of resource usage is a serious issue.

Thus in this work we discuss the current state of software energy consumption, and where will it go.

I. Introduction

Fundamentally all computation comes at a cost. It is of no surprise that electrical measurements of work correspond to computation as well. With the availability of smart phones, heavily parallelizable clusters, cloud-mad data centers, software and energy interact more readily than ever before. Energy comes at a cost to generate, to deliver, and to store. Delivery requires infrastructure, storage requires materials for batteries, and the by-product of energy consumption, heat, requires cooling. While hardware primarily consumes energy, it can only be as efficient as the software that commands it.

Software's interaction with energy is split among many contexts. Two important contexts are mobile applications and software services hosted within data centers. Other contexts include embedded sensors, the desktop, etc.

a) Data Centers: are limited by energy in terms of power limits of rack power systems as well as cooling. Typically energy accounts for 50% to 100% of the cost of purchased equipment over the equipment's lifetime [1]. Racks have limited energy hookups. Only so many power heavy units may be powered. Furthermore for every unit put in, the wasted heat must be addressed. A data center with poor cooling will pay even more in energy consumption due to the excessive use of the cooling system of each hosted server. Typically services offered by a data-center are software services and in many cases the services are dynamically provisioned on virtual machines or containers.

b) Mobile: applications are slightly different, their availability is affected by the availability of energy. Without battery energy left, no application could survive. The energy used by mobile devices is negligible, usually less than CFL light bulb while charging – yet the batteries are composed of potentially toxic and costly materials. Reducing mobile energy use leads to longer battery lives, combined with reduced battery replacement, and more availability for the end-user.

c) Embedded/Wireless Sensors: typically run on very low power computers and sensors that communicate information infrequently. The availability and reliability of these systems are directly affected by both hardware and software design.

Just these contexts alone motivate the importance of energy efficiency and the study of software energy consumption. Software consumption is inherently multidisciplinary as different engineers serve, rely on, and cater to other engineers. Hardware creators can only do so much until it becomes the responsibility of the software developer to develop software in an energy efficient and sustainable manner. Software engineering researchers have noticed this problem and have taken up the torch, thus accepting their responsibility for some of the energy consumption costs of applications.

Thus the audience of this paper is primarily software engineering researchers and developers, but the impact of this field is far more broad. End-users are affected by software energy consumption by the effect of desktop and laptop energy usage on their energy bills. The availability of end-user mobile devices is greatly affected by software energy consumption, whereby an inefficient program can practically leave some users stranded without the ability to communicate. Electrical engineers and computer engineers are affected by software energy consumption as they have to work hand in hand with software engineers to produce hardware that enables general purpose computation and yet still provides methods of achieving energy efficiency. If hardware designers are aware of the constraints of software developers they can address the needs of software developers and end-users as well. Managers and other stakeholders are affected by software energy consumption because they have to budget for data-center costs, and poor energy consumption will affect software sales. Furthermore as carbon taxes are being levied, energy consumption becomes an important budget item.

In this paper I introduce software energy consumption, discuss past and present challenges, works, and issues relevant to software engineering communities. Then I discuss my predictions for the future of software energy consumption, and where such research will go in the next decade.

II. Background

Energy is the effort expended to complete a task. For electricity we typically use joules (J), the energy unit of the International System (SI), to indicate the energy that a task takes. Power is the instantaneous rate of energy consumption or the work that is being done. Typically power is measured in Watts, which is the instantaneous amount of work done. The multiplication of power by time is energy, or energy is the integration of power over time. Sometimes energy is measured as watt-hours (e.g., 1Wh) by electricity providers where $1kWh = 3600J$. For long running services power is a common measure (average energy use per second), whereas for tasks with a clear beginnings and ends energy is a common measure – the cost of a task.

Software energy consumption is a kind of performance and thus part of the non-functional requirement (NFR) of efficiency. Generally we want software to consume the least amount of energy and have low power use. Software energy consumption testing is typically considered a kind of performance regression testing. This kind of testing typically is evolutionary [2] and seeks to compare performance between versions on the tasks of the product.

Benchmarking is another kind of regression testing that allows comparison between products. Benchmarking is less about comparing versions, than it is about comparing different implementations of the same task. Some energy research seeks to benchmark applications for energy efficiency [3], [4].

III. The Past and Present

There are many issues in energy-aware software engineering, green software engineering, and green-mining [2] ranging from the complexity of testing, dependency on hardware, dependency on the environment, or dependency on software. The generalizability of this research is hampered by the complexity, and the lack of availability of tools. All of these issues compound the difficulty of applying static or dynamic analysis to software energy traces.

A. Ranking Applications by Energy Efficiency

Consumers tend to lack information about software. When a consumer buys an oven, the oven is ranked by its energy efficiency. What if the same ranking existed for software? Research exists that seeks to rank software in terms of energy efficiency much in the way that energy stars [5] rates and ranks consumer products [3].

Three main challenges that face ranking software by energy efficiency include:

- Software executes more than one task
- Fair benchmarks for multiple products
- Efficiency per platform (Software/Hardware)

The challenges that software faces versus ovens is that software does multiple tasks and some of these tasks are quite distinct, for instance email clients retrieve emails, search emails, and viewing emails. Thus without agreement about the shared tasks, not every feature or task can be compared between products. This is complicated by the lack of standardization. Figure 1 demonstrates an example of how application energy rankings could be integrated into an App store: different apps that fulfill the same tasks could be measured on a per task basis, as to allow consumers to see the different efficiencies each app has to offer. Figure 1 shows 2 different email applications that have been measured for task based energy consumption. The first app is intended to be sparse in features, simple email client that due its apparent simplicity is quite energy efficient. The second app on the right is meant to be a full featured, easy to use, graphically brilliant email application that is less energy efficient for reading emails than the plain and simple email app. The app-store depicts both apps names, icons, and user ratings, followed by the version number of the apps. Underneath is a general energy rating composed of the per-task ratings. The per task ratings are measured and compared with other apps. The ranking must be on a per-task basis rather than a holistic basis, as certain apps will focus on specific behaviours and tune themselves for it. This kind of ranking should also be done on other dimensions of performance so that users can compare apps to each other comprehensively and across multiple dimensions. This kind of comparison would allow users to determine the right app for the right occasion – power efficient apps for energy-constrained travel, and power hungry but slick apps for home and work. An open question related to this Figure 1 is, “how do we summarize energy consumption when different tasks occur at different frequencies?”

Not all hardware is created equal and not all software works the same on all hardware. Thus if software executes differently on different hardware it must be measured, simulated, or estimated on that hardware. Thus when one ranks software, should it be invariant of the hardware? If so how should one normalize it [3], [6]? Zhang [3] poses a method of normalizing across platform with linear scaling – this technique is used by the Green-Miner [6] to normalize measurements from different smart-phones under test. Our models should address the hardware dependent performance versus the hardware invariant performance.

Furthermore software energy consumption is not stable across versions [2], [7], [8], [9]. Testing a single build of the software might not be enough, a partial or entire energy consumption profile should probably be built. Users of an app will probably care if there is change in energy efficiency [10].

There are three main works which attempt to benchmark or pose the software energy problem in a similar way to Energy Star rankings [5]. Amsel et al. [11] discussed green tracker and compared web browsers for energy consumption performance. Zhang et al. [3], [12] describe the differences between applications that do the same tasks yet perform differently in terms of energy use. They propose *software*

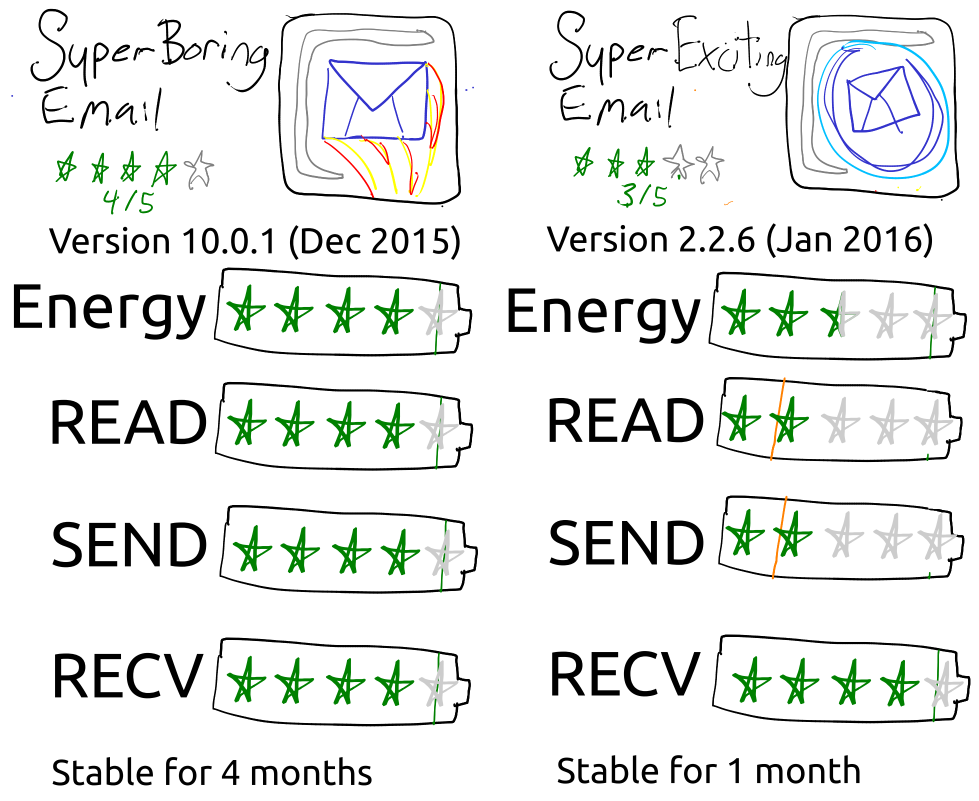


Fig. 1. Storyboard mock-up of future App ratings in the App Store

application energy consumption ratings (SAECR)/Green Star, a method to measure and compare and rank applications much like Energy Star [5]. Ecodroid [4] employed static and dynamic analysis to automatically rank applications – they did not use task based measurement."

Task based measurement is difficult because a task must be common across multiple software applications as to be important enough to be measured. Furthermore there is a granularity issue, at what level do we measure the task: per entity, per kilobyte, per task, per feature? When a new kind of application appears do we compare it to other applications based on shared features or shared tasks even though the intent of an application is the same? Task based measurement is a hard problem with many possible solutions. Sometimes one just needs to look at the intent, for instance imagine a new kind of video game, perhaps all that matters is not how the game operates, but that it provides entertainment – thus we would probably measure its energy performance for the task of entertaining.

Other stakeholders, such as manager, product owners, and developers might be concerned about other kinds of energy consumption – that of their virtual machines and services in the cloud and costs of such hosting. Thus not just mobile-apps should be ranked, but infrastructure software, middle-ware, operating systems, file-systems, and all of the components of cloud software distributions ought to be measured. This information might be less about sales and more about optimization and improved resource utilization. Tasks of service oriented software would be servicing an end-users session. Tasks of a middle-ware stack could be publication, delivery, and notification of workers.

Thus consumers need access to energy performance information and the app-store might be the perfect place to display such details, as shown in Figure 1. Whereas managers, software engineers, and system administrators might need this information from their OS providers and the software distributors, such as `apt` for Debian and Ubuntu Linux distributions.

B. Generalizable Models

One overarching goal of much of the energy consumption research is to produce a model that generalizes across many applications. The use-case of such a model is that developers do not have access to expensive hardware and cannot accurately measure the energy consumption of their applications – thus they must rely upon estimations based on different kinds of analyses and models. But these generalizable models suffer from the range of hardware, operating systems, environment, software domains, and versions of software.

In the mobile arena the wide-range of screen-sizes, memory sizes, and kinds of processors tends to hamper generalizable models. Furthermore the Android ecosystem is considered fragmented in terms of hardware and software [13]. Server-side, the difference between different x86 manufacturers chips can be significant. Thus there is much hardware variation in terms of primary components. This ignores the range of peripherals and I/O devices that be prevalent on mobile devices: GPS, motion sensor, touch sensors, light sensor, accelerometers, cameras, bluetooth, wifi, etc.

The issue of different operating systems is also relevant, Windows and Linux do not share the same code base and handle energy management differently. Android includes customizations distinct from Linux as well. Furthermore there are different versions and distributions of Linux, Windows, Android, OSX, and iOS. Thus measurements from one environment might not hold for another.

Furthermore generalizable models suffer from a lack of data. Energy traces are not prevalent in the operational data within Github git repositories or other publicly available repositories. Continuous integration tools tend not to measure or estimate energy. One possible repository of energy data, from the Carat project [14], is not publicly accessible to developers and researchers. Thus there is a real lack of software energy data available to researchers and what is available is not very comprehensive [15].

Too many models are very hardware dependent. For instance the models of Pathak et al. [16] require arduous component modeling. Much of the work of Hindle et al. [6] only is tested on a small subset of Android devices and platforms. Karan et al. [17], [18] built upon Pathak et al. [16] work and suggested a rule-of-thumb model based on system calls that works relatively well: if the system call count significantly changes between versions then the energy use between versions changes significantly. This rule-of-thumb specifically avoids mis-classifying many of the 90% of changes which do not affect the energy profile of an application. This model was extended and generalized as a regression problem by Shaiful et al. [19] who estimate not only change, but actual energy usage. The system call based models are general and relative to the products themselves, they model applications that face the user quite well. Yet the models fail to account for CPU use effectively without the use of counters. Models need to address what is generalizable and stable across subsets of hardware – what can accounted or controlled for – and what part of the models are hardware dependent. This knowledge enables generalization of models to similar devices.

While these models vary in their relevance, usefulness, and ability to deploy, will programmers want to apply these models?

C. How knowledgeable are programmers about energy?

While software engineering researchers are interested in software energy consumption are programmers knowledgeable or aware? Currently in 2015, the answer is a resounding, "not really."

Pinto et al. [20] studied StackOverflow [21], a question-answer site for programmers. They found that energy related questions were poorly answered and that many questions were asked.

Pang et al. [22] followed up and surveyed and interviewed programmers. Pang et al. found that programmers surveyed did not have much experience with software energy consumption. Not only did they lack experience but rarely were they asked to address software energy consumption. The programmers also said they would consider energy consumption when buying a mobile device.

Wilke et al. [10] corroborates the view of these developers. They found that App ratings on Google Play Store suffered when user commented on poor energy consumption behaviour. Khalid et al. [23] have made similar observations.

Many works aim to help developers by finding specific energy bugs [24]. Manotas et al. [25] provide suggestions for energy efficient collections. Others have suggested using genetic programming to optimize already existing programs [26]. Some works discuss the cost of using libraries that provide advertisements [27] and some works describe the costs and benefits of ad-blocking with respect to energy consumption [28]. Some work aims to optimize display usage through color choices [29]. While others help to provide feedback to developers if anything has changed [18].

D. Measurement

Software energy consumption needs to be measured. Many researchers use time as a proxy but for idle applications this might not be appropriate [3].

Hindle et al. [6] describes the green miner, a hardware-based continuous regression test framework. The green miner is a software queue for tests that enables deployment of tests onto a series of Android phones, enabling parallel execution of tests. Figure 2 depicts a screen-shot of a report from the green miner over a single test-run of an application. The time-line shows the power usage over time, followed by an energy consumed per component stacked bar plot. The energy consumed and power used is broken down by tasks within the test. At the end meta-data about the test is recorded. A similar work was presented by Banerjee et al. [30] whereby they use physical instrumentation.

Li et al. [31] tried to used high frequency measurements to attribute energy use to particular source lines, while Hao et al. [32] applied program analysis to estimate energy use. Gupta et al. [33] attempted to correlate measurements with library usage.

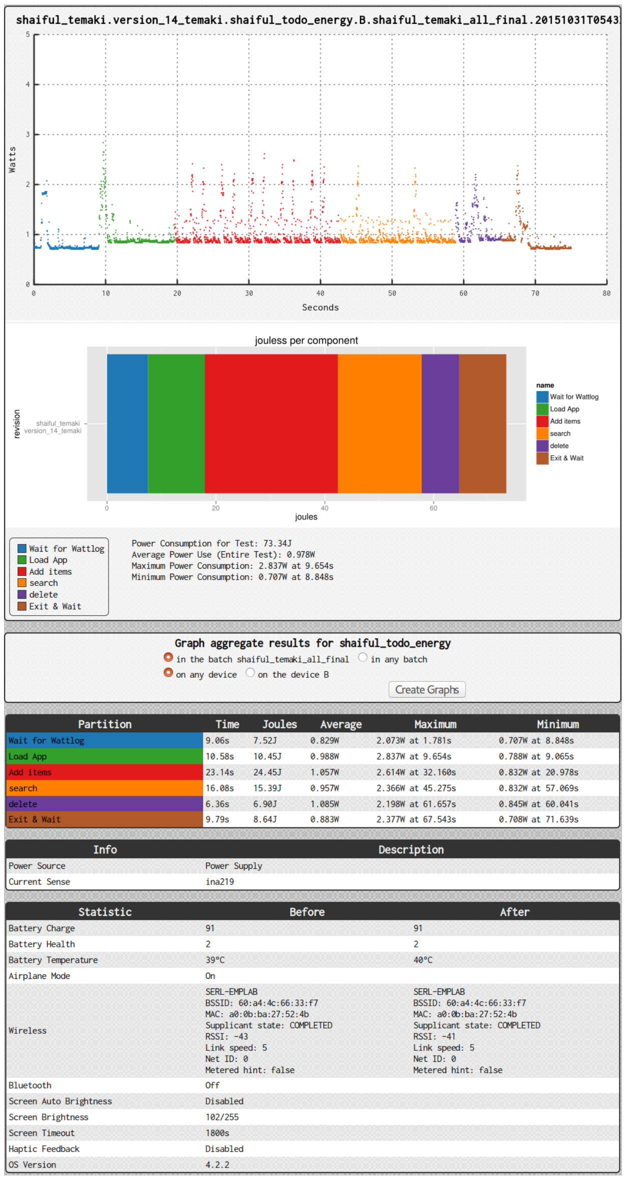
`LessWatts.org` from Intel develops and provides the PowerTop tool to estimate energy use at run-time based on ACPI information [34].

Thus there are many hardware methods of measuring energy but many are too complicated for programmers so they opt for either server hardware with instrumentation or for estimations from ACPI. Not everyone has electrical measurement expertise, thus when in doubt ask a colleague or an another engineer about measurement.

IV. Immediate Challenges

The field is currently immature. There is a lack of:

- Shared tools;
- Shared datasets;
- Benchmarks datasets;
- Agreement on methodology;
- Coherent community;



🟦 Wait for Wattlog
🟩 Load App
🟥 Add items
🟧 search
🟪 delete
🟫 Exit & Wait

```
Power Consumption for Test: 73.34J
Average Power Use (Entire Test): 0.978W
Maximum Power Consumption: 2.837W at 9.654s
Minimum Power Consumption: 0.707W at 8.848s
```

Graph aggregate results for shaiful_todo_energy

[✓ in the batch shaiful_temaki_all_final] in any batch
[✓ on any device] on the device B

Create Graphs

Partition	Time	Joules	Average	Maximum	Minimum
Wait for Wattlog	9.06s	7.52J	0.829W	2.073W at 1.781s	0.707W at 8.848s
Load App	10.58s	10.45J	0.988W	2.837W at 9.654s	0.788W at 9.065s
Add items	23.14s	24.45J	1.057W	2.614W at 32.160s	0.832W at 20.978s
search	16.08s	15.39J	0.957W	2.366W at 45.275s	0.832W at 57.069s
delete	6.36s	6.90J	1.085W	2.198W at 61.657s	0.845W at 60.041s
Exit & Wait	9.79s	8.64J	0.883W	2.377W at 67.543s	0.708W at 71.639s

Info	Description
Power Source	Power Supply
Current Sense	ina219

Statistic	Before	After
Battery Charge	91	91
Battery Health	2	2
Battery Temperature	39°C	40°C
Airplane Mode	On	
Wireless	SERL-EMPLAB BSSID: 60:a4:4c:66:33:f7 MAC: a0:0b:ba:27:52:4b Supplicant state: COMPLETED RSSI: -43 Link speed: 5 Net ID: 0 Metered hint: false	SERL-EMPLAB BSSID: 60:a4:4c:66:33:f7 MAC: a0:0b:ba:27:52:4b Supplicant state: COMPLETED RSSI: -41 Link speed: 5 Net ID: 0 Metered hint: false
Bluetooth	Off	
Screen Auto Brightness	Disabled	
Screen Brightness	102/255	
Screen Timeout	1800s	
Haptic Feedback	Disabled	
OS Version	4.2.2	

Fig. 2. Green Miner Example Test Run

- Methodological discussion about addressing threats;
- Lack of agreement on methodological threats.

All of these issues pile up into immediate challenges that face current software energy research as well as those works of the future. This section tries to illustrate the potential issues that the field faces.

A. Lack of Data

Currently there is very little data for researchers to work with. The current pattern is for researchers to setup a test and measure everything themselves. This is different than a lot of mining software repositories [35], [36] research – there is not a repository available here, unlike other kinds of dynamic analysis such as crash reports.

Each energy trace, such as the one depicted in Figure 2, from 1 test-run, contains many measurements of the same test over time. Typically these tests are re-run to address error in physical measurement and the environment. The re-running of the tests leads researchers to summarize the distribution of measurements leading to a collapse in the amount of data. 5000 kilobytes of energy measurements can be quickly collapsed down into 10 to 40 rows of summary statistics about the test-runs. The data is effectively limited by the researchers time and ability to run all the necessary tests. As dynamic analysis and tracing is typically used it takes a lot of run-time to execute tests.

Thus dynamic analysis takes time, but there is also a limitation in the number of applications that meet the requirements of the research. For instance if one is focusing on Android applications with available source code, the set of testable applications is quite limited. Furthermore given those applications very few come with tests so tests need to be generated.

Thus the field lacks data due to a lack of collecting existing data, a lack of sharing of existing data, a lack of appropriate applications to test, and a lack of available tests for these applications to enable dynamic analysis.

B. CPU is not enough

Many works – especially in the area of distributed computing – simply relate CPU time to energy [37], [38], [39], [40]. While this is correct for CPU bound processes, many applications are not CPU bound. Some are not even IO bound, they are event bound. Thus they have an idle cost, but not much in the way of CPU work. They might induce IO when woken up but for the most part most user facing applications are quite idle. Furthermore CPU use does not necessarily represent the activity of peripherals. In the case of GPU clusters, CPU use could be almost irrelevant as the GPU would be the dominant energy consumer.

If processes are CPU bound, optimizing and addressing their energy consumption is well supported by current benchmark tools and profilers. For a CPU bound process, improvement to its single-core run-time performance will usually improve its energy efficiency as well.

C. Virtualization

If one cares about sustainability [41], [42] and reducing the global energy consumption of computing, virtualization cannot be ignored. Many services online are virtualized, running on a virtual machine in the cloud, or within a container of a container service.

Measuring VMs and containers is quite difficult as resources are not equally shared [43]. Many clouds use oversubscription, whereby resources are over promised to many services with the hope that these services do not need all of these resources at the same time. With virtual machines CPU is often over subscribed while for containers both CPU and memory are oversubscribed.

With most of the world's services running within datacenters any savings in the resources used by a service has a potential for saving energy: when the CPU, memory, or peripherals such as hard-drives, network cards, and GPUs. The less use, the less heat, resulting in less cooling and more savings.

Currently there is some work on attributing energy consumption to virtualized machines [44], [43], but it is just the beginning of such research. Little to no work has been done to estimate the energy use of services within containers such as docker. At the moment it is very difficult to estimate the energy cost of a task that is virtualized or container-ized.

D. Multi-version analysis necessary

In a software product one change can fundamentally change the performance of the software [8]. The same is true for software energy performance. Thus to characterize the performance of a product by only its latest version is unfair. Projects such as Hadoop have had issues with performance regressions – Hadoop 2 on smaller clusters often performs worse on the same task than Hadoop 1 due to resource management [45].

Software is more than the just the current version – most developers exist in a continuously evolving context producing many builds and many versions at once. Versioning is a constant problem within software distributions such as Debian [46] or Ubuntu.

Thus one change can have a significant effect on performance of software, and that change might be required to address a raft of issues. This performance changing commit could negate past results [9]. Romansky et al. [9] investigated if every revision needed to be measured or just some, and found that the performance changing commits generally were either immediately corrected or initiated a long plateau of similar performance across subsequent versions.

For instance if one tests if a refactoring was impactful and just look at refactoring commits – what happens afterward [47]? Is the behaviour stable? Was there a bug? Just looking at the immediate before and after commits might not be enough to determine the actual effect of a design pattern or a refactoring.

Thus multi-version analysis adds more software to analyze, enables more data to be collected but also adds robustness against some threats to validity.

E. Non-determinism of hardware state

One problem with modern software is that it runs on complicated platforms. Furthermore the realistic scenario of running applications on various hardware can be complex.

One such difficulty is hardware power saving functionality which enables CPUs to use more or less voltage or to change clock rates, as well as enable low power or high latency mode in peripherals. In real world use these options are often on. Experimentally they are often turned off or set to a constant setting, but not always, and it is not always beneficial to create such an artificial setting [3].

Even if we start the CPU in a certain state for a test, the test input might induce a different CPU state [48]. Thus setting the CPU power state before a test might not be enough to ensure equivalent CPU power states during the test. Other sources of hardware sources of non-determinism are wifi networks and some disk I/O.

F. Non-determinism of software state

Software can exhibit non-determinism. Mobile platforms are quite adaptive and small changes in the environment can result in different behaviour. Furthermore events within the operating system are not always controllable or deterministic. Network communication is not deterministic as well, thus one serious confound is the non-determinism of software state in the OS alone. When this is combined with long running services non-determinism abounds.

If software communicates across the network, the network congestion, time of day, and availability of the access service could all have an effect. If software writes to a file-system, the current state of the file-system could determine how continuous or how fragmented a file is written to disk – more writes could lead to more fragmentation. Memory allocation could fragment memory leading to more work and compaction as time progresses. These issues are hard to address, and the most common solution, even utilized by micro kernel architectures, is to simply restart and throw away all that old state.

G. The need for science

Within this section I have brought up many issues, but how many have empirical evidence to demonstrate the dangers or costs of ignoring these issues. What if the measurements are strong against noise after enough runs are executed? Perhaps after 40 runs the initial state does not matter. Arcuri and Briand have provided practical guidelines for statistical tests within software engineering that are specific too but are still relevant to performance and software energy consumption [49].

Furthermore in terms of publications regarding energy what we need is more science. Not every energy consumption paper can be a tool paper. Sometimes a result or technique could be integrated into an existing tool or be deployed as tool, but that is a high bar when most bug prediction work never produces a deployable tool. Currently for software energy consumption research the bar is quite high, there is little taste or favour to scientific publications, such as the work of Li et al. [50], Romansky et al. [9], or Linares-Vásquez et al. [51], rather than tool publications such as Green Advisor [18]. Yet communities such as MSR [36] and ESEM [52] promote this kind of research with other kinds of nonfunctional requirements (NFRs) such a performance or maintainability have much empirical work behind them.

H. Community

Not only do we need more science in software energy consumption, we need more community support. There are some industry wide groups that discuss Green IT, sustainable IT infrastructure [53], [54], [55]. As of writing there is a smattering of specific sustainability and green IT conferences, none are truly coherent when it comes to software energy consumption, as each venues have different goals [56], [57], [58].

There are workshops such as GREENS [58], but the motivation to publish at GREENS is low when one can submit papers to other venues which garner more recognition. Furthermore in software engineering venues one could perceive there is a lack of a knowledge regarding software energy consumption, which mirrors the current reality of programmers' knowledge [22].

All of these factors lead to a software energy research diaspora, where sub-communities are made and results are quietly published but not noticed by other communities. It is almost as if researchers are publishing into a vacuum whereby other researchers do not see each other.

I. Impossible bar to reach, or potential paper?

These limitations should not scare anyone away from the field, in fact for empiricists and experimentalists these are papers in waiting. Many of these issues might not pose as significant effect as we think, or their effect might be avoided or controlled for methodologically. For any of these issues there is an impactful avenue of research found by asking the question, "Do we have to address this potential pitfall?" Potential paper topics that anyone, especially up and coming PhD students, could address:

- How to model the difference in performance between OSX, Windows, and Linux, or iOS and Android. What parts of energy models will change due to operating system or hardware, and what parts of the energy models will stay the same? How can we generalize across platforms?
- The effects of state on repeated tests. Benchmarks are often performed without restarting or without clearing caches. Methodologically what do we lose by ignoring this, versus what do we gain in test performance?
- The effects of differing wifi-state on energy tests. Wifi cards and radio-based networking tend to operate different in different contexts, such as closeness to a router or interference. How does this affect energy tests?
- Temperature, mobile devices, and energy consumption. Mobile devices typically lack temperature control mechanism and are often stored close to a warm human body, what is the impact of energy-testing at room temperature versus body temperature?
- Effective version test selection. How many versions do we really need to measure? Multi-version testing is expensive in terms of effort and time [59], what are effective selection and search strategies to reduce this work?
- Effect of hyper-visors on energy consumption. Much cloud computing uses virtualization – what is the impact of the hyper-visor on energy performance of a cloud server?
- How to control for non-determinism in disk I/O. File-systems are not necessarily deterministic as repeated writes can lead to fragmentation [60]. What is the energy impact of disk and file-system non-determinism on energy consumption? Does this have an affect on I/O based tests?
- How to control for non-determinism in network I/O. Network I/O often occurs out on the wild internet whereby traffic and congestion change hourly. When we run energy tests how much should we and can we control for these factors – and how much do we gain if we do?
- How to account for different background cloud utilization. Cloud computers often host multiple clients – how do different levels of tenancy affect energy consumption of a single virtual machine or service?
- What is the effect of isolation on our tests? The average user or cloud service will not be running software in a clean room environment. How does software energy consumption respond to the noise of real environments?
- Is there difference between software instrumentation and human input? If our tests are automated by faking user inputs, do the inputs and instrumentation use more or less energy?
- What issues affect ACPI energy estimates? ACPI is often used to estimate energy consumption but do we need to control for blind spots in ACPI – can these be addressed by better models?
- How many energy measurements do I need per version? When we wish to compare two versions how many measurements do we really need from each version? Can we determine the number of measurements dynamically to reduce work?
- Does the quality of test cases matter when measuring energy? Does code coverage matter for testing? How much exercise should a test do in order for its energy measurement to be representative, meaningful or comparable?

Thus these limitations should spur scientific research into the effects and costs of addressing and ignoring the issues brought up in this section. These limitations and proposal for future work segways into what will be expected in the future, next.

V. The Future

In this section I lay out my prediction for the future of software energy consumption research.

A. Multi-version analysis will be expected

In the future researchers will engage in multi-version analysis of performance and energy consumption. They will use multi-version analysis because a primary concern of energy consumption is performance regression e.g., "has performance worsened?" Multi-version analysis will also be used to increase the generality and robustness of their research. Instead of making claims about one snapshot of a program's performance researchers will establish the profile [9] of a program's performance. This is especially important in research that engages in factor analysis as it provides more measurement of the system but also protects against spurious factors being reported as significant.

B. An end to developer measurement

In the future we will never expect a developer to physically benchmark or measure their software. This will be the realm of technicians and researchers, not developers. Physical measurement will be avoided by easy to access services, better models of services and apps, and better software energy estimation frameworks that appropriately address the needs and limitations of developers.

a) No hardware Instrumentation: hardware is expensive, and it requires much knowledge and training to address hardware measurement. The developer of the future will not have to rely on expensive testing hardware, or the questionable measurements of their ACPI chip-sets.

b) Access to Hardware Regression Testing Services: if developers truly need physical measurement they will be able to outsource it. We expect in the future that services will be available that will be like the Green Miner [6] – developers will submit applications, specify the hardware to test on and simply wait for a result back from the framework. No awkward setup, no difficult testing. What physical measurements are made will probably be integrated into even better models.

For verification programmers will have the option to submit their application to a continuous integration, testing, and deployment service that will provide some hardware based measurement. Yet for the most part the future engineers need not worry about actually measuring software energy consumption.

c) Recommender Systems and Agents: Much like Clippy [61], integrated into Microsoft Office, developers should be treated with recommender systems and smart agents that can help guide their software development towards better energy performance. These systems could be integrated into IDEs as to provide immediate suggestions and hints to the developer.

C. Online Shared Repository

The future holds promise as large open shared repositories of dynamic traces of energy consumption are curated. Different platforms, different applications, different tests and different runs all aggregated in large online repositories of data. These shared repositories would allow the curation of community tuned models of energy consumption. Much like the PROMISE repository [62].

The repositories would allow the hours and hours that practitioners and researchers spend benchmarking and testing software to be used to develop better models. The variation in available runs alone would be intensely beneficial. Even the tests themselves could be shared, enabling further collaboration.

The future is crowd sourced and open shared traces available to all.

D. Cloud and Container Estimation

One of the largest concerns in the future will be the sustainability of software services [41]. There will be pressure from social causes, combined with carbon taxes and worldwide sustainability pressure to reduce carbon emissions. This will affect the software as a service market. Furthermore companies will be asked to estimate their energy use so they can argue if they are green and sustainable. The requirements of sustainable engineering will prompt for developer awareness of the issues and the ability to estimate the impact of services.

Thus all the difficulties mentioned before in Section IV-C will conflict with the requirement of energy estimation – programmers of the future will have to estimate or measure the sustainability of distributed software ecosystems. These ecosystems might not be fully subscribed to – many will be relatively idle services – but such estimates of energy consumption will be required.

Programmers will submit usage scenarios, configuration, and their software to a testing service that will estimate the energy usage of their services at different loads. This will require a new kind of continuous integration and deployment software to operate. Furthermore such a system will need measurement instruments to enable measuring, modeling, and estimation of software energy consumption.

E. Budgeted Software and Energy Requirements

More managers and customers will explicitly request software energy consumption be addressed in their applications. As Lago et al. [42] suggest, sustainability will be perceived as a software quality issue.

This will imply that not only will energy requirements exist, but likely services will be granted energy budgets that they have work within. It is likely that services provided by Amazon AWS and other cloud providers will start explicitly charging for energy rather than just CPU, Memory, IO and network usage. With this change in pricing part of the requirements elicitation process will be to define the energy budgets of a service.

F. Education

As Pang et al. [22] found, developers are not very aware of software energy consumption and thus if they were asked to act on it, as developers tend not to be very educated on the causes of software energy consumption. As of writing this, software energy consumption is a niche topic rarely taught to computer scientists or software engineers – although somewhat addressed in electrical engineering and computer engineering curriculums. Developers of the future will face the demand for sustainable systems, thus computer science and software engineering curriculums will change to address green software engineering.

VI. Conclusions

Software energy consumption research currently faces many challenges and threats to validity. Among these are attribution of energy use to processes, measurement of virtualized or containerized processes, estimation of energy use, and lack of freely available software energy tools that do not require physical hardware.

Methodologically software energy research is plagued by threats to generalizability regarding OS versions, application versions, environments, the variety of available hardware, and a lack of recorded operational data and measurements.

Currently there is a very limited research community who has done little to share data and tools. This is further compounded by a fragmentation of the community across numerous small conferences and workshops. Some of this fragmentation arises from the fundamental multidisciplinary aspects of software energy consumption: electrical engineers, computer engineers, and software engineers should work together to help each other address energy consumption holistically so that software can be written to take advantage of hardware advances, and so hardware can take better advantage of software knowledge for energy consumption and performance.

The future holds much promise for the field of software energy consumption as there are many hard challenges that need to be addressed. The programmers of the future will face sustainability as a requirement and will have to design software with energy efficiency in mind. These programmers not only will receive education, instruction, and training, they will have at their disposal powerful models and tools that are integrated into their development environment ever ready to provide them with software energy awareness when they need it.

Acknowledgements

Abram Hindle is funded and supported by a NSERC Discovery Grant.

References

[1] M. E. Jed Scaramella, "Solutions for the datacenter's thermal challenges," http://whitepapers.zdnet.com/abstract.aspx?docid=352318, January 2007, iDC white paper.

[2] A. Hindle, "Green mining: A methodology of relating software change to power consumption," in Submission to MSR 2012, http://softwareprocess.es/a/green-change-e.pdf.

[3] C. Zhang, A. Hindle, and D. M. Germán, "The impact of user choice on energy consumption," *IEEE Software*, vol. 31, no. 3, pp. 69–75, 2014. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/MS.2014.27

[4] R. Jabbarvand, A. Sadeghi, J. Garcia, S. Malek, and P. Ammann, "Ecodroid: an approach for energy-based ranking of android apps," in *Proceedings of the Fourth International Workshop on Green and Sustainable Software*. IEEE Press, 2015, pp. 8–14.

[5] E. Star, "Energy star: The simple choice for energy efficiency," 2016. [Online]. Available: https://www.energystar.gov

[6] A. Hindle, A. Wilson, K. Rasmussen, J. Barlow, J. Campbell, and S. Romansky, "GreenMiner: A Hardware Based Mining Software Repositories Software Energy Consumption Framework," in *Mining Software Repositories (MSR), 2014 11th IEEE Working Conference on*. ACM, 2014.

[7] A. Hindle, "Green mining: Investigating power consumption across versions," in *Proceedings, ICSE: NIER Track*. IEEE Computer Society, 2012, http://ur1.ca/84vh4.

[8] ——, "Green mining: a methodology of relating software change and configuration to power consumption," *Empirical Software Engineering*, vol. 20, no. 2, pp. 374–409, 2015. [Online]. Available: http://dx.doi.org/10.1007/s10664-013-9276-6

[9] S. Romansky and A. Hindle, "On improving green mining for energy-aware software analysis," in *Press of the 2014 Conference of the Center for Advanced Studies on Collaborative Research, IBM Corp*, 2014.

[10] 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*. IEEE, 2013, pp. 134–141.

[11] N. Amsel and B. Tomlinson, "Green tracker: a tool for estimating the energy consumption of software," in *Proceedings, CHI EA*. New York, NY, USA: ACM, 2010, pp. 3337–3342.

[12] C. Zhang, "The Impact of User Choice and Software Change and Energy Consumption," *University of Alberta, Edmonton, Alberta, Canada*, 2013.

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

[14] E. Peltonen, E. Lagerspetz, P. Nurmi, and S. Tarkoma, "Energy modeling of system settings: A crowdsourced approach," in *Pervasive Computing and Communications (PerCom), 2015 IEEE International Conference on*, March 2015, pp. 37–45.

[15] C. Zhang and A. Hindle, "A green miner's dataset: mining the impact of software change on energy consumption," in *Proceedings of the 11th Working Conference on Mining Software Repositories*. ACM, 2014, pp. 400–403.

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

[17] 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 *Press of the 2014 Conference of the Center for Advanced Studies on Collaborative Research, IBM Corp*, 2014.

[18] K. Aggarwal, A. Hindle, and E. Stroulia, "Greenadvisor: A tool for analyzing the impact of software evolution on energy consumption," in *31st IEEE International Conference on Software Maintenance and Evolution*. IEEE Computer Society, 2015.

[19] S. A. Chowdhury, L. N. Kumar, M. T. I. M. S. M. Jabbar, V. Sapra, K. Aggarwal, A. Hindle, and R. Greiner, "A system-call based model of software energy consumption without hardware instrumentation," in *Proceedings of the Sixth International Green and Sustainable Computing Conference (IGSC'15)*, 2015.

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

[21] "Stack Overflow," http://stackoverflow.com, 2008.

[22] C. Pang, A. Hindle, B. Adams, and A. E. Hassan, "What do programmers know about the energy consumption of software?" *IEEE Software*.

[23] H. Khalid, E. Shihab, M. Nagappan, and A. E. Hassan, "What do mobile app users complain about? A study on free iOS apps," *Accepted to be published in IEEE Software*, 2014.

[24] Y. Liu, C. Xu, and S. Cheung, "Diagnosing energy efficiency and performance for mobile internetware applications: Challenges and opportunities," *Software, IEEE*, vol. PP, no. 99, pp. 1–1, 2015.

[25] 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*, ser. ICSE 2014. New York, NY, USA: ACM, 2014, pp. 503–514. [Online]. Available: http://doi.acm.org/10.1145/2568225.2568297

[26] B. R. Bruce, J. Petke, and M. Harman, "Reducing energy consumption using genetic improvement," in *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO '15. New York, NY, USA: ACM, 2015, pp. 1327–1334. [Online]. Available: http://doi.acm.org/10.1145/2739480.2754752

[27] J. Gui, S. Mcilroy, M. Nagappan, and W. G. J. Halfond, "Truth in advertising: The hidden cost of mobile ads for software developers," in *37th IEEE/ACM International Conference on Software Engineering, ICSE 2015, Florence, Italy, May 16-24, 2015, Volume 1*. IEEE, 2015, pp. 100–110. [Online]. Available: http://dx.doi.org/10.1109/ICSE.2015.32

[28] K. Rasmussen, A. Wilson, and A. Hindle, "Green mining: energy consumption of advertisement blocking methods," in *Proceedings of the 3rd International Workshop on Green and Sustainable Software, GREENS 2014, Hyderabad, India, June 1, 2014*, H. A. Müller, P. Lago, M. Morisio, N. Meyer, and G. Scanniello, Eds. ACM, 2014, pp. 38–45. [Online]. Available: http://doi.acm.org/10.1145/2593743.2593749

[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 *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2015. New York, NY, USA: ACM, 2015, pp. 143–154. [Online]. Available: http://doi.acm.org/10.1145/2786805.2786847

[30] A. Banerjee, L. K. Chong, S. Chattopadhyay, and A. Roychoudhury, "Detecting energy bugs and hotspots in mobile apps," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, 2014, pp. 588–598.

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

[32] S. Hao, D. Li, W. G. J. Halfond, and R. Govindan, "Estimating Mobile Application Energy Consumption using Program Analysis," in *Proceedings of the 2013 International Conference on Software Engineering*, ser. ICSE '13, 2013, pp. 92–101.

[33] A. Gupta, T. Zimmermann, C. Bird, N. Naggapan, T. Bhat, and S. Emran, "Energy Consumption in Windows Phone," Microsoft Research, Tech. Rep. MSR-TR-2011-106, 2011.

[34] Intel, "LessWatts.org - Saving Power on Intel systems with Linux," http://www.lesswatts.org, 2011.

[35] A. E. Hassan, "The Road Ahead for Mining Software Repositories," in *Proceedings of the Future of Software Maintenance (FoSM) at the 24th IEEE International Conference on Software Maintenance*, 2008, pp. 48–57.

[36] MSR, "Mining Software Repositories," www.msrconf.org, 2013.

[37] C. Seo, S. Malek, and N. Medvidovic, "An Energy Consumption Framework for Distributed Java-Based Systems," in *ASE '07*, 2007, pp. 421–424.

[38] ——, "Component-level energy consumption estimation for distributed java-based software systems," in *Component-Based Software Engineering*. Springer, 2008, pp. 97–113.

[39] R. Joseph and M. Martonosi, "Run-Time Power Estimation in High Performance Microprocessors," in *Proceedings of the 2001 international symposium on Low power electronics and design*, ser. ISLPED '01. New York, NY, USA: ACM, 2001, pp. 135–140.

[40] V. Tiwari, S. Malik, A. Wolfe, and M. Tien-Chien Lee, "Instruction level power analysis and optimization of software," *The Journal of VLSI Signal Processing*, vol. 13, 1996.

[41] C. Becker, R. Chitchyan, L. Duboc, S. Easterbrook, B. Penzenstadler, N. Seyff, and C. C. Venters, "Sustainability design and software: The karlskrona manifesto," in *37th IEEE/ACM International Conference on Software Engineering, ICSE 2015, Florence, Italy, May 16-24, 2015, Volume 2*. IEEE, 2015, pp. 467–476. [Online]. Available: http://dx.doi.org/10.1109/ICSE.2015.179

[42] P. Lago, S. A. Koçak, I. Crnkovic, and B. Penzenstadler, "Framing sustainability as a property of software quality," *Commun. ACM*, vol. 58, no. 10, pp. 70–78, 2015. [Online]. Available: http://doi.acm.org/10.1145/2714560

[43] F. A. Moghaddam, P. Lago, and P. Grosso, "Energy-efficient networking solutions in cloud-based environments: A systematic literature review," *ACM Comput. Surv.*, vol. 47, no. 4, p. 64, 2015. [Online]. Available: http://doi.acm.org/10.1145/2764464

[44] F. A. Moghaddam, T. Geenen, P. Lago, and P. Grosso, "A user perspective on energy profiling tools in large scale computing environments," in *2015 Sustainable Internet and ICT for Sustainability, SustainIT 2015, Madrid, Spain, April 14-15, 2015*. IEEE, 2015, pp. 1–5. [Online]. Available: http://dx.doi.org/10.1109/SustainIT.2015.7101364

[45] A. H. Ivanilton Polato, Denilson Barbosa and F. Kon, "Hadoop branching: Architectural impacts on energy and performance," in *Proceedings of the Sixth International Green and Sustainable Computing Conference (IGSC'15)*, 2015.

[46] M. Claes, T. Mens, R. Di Cosmo, and J. Vouillon, "A historical analysis of debian package incompatibilities," in *Mining Software Repositories (MSR), 2015 IEEE/ACM 12th Working Conference on*. IEEE, 2015, pp. 212–223.

[47] C. Sahin, L. Pollock, and J. Clause, "How do code refactorings affect energy usage?" in *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ser. ESEM '14. New York, NY, USA: ACM, 2014, pp. 36:1–36:10. [Online]. Available: http://doi.acm.org/10.1145/2652524.2652538

[48] S. A. Chowdhury, V. Sapra, and A. Hindle, "Is HTTP/2 more energy efficient than HTTP/1.1 for mobile users?" *PeerJ PrePrints*, vol. 3, p. e1280, 2015. [Online]. Available: http://dx.doi.org/10.7287/peerj.preprints.1280v1

[49] A. Arcuri and L. Briand, "A practical guide for using statistical tests to assess randomized algorithms in software engineering," in *Software Engineering (ICSE), 2011 33rd International Conference on*. IEEE, 2011, pp. 1–10.

[50] 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*. IEEE Computer Society, 2014, pp. 121–130. [Online]. Available: http://dx.doi.org/10.1109/ICSME.2014.34

[51] 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*, ser. MSR 2014. New York, NY, USA: ACM, 2014, pp. 2–11. [Online]. Available: http://doi.acm.org/10.1145/2597073.2597085

[52] *International Symposium on Empirical Software Engineering and Measurment*, 2015. [Online]. Available: http://esem-conferences.org/

[53] S. Murugesan, "Harnessing Green IT: Principles and Practices," *IT Professional*, vol. 10, no. 1, pp. 24–33, 2008.

[54] Alliance to Save Energy, "PC Energy Report 2007: United States," http://www.climatesaverscomputing.org/docs/Energy_Report_US.pdf, 1E, Tech. Rep., 2007.

[55] P. Kurp, "Green computing," *Communications of the ACM*, vol. 51, no. 10, pp. 11–13, 2008.

[56] *GREENCOM '12: Proceedings of the 2012 IEEE International Conference on Green Computing and Communications*. Washington, DC, USA: IEEE Computer Society, 2012.

[57] *Proceedings of the Sixth International Green and Sustainable Computing Conference (IGSC'15)*, 2015.

[58] "Fourth international workshop on green and sustainable software (GREENS 2015)," 2015.

[59] S. Romansky and A. Hindle, "On improving green mining for energy-aware software analysis," in *Proceedings of 24th Annual International Conference on Computer Science and Software Engineering*. IBM Corp., 2014, pp. 234–245.

[60] P. Greenawalt, "Modeling power management for hard disks," in *MASCOTS '94., Proceedings of the Second International Workshop on*, Jan 1994, pp. 62 –66.

[61] E. Horvitz, J. Breese, D. Heckerman, D. Hovel, and K. Rommelse, "The lumiere project: Bayesian user modeling for inferring the goals and needs of software users," in *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*. Morgan Kaufmann Publishers Inc., 1998, pp. 256–265.

[62] T. Menzies, R. Krishna, and D. Pryor, "The promise repository of empirical software engineering data," North Carolina State University, Department of Computer Science, 2016. [Online]. Available: http://openscience.us/repo