GreenMiner: A Hardware Based Mining Software Repositories Software Energy Consumption Framework

Abram Hindle, Alex Wilson, Kent Rasmussen, E. Jed Barlow,
Joshua Campbell, Stephen Romansky
Department of Computing Science
University of Alberta
Edmonton, Canada
{hindle1, aewilson, krasmuss, ejbarlow, joshua2, romansky}@ualberta.ca

ABSTRACT

Green Mining is a field of MSR that studies software energy consumption and relies on software performance data. Unfortunately there is a severe lack of publicly available software power use performance data. This means that green mining researchers must generate this data themselves by writing tests, building multiple revisions of a product, and then running these tests multiple times (10+) for each software revision while measuring power use. Then, they must aggregate these measurements to estimate the energy consumed by the tests for each software revision. This is time consuming and is made more difficult by the constraints of mobile devices and their OSes. In this paper we propose, implement, and demonstrate Green Miner: the first dedicated hardware mining software repositories testbed. The Green Miner physically measures the energy consumption of mobile devices (Android phones) and automates the testing of applications, and the reporting of measurements back to developers and researchers. The Green Miner has already produced valuable results for commercial Android application developers, and has been shown to replicate other power studies' results.

Categories and Subject Descriptors

D.4.8 [**Performance**]: Energy; D.2.5 [**Testing**]: Regression

General Terms

Performance

Keywords

Software Energy Consumption; Software Change; Android

MSR '14, May 31 - June 07 2014, Hyderabad, India

ACM 978-1-4503-2863-0/14/05...$15.00.
http://dx.doi.org/10.1145/2597073.2597097.

1. MOTIVATION

In Free/Libre Open Source Software there tends to be a lack of historical records regarding testing results, testing performance, dynamic analyses of test runs, and power use performance profiles. Run-time data, such as the results of unit tests, are often thrown away once they are seen by developers who are interested. Crash reports are the most common form of run-time data that is stored. Tools such as Launchpad [1] enable the reporting and aggregation of crash reports. While some exceptions such as Mozilla smoke tests [2] and CPAN [3] smoke tests exist, for the most part there are not many performance oriented data sets available in the software repositories that the field of *Mining Software Repositories* (MSR) mines. Due to the lack of performance data and due to the severe lack of power use/energy consumption related data the task of *green mining* [11], studying the relationship between software changes and software energy consumption, is difficult and time-consuming for both researchers and developers.

Studying software power use, especially mobile software power use is incredibly important because over 20% of the US population owns and uses smart phones [20]. Thus users rely on these mobile computers for not just games, but for social and emergency connectivity. In the Android marketplace, battery saving apps such as Juice Defender [6] have been downloaded over 10 million times, indicating that battery life is a concern for many smartphone users.

MSR research has typically focused on data that was already available and stored in software repositories. Green mining must accommodate the fact that power use data is missing from these repositories and thus must be generated. To generate power use data one must typically 1) write a test; 2) setup a test environment; 3) run an application; 4) monitor its power use; and then, 5) record, report, and aggregate the results with other test runs. There is noise and some level of non-determinism in power instrumentation and thus many measurements must be combined into a single estimate. These factors make testing for power performance difficult. The need for running repeated tests means that it is also time consuming; many of these observations have been confirmed by Dong et al. [3].

[1] Launchpad `https://launchpad.net/`
[2] Mozilla Smoke Tests: `https://moztrap.mozilla.org/results/runs/`
[3] CPAN Smoke Tests: `http://www.cpantesters.org/distro/S/Slurp.html`

In this paper we propose the first dedicated hardware/-software mining software repositories test harness. This is a dedicated hardware testbed meant to enable the task of green mining: the study of relationships between software change and software power use. While others such as Shang et al. [19], Dyer et al. [4], and Gousios et al. [7] have leveraged cloud computing infrastructure such as Hadoop, mongodb, mysql and even bittorrent to store, distribute, collect, and execute large queries on MSR-relevant data, cloud computing is less than helpful to green mining. Virtualization and power estimation tend to rely on counter based models [9, 1] to estimate supposed power use rather than measuring actual power use. Thus green mining faces an interesting limitation in that instrumentation of virtualized computing is currently limited and difficult. Thus, one must resort to physically instrumented computers to measure power use.

We propose a dedicated hardware test harness that uses physical instrumentation of the power circuitry of the measured mobile devices (Android smartphones) to automate the power use testing of software such as apps that run on Android. This **Green Miner** framework [4] includes software components that schedule and run large test jobs spanning 100 or more revisions and execute numerous tests. Furthermore smartphones are not the easiest devices to use and thus automating their use is extremely helpful for testing. The Green Miner framework can aggregate, analyze and graph the results allowing researchers to poke and prod at the data both directly and visually.

Because of the lack of virtualization support, the need to run many 1000s of tests quickly, the lack of pre-existing data, the data processing requirements, the need to obtain results quickly, the need to change tests often, and the need for stable instrumentation, we built the Green Miner framework.

Our contributions include:

- The introduction of the first dedicated hardware MSR-based test harness: Green Miner.
- A description of Green Miner, and instructions on how to build Green Miner.
- Advice for anyone attempting Green Mining.
- Case studies showing how one can use Green Miner for MSR experiments.
- An investigation of how the visual aspects of a UI affect software power use.

2. PREVIOUS WORK

There is much previous work relevant to special configurations for mining, mining for performance, measuring energy consumption and web-visualization of MSR data.

Energy is the cost or the ability to do work, and is measured in watts while power is the rate at which energy is converted or consumed, and is measured in joules. One joule is exactly equal to one watt for one second. Executing a task on a mobile device consumes energy, of which there is a limited amount stored in the device's battery. While the task is being completed, the device draws power from the battery which is the rate that the energy is being consumed. Mean watts refer to the mean power draw required during a task, while joules refer to the total energy consumed by the end of the task. For example, mean watts determines how many minutes a user can listen to music on their mobile device before the battery is depleted, while joules determines how much energy is consumed simply by starting their music player application.

[4]Green Miner is available here! `http://greenminer.softwareprocess.es/`

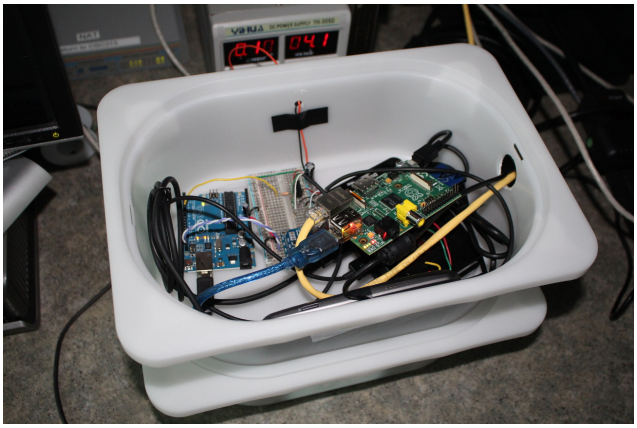

Figure 1: The innards of a Green Miner hardware client unit. Consisting of an Arduino, a breadboard with an INA219 chip, a Raspberry Pi running the Green Miner client, a USB hub, and a Galaxy Nexus phone which is connected to a high-current 4.1V DC Power Supply.

2.1 MSR Miners

In MSR there are many examples of researchers using cloud computing, distributed computing, and cloud computing frameworks like Hadoop and pig [18] to answer MSR relevant queries on large MSR data sets.

GHTorrent [7] is an excellent example of a MSR collection rig that collects and stores data from the Github "Firehose." It uses dedicated computers to monitor, store, and distribute Github data that is currently larger than 100GB. GHTorrent is an example of using distributed or cloud computing to collect and serve MSR data and requests.

BOA [4] is a query system that can aggregate, count and query a wide range of already extracted repositories using a distributed map-reduce query language invented for the task of mining software repositories. It compiles down in to map-reduce jobs using Hadoop and then executes these jobs on many cloud computers using the Hadoop framework.

Sheng et al. [19] have used Hadoop within their own research lab to answer large questions using the idle processing power of lab computers. In particular they investigated log message evolution within Hadoop itself.

2.2 Power Use

There is much work in terms of Android power use. PowerTutor by Zhang et al. [21] is an Android software power monitor that augments ACPI power readings with machine learning to improve accuracy. ACPI readings can be problematic because ACPI relies on chipsets and circuitry which have only the minimal instrumentation required by mobile device end-users.

Powertop [13] is a popular power monitor that runs on any GNU/Linux computer with ACPI. It reports the processes and drivers that produce events and CPU wake-ups. Based on this information it tries to estimate energy con-

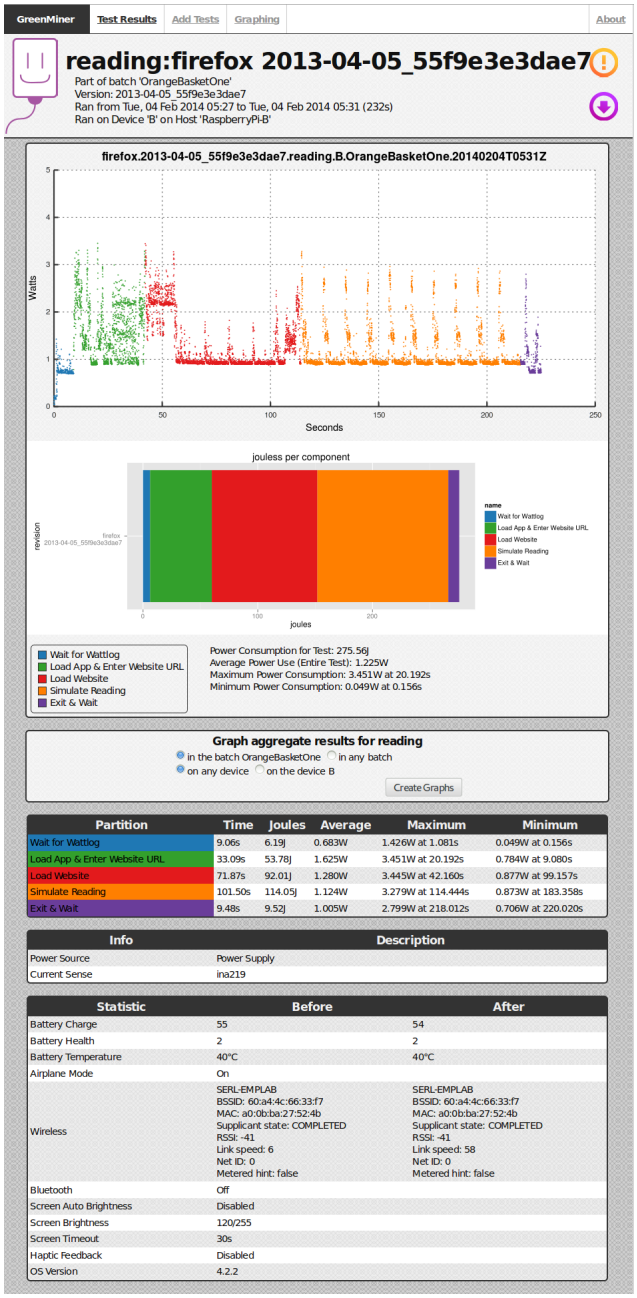

Figure 2: Green Miner Web service: Easy to use test results.

sumption. Powertop is used by industry, notably by Michael Larabel [14] who executed studies, similar to green mining, on various versions of Ubuntu.

Hao et al. [10] leveraged program analysis and heavily instrumented Android devices to benchmark Java VM bytecode. They then estimated method power use based on those benchmarks. The device they employed was not an Android phone but rather a dedicated Android testing device instrumented by the manufacturer. Green Miner differs because it can potentially measure any mobile device that exposes access to a DC power supply or battery.

2.3 MSR and Performance

Gupta's [8] work on mining Windows Phone 7 energy consumption traces centres on determining which modules are causing power usage. They generated these traces through hardware instrumentation while repeatedly running benchmarks of calls. Their work differs from ours because multiple Green Miner testers are used and they are controlled by a software controller that queues and runs test cases.

Sheng et al. [19] were primarily studying performance leveraging logs and the evolution of log messages. They had to build and test many versions of the software, much like our frameworks had to.

Hindle's [12, 11] work on Green Mining combined MSR research and power use performance research to study software change and evolution relevant to power use performance.

2.4 MSR and Visualization

One of the most successful MSR fields is that of visualization. It is widely adopted on the web and utilized in industry by companies such as Github. [5] Many of these visualizations end up embedded into web services that allow querying and reporting.

Sakamoto et al. [17] describe a web framework for browsing and visualizing MSR-relevant data. They provide views of metrics and rely on Google charts.

D'Ambros et al. [5] describe moving some of their visualization framework to the web and difficulties involved. They provide a large number of visualizations based on the very powerful Churrasco and Small Project Observatory [15]. The Small Project Observatory allows for the visualization of data from multiple software development projects and thus is quite relevant since Green Miner is meant to compare between versions and applications.

3. TEST HARNESS

With the goal of running a variety of automated power usage tests on Android, a test harness was developed which automatically executes Android application tests using their user interface while measuring their power use. The test harness comprises a single web server for storing data and scheduling test runs, and four clients which run tests. Each client system includes a Raspberry Pi for collecting, uploading data, and starting tests; a Galaxy Nexus running Android OS 4.2.2; an Arduino Uno; and an Adafruit INA219 current sensor for monitoring power usage. In place of a battery, each Galaxy Nexus is connected via the current sensor to a lab bench DC power supply, set to supply 4.1V. This setup removes one possible source of variation in the test data – battery voltage and internal resistance. The Arduino records the amperage, wattage, and voltage fifty times per second during testing, with a resolution of approximately 0.1 milliamps, and 0.01 volts.

3.1 Equipment

The physical setup of one of the Green Miner clients is depicted in Figure 1. The DC power supply is glowing in the background and the Galaxy Nexus, Raspberry Pi, INA219, USB Hub, and Arduino are all visible inside of the white plastic container.

[5]For an example of Github's visualizations see: https://github.com/abramhindle/mostitch/graphs

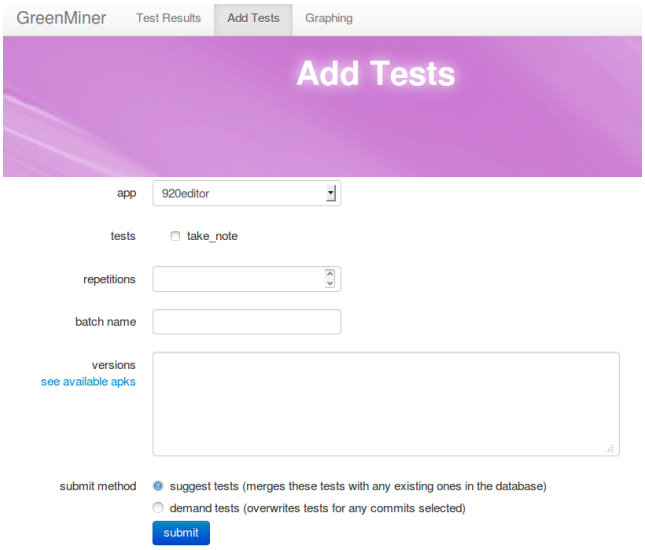

Figure 3: The Green Miner Web service allows tests to be scheduled easily.

A Raspberry Pi was used for collecting data, uploading data, and executing tests on an Android device. The Raspberry Pi was chosen because it is a low-power ARM computer that can run GNU/Linux, and thus it could run ADB (Android Debug Bridge) as well as the scripts necessary to interact with our Green Miner web service. The Raspberry Pi was somewhat problematic because it needed to be provided with a steady 5V USB power source. Plugable's 7 Port High Speed USB 2.0 Hub with 3A Power Adapter was powerful enough to handle the Raspberry Pi. This powered the Raspberry Pi and attached Arduino without problems. Transcend 8GB SD Cards were used for the Pi's file system. The Raspberry Pi's I2C pins are not suitable for instrumentation, so the Arduino is connected to the INA219 instead.

The Android device we chose was the Samsung Galaxy Nexus running Android OS 4.2.2. Originally we instrumented the battery itself, but it was found that batteries lead to unstable measurements due to large variations in voltage and internal resistance, thus it was necessary to simulate a battery by using a high-current DC power supply. The batteries were removed from the Galaxy Nexus and wires were soldered directly to the battery contacts. When provided 4.1V from the power supply the phone's software was not aware that the battery had been removed. The power leads are connected to the Adafruit INA219 IC to measure power usage. The phone was connected via the USB hub to the Raspberry Pi so that the Raspberry Pi could send applications and UI test scripts to it. However this connection can be interrupted by a transistor controlled by the Arduino, which itself is controlled by the Pi.

The original method for measuring current was to measure the drop in voltage across a shunt resistor and amplify that signal for an ADC on the Arduino to report, and then take a parallel ADC reading for voltage. To measure wattage one needs both amperage and voltage. Unfortunately this led to a complicated tuning setup, so the INA219 current sensor was used instead. The INA219 is an IC that measures voltage and current and reports the values over I2C (a digital serial protocol used to communicate serial data between ICs). The INA219 came with its own shunt and was far more reliable and needed far less tuning. Other amplifiers that were used seemed to be temperature sensitive. The INA219 chip reports current and voltage measurements over I2C to the Arduino. This instrumentation is desirable because it doesn't depend on the instruments and software in the phone. 1 INA219 was used per Android phone.

The Arduino was an Arduino Uno flashed with our custom software that fulfilled two tasks: switching on and off the USB connection for the phone and reading the power measurements via I2C from the INA219 chip. Both of these functions are ultimately controlled and reported to to the Raspberry Pi by a USB Serial connection. The Arduino Uno is pictured in Figure 4 connecting to the INA219. Figure 5 depicts the Arduino's circuit for enabling and disabling the USB connection to the phone using a TIP127 transistor. By default the smartphone is connected, but if the Arduino signals the transistor will block the USB connection. This is done in order to prevent the phone from using USB (charging) during tests. 1 Arduino was used per phone.

The power supply used was a Weber Displays YIHUA 305D Lab Grade 30V 5A Digital Precision DC Adjustable Power Supply. Multiple phones can be powered with 1 power supply (2-3). The voltage is stable. Even when the phone uses more power than could have been provided over USB, the voltage is consistent. The power supply's output was set to 4.1V, the upper end of the voltage that a nearly charged battery produces. Some apps, when combined with the phone's cellular network radio, would cause the phones to draw a high current briefly. This high current is easy for a cellphone battery to handle but hard for a DC power supply to handle, and impossible to provide over USB. Thus, a powerful power supply that could handle the peak current of two phones was chosen. In our testing, smaller power supplies could not handle powering the phones and the phones would turn off. A capacitor of 1000µF was placed across the power supply leads, acting as a low-pass filter and current buffer for the power supplied to the phone.

Thus each client required a USB hub, a Raspberry Pi, an Arduino Uno, an INA219, a suitable transistor, an Android phone and a stable source of DC power at typical battery voltages.

3.2 Power Measurement

Voltage and amperage measurement is the task of the INA219 IC. It internally averages 32 readings to produce 50 readings per second. The resolution is 0.01V for voltage and 0.1mA amperage measurements. The INA219 was also set up to read voltages between 0 and 16V at an amperage range of approximately 0 to 1.3A. The Arduino Uno uses these 50 readings per second to calculate a wattage from the root mean squared of the V and A signals, reporting over USB serial 50 times a second to the Raspberry Pi 3 values: the current Wattage, mA, V and the Arduino's unique name. These results are recorded by the Raspberry Pi. The Raspberry Pi then packages this stream of data into a test report that includes other metadata such as the timing of the test, the state of the phone before and after the test, the totals, the measurements per task, which device was used, which Arduino was used, the name of the test, the run number, the application under test, the version of the appli-

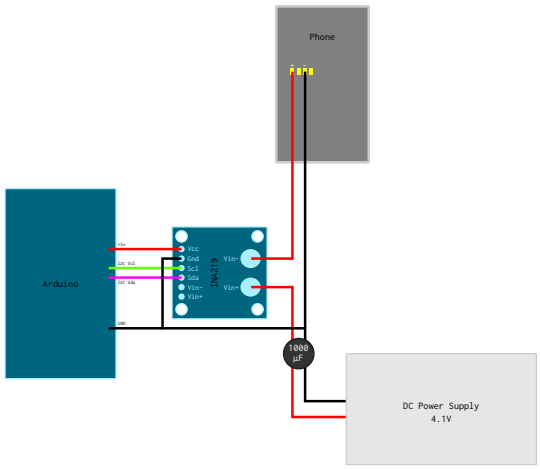

Figure 4: INA219 Circuit for power measurement from a Galaxy Nexus smartphone

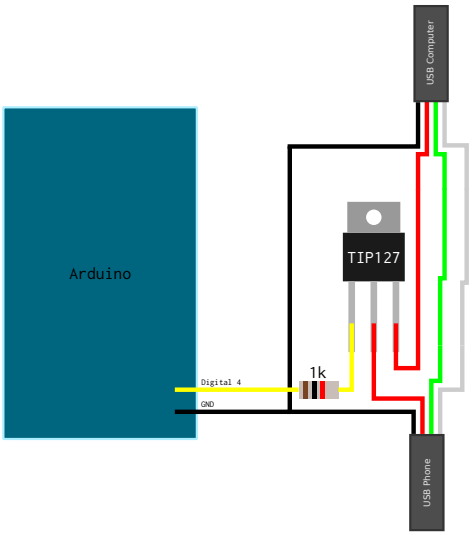

Figure 5: Transistor Switch Circuit for connecting and disconnecting the USB cable between the Raspberry Pi and the smartphone.

cation, the configuration of the test script, etc. All of this information plus the readings is then packaged and submitted to the web service. Multiple measurement clients can run simultaneously, thus allowing multiple parallel tests.

3.3 Test Definition

Tests are defined using the rudimentary Unix shell available on Android. These scripts are uploaded to the Android device prior to a test being run, and started just before the USB is disconnected. A number of command-line utilities are used to launch apps, create input events, and wait. Some of these command line utilities (such as 'am' and 'pm') are shipped standard with Android. Some of the utilities were custom-written in C for these experiments, including a high-precision sleep routine and input event injector. Each test is divided into a number of components, so that the different phases of the test (e.g., starting the app, loading a webpage, etc.) can be analyzed separately. These components are visible by different colours in the easy-to-read power use report on the web service as shown in Figure 2. In addition, each test can perform actions on the Pi before and after the Android script is run.

The test code is distributed to the Raspberry Pi devices via Git. When a test is scheduled by the web service and the Raspberry Pi client agrees to run the test, it is deployed from its test repository onto the Android device.

Test requests are posted, processed and sub-jobs are given out to available Raspberry Pis by the web service. The clients distribute the appropriate version and test code to the Android device under test. The Raspberry Pi test client monitors, records, annotates and uploads the test results back to the Green Miner web service.

3.4 Test Device Specialization

The benefit of using only one model of phone is that the hardware and operating system are the same. We also instrumented a Samsung Galaxy S2, however, it has a different screen resolution than the Galaxy Nexus phones. Tests have to be specialized and made device specific by adjusting, for example, touch locations on the screen. If tablets were instrumented, one would need to address the different UI layout and sizes in the test definitions.

4. GREEN MINER WEB SERVICE

The Green Miner web service enables the distribution and collection of green mining tests and their results. The web service is responsible for the aggregation, persistence, storage, analysis of test data, as well as scheduling test runs. It also plots test results and provides access to raw and aggregate test data.

4.1 Individual Test Report

The web service's main page lists recent tests and allows one to click and read the report about an individual test run. An example of such a report is depicted in Figure 2.

This report contains a trace of the power use over time, partitioned by testing task. Each partition is summarized by mean-watts and joules consumed as well as its task label. A stacked plot shows proportionally each task's energy consumption relative to the whole test. The type of test, how the test was powered and meta-data before and after the test run is also listed. This meta-data includes the current WiFi state at the beginning and end of the test, because if a WiFi router crashes results may be invalid.

At the top of the report shown in Figure 2 details about the software and the test run are provided. The exclamation mark button at the side is clickable and will flag a test as bad. If there was an error during testing and the result should not be used (machinery crashed, errors, debugging) then the flagging of a test will remove it from the test list and any analysis. The test can be manually recovered if necessary. The other button is a download button that enables anyone browsing to download the actual meta-data of the test so they may perform their own analysis.

4.1.1 Job Queue

One important task of the web service is help a researcher or developer to define a batch of tests to be run and analyzed. The Green Miner web service provides a user interface to schedule such tests. Figure 3 depicts the UI for adding tests

to be scheduled and run by the test clients on the Android devices. One first selects the product one wants to test. The product will be associated with a set of tests that can be run. These tests can be selected by clicking the checkboxes of the tests. Then the number of repetitions can be set so that tests can be run multiple times for each application version to ensure safe estimates of power use. The next field allows one to choose application versions to be tested.

For programs like Fennec, build jobs ran on other machines that then uploaded binaries of each Fennec version to the web service. For instance, Figure 6 depicts 685 versions of Firefox Fennec, queued up and tested by submitting one of these test request forms.

Once a test batch is submitted, it is broken up into component tests and distributed to running clients. Clients take the tests and report back the results, removing the test from the test queue. Test run order is randomized to reduce biases such as caching, temperature, and time of day.

4.2 Visualizations

Once a few tests have been run most developers and researchers want to see results. One test result tells part of a story but it does not show trends or evolution. Aggregation of many tests is the corner stone of green mining, thus no Green Miner would be complete without aggregation, reporting and visualization.

The most basic visualizations produced are individual test reports such as the Firefox test report in Figure 2. However, the Green Miner web service provides many kinds of aggregate visualizations, spanning more than one test. Both predefined and custom aggregate visualizations are available. Once a request for aggregate results is made, either by the main menu as shown in Figure 2, or by submitting an aggregate query, the user is shown the results of the aggregation as they are being generated, as in Figure 12. Sometimes aggregation is time consuming because thousands of time-series are being analyzed. To produce Figure 7, over 8000 tests were aggregated by the web service. For custom analysis, the Green Miner web service supplies the underlying data behind the aggregate plots in CSV format. In fact, these CSV datasets are available by URL on the Green Miner web service, enabling 3rd party web services to analyze and visualize the data. This includes visualization frameworks and web services such as `d3.js` [6] or Google charts [17] to plot the raw data.

Figure 7 shows 685 box-plots, ordered by application version, of the distribution of mean watts and joules consumed for the web page reading test on Firefox Fennec. There are large areas of little variation, but there are also extreme changes. Sometimes these extreme changes are caused by the application or the test case crashing. Plots like Figure 7 provide an immediate summary of results and allow developers and researchers to decide upon further analysis or testing.

Figure 6 shows 2 views of 8600+ tests run on 685 versions of Firefox Fennec. These plots are ordered by test task and then by version: each test task is shown as a range of colourful box plots from orange to blue to pink, where orange represents early versions of Fennec and pink represents later versions of Fennec. The top plot shows the mean watts used during that portion of the test. The bottom plot shows the energy consumed during each task of the test. The data in the bottom plot is equivalent to the data in the top plot times the time it took to complete a task. Note that the high power (top plot) portions of the test obviously do not run as long as the lower-power portions and thus they consume less energy (bottom plot) than the longer-running portions. An interesting observation that can be made from the Mean Watts plot in Figure 6 is the obvious tail power state [16] that occurs after application exit.

Other visualizations available not shown for space concerns include:

- T-test similarity matrices, showing whether different application versions are statistically significantly different or not.
- Stacked box-plots of energy consumed by each task, relative and absolute.
- Application similarity matrices, showing whether any pair of applications are similar based on a pairwise t-test.
- Run count plots that show how many runs each application version received in a certain batch.
- Per-device aggregate plots comparing test clients.

4.3 Multi-System Measurement Normalization

Due to manufacture variations and error tolerances within electrical components not all devices are created equal, even if they are the same model. Thus to parallelize test results and normalize the results 1 phone, client, testbed, and instrumentation, is chosen to act as the gold standard phone to which all other data is normalized. This is done by creating linear regression models and using them to normalize the mean watts, joules and mean amps readings of data from other devices. The slope of these models, for mean watts, tends to range from 0.94 to 0.99 (almost identical) and the intercepts range from -0.04 W to to 0.02 W. The constant offset given by the intercept is often the majority of the correction needed to normalize the measured results. The R^2 values are above 0.95 for our linear regression models.

5. PIXEL POWER CASE STUDY

To demonstrate the kind of experiments Green Miner is useful for a full case study that studied Firefox's UI evolution was attempted.

One candidate for power optimization, as shown by Zhong et al.[22] is the graphical user interface (GUI). One of the goals of Green Mining is to estimate the power use of software without compilation or testing; to meet this goal with respect to user interface design, a firm correlation between user interface change and energy consumption must first be established. Although the power usage of organic LED (OLED) displays has already been successfully modelled from a software perspective [2], a question remains regarding how small changes in the user interface of a real application can affect energy consumption. In this paper we verify the relationship between OLED display content and power consumption, and use this result as the basis for a study of the power consumption and evolution of the user interface in the Firefox browser on Android.

5.1 Test Description

Our initial test was designed to measure the impact on power consumption of both high contrast changes (white vs. black backgrounds), and the evolving Firefox GUI. Firefox was launched and navigated to two pages, one after the other.

[6] D3.js `http://d3js.org`

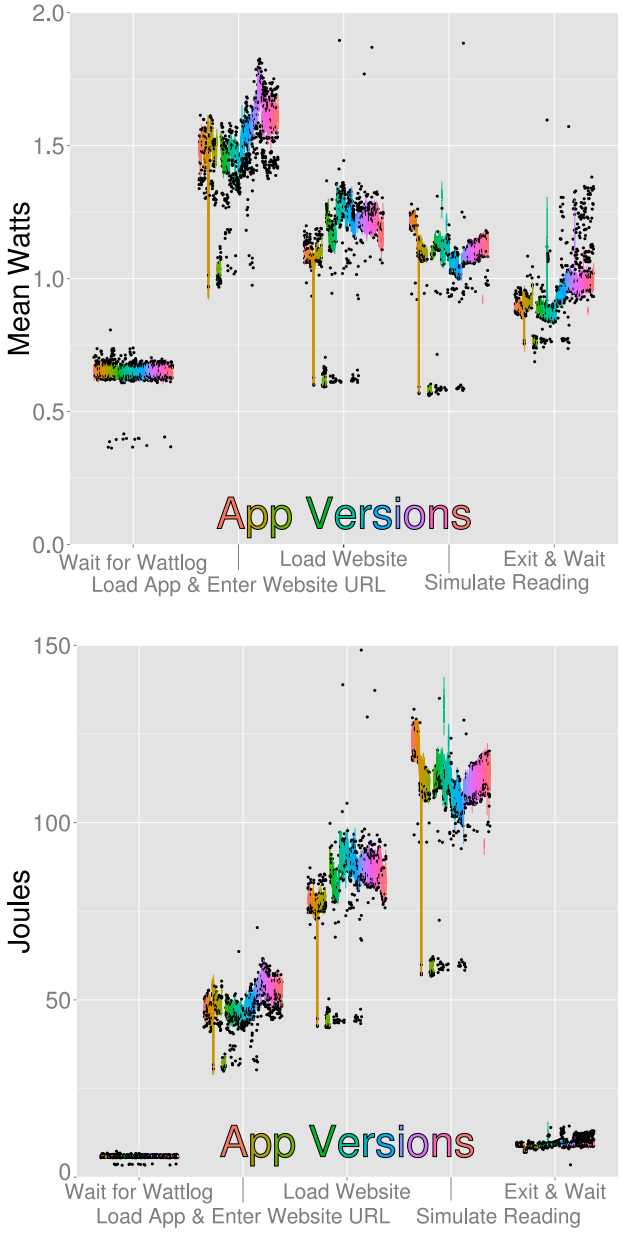

Figure 6: 685 Versions of Firefox Fennec, each tested more than 10 times with a webpage reading test, both mean watts and joules, separated by test task. This is output directly from Green Miner.

After waiting long enough to be reasonably certain that the page would be finished loading, the phone remained idle on the page for 120 seconds, while its power usage was monitored. Each page displays the same text, one in black with a white background, and the other with the colours inverted.

For our testing, we selected a number of Android binaries from Mozilla's official nightly builds of Firefox for Android. The versions used were chosen such that they represented every GUI change that might affect the test. For each change, the first version to include the change was selected, along with the most recent version that did not. This configuration

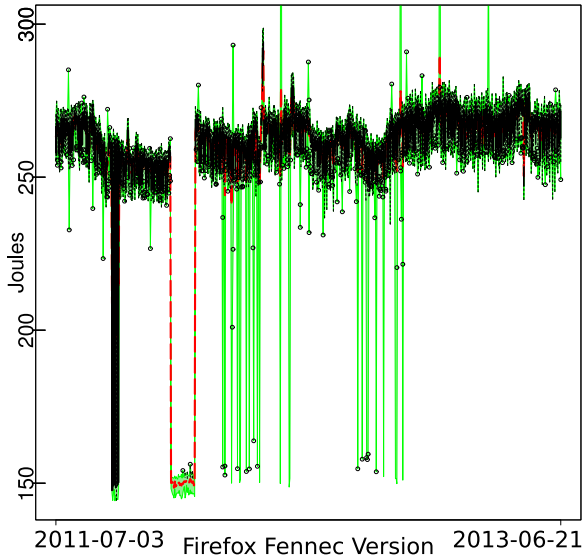

Figure 7: 685 Versions of Firefox Fennec, each tested more than 10 times each with a webpage reading test, showing joules consumed for the entire test. This is output directly from Green Miner.

allows for paired t-testing, and was designed to eliminate the majority of other software change effects. Because Firefox has a relatively minimal user interface, serving mainly as a frame to the webpages it displays, the interface evolution among all of the tested versions can easily be summarized and is shown in Figure 8.

Once these tests were completed, the question arose of whether any potential power differences caused by changes to the GUI were being dominated by changes to the software. To answer this question, a similar test was created where screenshots of each Firefox version displaying the black text on a white background were opened in Android's built-in image viewer. By replacing Firefox with the image viewer, this test removes any variation in energy consumption which might be produced by Firefox changes unrelated to the GUI design.

5.2 Methodology

The Green Miner test harness was utilized for this experiment. Screen timeout on the Android devices is set to 30 minutes, and screen brightness is set to 120 out of 255. Each Firefox version was repeatedly installed, ran, tested, and uninstalled. Uninstalling helps avoid cache effects. Finally, the data was uploaded to the Green Miner web service.

5.3 Results

Our original test allowed for two different statistical analyses: testing for a difference in mean power consumption between black text on white and white text on black, and testing for a difference in mean power consumption between versions. For the first result, mean difference in mean power consumption in watts between displaying black text on a white background and white text on a black background, a paired student's t-test produces a 99.999% confidence interval of -0.2352W to -0.2352W: the white text on a black background uses at least 0.2W less than black text on white. This is an encouraging result which offers very strong sup-

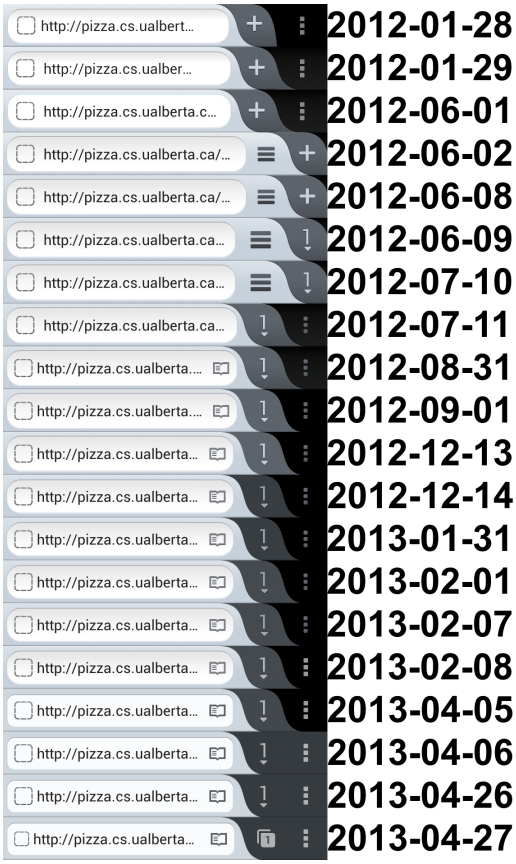

Figure 8: Firefox Android Fennec GUI Evolution.

port for the link between the display contents and energy consumption, and serves as a motivating example for the exploration of GUI energy consumption. This result also cross-validates with the results of Dong et. al.[2].

The second statistical analysis of this data attempts to determine which changes to the Firefox GUI produce a significant change in power consumption. By testing power consumption of either black on white or white on black text across each GUI change, we hope to determine which GUI changes, if any, might cause a significant change. With this data, however, no changes yield statistically significant results. Some changes produce an apparent difference in means under a student's t-test, but with too much variance within the data to produce acceptably low p-values.

When considering these results, in light of the white vs. black text results, we were still unsatisfied with our ability to answer the question of how these GUI changes were affecting power consumption. This prompted us to create a new test using the image gallery application to display screenshots of each Firefox version, thus isolating GUI changes from software changes. However, when running this test on multiple devices with a medium level of brightness, the results were still lacking in significance. With our final image gallery test, we attempted to further reduce any possible sources of variation beyond the display content. Only a single device was used, and the screen brightness was set to the device maximum so that display contents might be more of a dominant factor.

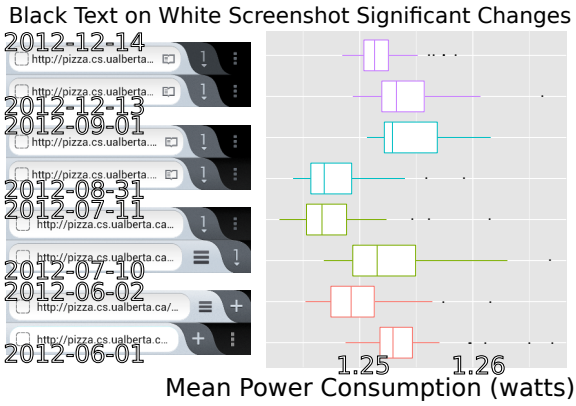

Figure 9: This graph combines box-plots for mean watts in the gallery test, with the relevant Firefox UI on the left. Only significant changes are shown.

With 35 runs per Firefox version, these final results show four of the ten changes as being significantly different in a two-sided student's t-test with p-values below 0.05. More interesting is that although the two changes we suspected would have the largest impact (2012-06-01/02, 2012-07-10/11) were indeed significant, the first of these produced an estimate for the mean difference which depicts a decrease in power consumption, as opposed to the increase we expected. In addition, two other changes came back as significant (2012-08-31/09-01, 2012-12-13/14) which constituted only minor changes in GUI, which we did not expect to be significant. All results are summarized in Figure 10, and Figure 9 contains only the changes we found to be statistically significant. Note that, because website rendering changed several times across the UIs tested, comparisons between GUI revisions which are not paired together may not be valid.

5.4 Conclusion: Pixel Power

With our initial familiarity of power use on OLED displays, the significant GUI changes were still surprising. This result reinforces the need for automated analysis of UI changes for energy efficiency because we have shown with the help of Green Miner that some UI changes have a significant effect on the power use of an application.

6. DISCUSSION

6.1 Industrial Success

The Green Miner framework lead to one initial industrial success! During one experiment on the efficiency of different applications executing the same task, one of the authors of this paper, Kent Rasmussen, found that one of the popular Reddit reader apps, had abnormally high power usage while browsing Reddit, even compared to web browsers. Figure 11 shows the anomalous behaviour in red. While adding a comment to a Reddit discussion, the power use was quite high compared to reading the article, suggesting something was amiss. Armed with data extracted by the Green Miner framework, this possible performance issue was reported to the developers who then investigated the cause. The developers were grateful and provided a patched version where the regression had been solved (depicted as the blue behaviour in Figure 11). The developers said that old code that used

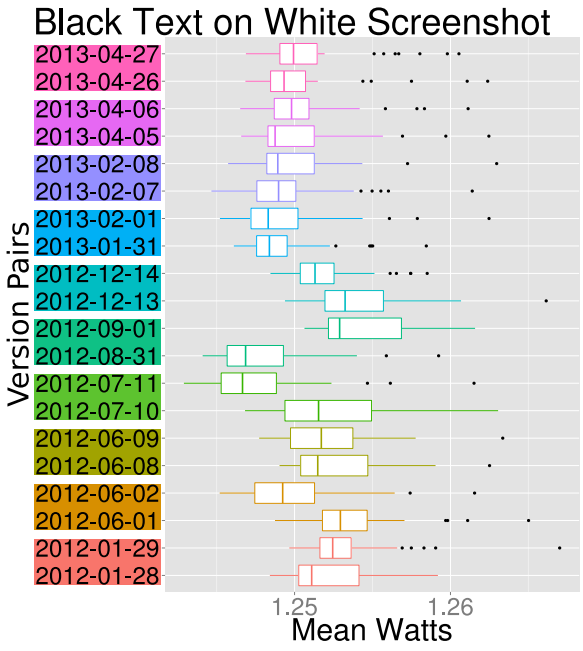

Figure 10: This graph displays box-plots for mean watts in the gallery test, on a single device. Results in each pair were t-tested for significant difference.

the GlobalLayoutListener was being called excessively leading to increased CPU usage, thus the regression was solved by removing the redundant code. Thus the Green Miner framework helped debug a regression in a popular commercial mobile app simply by comparing the power use of similar tasks!

6.2 Lessons Learned

Improve reliability with strong physical connections. Initially we tried to emulate a battery by building objects that had the same dimensions as batteries so they could slip inside of an Android phone. This was problematic because the connection between the fake battery and the phone was sensitive to movement, vibration, heat and was prone to popping out. By soldering wires directly to the phone's battery

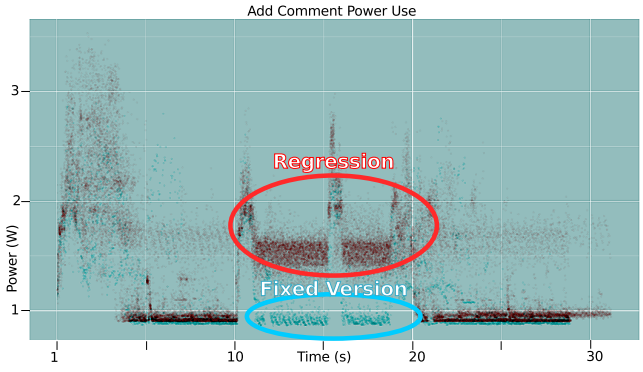

Figure 11: Green Miner results showing the difference between two versions of a popular Reddit application.

contacts we guaranteed a good physical connection. This enabled us to power the phone without using USB, which is too limited to successfully use a phone without a battery.

Batteries are not reliable. The goal was to simulate regular operating environments and in doing so the battery, built-in circuitry, and ACPI interface were originally used. Unfortunately, neither the batteries, nor the energy estimation circuitry attached to them were sufficiently reliable for Green Mining. For instance, recharging a battery does not always return it to its former level of charge. Initially, recharge scripts were used that would recharge batteries to 90% between tests, but over time the maximum charge of the battery would dip lower and lower. Additionally, the voltage did not remain constant within each test run. This meant that the watts used dropped over time as the battery was producing less voltage, in turn causing undesirable artifacts in the resulting data. Because of these problems, relying on batteries led to results that were erratic and hard to reproduce. The batteries were initially replaced with AC to DC wall socket (also known as "wall wart") power supplies, but these supplies could not handle the high current spikes that phones often require. Batteries can handle these spikes reliably but many wall socket transformers cannot handle such a fast change in load. Eventually the batteries were replaced with a stable lab bench DC power supply that was hardy enough to handle the load of multiple phones.

The voltage of a charged battery must be measured before replacing it with a DC power supply. The batteries used were rated for 3.7V but in fact, 3.7V was the low end of the voltage expected by the phone. A supply of 4.1V worked well and caused the phone to think that the fake battery was of moderate health.

Raspberry Pis need to be properly powered. If Raspberry Pi computers do not have a strong power supply they will crash and reboot during certain tasks or in certain configurations. An externally powered USB hub from Plugable solved this problem for this instance of the testbed.

Ground differences matter. The difference between 2 grounds can be measured with a multimeter. An early setup consisted of multiple devices hooked into one computer, but when the ground difference was measured it was over 0.5V. This meant that the grounds were not at 0V but they were 0.5V away from each other. This was a concern to our setup because it means that the 0V measured by an instrument might not be accurate, it could be off by 0.5V in either direction. Additionally the difference in voltage causes a current between the two grounds which also disrupts the current measured by an instrument. Thus wattage calculations could be very far from the actual wattage being demanded by the phone. The issue of a single machine with different ground potentials was solved by by using a single computer for each testbed and wiring the ground planes of every device (phone, Arduino, hub, pi, power supply, and power meter) in a testbed together.

There is error in everything and even the same model of a device can behave differently. Every resistor and every IC tend to be published with spec-sheets that describe their error or tolerance to error. Error free manufacturing and calibration is impossible, thus given 4 Galaxy Nexus phones they will have slightly different components, or even if they have the same components, manufacturing error margins dictate they will act slightly differently. Resistors are commonly manufactured with error margins of 5% or 1% and the ag-

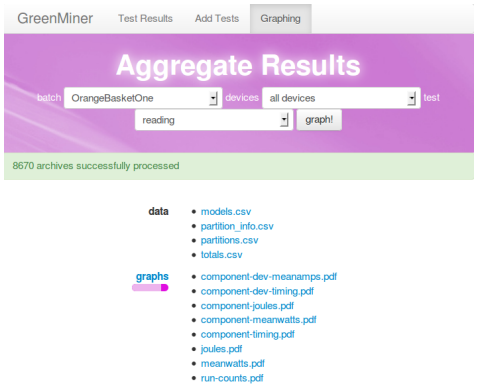

Figure 12: Green Miner Aggregate Results provide both plots of the aggregates as well as the raw data.

gregation of these components within something like an Android device will mean that different devices will produce slightly different power measurements even if they are all the same model. This caused the need for power measurement normalization. In our case we took 1 device as the ideal and then made linear models for the other devices to emulate the ideal device. The linear model will scale wattage appropriately to deal with these possible unavoidable manufacturing variations and variations in our testbeds.

7. CONCLUSIONS

In conclusion we presented the plans, design, and implementation of the first dedicated hardware embedded MSR test harness: Green Miner. Furthermore we discussed practical experiences and problems that were encountered and could be encountered by other interested developers and researchers.

The Green Miner is a hardware / software framework that physically measures power use, while also adding a continuous-integration-like test queuing system to it, as well as a data aggregation and an analysis suite to interpret the extracted data. Because of its parallel and continuous properties we can run power tests very quickly and in less time than before.

To aide parallel testing we came up with a simple linear model technique to normalize measured data based on the biases and errors inherent in the devices under test. Manufacturing is not perfect and we need to normalize the measurements to address this issue and enable parallelism.

We demonstrated the effectiveness of the framework on the problem of studying UI evolution, and found a few instances where UI changes could have lead to a change in energy consumption.

8. ACKNOWLEDGEMENTS

Abram Hindle is supported by an NSERC Discovery Grant. E.J. Barlow was supported by an NSERC USRA Grant. Thanks to Taras Glek from Mozilla for commentary.

9. REFERENCES

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

[2] M. Dong, Y.-S. K. Choi, and L. Zhong. Power modeling of graphical user interfaces on oled displays. In *Proceedings of the 46th Annual Design Automation Conference*, DAC '09, pages 652–657, New York, NY, USA, 2009. ACM.

[3] M. Dong and L. Zhong. Self-constructive, high-rate energy modeling for battery-powered mobile systems. In *Proc. ACM/USENIX Int. Conf. Mobile Systems, Applications, and Services (MobiSys)*, June 2011.

[4] R. Dyer, H. A. Nguyen, H. Rajan, and T. N. Nguyen. Boa: a language and infrastructure for analyzing ultra-large-scale software repositories. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 422–431. IEEE Press, 2013.

[5] M. D'Ambros and M. Lanza. Distributed and collaborative software evolution analysis with churrasco. *Science of Computer Programming*, 75(4):276–287, 2010.

[6] Google. Juice defender - android apps on google play, August 22 2013. `https://play.google.com/store/apps/details?id=com.latedroid.juicedefender&hl=en`.

[7] G. Gousios and D. Spinellis. Ghtorrent: Github's data from a firehose. In *Mining Software Repositories (MSR), 9th IEEE Working Conference on*, pages 12–21. IEEE, 2012.

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

[9] S. Gurumurthi, A. Sivasubramaniam, M. Irwin, N. Vijaykrishnan, and M. Kandemir. Using complete machine simulation for software power estimation: the SoftWatt approach. In *Proc. of 8th Int. Symp. High-Performance Computer Architecture*, Feb 2002.

[10] 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*, ICSE '13, pages 92–101, Piscataway, NJ, USA, 2013. IEEE Press.

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

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

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

[14] M. Larabel. Ubuntu's power consumption tested. `http://www.phoronix.com/scan.php?page=article&item=878`, Oct 2007.

[15] M. Lungu, M. Lanza, T. Gîrba, and R. Robbes. The small project observatory: Visualizing software ecosystems. *Science of Computer Programming*, 75(4):264–275, 2010.

[16] A. Pathak, Y. C. Hu, M. Zhang, P. Bahl, and Y.-M. Wang. Fine-Grained Power Modeling for Smartphones using System Call Tracing. In *Proceedings of the sixth conference on Computer systems*, EuroSys '11, pages 153–168, New York, NY, USA, 2011. ACM.

[17] Y. Sakamoto, S. Matsumoto, and M. Nakamura. Integrating service oriented msr framework and google chart tools for visualizing software evolution. In *4th Int. Workshop on IWESEP*, pages 35–39. IEEE, 2012.

[18] W. Shang, B. Adams, and A. E. Hassan. Using pig as a data preparation language for large-scale mining software repositories studies: An experience report. *Journal of Systems and Software*, 85(10):2195–2204, 2012.

[19] W. Shang, Z. M. Jiang, B. Adams, A. E. Hassan, M. W. Godfrey, M. N. Nasser, and P. Flora. An exploratory study of the evolution of communicated information about the execution of large software systems. In *WCRE*, pages 335–344, 2011.

[20] A. Smith. Smartphone ownership 2013. June 5 2013. `http://pewinternet.org/Reports/2013/Smartphone-Ownership-2013/Findings.aspx`.

[21] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, and L. Yang. Accurate online power estimation and automatic battery behavior based power model generation for smartphones. In *Proceedings of CODES/ISSS'10*, pages 105–114, New York, NY, USA, 2010. ACM.

[22] L. Zhong and N. K. Jha. Graphical user interface energy characterization for handheld computers. In *Proceedings of the 2003 International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, CASES '03, pages 232–242, New York, NY, USA, 2003. ACM.