Using System Calls to Predict Changes in Application Energy Consumption Profiles

by

Karan Aggarwal

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

University of Alberta

Abstract

Battery is a limited resource that affects the smartphone availability. Hence, it is the responsibility of developers to develop and maintain energy efficient applications to enhance end user satisfaction. As the impact of software code change on energy consumption is not known, developers need special instrumentation to assess the impact of change on their application's energy consumption profile. Unfortunately, this instrumentation is costly and generally not available. In order to address this issue, we use system calls to predict the impact of code changes on the energy consumption profile. We find that significant changes to system call profiles often induce significant changes in energy consumption. Using this simple observation, we introduce *GreenAdvisor*, a first of its kind tool that systematically records and analyzes an application's system-calls in order to predict whether the energy-consumption profile of an application has changed. The *GreenAdvisor* tool was evaluated using a user study whereby numerous software teams, whose members used *GreenAdvisor* while developing Android applications to examine the energy-consumption impact of selected commits from the teams' projects. The evaluation confirms the usefulness of our tool in assisting developers analyze and understand the energy-consumption profile changes of a new version. This work is useful for the developers who are grappling with paucity of tools or knowledge to find out impact of changes on their application's energy consumption.

Preface

The user study of which is a part of this thesis, received research ethics approval from the University of Alberta Research Ethics Board, Project Name "Energy consumption and Code Syntax Error Location in Undergraduate Software Engineering Courses", No. Pro00050197, 17th October, 2014.

A version of Chapter 3 was published [3] as "The Power of System Call Traces: Predicting the Software Energy Consumption Impact of Changes" at Centre for Advanced Studies Conference(CASCON), 2014 and has contributions from Joshua Charles Campbell, Dr. Abram Hindle, and Prof. Eleni Stroulia with concept formation and manuscript composition.

A version of Chapter 4 and Chapter 5 has been accepted [2] as "GreenAdvisor: A Tool for Analyzing the Impact of Software Evolution on Energy Consumption", to be published at International Conference on Software Maintenance and Evolution(ICSME), 2015. Dr. Abram Hindle and Prof. Eleni Stroulia were the supervisory authors involved with concept formation and manuscript composition.

Dedicated to my parents, my brother, and my sister-in-law for their love and support.

Acknowledgements

I would like to thank my supervisors Dr. Abram Hindle and Prof. Eleni Stroulia for their great supervision. This thesis would not had been possible without their patient and quality guidance. I would like to express my gratitude to them for introducing me to the area of Mining Software Repositories, providing research opportunities, and teaching me the skills required for conducting research.

I am also grateful Alexander Wilson and Joshua Campbell, who helped me in getting familiar with Green Miner, and also with debugging Green Miner at times. I would love to express my gratitude to Gregory Burlet for managing the collection of consent forms and feedback questionnaire for the user study, without which this thesis would not had been possible.

Last but not least, I owe a great to my family who supported and encouraged me in every way possible.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Software undergoes continuous changes through evolutionary software development processes and maintenance activities such as bug fixing. The impact of these changes to the application behaviour can be extensively and thoroughly tested, using a variety of available tools. However, the consequences of changes to the application's energy consumption are extremely difficult to predict and, to date, there have been little research or tools to help developers with this challenging task.

This problem manifests itself as a great challenge in the context of mobile applications. Smartphones and tablets have witnessed an exponential rise in usage in past few years, surpassing the 1 billion user mark in 2013, making a huge impact on human life. According to Pew [32], as of May 2013, 63% of adult smartphone owners (in North America) use their phones to go online and 34% of smartphone users access Internet mostly using their phones instead of a more "traditional" device such as a desktop or laptop computer. This usage data has motivated the development of sophisticated and complex mobile applications, offering a multitude of information-access, entertainment, and education functionalities. The more sophisticated these applications become, the more they demand of the mobile device battery. Battery life, which affects a mobile device's availability, is a critical factor in user satisfaction of a mobile device and impacts the ratings of an application [39]. Despite its importance, developers are generally unable [26, 30] to estimate the energy consumption of their applications, and predict the impact of code changes.

1.1 Software Code Evolution and Energy Consumption

Some research effort has been devoted to the examination of factors affecting the energy consumption of mobile applications and constructing models to estimate energy consumption of smartphones [13], [37], [29]. However, relatively little attention has been focused on the effect of code changes on an application's energy consumption profile —the historical or expected energy consumption of an application. Most of previous work has been focused on the energy consumption of applications to help the end user track the energy consumption of their smartphone [28], [42], [24], rather than helping the developers understand the impact of their code changes on the energy consumption profile of their applications. Very little work has been done to help the applications developers develop energy-efficient applications specially in view of developer unawareness of the factors affecting energy consumption of their applications [30].

Most notably, Hindle *et al.* [17] proposed the *Green Mining* methodology, for analyzing the impact of code changes on energy consumption: intuitively, this methodology proposes that the energy consumption of multiple versions of the software should be measured. Software energy consumption across versions was correlated with multiple software metrics in order to recognize which metrics might be potentially useful proxies for estimating the size and direction of the energy-consumption impact of code changes.

Zhang *et al.* [40, 41] demonstrated the impact of user choices on software energy consumption by using competing applications and user actions. They also propose a energy ratings benchmarks for applications just as done for standard electrical appliances to motivate developers for creation of energy efficient applications.

In order to help developers to construct energy efficient applications, it is important for them to track their application's energy consumption during the development process. This can be achieved however only using expensive instrumentation, which is most likely not available, neither practical for application developers. In this work, we propose a method and introduce a tool based on our method that can be used by developers to estimate if their source code changes cause changes in their software's energy profile: the profile of energy consumption during runtime. We rely on dynamic analysis of application runs to estimate energy consumption changes caused by software change. We use dynamic analysis by tracking system calls, that are the calls made by an application to access the system resources like memory and network. System calls are essentially signatures of resource usage of an application.

In this thesis, we use records of system calls to model software energy consumption over multiple versions. Our results point towards a relationship between the change in energy consumption and change in the system call invocation profile. The central contribution of this work is *Rule of Thumb*, which states that "a signicant change in system call counts implies a signicant change in energy consumption", where significant implies statistically significant change. Using these findings, we created a tool to help developers in keeping track of the energy profile of their applications in the absence of the expensive instrumentation required to do so.

1.2 Thesis Overview

- Energy consumption and system calls: Using green mining methodology across multiple versions of two applications, we tried to correlate system-call counts with the energy consumption. A simple and easy to use qualitative *Rule of Thumb* was proposed to predict whether energy consumption will change or not based on the system-call counts. This simple rule can potentially be very useful since developers can use this in absence of any expensive and rarely available energy measuring instrumentation.

- *GreenAdvisor* tool: Using the *Rule of Thumb* we developed a tool called *GreenAdvisor* that can be used by developers to keep track of the energy profile of application to notice any significant changes when a code change is made. Architecture and working of the tool has been described.

- User study: The tool is evaluated using a user study and so is the *Rule of Thumb* using the student user projects. The results demonstrated the usefulness of the tool, and also validate the effectiveness of the *Rule of Thumb*. Then, a improved model that uses *Rule of Thumb* with a logistic regression model to predict the direction of energy change along with the significance is proposed and evaluated.

- Conclusions and future work: Finally, conclusions from our findings are presented in the final chapter and future directions for research in the field have been suggested.

1.3 Contributions

This thesis makes the following important contributions:

- We demonstrate that system call profiles can be used to model changes in energy consumption profiles for Android applications. We propose, test and verify a simple *Rule of Thumb*, which makes it very easy for developers to track significant changes to the energy profile of their applications.

- We introduce *GreenAdvisor* tool that uses *Rule of Thumb* to indicate the potential significant change in energy profile of Android applications and the code change that might be responsible for that, helping developers track the evolution of their applications.

- We conducted an user study that evaluated the usefulness of the tool, and explored the general awareness about the energy efficiency affecting factors among the developers. The user study also validates the *Rule of Thumb*.

1.4 Thesis Organization

The rest of this thesis is organized as follows: Chapter 2 details the research that is related to this thesis, from different perspectives; Chapter 3 presents first half of the thesis establishing relation between system calls and application energy consumption; Chapter 4 presents the second half of the thesis with the description of *GreenAdvisor* tool; Chapter 5 presents an user study conducted to gauge the useful of the tool; we conclude this thesis and discuss the future work in Chapter 6.

Chapter 2

Background and Related research

In this chapter, background required to understand the concepts used in thesis are described along with the related works that describe the context of this thesis in the existing literature. Section 2.1 describes the system calls and their position in the architecture, the software energy consumption, and statistical models used in the thesis. Section 2.2 is organized into works that have considered instructions level energy modeling, hardware-component based energy modeling, hybrid models, effect of code transformations on energy consumption, mining energy related textual data, and mining software repositories.

2.1 Background

In this section, we discuss the system calls, the software energy consumption, and statistical models used in the thesis.

2.1.1 System Calls

System calls form an API for user-space applications to access the services, abstractions and devices that are managed by the kernel and the rest of the OS. System calls are standard functions provided by the OS kernel to user processes. Usually such system calls are provided for: communicating with the hardware, for example, accessing the hard disk; creating and executing new processes; managing memory use; sending and receiving data to and from

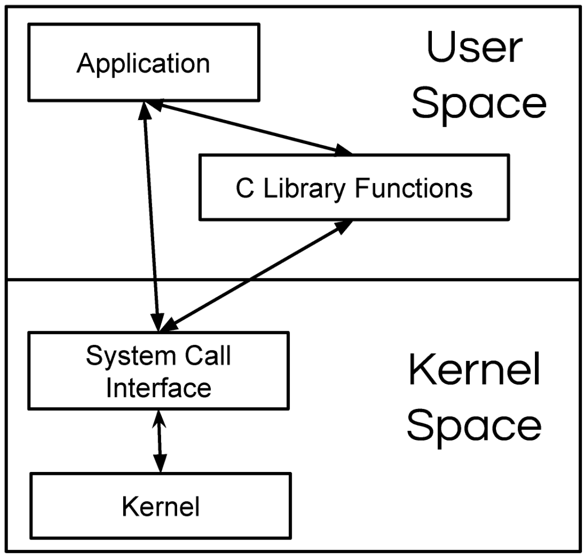


Figure 2.1: Interaction between applications, C library functions, system calls, and the kernel.

other processes; and receiving notifications for events [38]. Figure 2.1 presents the relationships among user applications, C library functions, system calls and the OS kernel [38]. System calls are typically called by a library, such as the C standard library, graphics libraries, network libraries, inter-process communication (IPC) libraries, and occasionally by the application itself.

One can expect different versions of software to invoke different system calls during their execution if they differ from each other in terms of the services they use and how they access these services.

2.1.2 Software Energy Consumption

Energy refers to the cost or ability to do work while power is the rate of energy use. Power is measured in watts (W) while joules (J) are the cumulative watt-seconds measured to do work. If a process uses 4W of power for 10 seconds it consumes 40J of energy. Mean-watts are often used when tests are of the same length of time. Mean-watts is the average power used for any moment of a task. Mean-watts multiplied by seconds produces the energy consumed in units of joules during a test. The energy consumption of software systems can be viewed as the energy consumed by the software to provide a certain amount of work or service over time.

2.1.3 Simple Linear Regression

Linear regression attempts to model the relationship between two variables by fitting a linear equation to observed data. An independent variable is an input to the model and considered to be the cause that is hypothesized to produce a change or effect on the variable, referred to as the dependent variable. Linear regression fits a straight line through the set of data points in such a way that makes the sum of squared residuals of the model (vertical distances between the points of the data set and the fitted line) as minimal as possible. Equation of straight line is:

$$y = a \cdot x + b \tag{2.1}$$

where, x is the independent variable while y is the dependent variable. The slope of the fitted line, a, is equal to the correlation between y, x data points adjusted by the ratio of standard deviations of these y to x. The fitted line passes through the centre of mass of the given data points and hence, the intercept value is adjusted accordingly.

Simple linear regression assumes a linear relationship between the two variables. R^2 is a goodness of fit measure for the linear regression model with the given data points used to construct the model, a statistical measure of how close the data points are to the regression line. It is a ratio of explained variance, *i.e.,* variance of dependant variable as obtained from linear model with respect to mean of data, to the variance observed in the dependant variable in given data points with respect to the mean. High R^2 values usually reflect good fit, and low R^2 values a bad fit. This technique when extrapolated to model relationship of dependant variable with multiple independent variables is called multiple linear regression.

2.1.4 Logistic Regression

This is a classification model that uses probabilistic modeling to classify dependant variable into two classes. It predicts the probability of the dependent variable to exist in either of classes using the independent variable(s) while the simple linear regression used OLS to find the best fitting line. Hence, in simple linear regression, the relationship between the dependent and the independent variables is assumed to be linear, this is not the case in logistic regression. The logistic regression uses logit function to model the probability of a variable to exist in either of the classes. The logit function is natural logarithm of odds of occurrence of the event:

$$P = \frac{e^{\alpha+\beta x}}{1 + e^{\alpha+\beta x}} \tag{2.2}$$

where P is the probability of one of the events, α and β are the parameters of the model (as in normal linear regression). The value of α yields P when x is zero, and β indicates how the probability of the event changes when x changes by a single unit. As the relation between x and P is non-linear, β does not have a linear interpretation in this model as it does in ordinary linear regression.

The logit function maps any value of variable to values between 0 to 1. The regression coefficients of the logistic function are calculated by using maximum likelihood estimation by applying an iterative process. Starting with initial seed values of coefficients, the coefficients are updated to reach maximum probability density state. The probability function being modeled as a linear combination of the independent variables along with their coefficients.

Hence, the logistic regression estimates the probability of occurrence of event over the probability of non-occurrence and takes a threshold to translate the predicted probabilities as a success or failure.

2.2 Related works

In this section, existing literature related to this thesis has been presented. Te existing literature has been divided into four topics: modeling energy consumption, mining software repositories, mining energy related natural language text, and effect of code transformations on energy consumption.

2.2.1 Modeling Energy Consumption

In this section, works related to modeling the energy consumption of devices have been described divided into three types: Hardware component based, instruction level based, and hybrid models.

2.2.1.1 Hardware Component Based Energy Consumption Models

A large number of models have been proposed for modelling the energy consumption of devices based on their individual hardware components. It involves collecting the energy statistics by profiling the sample runs of the applications on the device. Using these energy statistics and modeling like linear regression, energy consumption models are built that can be used to predict the energy consumption values of the future device usage based on individual component usage.

Flinn *et al.* [10] built a profiler tool called, PowerScope, the first of the hardware component based power profilers. It is based on modelling individual hardware components mainly — CPU, network, and disk, tested on laptop. They measure the power by tracing the current across a constant voltage supply. To demonstrate their tool, they took a case of study of video streaming by using a video with two different resolutions. They vary the test conditions like using lossy compression, reducing the display size, and modifying the network interface to demonstrate their tool's efficiency in measuring the reduced power consumption.

Gurumurthi *et al.* [13] have built a profiler, SoftWatt on the SimOS platform. They not only model the individual components namely — CPU, memory, and disk, but also take into consideration the interaction between these components. The authors have modelled the energy consumption into — Kernel energy consumption, which is practically constant, and external I/O system calls, which bring an element of variability because on the varied size of the data transfer blocks. They also found that synchronization operations are quite expensive in terms of energy consumption. SimOS collects the energy statistics and Softwatt reports those statistics using its analytical energy models.

Carroll *et al.* [7], have constructed energy consumption models by devising test cases on a Freerunner device, modified to run without the battery. They have constructed the model using the components like CPU, memory, and screen. by using tests like, emailing, voice calling, web browsing, messaging, audio playback, and video playback. Depending on the workloads, either screen or CPU have a major contribution to energy consumption. They also compared HTC Dream and Nexus One, running on battery using the above tests.

Shye *et al.* [37], built a logger application for G1 Android HTC phones, that was used to collect the energy consumption from 20 users who had installed the application, uploaded to server. Using these statistics, they construct a linear regression model based on hardware components like wifi, EDGE, touch screen , CPU, SD Card, and music player. Their model performed well when used to predict real world energy consumption by achieving high accuracies. They also proposed optimizations to save on screen energy consumption, by analysing their application's data.

Zhang *et al.* [42], built two online profiler tools, PowerTutor and PowerBooter. PowerTutor is based on integration of energy models for components in Android smartphones as the power consumption of each hardware component — CPU, wifi, audio, LCD, GPS, and cellular using linear regression. PowerBooter is a constructed as a more general tool, which builds the model based on battery discharge profile while linking it with components. Though they have shortcomings like susceptibility to variance in battery's internal resistance, they present very easy method to compute the energy consumption without any special instrumentation.

Dong *et al.* [8] implemented a self-constructive energy model for Linux-based mobile systems. The energy model, called Sesame, generates energy models for mobile systems without external power measurement. Sesame collects system statistics and applies the Advanced Configuration and Power Interface (ACPI) to gather the predictors for the energy model. Energy readings are collected through the smart battery interface, and linear regression energy models are generated based on the collected data.

Mittal *et al.* [24] have built WattsOn an energy consumption model on Windows Phone platform. It builds upon specific components — 3G, wifi network, screen, and CPU. They also discovered the issue of tail energy consumption, the high energy state of the components after a task has been completed. They introduce resource scaling to generalize WattsOn for real phones to account for the underestimation of energy consumption on emulators.

Some of the hardware components feature a tail energy phenomenon [4, 29]. Tail energy phenomenon is exhibited when a component stays in a high energy state after the completion of an operation even though it is no longer being used. Thus, these simple energy consumption models above are unable to capture this phenomenon in the models.

2.2.1.2 Instruction Level Energy Modeling

For applications running in JVM, works described below utilized the Java bytecode instructions to build energy models for software systems.

Seo *et al.* [35, 36] have implemented an energy consumption model for Java-based software systems running on distributed devices. This energy model consists of three components — computational energy cost, communication energy cost, and infrastructure energy overhead. This energy consumption model makes accurate estimates which fall within 5% of the actual energy cost for an application. However, the model is highly dependent on the hardware and JVM.

Hao *et al.* [15] built an energy consumption model, *eCalc*, for Android applications that measures CPU energy consumption at two levels: the whole program and the method by performing a program analysis based on bytecodes. The approach in *eCalc* is similar to Seo *et al.* [36]. It is able to estimate energy consumption within 9.5% of the actual value.

An extension of *eCalc* implemented by Hao *et al.* [14], called *eLens*, combined program analysis with instruction-based energy modelling. *eLens* is able to estimate the energy consumption of hardware components besides the CPU and has fine-grained energy profiling on multiple levels. Li *et. al.* [21] extended this work by profiling bytecode instructions, and estimating the energy consumption of each line of source code. Their profiler application, *vLens*, is able to estimate energy consumption with high accuracy.

Instruction-based power models are often designed for software running in a JVM. Some of the applications in this paper do not run in a JVM.

2.2.1.3 System Call based Models

Hybrid models use hardware component based modeling with system calls data to build energy models. A hybrid system-call-based power model that utilizes hardware states was proposed by Pathak *et al.* [29]. They applied system call tracing to model the energy consumption of applications running on smartphones. First, they studied the power behaviour of components in a smartphone and showed that several components have tail power states (a component stays in high power state for a period of time after use), system calls that do not imply utilization can change power states, and several components do not have quantitative utilization. They conclude that energy linear models based on correlating utilization with energy consumption are not accurate. They built energy models by modelling the power states and generate finite state machines (FSMs) of each system call for each component in a smartphone, and then integrating all these FSMs to model a FSM for each individual component which are used to construct a FSM for the particular smartphone. Thus, based on the FSM of a certain smartphone, the system's current state can be identified to estimate the energy consumption of an application. Their results show improved accuracy compared to an approach [37] based on linear regression modelling. It was extended [28] to implement a fine-grained energy profiler for smartphone using FSMs. This energy profiler, *eprof*, can work on both Android and Windows Mobile phones to estimate the energy consumption of smartphone applications.

Similar to this system call based model, we also trace system calls and correlate them with software energy consumption. However, this thesis relies on the aggregate count of system call invocations and studies system call profiles across multiple versions of existing products. Moreover, this thesis is focused on developers and predicting energy consumption change with change in code base of individual applications, rather than helping end users track the energy consumed by their smartphone.

2.2.2 Mining Software Repositories

Mining software repositories research uses data mining and machine learning techniques to explore the data generated during the software development processes in repositories like version control system, tracking systems, and mailing lists. By analyzing this data generated during the software development process, researchers have helped the software engineering community in better understanding of the software processes to provide a feedback for better practices. These techniques have been leveraged to solve problems such as detection of bugs, defect prediction. However, use of these techniques in energy-aware development has been very limited.

Gupta *et al.* [12] used energy traces and execution logs in Windows phone to build energy models. They use some some tests to log the power traces, i.e. measurement of power consumption over time and corresponding execution logs (containing the execution sequence of executable files). They used linear regression models and decision trees to find the most power intensive modules/executables, which was quite similar to the developer perception of those modules. They also try to model effect of coupling modules on the energy consumption with a good accuracy. By clustering the power traces, they identified some anomalous traces, which turned out to be related to the bugs in the corresponding modules. They demonstrate the effectiveness of their model by predicting the power consumed by test sequences, and get quite high rank correlations to the actual power consumption.

Hindle [16, 17] proposed a detailed methodology, called *Green Mining*, to collect the energy consumption statistics and try to relate them to software metrics over versions of the software. Though, they found a weak relationship between the two, author has proposed a future direction to the field. By following the methodological process of Green Mining, researchers can help the developers to understand what type of changes to the source code can have an effect on the energy consumption.

Hindle *et al.* [18] also created the Green Miner test-bed which has a dedicated hardware infrastructure to measure the energy consumption of Android applications across multiple versions. They demonstrated that multiple test runs are required to reliably calculate the energy consumption of an application on a smartphone. Green Miner can measure up to 50 power readings every second. A constant voltage is provided to the phone which draws varying current according to phones requirements. Authors validated the platform using case study on firefox application to cross validate results with previous work. Besides this, an energy bug was also observed for the reddit application that was validated by the developers of the application. This test bed was used for measuring the energy consumption of all the applications used in this thesis.

Zhang *et al.* [40, 41] demonstrated the impact of user choices on software energy consumption by using competing applications and user actions. They also propose a energy ratings benchmarks for applications just as done for standard electrical appliances to motivate developers for creation of energy efficient applications.

Li *et al.* [20] observed that on a dataset of 405 applications that on an average idle state consumes 61% of total energy consumed by an application, with network being the most energy consuming component. They also observed that API invocations consumed most amount of energy than the developer code.

2.2.3 Mining Energy Related Natural Language Text

This section presents the work that have used textual data from software repositories or online developer discussion forums to report interesting findings about the software energy consumption.

Pinto *et al.* [30] analyzed a dataset of more than 300 questions, and 800 users on the StackOverflow for the energy consumption related discussions among the developers. They found that developers were generally unaware of the factors affecting energy consumption of their applications based on the quality of their answers though they had vague idea about some optimizations that can potentially affect the energy consumption of their applications. Developers harbour misconceptions like treating performance as a measurement for energy consumption and using languages like Java to reduce energy consumption. They also discovered that developers were facing lack of tools for energy aware development.

Malik *et al.* [22] extended this work by analyzing the whole of *StackOverflow* dataset instead of selected questions. They found that majority of energy-related questions are implicit in nature, *i.e.,* the primary goal of the question is not energy related but as a side-goal. Though they make up less than 1% of all posts on the *StackOverflow*, for the mobile OS projects of Android and iOS they make up 21.9% and 15.2% of the posts respectively. The developers mostly discuss about optimizations regarding data updates, synchronization, utilizing sensors, radio utilization, and memory use.

Pathak *et al.* [27] analyzed the discussion data from 4 online mobile user forums and bug repositories for *eBugs*, that is the bugs that cause applications to consume unexpectedly high amounts of energy. The *eBugs* were classified into taxonomies ranging from hardware defects like battery issues to software defects like OS configurations to high state energy consumption like no-sleep bugs.

Wilke *et al.* [39] used comments on Google play applications to analyze how users think about energy consumption of applications. They found that energy efficiency related issues negatively impact the user rating of the applications. They also discovered that paid applications had no different energy-efficiency related complaints by user as the free ones, hence demonstrating obliviousness to energy-efficiency driven development among application developers.

Moura *et al.* [25] analyzed selected commits from selected Github projects for energy related commit messages in order to understand the behaviour of developers for introducing energy optimizations in their applications. They found that developers performed 12 very diverse types of energy related optimizations like voltage/frequency scaling of CPU operations, disabling features, tackling idling, fixing energy bugs. Most of these optimizations are at the low level, *i.e.,* at Kernel/OS level and developers don't use much of application level optimizations.

Pang *et al.* [26] performed a survey on 100 developers to ascertain the level of knowledge of software energy consumption among developers. It was observed that developers have limited knowledge about energy efficiency of their software and possess limited awareness about the best practice for energy-efficiency of software.

2.2.4 Effect of Code Transformations on Energy Consumption

In this section, works investigating the effect of code transformations on energy consumption have been described with a emphasis on refactorings. Refactorings [11] are code transformations that change the structure of the application but not its behaviour.

Sahin *et al.* [34] investigated the effect of common code refactoring on energy consumption using Java applications. They used refactorings like local variable to field and extracting local variable to investigate its effect on energy consumption of selected Java applications. They found that refactorings can significantly change the energy consumption of an application, though the exact impact varies depending on the application and the location of the refactoring.

Banerjee *et al.* [5] introduced the terms energy hotspots and energy bugs. Energy hotspots can be defined as as a scenario where the smartphone starts consuming abnormally high amount of energy though the utilization of its hardware resources is low. While energy bug leads the phone to a higher energy state by preventing phone from becoming idle because of a bug in the application code. They introduce a testing framework that can be used to detect both type of anomalies in the application.

Kwon *et al.* [19] investigated using distributing to cloud the code suspected by developers to be energy hotspots. Using a automated approach to transfer the application functions from the energy hotspots they were able to reduce the energy consumption of applications substantially. However, this method is only cost effective for the applications requiring lots of computation. For the applications with average or little computation requirements, the network costs impose more energy costs than the benefit from distributed computation.

Pinto *et al.* [31] describe the work done in refactoring code and its standing with respect to impact on the energy consumption of applications. They have identified the areas where some works on energy consumption modeling have been tried and how refactoring can possibly be utilized. A lot of opportunities exists in creation of refactoring tools with energy efficient techniques integrated like suggesting energy efficient APIs, energy efficient obfuscation, and synchronizing code to perform bulk I/O operations.

Brandolese *et al.* [6] used assembly instructions to model energy, augmenting them with instruction and cache misses. The instructions are modeled on CPU, memory and bus usage. Instruction miss occurs when the particular I/O bus or CPU is busy and hence prolonging the execution. Cache misses occurs when the data being accessed has to be fetched into cache. They used two code transformations — loop fusion and loop unrolling to optimize the energy consumption successfully.

2.3 Chapter Summary

In this chapter, literature related to this thesis was discussed. The literature was viewed from different aspects: energy consumption models, mining software repositories, effect of code transformations, and mining energy related natural language text. These prior works have attempted to build models though mostly catering to end user to track energy consumption of their device as shown in Section 2.2.1. Other works in Section 2.2.3, 2.2.4 and 2.2.2 show that developers are realizing the importance of energy-efficiency of their applications but are unable to follow energy-efficiency driven development for the lack of tools. Also, they lack proper knowledge about the energy-efficient development and have misconceptions because of the lack of curriculum in the same. In this thesis, we have attempted to provide one such tool to assist developers in energy-efficient development.

Chapter 3

System Calls and Energy Consumption

A lot of research effort has been devoted to model energy consumption on smartphones based on hardware components (Section 2.2.1.1), system run-time statistics [8], finite state machines (Section 2.2.1.3), and byte code instruction usage (Section 2.2.1.2). However, very few studies [16] have investigated the impact of software change on application energy consumption based on actual evolution of software projects. Much is to be learned by measuring energy consumption of multiple software versions to discover potential correlations between software metrics and energy consumption. It was demonstrated [18] that multiple test runs are required to reliably calculate the energy consumption of an application on smartphones. This measuring equipment which adheres to reliable and repeated testing requirement is generally not available to developers.

We want to help developers estimate if their source code changes induce changes in their software's energy consumption profile. Thus, we attempt to rely on dynamic analysis of test cases to estimate energy consumption changes caused by software change. In order to do so, we use dynamic analysis using system calls, that are used by applications to access services offered by operating systems like hardware control, and communication.

Next, we describe Green Miner [18] a platform used to collect the energy consumption and system call statistics. Then, we describe a case study that examines two open source systems — `Firefox` and `Calculator` . Using sample usage scenario on two applications we collect the data for energy consumption usage and system call traces. Using this data, we model the energy consumption with the help of system call invocation counts. Lastly, we propose a very simple and practical *Rule of Thumb* model that can be used by developers to determine significance of change in their application's energy profile when a code change is made. *Rule of Thumb* states that "a signicant change in system call counts implies a signicant change in energy consumption".

This chapter is organized as follows: Section 3.1 details of the Green Miner platform used to collect the statistics used in this thesis; Section 3.2 contains the case study on two open source applications, the usage scenarios, energy consumption modeling, and results of the study.

3.1 Green Miner

Green Miner is a test bed consisting of four Android clients. Each client consists of a *Galaxy Nexus* phone controlled by a Raspberry Pi that starts the tests, collects and uploads the data onto a centralized server, and an Arduino for measuring energy consumption. *Green Miner* takes a sequence in the form of a script that is used to replay the sequence of taps and swipes on an actual phone. It runs the sequence on the test bed phone by injecting touch events into the phone's input event system, from the shell script. It creates touch events on the phone, swiping actions on the phone screen and text entry on the phone's on-screen keypad.

In order to create the sequence of a user using the phones in a typical scenario, the screen pixel positions of each of the touch events are recorded. For example, to emulate a tap action on the phone, the pixel position on the screen for the tap is recorded. Similarly, for a swipe action, the starting and ending pixels on the screen are recorded. The emulator provides the option to record the pixel positions at each tap or swipe on the emulator screen. The Android emulator is available with the standard Android Studio platform. By recording these positions and actions, the shell script is created which is used on the phone to run these test-cases.

3.2 Case Study

3.2.1 Methodology

In this section, the Green Mining methodology, as proposed by Hindle [17] is described that was followed in this thesis. This section describes the procedure for creating test scripts and collecting the energy and system call statistics for multiple versions. Data was collected using the following procedure:

1. Select the application, and build multiple versions;
2. write the test script to drive the application;
3. configure the Green Miner; and
4. collect and analyze the results.

3.2.1.1 Selecting the Application and Building Multiple Versions

Two open source applications with multiple versions were chosen for these tests: `Calculator` [1] and `Firefox` [2] for Android. Both applications are widely used, `Calculator` for calculations and `Firefox` for browsing the web. 101 `Calculator` versions were retrieved from its GitHub repository. These versions were commited between January 1, 2013 and February 5, 2013. Using the Android NDK tools 101 `Calculator` APKs (Android Packages) were built. APKs are similar to Java `jar` packages or Debian `deb` OS packages, except that APKs are used only in Android.

Similarly, `Firefox` application versions were obtained by building the nightly commits in the `Firefox` repository from March 7, 2011 to November 17, 2011. 156 versions of `Firefox` were built, each separated by one commit.

3.2.1.2 Devising the Test Sequence

The test scripts were created to simulate realistic usage of the application by an average user. The `Calculator` application is used for calculations that a user might execute daily such as unit conversions or tax calculations. The

[1]https://github.com/Xlythe/android_packages_apps_Calculator
[2]https://wiki.mozilla.org/Mobile/Fennec/Android

`Calculator` application test does some simple conversion calculations such as converting miles to kilometres, gallons to litres, calculating tax amounts and solving an equation using the quadratic formula. This test has a duration of 125 seconds.

The primary use of `Firefox` is to browse web pages full of multimedia such as images. To simulate an average user reading a web page with the `Firefox` application, the script opens the Wikipedia page about American Idol, and emulates reading action for 4 minutes. The script scrolls the page by swiping the page down and clicks on the screen at regular intervals of 10 seconds to prevent the phone from sleeping. The webpage is stored on a local server accessible to devices under test. This prevents the webpage from changing and also prevents varying server response times and network congestion from affecting the data collected.

3.2.1.3 Configuring the Green Miner

The *Green Miner* test bed [18] was used for testing the applications. For system call tracing, `strace` [3] was employed, and cross-compiled for Android. It is a tool used to record the system call, its arguments and its return value called by a running process or application. It is used by developers to keep track of the system calls being made and detect any unusual behaviour of their applications to facilitate debugging. The `strace -c` option was used to obtain system call counts.

3.2.1.4 Running the Tests

As there is a variation in power measurements due to factors unrelated to the software being tested, multiple tests for the same version need to be run. In these experiments, 10 tests per version were run, which is the minimum number required to record reliable mean values of energy consumption. From these tests the mean energy consumption is calculated for each version. Because `strace` adds its own energy overhead during the tests, separate test runs are required for collecting the system call data. 10 runs per version are run with

[3]http://sourceforge.net/projects/strace/

the `strace` tool in the background to obtain the system call profiles in order to observe whether the system call profile is stable or not.

3.2.2 Dataset

Dataset obtained from the methodology described previously, contained 10 tests each for energy consumption and system call trace measurements for 101 versions of `Calculator` and 156 versions of `Firefox` . Therefore, there were a total of 2020 and 3120 test runs for the `Calculator` and `Firefox` applications, respectively. A total time of 400 hours was needed to run these tests.

3.2.3 System Call Profile Stability

In our tests, the `Calculator` application invoked 46 different system calls and `Firefox` invoked 91 different system calls. The system calls that had on an average less than 10 calls per version were filtered out, leaving 25 system calls for `Calculator` and 53 for `Firefox` . Though it is generally expected that system call counts are stable across different runs, the number of system call invocations showed some variation. Different system calls have different variances. Figure 3.1 and Figure 3.2 shows the average of the variances of the 10 counts for each version over all versions of `Calculator` and `Firefox` respectively, relative to the average of the means. Since, The mean and variance of the distribution of system call count measurements across 10 runs for each of the version is measured; these variance values are then represented as percentage of those mean values for each of the system call. As system call counts vary drastically across versions and between different system calls, we normalise variance as a percentage of mean, so that the variances are scaled relative to the mean. These variances as percentages of mean system call count across the versions have been shown in the graphs. These graph show that the system calls do exhibit a little instability, just like the energy consumption measurements. Hence, they need to be averaged across the runs in order to get stable representative values for the the system call counts for a version.

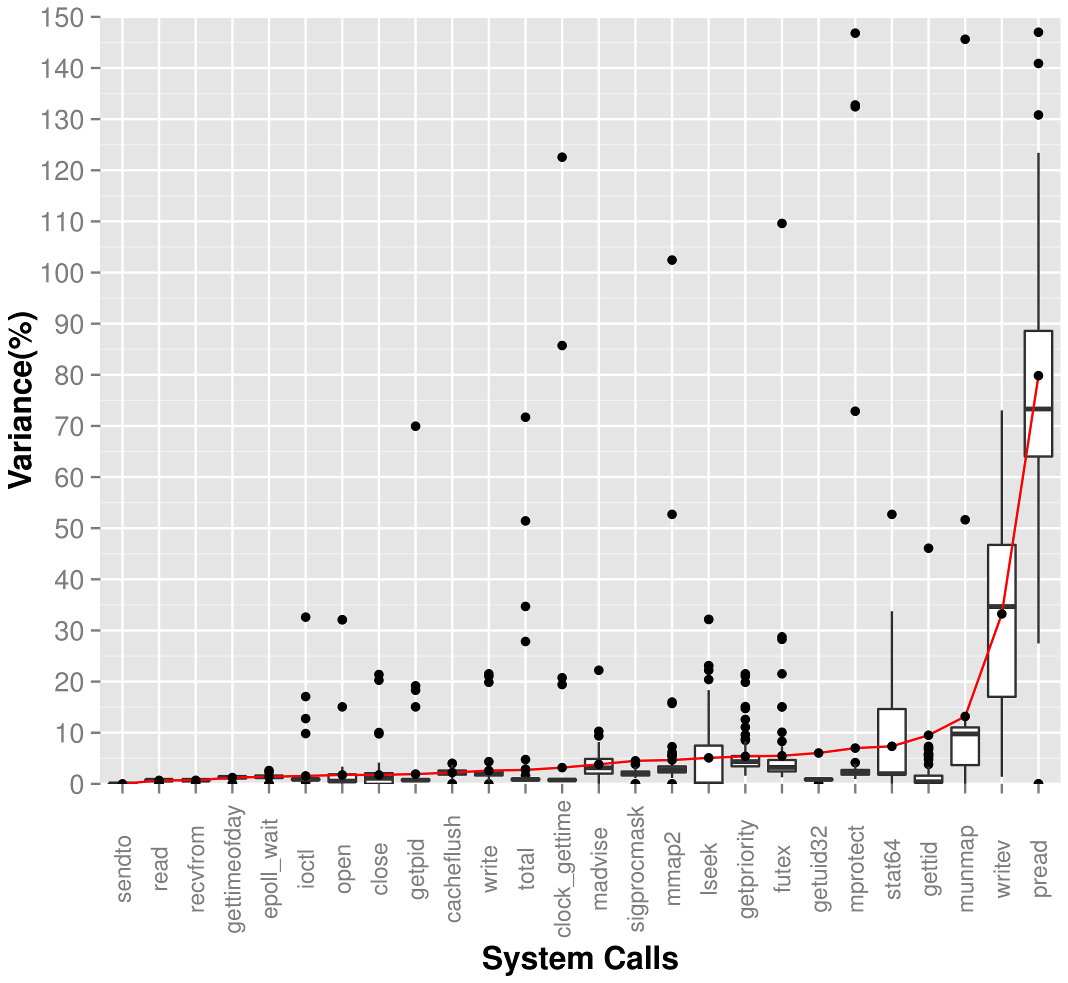


Figure 3.1: Variance as a percentage of mean value across 10 runs per system call per version for the `Calculator` application. The X axis refers to the system calls, while the Y axis refers to variance in percentage.

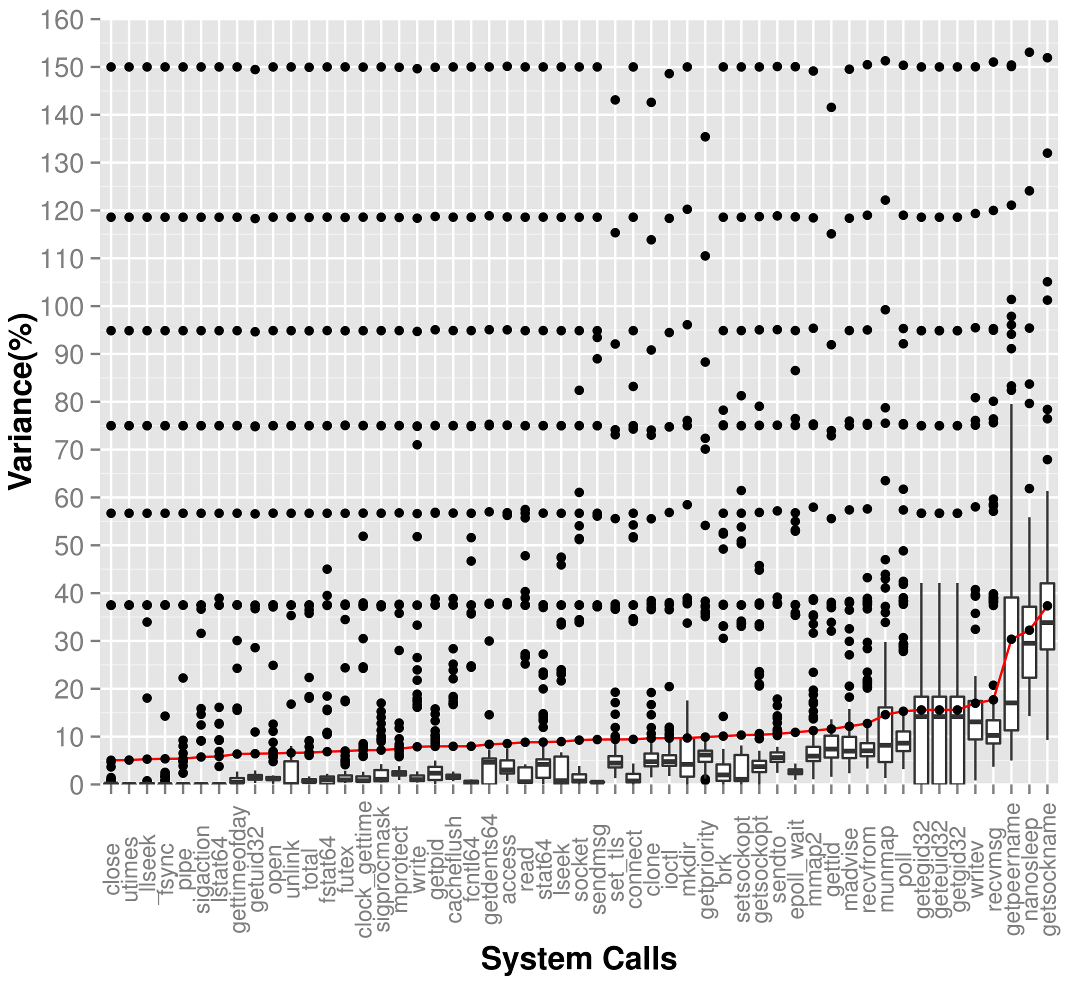


Figure 3.2: Variance as a percentage of mean value across 10 runs per system call per version for the `Firefox` application. The X axis refers to the system calls, while the Y axis refers to variance in percentage.

3.2.3.0.1 Calculator: 14 of the 25 system calls had an average variance $\leq$ 3% between versions. However, three system calls, `munmap`, `pread`, and `writeev` have average variances of 13%, 79%, and 33%, respectively, across all versions. 4 system calls had an average variance between 3% and 5%. Finally, 4 system calls had an average variance between 5% and 10%. The total system call count, the sum total of all system calls, has a mean variance of 2% across all versions.

3.2.3.0.2 Firefox: 33 of the 52 system calls had a mean variance between 5% and 10% and the remaining 19 had a mean variance $\geq$ 10% when variances

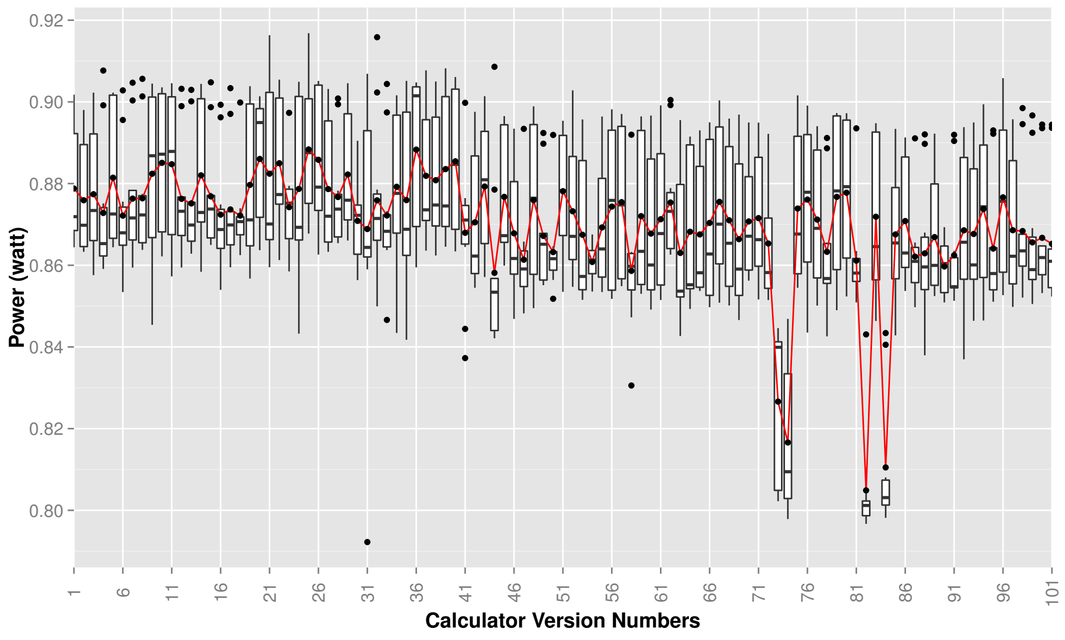


Figure 3.3: Box plots of the mean watts used for each version of the `Calculator` application. Each of the 101 versions was measured 10 times. The X axis represents the version numbers and the Y axis represents the energy consumption.

were averaged over all versions. The `Firefox` application shows much higher variability than the `Calculator` application.

3.2.4 Energy Consumption

The energy consumption distribution as shown in Figure 3.3 and Figure 3.4 were obtained by running the tests on *Green Miner*. In order to determine whether the differences between any two versions were statistically significant, pairwise Student's *t*-tests were performed for each pair of `Calculator` and `Firefox` versions, as shown in figure 3.5.

3.2.4.0.3 Calculator: Figure 3.3 depicts a sudden drop in energy consumption at versions 73, 74, 82, and 84. Figure 3.5 shows that for versions after version 45, many versions are statistically different than the versions before version 45. The authors looked within the code repository and determined that this is because the screen display density was reduced in version 45, leading to a slight difference in the energy consumed. Versions 73, 74, 82 and 84

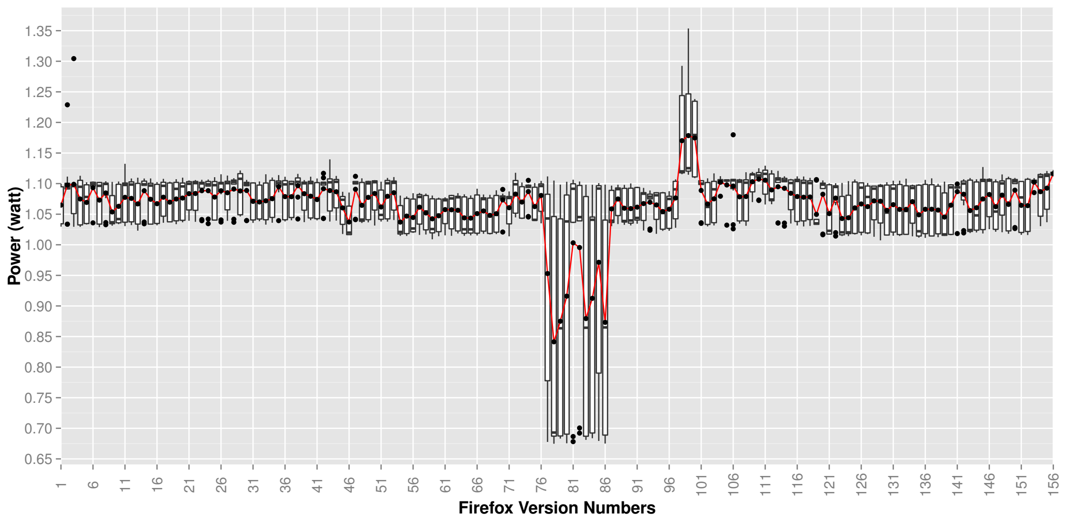


Figure 3.4: Distributions of the mean watts consumed per version of `Firefox` application over 10 tests for 156 versions. The X axis represents the version numbers and the Y axis is average power use.

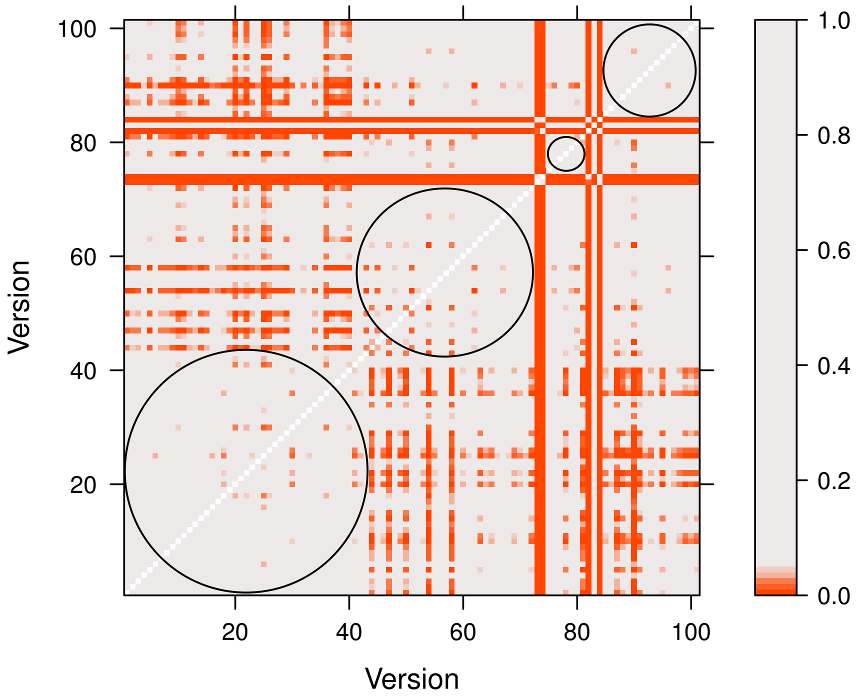


Figure 3.5: Pairwise Student's t-test for power distribution of each version pairs for `Calculator` . The X axis and the Y axis represent the version numbers. Orange squares indicate pair of versions are significantly different (p-value ≤ 0.05) after correction for multiple hypotheses. Black circles represent the clusters of versions having similar energy profile.

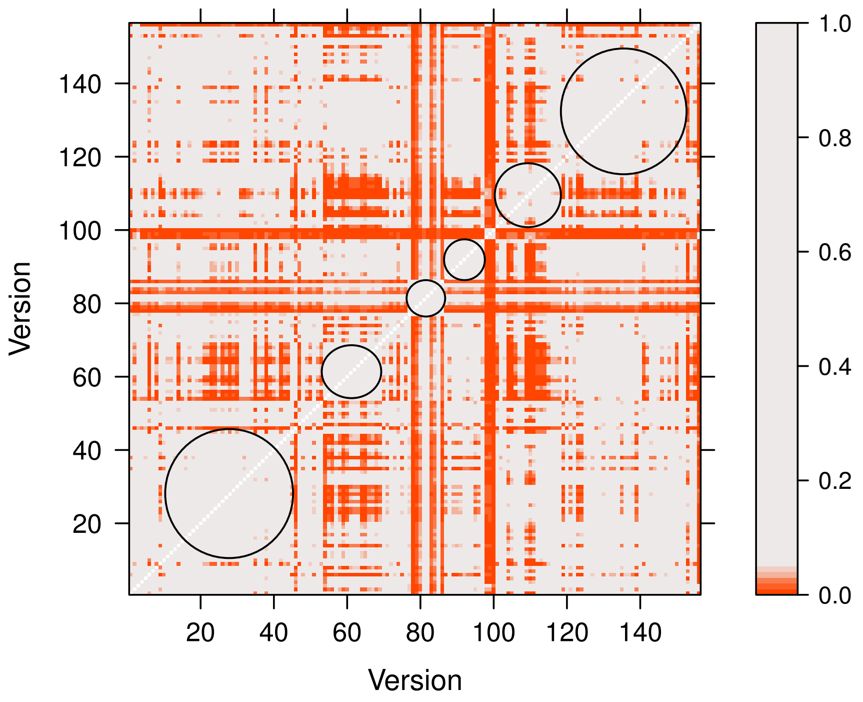


Figure 3.6: Pairwise Student's t-test for power distribution of each version pairs for `Firefox` . The X axis and the Y axis represent the version numbers. Orange squares indicate pair of versions are significantly different (p-value ≤ 0.05) after correction for multiple hypotheses. Black circles represent the clusters of versions having similar energy profile.

are versions that failed during testing and form their own cluster. These four versions are significantly different from all the other versions in terms of energy consumption. The errors in these four versions could have been the result of two large, incomplete refactoring efforts, one which started at version 73 and another which started at version 82.

3.2.4.0.4 Firefox: Versions after version 56 are significantly different from the versions before version 56 as shown in figure 3.5 and 3.4. Also, version 46 is significantly different from versions 1 to 56 in terms of power use. In version 56, a bug was fixed which added `.net` at the end of the domain name whenever the active tab was switched before loading the website by removing the buggy code. This contributed to the reduced energy consumption in versions 56 onwards.

Energy consumption increases again after version 70, when functionality to check for plugin crashes was added. Versions 77 to 86 contained a bug that was introduced during refactoring and was finally fixed in version 87. Versions 98 to 100 contained bugs that crashed `Firefox` completely.

System calls are not stable and exhibit variation like energy consumption measurements. Hence, these system call counts need to be averaged similar to the energy consumption values. The mean energy consumption values are very different in both the applications. Using these measurements, we next analyze the relationship between energy consumption and system call invocation counts.

3.2.5 Linear Regression Modeling

In order to test the relationship between energy consumption and system call counts, two models were used. A linear model was created using the ordinary least-squares error (OLS) regression method for individual system calls, relating the mean energy consumption and mean system call counts for that system call for both the application data.

3.2.5.1 Calculator

By analyzing system calls for cross-correlations it was found that a large number of the 25 system calls are highly correlated with each other: most system calls have a high Spearman's correlation coefficient ($|\rho| \geq 0.8$) with at least one other system call. Hence, for 25 system calls, single variable models were constructed relating energy consumption to the number of invocations of that system call. A linear model of the form $y = b_1 \cdot x + b_0$ was created, where y is the mean power use of a version, x is the mean system call count for a particular system call for that version, b_0 is the intercept, and b_1 is the slope coefficient. 10 such models are depicted in Table 3.1. Table 3.1 also shows the mean number of invocations of the system call across the versions as $\bar{x}$, and mean power contribution of the system call, $b_1 \cdot \bar{x}$, as estimated from the linear models in the last column.

System Call	Spearman's ρ	R^2	Coefficient b_1	$\bar{x}$	$b_1 \cdot \bar{x}$
`mmap2`	0.670	0.764	8.87×10^{-5}	511.30	0.0453
`open`	0.651	0.819	5.03×10^{-4}	115.73	0.0582
`close`	0.048	0.695	1.44×10^{-3}	32.72	0.0470
`epollwait`	0.031	0.684	1.06×10^{-5}	5113.56	0.0543
`mprotect`	0.613	0.756	5.83×10^{-6}	8438.49	0.0491
`recvfrom`	0.060	0.698	7.64×10^{-6}	7213.38	0.0550
`writev`	0.645	0.391	6.57×10^{-4}	29.03	0.0190
`cacheflush`	0.498	0.745	1.27×10^{-5}	3954.34	0.0500
`ioctl`	0.201	0.740	3.79×10^{-6}	14885.60	0.0560
`total calls`	0.373	0.740	5.64×10^{-7}	98155.94	0.0550

Table 3.1: Linear model summary for selected system calls for `Calculator` application

Selected system calls are shown in Table 3.1 and their descriptions in Table 3.2. 12 out of 25 of the system calls correlate with (Spearman's $|\rho| \geq 0.3$) the energy consumption, while the other 12 have $|\rho|$ values between 0 and 0.3 with 1 being negatively related to energy consumption. However, all 25 system calls have high R^2 values. 20 of the 25 models had $R^2 \geq 0.6$. A model relating the mean sum total of all system call counts to power was also constructed, as shown in Table 3.1. Sum count of all system calls correlates weakly ($|\rho| = 0.37$) but its linear relationship has a higher R^2 value of 0.74. Though the linear model coefficient b_1 for almost all the system calls is small, they are statistically significantly different from zero with p-value ($\leq 1 \times 10^{-12}$). The coefficients are small because the number of call invocations are high and the units used (mean watts) are large. As the linear models have robust R^2 values, they capture a relationship between the system call counts and energy consumption of the `Calculator` application.

3.2.5.2 Firefox

By using similar one variable linear models it was found that only 10 out of 53 of the system calls are mildly correlated (Spearman's $|\rho| \geq 0.3$) to energy consumption, while 36 have $|\rho|$ values between 0 and 0.3 and the remaining 7 are negatively correlated. However, most of the linear models have low R^2 values, most of them with R^2 less than 0.3. A linear model based on the sum

System Call	Description [23]
`mmap2`	map files or devices into memory (called to allocate memory and load libraries)
`open`	open a file descriptor
`close`	close a file descriptor
`epollwait`	Wait for events on the epoll file descriptor
`mprotect`	set protection on a region of memory
`recvfrom`	receive messages from a socket
`writev`	write data into multiple buffers
`cacheflush`	flush contents of instruction and/or data cache
`ioctl`	performs device-specific I/O operations

Table 3.2: Selected system calls with their descriptions from the `Calculator` application test case.

System Call	Spearman's ρ	R^2	Coefficient b_1	$\bar{x}$	$b_1 \cdot \bar{x}$
`_llseek`	0.100	0.040	1.30×10^{-4}	208.45	0.029
`brk`	-0.170	0.001	-8.07×10^{-5}	72.175	-0.005
`mmap2`	0.360	0.257	3.50×10^{-4}	546.95	0.190
`ioctl`	0.407	0.534	1.77×10^{-4}	1413.16	0.250
`epollwait`	0.088	0.274	3.41×10^{-5}	6293.50	0.210
`ftruncate`	0.001	0.269	2.64×10^{-7}	10.23	0.000
`fsync`	0.044	0.126	1.00×10^{-3}	78.46	0.120
`close`	-0.030	0.016	3.10×10^{-6}	8117.55	0.020
`fstat64`	0.060	0.014	1.57×10^{-5}	1517.78	0.023
`dup2`	0.235	0.008	1.40×10^{-3}	11.43	0.002
`write`	0.199	0.072	5.63×10^{-6}	7313.99	0.041

Table 3.3: Linear model summary for selected system calls for `Firefox` application

total of all system calls has a correlation of 0.606 and R^2 value of 0.31. The coefficient, b_1 is statistically significant with very small p-values($\leq 1\times10^{-12}$).

10 selected linear models are shown in table 3.3 and the system calls used are described in table 3.4. Table 3.3 also shows the mean number of invocations of each system call across all versions as $\bar{x}$ and the mean power contribution of each system call estimated by the linear models ($b_1 \cdot \bar{x}$) in the last column.

3.2.6 Logistic Regression Modeling

A logistic regression model was constructed by relating a significant change in energy consumption (compared to a reference) to a significant change in system call counts (compared to the same reference).

System Call	Description [23]
`_llseek`	reposition read/write file offset
`brk`	change data segment size
`mmap2`	map files or devices into memory
`ftruncate`	truncate a file to a specified length
`ioctl`	performs device-specific I/O operations
`epollwait`	Wait for events on the epoll file descriptor
`fsync`	synchronize a file's in-core state with storage device
`close`	close a file descriptor
`fstat64`	get file status
`dup2`	duplicate a file descriptor

Table 3.4: Selected system calls with their descriptions from the `Firefox` application test case.

Developers are concerned about the effect that their code changes have on their application's energy consumption. Logistic regression models were created to address the question, "*can we predict whether our application's energy consumption changes or not by using changes in system call counts?*"

Logistic regression requires a binary classification. Thus, each application version's energy consumption is characterized as *high* or *low*. Using the mean system call counts from all versions as a reference, the difference between the system call count for each version and the reference is calculated. High energy consumption is defined as being more than the mean, and low energy consumption is defined as being less than the mean. This difference is used as the independent variable in the logistic regression.

Most system calls are highly correlated with other system calls. Thus, a model with a reduced number of system call counts as features can be constructed. An iterative approach was used to choose the appropriate independent variables (system call counts) for the model. This approach is to add system calls as independent variables one by one. If all system calls in the model were significant the new system call was kept as an independent variable. If not all system calls were significant to the predictions of the model, the insignificant system calls were removed. The final model is the one with the largest number of significant system calls.

3.2.6.1 Calculator

257 models using 5 system calls were found. No model was found where all 6 or more system calls were still significant. In order to select the best model out of these models, mean accuracy under 10-fold cross validation was used. The dataset is quite small and the variability in the cross validation error is high. To counter this variability, 10-fold cross validations were run 100 times and their accuracies were averaged. The model using `recvfrom`, `munmap`, `cacheflush`, `gettid`, and `epoll_wait` had the highest average accuracy of 87.7%. The model with the lowest accuracy using only significant system calls had an accuracy of 75.6%. The top 3 most common system calls in all 257 models were `write`, `gettimeofday`, and `getpid`. Using the most accurate model, *it is possible to predict the direction of the change in energy consumption relative to the reference energy consumption with high accuracy.*

3.2.6.2 Firefox

128 models containing 9 system calls were obtained. No model had 10 or more system calls that were statistically significant. Each of the 9 features is the change in the system call count of a version with respect to the mean call count of that system call across all versions. The highest average accuracy was 80.4% from the model with `recvmsg`, `lseek`, `read`, `gettimeofday`, `futex`, `epoll_wait`, `clock_gettime`, `cacheflush` and `access` as features. The lowest accuracy was 68.07%. The three most common system calls among the models were `ioctl`, `close` and `lseek`.

Using the above logistic regression models, it is possible to predict the direction of the change in energy consumption relative to the reference energy consumption mean with a high accuracy.

3.2.7 Rule of Thumb

Developers should not be required to use expensive and specialized power testing equipment in order to make predictions about the changes in energy usage caused by their changes to the application source code. One of the goals of this paper is to provide developers with a “rule of thumb” in order to make this prediction quickly. To provide this rule, the Student’s t-test was used to examine the relationship between significant changes in system call usage and significant changes in energy consumption.

In order to create a rule useful in real-world development processes, only consecutive versions were considered. 100 Student’s t-tests were performed for `Calculator` and 155 Student’s t-tests were performed for `Firefox` . Each test was done for a consecutive pair of versions to determine whether the version succeeding the change was significantly different from its predecessor. For each version, 10 power measurements were taken and compared with the 10 power measurements for the next version.

t-tests were also performed for each of the system calls, using the 10 call invocation counts to determine whether the usage of that system call changed significantly between any two consecutive versions. The null hypothesis (H_0) is the anti-thesis of the alternative hypothesis, that is the real claim being made, or the real hypothesis, *i.e., Rule of Thumb* in this case. p-value can be described as the probability of rejecting the null hypothesis when the null hypothesis is true, so that if p-value is low, it is safe to reject the null hypothesis. This threshold for low is decided by level of significance, α. Alternatively, p-value can be interpreted as the probability of observing the given data, if null hypothesis is true. A p-value of 0.05 indicates that there is 1 in 20 chance of observing the given data, if the null hypothesis is true. Since, we perform 10 runs per version to determine whether the measurements are statistically significant different, it would translate to a probability of 0.1 for observing extreme data, assuming that nine runs out of the ten produced extreme measurement with respect to the null hypothesis. An α of 0.05 assumes 95% measurements exhibiting extreme error with respect to null hypothesis, which is conservative given our number of runs. Hence, an α of 0.05 was chosen as a p-value threshold for establishing whether consecutive versions are significantly different or not. Since consecutive versions are tested, the control group is different for each t-test. Therefore, multiple test correction is not required.

The purpose of this procedure is to determine whether a significant change in system call use and a significant change in energy profile usually occur together. In order to establish the usefulness of the rule of thumb, precision, recall and specificity were calculated:

$$\begin{aligned}
Precision &= \frac{SS}{SS + SN} \\
Recall &= \frac{SS}{SS + NS} \\
Specificity &= \frac{NN}{NN + SN} \\
F_1 &= 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}
\end{aligned}$$

SS is the number of times the energy consumption difference as well as the system call count difference are significantly different; NS is the number of times the energy consumption is significantly different but the system call profile is not; SN is the number of times the system call profile is significantly different but the energy consumption is not; and NN is the number of times that neither are significantly different.

All four statistics — precision, recall, specificity, and F_1 range from 0 to 1 with 0 being the worst and 1 being the best possible value. High precision indicates that positive results from our rule of thumb indicate a significant change in energy consumption. High recall indicates that there were few significant energy consumption changes that the rule of thumb missed. High specificity indicates that our *Rule of Thumb* generates few false positives. F_1 being weighted mean of precision and recall, is a measure of accuracy. Higher the F_1, more balanced and accurate the model is.

In other words, if a developer used our rule of thumb to decide when to perform energy consumption tests, with high precision, more of the time they spent doing energy testing would yield significant results. With high recall, they would find more of the significant results that could be found. With high specificity, they would spend less time testing insignificant results. Table 3.5 summarizes these statistics for our *Rule of Thumb* on selected system calls on both applications.

3.2.7.1 Calculator

The highest recall of 0.909 was obtained using the system call `cacheflush` while the lowest was obtained by using `sendto` which had a constant number of invocations, 182, in all versions. The highest precision of 1 was obtained using the system calls `pread` and `stat64` while the lowest was obtained by using `sendto`. The highest specificity was obtained by using `cacheflush` while the lowest specificity was obtained by using `sendto`. For some system calls, changes in system call usage predicted changes in energy usage with high precision and recall while having high specificity as shown in Table 3.5.

3.2.7.2 Firefox

The highest recall, 0.6, is obtained using the system calls `lstat64`, `pipe`, and `utimes`. The lowest recall of 0.1 was obtained using seven different system calls. The highest precision, 0.263, was obtained using `ioctl` while the lowest was obtained using `stat64`. The highest specificity of 0.96 was obtained using `ioctl`, while using `cacheflush` had the lowest at 0.93. Values for precision and recall are lower than those obtained for the `Calculator` application, however specificity is higher.

Sum of Calls in Table 3.5 refers to sum total of counts of all system calls, and can be used in a situation where the developer has no information about what system call will be best to use.

Given our threshold of $\alpha = 0.05$, `Calculator` 's energy consumption changed significantly 11 out of 100 times, or 11% and `Firefox` 's energy consumption changed significantly 10 out of 155 times, or 6.4%. Thus, by randomly guessing version pairs we would get a precision of 0.11 for `Calculator` and 0.06 for `Firefox` . Because the number of commits that change energy consumption significantly is low and energy testing is expensive, we expect a much higher specificity than recall. The unbiased coin flip detects on an average of 50% of the significant change power versions, giving a recall of 0.5. Random guess gives equal number of false negatives and true negatives, giving a specificity of 0.5 also. Regardless of data, in case of randomly guessing,

`Calculator` application				
	Precision	Recall	Specificity	F_1 score
`sendto`	0	0	0.89	0
`stat64`	**1.00**	0.55	0.98	**0.71**
`cacheflush`	0.34	**0.91**	**0.98**	0.56
Sum of calls	0.35	0.72	0.96	0.47
Coin flip	0.11	0.50	0.50	0.18
`Firefox` application				
	Precision	Recall	Specificity	F_1 score
`fcntl64`	0.04	0.10	0.94	0.057
`ioctl`	**0.26**	0.50	**0.96**	0.34
`lstat64`	0.08	**0.60**	0.95	0.14
Sum of calls	0.18	0.60	0.97	0.27
Coin flip	0.06	0.50	0.50	0.11

Table 3.5: *Rule of Thumb* — precision, recall, specificity, and F_1 score for the *Rule of Thumb*, using best and worst system calls for the `Calculator` and `Firefox` applications.

recall, and specificity must sum to 1. An unbiased coin flip usually performs worse than our rule of thumb with a randomly selected system call. System calls that are constant between versions perform worse than a random guess. In practice, the random guess should be biased to predict lesser number of significant changes, thereby reducing recall and increasing specificity to limit testing cost. Even sum of system calls outperforms the unbiased coin flip on all three statistics. Even if a biased coin flip were to be used, the rule of thumb using the sum total would perform better.

3.2.8 Discussion

The results in the previous sections show that system call counts are somewhat stable but still have variance. This observation implies that repeated measurements are required in order to address the high variability of some of the system calls counts. The linear models obtained show that many system calls correlate with energy consumption. This was demonstrated by high R^2 values obtained with the `Calculator` application models. However, in the case of `Firefox` , the linear models do not perform as well, having both lower correlations and lower R^2 values.

The logistic regression models successfully determined if a change in the system call profile would lead to a significant change in power consumption with accuracies between 75.6% and 87.7% for the `Calculator` application and accuracies of 68.07% to 80.04% for `Firefox` . Our logistic model is able to predict whether power usage will change significantly with high accuracy using the changes in system call usage.

While it might be the case that an application is using a lot of power doing CPU-only computations while not making system calls, this is rare because usually CPU-heavy computations involve memory management system calls such as `sbrk`. Even in the case that they do not, this type of application behaviour will still be evident in the system call profile. Waiting for system call completion is usually the only way that an application can allow the CPU to idle. Thus, we expect system calls like `epollwait`, which is specifically used to idle the application, to have a positive correlation with power use. This is because in order for the application to sleep many times, it must wake up and perform computations many times. And in fact, this is what was observed in Table 3.3.

I/O system calls can also be an indicator that an application is using power-hungry peripherals. For example, an application might use `write` to send data to a remote computer over the network, activating the phone's wifi transmitter, which requires power. Thus, a network application like `Firefox` should show a positive correlation between `write` calls and power consumption, which is what was observed.

Some system calls have a negative correlation with power use. This correlation is most likely caused by an application using alternate methods which use less power to achieve the same result. Consider the case of `Firefox` , which, like all browsers, has a cache. If `Firefox` can retrieve data from its cache it doesn't need to use wifi as much, saving power.

The rule of thumb, "energy profile usually changes significantly when the system call profile changes significantly," is supported by both datasets, however it achieves higher precision, recall and specificity on `Calculator` than on `Firefox` . Since rule of thumb outperforms the random guess, it provides developers an insight to review their last change with the application's energy consumption perspective, resulting in large time savings, when instrumentation is not available or not feasible.

Thus, developers who keep track of system call profiles for their application can make an informed decision on when more expensive power testing may be useful. Profiling system calls is less resource intensive than setting up a special hardware test bed and using power instrumentation to profile power consumption. The rule of thumb requires only that developers track system call profiles, which is very simple.

3.3 Chapter Summary

In this chapter, the relationship between system call invocations and energy consumption across multiple versions of two Android applications was investigated. System calls were found to suffer from limited variability, thus by relying on mean system calls numbers one could linearly model the relationship between system calls and energy consumption. Most system calls are mildly correlated to the energy consumption.

Change model using logistic regressions to predict if a new version has significantly different energy consumption compared to an older version based on the difference in system call invocations, achieved high accuracy.

Our proposed hypothesis that a significant change in an application's system call profile predicts a change in the application's energy consumption profile, is supported by the results obtained. We demonstrated a relationship between energy consumption and system call profiles, and propose a practical method for developers to track their energy profiles.

Chapter 4

Green Advisor

In the previous chapter, the green mining methodology [16] was used to correlate system-call counts with the energy consumption of android applications. A simple and easy to use qualitative *Rule of Thumb* was proposed to predict whether energy consumption will change or not based on the system-call counts. Even though this is a very simple model, and a rule of thumb, it can potentially be very useful since developers have been found to be generally unaware of the factors affecting energy consumption of their applications [30] [26].

Most of the research effort and tools have been devoted to model energy consumption to help end user track their smartphone's energy consumption [42], [13], [7], [37], [24]. However, there has been very little effort to help developers track and optimize energy consumption profile of their applications. For this reason, developers are facing a lack of tools and knowledge for energy aware development [30]. Given this context, and motivated by the results from the case study, a tool called *GreenAdvisor* was developed to operationalize this *Rule of Thumb* model. In the context of evolutionary software development, it compares the system-call logs of subsequent commits and predicts how the new version's energy-consumption profile will compare to that of the previous version. In this chapter, we describe the architecture and working of the tool.

This chapter is organized as follows: Section 4.1 describes the *jUnit* framework used in this tool; Section 4.2 describes the architecture of the *GreenAdvisor*; Section 4.3 describes how code changes are related to system calls; Section 4.4 details the work-flow of *GreenAdvisor*.

4.1 *jUnit*

jUnit[1] is a unit testing framework available for the Java programming language. Unit testing is a software testing process in which the smallest possible fragments of an application termed as units, are independently inspected to determine whether they are performing desired operations. It is used extensively in test-driven development. Test-driven Development(TDD)is the process of developing code according to functionality that is required to be added and passing it through testing to make sure that it meets the requirements. In TDD, test cases are written first to make sure that the existing code fails to meet it. Then, application code is developed to ensure that the test case is validated. This ensures development that is more focused on requirements than the conventional development style. *jUnit* provides such a framework to test unit code on Java.

Since, Android has its own libraries, Android *jUnit* testing framework have been devised to provide unit testing to Android developers. Using this framework, developers can test the aspects of their application units. *jUnit* requires developers to write test cases in addition to their application code. These test cases would cover all aspects of the application if written in a test-driven development fashion.

GreenAdvisor assumes that *jUnit* framework is used to develop test cases for the Android application under consideration. Next, the architecture of the *GreenAdvisor* tool has been described.

4.2 Overview

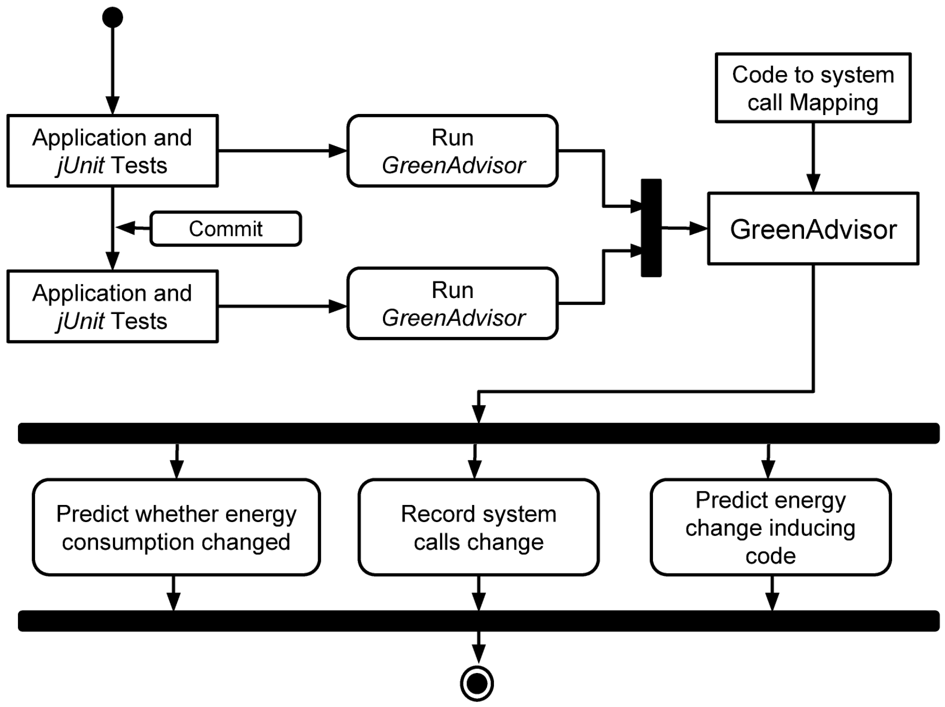
The *GreenAdvisor* tool has been built to enable programmers to apply the *Rule of Thumb* model on their own Android applications without the need for expensive hardware based energy consumption instrumentation. It profiles the system-calls of application test cases, and uses the *Rule of Thumb* [3] to alert the application developer about a possible change in the energy-consumption

[1]http://junit.org/

profile of their application. By profiling the system-call traces between application versions, it alerts the developer about possible changes in energy consumption profile. If *GreenAdvisor* predicts change in application's energy consumption profile, it also highlights the code that might be responsible for that change using a simple bag-of-words model to relate system-calls to Java API calls.

The *GreenAdvisor* uses the *jUnit* tests constructed by the application developers as test cases for application usage. This tool uses application executables (.apk files in Android) and the application's Android *jUnit* test cases to obtain system-call trace profiles. While the *jUnit* test cases are executing on the application, it uses *strace* program to profile the system-call counts of the tests. These test runs are performed multiple times, the default being 5 times per commit-version to obtain both an average and a distribution of measurements. These system-call measurements are compared together using a Student's t-test in order to apply the *Rule of Thumb*. Since, we have to use 5 runs per commit-version so that its less cumbersome for developers to monitor their application, we use a less conservative level of significance α, of 0.1 instead of 0.05, as we have reduced our runs by half from 10 to 5. A p-value of 0.1 indicates that there is 1 in 10 chance of observing the given data, if the null hypothesis is true. Since, we perform 5 runs per version to determine whether the measurements are statistically significant different, it would translate to a probability of 0.2 for observing extreme data, assuming that four runs out of the five produced extreme measurement with respect to the null hypothesis. An α of 0.1 is conservative, and hence, used for the t-tests.

The *GreenAdvisor* generates a report as shown in Fig. 4.2, listing the system-calls that have changed significantly, and the source code in the `diff` between versions that potentially causes the functional changes in the application. For example, if calls to the `open` system-call increase significantly, the tool indicates to the developer that they might be performing more number of open file operations. In the generated report, it also highlights the code change that might be responsible for the change in the system call profile using a bag-of-words model. The work-flow of *GreenAdvisor* is shown in Figure 4.1.



Application and *jUnit* Tests
Run *GreenAdvisor*
Commit
Application and *jUnit* Tests
Run *GreenAdvisor*
Code to system call Mapping
GreenAdvisor
Predict whether energy consumption changed
Record system calls change
Predict energy change inducing code

Figure 4.1: Activity diagram of workflow of the *GreenAdvisor* tool.

System Call	Code Expressions
`fsync`	.sync()
`open`	.open()
`close`	.close()
`getsockname`	.getSocketLocalAddress(); .getsockname()
`read`	.read(); .readLine()
`access`	new FileInputStream(); new FileOutputStream(); new InputStream();
`recvmsg`	.recvmsg(); receive()
`writev`	.write();new BufferedWriter();new PrintWriter(); .flush()

Table 4.1: Code expressions for selected system calls

4.3 Mapping Code to System Calls

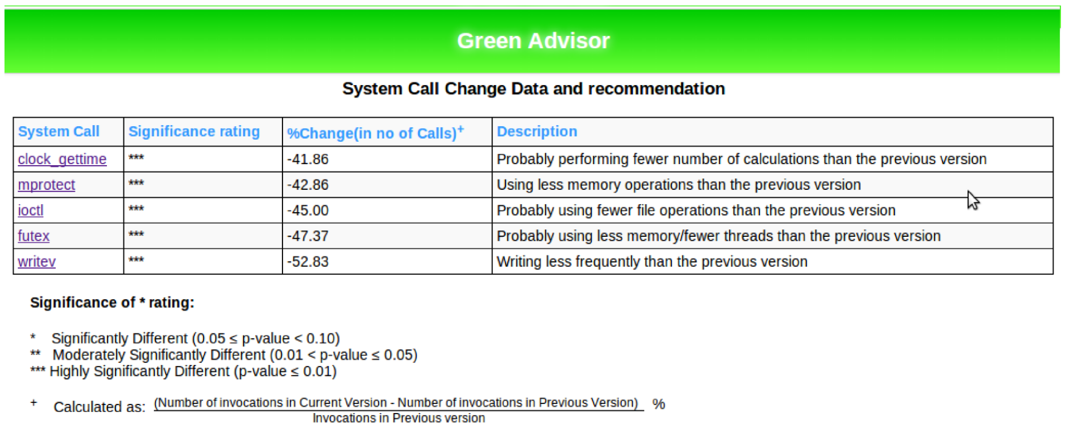
GreenAdvisor uses bags of words, dictionaries, related to a system-call to annotate source code relevant to that system-call. The bag-of-words are created from observed system-calls and the Java APIs that potentially generate these system-calls. This is currently a manual and ad-hoc process where by keywords, identifiers, APIs, and terms from Java code and Android APIs are grouped into bags-of-words relevant to a particular system-call. Keywords and identifiers from file I/O APIs, and database APIs such as `FileInputStream` and Android Preferences API have been inserted into bags-of-words for `open`, `write`, and `read` (depending on the exact identifier). Some system-calls such as `close` might not be explicitly called (finalizers called during garbage collection). These word bags have been built manually, and could be improved by a variety of automated techniques.

If there is a significant change in `open` system-call, that might be mapped to `FileInputStream`'s `open()` method. Some examples of the mappings are provided in Table 4.1. If a system-call is found to have been changed significantly, the tool searches for these Java method(s) corresponding to that system-call in the `diff` of application code between code prior to commit and code after the commit. If any of those identifiers are found in the `diff`, they are indicated in the report with that system-call. By indicating the changed code that might be behind the expected energy consumption change the tool would help developers review their code change, and possibly optimize it. A code segment highlighted by the tool in a sample run is shown in Figure 4.3.

4.4 Workflow

The *GreenAdvisor* workflow involves the following step sequence.

1. It takes as input the application executable APKs and *jUnit* tests.
2. Next, it records the system-call-profile for selected (possibly all) versions of the application by executing test runs multiple times.


Green Advisor

System Call Change Data and recommendation

System Call	Significance rating	%Change(in no of Calls)$^+$	Description
clock_gettime	***	-41.86	Probably performing fewer number of calculations than the previous version
mprotect	***	-42.86	Using less memory operations than the previous version
ioctl	***	-45.00	Probably using fewer file operations than the previous version
futex	***	-47.37	Probably using less memory/fewer threads than the previous version
writev	***	-52.83	Writing less frequently than the previous version

Significance of * rating:

* Significantly Different (0.05 ≤ p-value < 0.10)
** Moderately Significantly Different (0.01 < p-value ≤ 0.05)
*** Highly Significantly Different (p-value ≤ 0.01)

$^+$ Calculated as: $\frac{\text{(Number of invocations in Current Version - Number of invocations in Previous Version)}}{\text{Invocations in Previous version}}$ %

Figure 4.2: A sample report generated by the *GreenAdvisor* tool indicating a change in system-call counts.

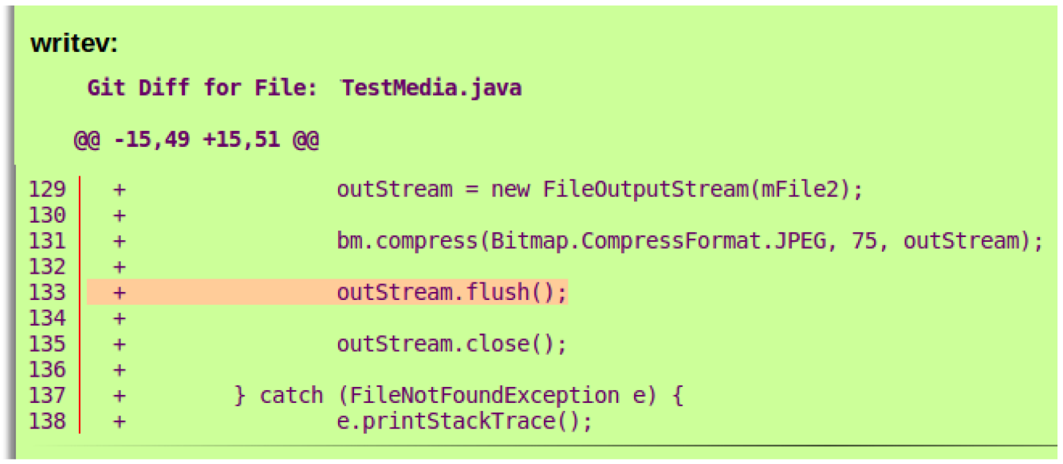
3. It compares the system-call profiles of subsequent application versions and using the *Rule of Thumb* predicts whether a significant energy-consumption change may have occurred between the two versions.

4. Next, it uses a pre-defined bag of words to map the changed system-calls to corresponding Java methods in the application, to indicate which segments of the code in the commit `diff` might be responsible for the change.

5. Finally, it generates a report containing the system-calls that have changed and the code changes that might be responsible for these changes.

First, the tool takes as input the path of the directories where the application project and the *jUnit* test project are present. It finds the application executables(APKs) for both of the application and test project from within those directories. Then using the `InstrumentationTestRunner`, the tool executes the test cases on the emulator/device. The tool records the system-call invocation counts being made by the application in a database using the `strace` tool that traces all the system calls being invoked by the application process.

Now, if the developer makes a commit, *i.e.*, makes a change in the application code and wants to determine if this has induced any change in the energy consumption of the application. In order to do this, tool would be re-run on



writev:

```
    Git Diff for File:  TestMedia.java

   @@ -15,49 +15,51 @@
+                outStream = new FileOutputStream(mFile2);
+
+                bm.compress(Bitmap.CompressFormat.JPEG, 75, outStream);
+
+                outStream.flush();
+
+                outStream.close();
+
+        } catch (FileNotFoundException e) {
+                e.printStackTrace();
```

Figure 4.3: A code segment of `diff` highlighted by the *GreenAdvisor* tool indicating the likely source of the `writev` system-call.

the updated application and will record the system call counts being invoked by the updated application.

Using the system call data for the previous version on which *GreenAdvisor* was ran, the tool determines the change in the system call profile and the determines all the system calls that exhibited significant change in new commit-version. If there is any such system call, the tool indicates that there is a change in the energy consumption profile of the application in the new commit-version. All this information is displayed in the report generated by the tool as shown in Figure 4.2 along with the description of the system call(s) that exhibited the significant change. For example, if `writev` system call is observed to have a reduction in invocation count, the tool indicates that the latest commit-version might be performing fewer write operations than the previously recorded commit-version.

Next, using the *Git* `diff`, the tool determines the code that was changed in-between the two commit-versions. Using this and the bag-of-words model as described in previous section, *GreenAdvisor* determines the code that might had been responsible for the change in the call invocation counts of the the system call(s) identified in the previous step. This is indicated in the report as well, as shown in the Figure 4.3 for the `writev` system call.

Hence, this way developers can keep a track of the change in energy consumption profile of the application they are developing and also possibly the code that might be responsible for that change. Instructions to run the tool, and sample report can be found at [1].

4.5 Chapter summary

In this chapter, using results from previous chapter a first of its kind, system-call profile based tool *GreenAdvisor* has been introduced that predicts changes in energy profile of applications with change in code base. *GreenAdvisor* tool uses an application's system-call profile to warn developers about the possible change in energy consumption profile of your applications. The *GreenAdvisor* tool implements and employs an augmented *Rule of Thumb* proposed in previous chapter. *GreenAdvisor* also indicates the possible code that might be responsible for that change by using pre-defined code patterns for a set of system-calls using a bag-of-words model. In next chapter, we evaluate *GreenAdvisor* using a user study.

Chapter 5

Evaluating Green Advisor

In order to measure perceived usefulness of *GreenAdvisor* a user study has been performed on *GreenAdvisor* to be evaluated against the projects of several software teams, while at the same time probing the developers' understanding of the factors that affect the energy consumption of their applications. Furthermore, the original *Rule of Thumb* model has been augmented to predict not only whether the application energy consumption will change but also the direction of the change, and this new model was evaluated on important commits.

Using the *GreenAdvisor* tool, a short user study was conducted with 77 third year university students, forming 11 student teams developing geo-location-aware question-and-answer software Android applications as their course projects while 2 teams developed a travel expense claim application. Using these projects, experiments were run to evaluate the prediction models. This chapter describes our data-collection method, our survey, and experiments run on the 13 projects.

This chapter is organized as follows: Section 5.1 describes the user study in an undergraduate class setting, methodology used, and results of the study; Section 5.2 describes the proposed improved model and its evaluation.

5.1 User Study on the *GreenAdvisor* Tool

In order to evaluate the *GreenAdvisor* tool, a voluntary survey was distributed among the 3rd year undergraduate class of an introductory software engineering course, CMPUT 301 at University of Alberta during Fall'14 and Winter'15 terms. A demonstration of *GreenAdvisor* was given in the class and its usage was explained to the students. The student teams were asked to run the tool on selected commits that they thought might change the energy consumption of their applications. They were also asked to examine whether the code segments identified by the tool matched with their expectations of the energy consumption change inducing code. The survey forms can be found in Appendix A.4. Students could run the *GreenAdvisor* tool in the lab or at home in their own time. Based on this exercise, they were asked to evaluate the tool with respect to its effectiveness in predicting the energy change and in identifying the code that induced that energy change. They were also asked to write a general paragraph about the factors they thought affected the energy consumption of their applications and their views on the tool. This survey was approved by Ethics board at University of Alberta (ID: Pro00050197).

77 students (13 teams) consented for the study, and out of those 42 students (7 teams) filled in the survey questionnaire. The effort per student on the course project can be estimated to be around 80 hours per student, approximately 480 person-hours per team, throughout the term. The combined effort of all the teams who participated was more than 6160 person-hours. Time was estimated based on student feedback forms.

5.1.1 Student Projects

All of the 11 team projects in Fall'14 aimed to build an online question-and-answer forum, accessed through a mobile application from any location. A set of application requirements were given to the students, including posting questions, searching for questions, browsing questions, posting replies to questions, sorting questions/answers, favouriting questions, upvoting/downvoting answers, attaching geolocation to questions and so on. All 11 teams used the same requirement specifications. There were three course-project deliverables (project parts) during three months of project duration.

The projects of 2 teams that participated in the study in Winter'15, aimed to build a travel expense claim submission application for an organizational

Table 5.1: Number of commits, Lines of Code (LOC), and Number of files of each teams' project.

Team	#Commits	LOC	#Files
1	252	3576	70
2	573	9568	196
3	840	6739	81
4	637	7827	120
5	626	6077	119
6	425	10196	154
7	458	6011	103
8	549	4420	86
9	390	9243	153
10	555	7543	112
11	391	5999	109
12	704	9269	185
13	382	10708	137
Average	521	7475	125

setting. The set of requirements included creating claims, submitting claims, approving claims, providing offline functionality, adding geolocation to the travel destination and so. There were three course-deliverables, similar to the Fall'14 term projects.

Students started writing *jUnit* tests after the first deliverable. The *jUnit* tests were written explicitly to test the use-cases and user-stories of the project. Test-driven development was promoted during this period. Between the deliverables, they developed key application features such as user location recording, online storage with Elasticsearch, taking photos and offline storage. Table 5.1 shows number of commits for each of the projects. The number of commits vary widely across projects from 252 commits to 840 commits.

5.1.2 Expert Use of the Tool

In parallel with our survey of the student developer teams, some distinct commits from each of the 13 projects (from teams that had consented) were selected manually to evaluate the *Rule of Thumb* and on a much larger dataset than our previous study.

5.1.3 Selection of Commits

As each project has 100s of commits, it is infeasible to run experiments on all the commits as most of the commits represent very trivial changes, and hence, there is not much difference between successive commits. Therefore, only large-sized and deliverable commits, important to deadlines were evaluated. Prior work [33] suggests that estimating energy profiles based on sub-selecting commits does not heavily harm the accuracy of energy consumption profile estimation. In fact, energy consumption profiles tend to consist of plateaus of constant energy consumption with intermittent changes in energy consumption. For each of the deliverables, the commits that made the largest change to the code base were selected. In addition, the largest commits in between two deadlines were selected, as they are likely to represent an appreciable change in the application code — this was an attempt to ensure uniform selection as prescribed by Romansky *et al.* [33]. Finally, the commit messages were examined and, if any appeared important, the corresponding commits were chosen as well. Using this procedure, around 10 commits per project were gathered. It was also made sure that the commits selected had no change in the *jUnit* tests, so that the energy consumption values in the commit-versions considered are comparable and reflects changes only due to change in the application code only. These commits were generally large ones, except the ones submitted as course deliverables. If a commit version that was submitted as deliverable did not had any functional change in the application, the last commit that made a code change in the application code was selected. In many cases, the large commits introduced some errors like build errors, *jUnit* test crash errors, so that the next available functional commit was selected. Most of the commits selected were after the manual inspection of commit messages.

5.1.4 Evaluating *Rule of Thumb*

The *Green Miner* test bed [18] was used for obtaining the energy consumption measurements of the various versions (commits) of the applications using the methodology described in previous chapter. It was configured to run *jUnit* tests. The selected versions were run to obtain the energy consumption values. With the help of *GreenAdvisor* the system-call profile counts were obtained. Multiple energy consumption measurements recorded from *Green Miner* were used to establish if the two commits had statistically significantly different energy consumption by performing *t*-tests. Then, *t*-tests were performed on the multiple system-call counts to observe whether the system call profiles were statistically different from the previous commit or not.

In summary, for each of the 13 projects, 10 versions of the project were built, each version was tested and its energy consumption measured 10 times, then its system-calls were measured 10 times, resulting in 1300 energy consumption measurements and 1300 system-call measurements across 13 projects. Using these measurements, four metrics described in the previous section, precision, recall, specificity, and F_1 score, were calculated for evaluating *Rule of Thumb* model on each of these projects.

5.1.5 Results

In this section, results obtained from the student survey of use of tool, and the evaluation of *Rule of Thumb* on student projects has been described.

5.1.5.1 User Study

All teams in the class, received the *GreenAdvisor* tool in the lab-sections for the course and used the tool in the lab and on their own computers. Out of the students (13 teams) that consented, 42 students (7 teams) submitted the survey forms. Each of the teams identified some commits they thought could have caused a change in the energy-consumption profile of their application. Then, on these commits, teams used the *GreenAdvisor* tool to generate report and prediction about the energy consumption change. Out of 7 teams, 4 agreed, while 3 disagreed, that tool was able to identify code associated with change in system-calls. Out of those 3, 1 team was not able to see the associated code with the system-calls because those system-calls did not had a pre-defined code patterns in bag-of-words model. The other two teams expected change in energy consumption of their application while the tool predicted no change. Out of 7 teams, 5 teams said that the tool was able to indicate energy consumption change regularly, while other 2 reported that it worked only sometimes. Results are summarized in Table 5.2. have

Empirical verification: the commits mentioned by the groups were examined to identify whether they really exhibited an energy-consumption change, as the student developers thought. Only 5 out 7 identified commits (across 7 projects) exhibited a change in the energy-consumption profile of the application. 2 out of 3 teams that had reported that the tool was unable to predict energy consumption changes, had identified commits that did not produce energy consumption changes.

The students were asked to provide a paragraph describing their use of the tool. Most students described the system-calls identified by the tool. Some of the students described some ideas on they could optimize the energy consumption of their applications like minimal use of GPS, using dark colors, optimizing local storage update with the remote server – matching some of the suggestions of Pinto *et al.* [30]. Interestingly, students did not indicate that they knew anything about these factors until they had done their own research online, reading literature and forums, in order to produce these paragraphs.

Hence, 5 (30 students) out of 7 student groups (42 students) identified commits in their own project that induced a change in energy consumption; 4 (24 students) out of 7 groups felt that the bag-of-words system-call identifying code approach worked; 5 (30 students) out of 7 student groups felt that the *GreenAdvisor* tool was able to indicate energy consumption changes regularly. While some of the results are mixed, the tool was usable for most of the students and was able to identify relevant system-calls for multiple teams. The user-study indicated that more work could be done on system-call inference from changed source code. This student feedback serves mostly as a qualitative form of feedback and validation. The next section, when combined with these results, confirms that this method indeed is capable of working for actual developers.

Table 5.2: Survey questionnaire responses

Question	Team 2	Team 3	Team 4	Team 9	Team 11	Team 12	Team 13
The tool is able to identify the code associated with the change in system calls. How much do you agree with the above statement?	Disagree	Disagree	Agree	Strongly Disagree	Agree	Agree	Agree
In your opinion, how well does the tool predict whether a source-code change affects energy consumption?	Sometimes [20-49%]	Regularly [51-80%]	Regularly [51-80%]	Sometimes [20-49%]	Regularly [51-80%]	Regularly [51-80%]	Regularly [51-80%]

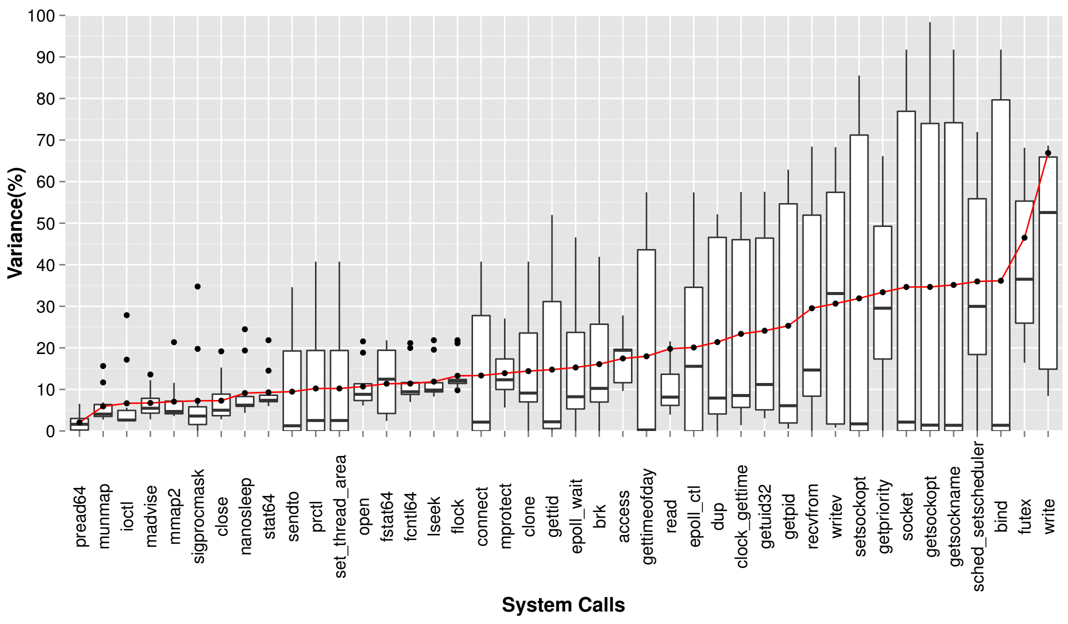


Figure 5.1: Variance as a percentage of mean value across 30 runs per system-call per version for all the applications considered. The X axis refers to the system-calls, while the Y axis refers to variance in percentage.

Table 5.3: Precision, recall, and specificity values of prediction of energy change using Rule of Thumb model for each team project

Team	Rule of Thumb			
	Precision	Recall	Specificity	F_1
1	0.33	0.66	0.75	0.44
2	0.50	1.00	1.00	0.66
3	0.44	1.00	1.00	0.61
4	0.11	1.00	1.00	0.20
5	0.43	1.00	1.00	0.60
6	0.50	1.00	1.00	0.66
7	0.57	1.00	1.00	0.73
8	0.66	1.00	1.00	0.80
9	0.44	1.00	1.00	0.61
10	0.33	0.66	0.75	0.44
11	0.43	1.00	1.00	0.60
12	0.57	1.00	1.00	0.73
13	0.43	1.00	1.00	0.60
Average	0.44	0.95	0.96	0.59

5.1.5.2 Rule of Thumb on Student Projects

In order to evaluate the *Rule of Thumb* model on these 13 projects built by 77 students, energy consumption and system call counts for the selected commits were used. Using the system call counts, the *Rule of Thumb* predicts when the energy consumption of the application is changed significantly. This prediction is verified by obtaining the actual energy consumption measurements from the actual execution of application versions (commits) on the Green Miner.

First the question of system-call stability across 13 projects was investigated. The variance in system-call values are shown in Figure 5.1 by using 30 runs per commit-version of each application. As can be observed, the variation values are quite similar to ones obtained before as shown in Figure 3.1 and Figure 3.2.

4 statistical measures — precision, recall, specificity, and F_1 score are used to measure the effectiveness of the rule of thumb. Average precision was 0.44, recall 0.95, specificity 0.96, F_1 score 0.59 across the 13 projects as shown in Table 5.3. As can be observed that recall and specificity values are quite high, though the precision and F_1 scores are not that high, similar to the results obtained previously in Chapter 3.

11 projects from *Fall'14* term varied in their implementation, though the requirement specifications were exactly same for all the projects; similarly for the two projects from *Winter'15* term. Students were expected to create a questions-answer application according to these user specifications. In all the projects, they had to use an *Elasticsearch* [1] server provided by the instructors to store their application data remotely. In absence of a connection to server, the application is expected to store the data locally on the phone. They were also expected to record and display the location of user in the application using GPS.

Most of the projects had bright screen colors, while only one had dark colors. Some of the applications frequently updated the server with the local data while some waited a bit more. The frequency of accessing the GPS

[1] https://www.elastic.co/products/elasticsearch

and updating the location of user also varied across the projects. Thus main variation points for energy consumption were primarily network IO, disk IO, GPS peripherals. The applications were not heavily CPU-bound or memory-bound. These choices impacted the energy consumption of these applications.

An observation made across all of the projects was that once students started using *Elasticsearch* server, their application's energy consumption increased significantly as expected. Using Elasticsearch meant that the students had to rely on more network IO. Another observation was that their applications' energy consumption increased once geo-location and GPS tracking was added to the applications. In one of the applications, a decrease in energy consumption was observed when the geo-location usage was disabled between certain versions of the application. Project 6 exhibited dark screen colors, and was found to have lower energy consumption than the other projects, though it had some commits where the energy consumption of the application was quite high. When geo-location was introduced to project 6, the mean power use went up to 1.6W, inducing more energy consumption. The normal power use was 0.8W prior to this commit, half of the energy consumption. Other projects had mean power use around 0.9W, though this might also be possible because of difference in *jUnit* tests and consequently the different test sequences.

5.1.6 Discussion

In this thesis, a first of its kind of tool – *GreenAdvisor* that can be used by developers to track energy consumption profile of their applications is introduced. The tool can predict the whether the energy consumption of application has been changed, but also tries to identify the source code changes that might have induced that change in energy consumption profile. While, most of prior work focused on providing a perspective to users about the components or applications utilizing energy of their mobile phones, this work provides a simple method and a handy tool for application developers to estimate changes in their applications energy consumption profile while developing their applications, without use of any hardware instrumentation.

For the survey, students were asked to identify the code commits that they expected to have induced change in their application's energy consumption profile. The survey was overall positive with most of them agreeing to the tool's ability to predict change in energy consumption profile. The cases where the tool indicated no change were indeed true though the students thought otherwise. However, tool failed to identify the lines of source code that induced changes consumption for a number of the responses because not all of the system-calls could be easily mapped to Java API calls by the authors. These results prompt further investigation into better identifying the source of system-call changes in source code.

From the student write-up about the tool and energy aware development, it is clear that students have only a rough idea of the impact of application design, and implementation decisions that could impact their application's energy profile though they did not used it in practice. Similar observation was made by Pinto *et. al.* [30] on *StackOverflow*, though their dataset was potentially from experienced developers rather than the undergraduate students considered in this study. This calls for an energy-aware development curriculum in the software engineering courses.

Using *Rule of Thumb* model to predict the occurrence of significant change in energy consumption of the application, high specificity was observed across all 13 systems, as was observed across 2 larger android applications in prior work [3]. High specificity indicates that our *Rule of Thumb* generates few false positives, and hence developers spend less time testing insignificant change commits. Profiling system-calls using *GreenAdvisor* is less resource intensive than setting up special hardware test-bed like Green Miner, and utilizing that to instrument energy consumption. The *Rule of Thumb* requires only that developers track system-call profiles, which is very simple with the help of *GreenAdvisor* tool, that additionally gives them a hint of the code that might be responsible for energy consumption change.

Table 5.4: Selected system-calls with their descriptions from the improved prediction model

system-call	Description [23]
`open`	open a file descriptor
`cacheflush`	flush contents of instruction and/or data cache
`mmap2`	map files or devices into memory
`epollwait`	wait for events on the epoll file descriptor
`write`	write selected bytes into file descriptor
`getpid`	returns the process ID of the calling process
`getpriority`	returns the current priority for a process
`sendto`	used to transmit a message to another socket
`munmap`	creates a new mapping in the virtual address space
`nanosleep`	suspends the calling thread for specified time
`clock_gettime`	finds the precision of the specified clock
`sigprocmask`	fetch and/or change signal mask of calling thread

5.2 Improved Model

Rule of Thumb only predicts whether the energy consumption of application has changed significantly or not. However, developers are more interested in the direction of change in energy consumption. They are more interested in knowing if their application's energy consumption has increased or decreased significantly. Given that *Rule of Thumb* has high specificity, *i.e.,* in the case where the *Rule of Thumb* predicts no significant change in energy consumption, developers can be sure about that prediction. However, in the case where *Rule of Thumb* predicts significant change in energy consumption, there is no indication whether the energy consumption has increased or decreased. Thus the model was augmented by logistic regression models. These models were constructed using the system-calls and energy consumption measurements data of the `calculator` and `firefox` applications from the previous Chapter 3, and the dataset from these 13 student projects. The logistic regression model takes as input the change in average system-call counts for consecutive versions and predicts whether the energy consumption has increased or decreased.

The following three models were built by using different data for training, and testing set as below:

Table 5.5: True positive and false positive rates of prediction of direction of energy change using the improved model. FF indicates Firefox, Calc indicates Calculator, P indicates the 13 student projects, 0.5P indicates half of the projects. Case 3 employs 10-folds cross validation.

Case	Train	Test	Precision	Recall	Specificity	F_1
1	FF Calc	P	0.50	0.43	0.62	0.46
2	FF Calc 0.5P	0.5P	1.00	0.33	1.00	0.50
3	P	P	0.63	0.53	0.53	0.58

- Trained on `calculator` and `firefox` datasets and tested on student projects.
- Trained on `calculator`, `firefox` and, randomly selected 50% of commits from combined 13 student projects dataset and tested on a remaining 50% commits student projects dataset.
- Trained and tested on student projects dataset using cross validation.

These logistic regression models are used in the case where tool predicts significant change in energy consumption to predict the direction of change.

5.2.1 Results

Logistic regression models were built to predict the direction of change in energy consumption *i.e.,* increase or decrease when the *Rule of Thumb* predicts significant change in energy consumption. Only those system calls were used to build the model who were individually statistically significant with energy consumption ($\Delta energy \sim syscalls$). 12 such calls were identified in cases `1` and `3`, while 7 in case `2`. These 12 calls common to all the models are shown in Table 5.4.

The results are shown in Table 5.5. The cases in the Table 5.5 refer to the three experiments corresponding to the three different testing and training sets. In case `1`, the model was trained on the `firefox` and `calculator` dataset to be tested on the projects dataset. In case `2`, the model was trained on the `firefox`, `calculator` and 50% randomly chosen dataset from the student projects to be tested on the remaining projects dataset. In case `3`, the model was trained and tested on the combined dataset of all the student projects using 10-folds cross validation. From all the three cases, as it can be observed, by using more of projects dataset, the precision, specificity increases, hence better accuracy. Hence, this logistic regression model can be augmented with the *Rule of Thumb* in the case where *Rule of Thumb* predicts significant change in energy consumption of the application. Using the *Rule of Thumb*, the commits can be separated into versions where energy consumption changed significantly or did not change significantly. As *Rule of Thumb* is highly accurate for predicting the non-significant model change cases, the logistic regression model is applied on the cases where *Rule of Thumb* predicts significant change in energy consumption. The logistic regression model then predicts the direction of change on these cases *i.e.,* whether the energy consumption increased or decreased. Using this two layer model, it can be predicted whether the energy consumption of the applications changed or not, and if it did, to predict the direction of change.

5.2.2 Discussion

An improved model using the logistic regression to predict the direction of energy consumption change(increased or decreased) was proposed and evaluated. Though not exceptionally accurate, this model enhances the *Rule of Thumb* model by predicting direction of change in the cases where energy consumption of applications might had changed significantly. The logistic regression model was created and tested using foreign projects profiles, indicating that there is information to learn from the available corpus of software in the wild. Furthermore, it hints at potentially universal system-call models, rather than application specific ones.

5.3 Threats to Validity

Internal validity is constrained by our choice of project requirements given that all the projects had the same requirements. Also, the choice of commits-versions selected for investigation of *Rule of Thumb* model presents a validity construct.

The external validity is constrained by the test construction. It is possible that the tests students wrote had limited coverage of the applications' features. However, students were expected to use test driven development, and were marked on the quality of the tests. Hence, the tests covered most parts of the application. External validity is hampered by our use of student projects [9]. Some students are exceptional programmers, already employed in industry, many are not. Hence, the level of programming skills exhibit a high variations between teams as well, in contrast to industry where people working on a project are more likely to have have much better programming skills than novices.

5.4 Chapter Summary

In this chapter, the *GreenAdvisor* introduced in previous chapter was evaluated using a short survey conducted on students in an undergraduate class whose results were positive. Additionally, *GreenAdvisor* and its *Rule of Thumb* model were evaluated through a series of experiments indicating that they can accurately find commits that change energy consumption profile and they can determine the direction of the change. These experiments further confirm the generality and usefulness of *Rule of Thumb* model on much more number of applications than our previous work. An improved model of *Rule of Thumb* was also proposed and evaluated for predicting direction of change apart from the significance of change in the previous model.

Chapter 6

Conclusions and Future Work

This thesis investigated the relationship between system call invocations and energy consumption. The first half of the thesis, studied the relationship across multiple versions of two Android applications: `Calculator` and `Firefox`. Motivated by these results, and lack of tool support for developers, we introduced *GreenAdvisor* tool in the second half of the thesis which helps developers track impact of their code changes on energy consumption of their applications.

By relying on mean system calls counts, we modeled the relationship between system calls and energy consumption. Most system calls are mildly correlated to the energy consumption. We proposed a very simple *Rule of Thumb* model that can be leveraged by developers in absence of special instrumentation that is required to track the energy consumption profile of applications. Most of the tool support is only available for end users to track the energy consumption statistics of their smartphones, but few for the developers that hinders the energy-efficiency driven development. With this background, we introduced first of its kind, developer-centric system-call profile based tool *GreenAdvisor* that predicts changes in energy profile of applications with change in code base by using *Rule of Thumb* model. *GreenAdvisor* also indicates the possible code that might be behind that change from pre-defined code for a set of system-calls using a simple bag-of-words model. Tool is accessible at [1], with instructions to setup the initial environment.

GreenAdvisor and its *Rule of Thumb* model were evaluated through user study and a series of experiments indicating that they can accurately find commits that change energy consumption profile and they can determine the direction of the change. Hence, in the absence of any instrumentation, *Rule of Thumb* allows the developers to track energy consumption profiles of their applications. An improved model using logistic regression can be used with *Rule of Thumb* to predict the direction of change of energy consumption as well. The proposed method and tool are helpful for developers to trace the energy profile of their applications without using any special instrumentation that is rarely accessible to developers.

This thesis helps developers determine energy consumption profile changing commits by aggregating a profile of system-call counts over many versions by applying a simple *Rule of Thumb* model and provides a practical first of its type *GreenAdvisor* tool that can be used to that end. Hence, this study demonstrates a relationship between energy consumption and system call profiles, providing a promising research direction, and a practical method for developers to estimate their code change's impact on energy consumption by using system calls profile.

6.1 Future Work

This work can be extended by more accurate and general prediction models. These models can be made using system-calls using data from a variety of applications like gaming, browser, weather, video players, music players to have general dataset of system calls and energy consumption. In this work, counts of the system calls were used to build simplified models. Another direction of research can be temporal mining of system call sequences to energy consumption. However, as the sequences are variable even for the exactly same conditions, as observed in Chapter 3 and Chapter 5, this might require some extra measures.

In this work, the commonly used Java methods were mapped to system calls manually. This could be improved by devising techniques that automatically associate API and common patterns to system calls for detection of energy change inducing code.

Bibliography

[1] Karan Aggarwal. Greenadvisor. https://github.com/kaggarwal/GreenAdvisor, 2014.

[2] Karan Aggarwal, Abram Hindle, and Eleni Stroulia. GreenAdvisor: A Tool for Analyzing the Impact of Software Evolution on Energy Consumption. In *International Conference on Software Maintenance and Evolution (ICSME)*. to be published, 2015.

[3] Karan Aggarwal, Chenlei Zhang, Joshua Charles Campbell, Abram Hindle, and Eleni 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.

[4] Niranjan Balasubramanian, Aruna Balasubramanian, and Arun Venkataramani. Energy Consumption in Mobile Phones: A Measurement Study and Implications for Network Applications. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, IMC '09, pages 280–293, New York, NY, USA, 2009. ACM.

[5] Abhijeet Banerjee, Lee Kee Chong, Sudipta Chattopadhyay, and Abhik Roychoudhury. Detecting energy bugs and hotspots in mobile apps. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 588–598. ACM, 2014.

[6] Carlo Brandolese, William Fornaciari, Fabio Salice, and Donatella Sciuto. Analysis and modeling of energy reducing source code transformations. In *Proceedings of the conference on Design, automation and test in Europe-Volume 3*, page 30306. IEEE Computer Society, 2004.

[7] Aaron Carroll and Gernot Heiser. An Analysis of Power Consumption in a Smartphone. In *Proceedings of the 2010 USENIX conference on USENIX annual technical conference*, USENIXATC'10, pages 21–21, Berkeley, CA, USA, 2010. USENIX Association.

[8] Mian Dong and Lin Zhong. Self-Constructive, High-Rate Energy Modeling for Battery-Powered Mobile Systems. In *MobiSys '11*, pages 335–348, New York, NY, USA, 2011. ACM.

[9] J. Feigenspan, C. Kastner, J. Liebig, S. Apel, and S. Hanenberg. Measuring programming experience. In *Program Comprehension (ICPC), 2012 IEEE 20th International Conference on*, pages 73–82, June 2012.

[10] Jason Flinn and M. Satyanarayanan. PowerScope: A Tool for Profiling the Energy Usage of Mobile Applications. In *Proceedings of the Second*

IEEE Workshop on Mobile Computer Systems and Applications, WMCSA '99, 1999.

[11] Martin Fowler. Refactoring: Improving the design of existing code, 1997.

[12] Ashish Gupta, Thomas Zimmermann, Christian Bird, Nachiappan Nagappan, Thirumalesh Bhat, and Syed Emran. Detecting energy patterns in software development. *Microsoft Research Microsoft Corporation One Microsoft Way Redmond, WA*, 98052, 2011.

[13] Sudhanva Gurumurthi, Anand Sivasubramaniam, Mary Jane Irwin, N. Vijaykrishnan, Mahmut Kandemir, Tao Li, and Lizy Kurian John. Using Complete Machine Simulation for Software Power Estimation: The SoftWatt Approach. In *Proceedings of the 8th International Symposium on High-Performance Computer Architecture*, HPCA '02, 2002.

[14] Shuai Hao, Ding Li, William G. J. Halfond, and Ramesh Govindan. Estimating Mobile Application Energy Consumption using Program Analysis. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 92–101, 2013.

[15] Shuai Hao, Ding Li, William G.J. Halfond, and Ramesh Govindan. Estimating Android Applications' CPU Energy Usage via Bytecode Profiling. In *First International Workshop on Green and Sustainable Software (GREENS), in conjunction with ICSE 2012*, June 2012.

[16] Abram Hindle. Green Mining: A Methodology of Relating Software Change to Power Consumption. In *MSR*, pages 78–87, 2012.

[17] Abram Hindle. Green Mining: Investigating Power Consumption across Versions. In *Proceedings of the 2012 International Conference on Software Engineering*, ICSE 2012, pages 1301–1304, Piscataway, NJ, USA, 2012. IEEE Press.

[18] Abram Hindle, Alex Wilson, Kent Rasmussen, Jed Barlow, Joshua Campbell, and Stephen 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.

[19] Young-Woo Kwon and Eli Tilevich. Reducing the energy consumption of mobile applications behind the scenes. In *Software Maintenance (ICSM), 2013 29th IEEE International Conference on*, pages 170–179. IEEE, 2013.

[20] Ding Li, Shuai Hao, Jiaping Gui, and William GJ Halfond. An empirical study of the energy consumption of android applications. In *Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on*, pages 121–130. IEEE, 2014.

[21] Ding Li, Shuai Hao, William GJ Halfond, and Ramesh Govindan. Calculating source line level energy information for android applications. In *Proceedings of the 2013 International Symposium on Software Testing and Analysis*, pages 78–89. ACM, 2013.

[22] Haroon Malik, Peng Zhao, and Michael Godfrey. Going green: An exploratory analysis of energy-related questions. In *Proceedings of the 12th Working Conference on Mining Software Repositories*. to appear.

[23] The Linux man-pages project. Linux man pages online. http://man7.org/linux/man-pages/, 2013.

[24] Radhika Mittal, Aman Kansal, and Ranveer Chandra. Empowering Eevelopers to Estimate App Energy Consumption. In *Proceedings of the 18th annual international conference on Mobile computing and networking*, Mobicom '12, pages 317–328, New York, NY, USA, 2012. ACM.

[25] Irineu Moura, Gustavo Pinto, Felipe Ebert, and Fernando Castor. Mining energy-aware commits. In *Proceedings of the 12th Working Conference on Mining Software Repositories*. to appear.

[26] Candy Pang, Abram Hindle, Bram Adams, and Ahmed E Hassan. What do programmers know about the energy consumption of software? *PeerJ PrePrints*, 3, 2015.

[27] Abhinav Pathak, Y Charlie Hu, and Ming 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*, page 5. ACM, 2011.

[28] Abhinav Pathak, Y. Charlie Hu, and Ming Zhang. Where is the Energy Spent inside My App?: Fine Grained Energy Accounting on Smartphones with Eprof. In *Proceedings of the 7th ACM european conference on Computer Systems*, EuroSys '12, pages 29–42, New York, NY, USA, 2012. ACM.

[29] Abhinav Pathak, Y. Charlie Hu, Ming Zhang, Paramvir Bahl, and Yi-Min 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.

[30] Gustavo Pinto, Fernando Castor, and Yu David Liu. Mining questions about software energy consumption. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 22–31. ACM, 2014.

[31] Gustavo Pinto, Francisco Soares-Neto, and Fernando Castor. Refactoring for energy efficiency: A reflection on the state of the art.

[32] Pew Research. Smartphone ownership 2013, 2013.

[33] Stephen Romansky and Abram 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.

[34] Cagri Sahin, Lori Pollock, and James Clause. How do code refactorings affect energy usage? In *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, page 36. ACM, 2014.

[35] Chiyoung Seo, Sam Malek, and Nenad Medvidovic. An Energy Consumption Framework for Distributed Java-Based Systems. In *ASE '07*, pages 421–424, 2007.

[36] Chiyoung Seo, Sam Malek, and Nenad Medvidovic. Component-level energy consumption estimation for distributed java-based software systems. In *Component-Based Software Engineering*, pages 97–113. Springer, 2008.

[37] Alex Shye, Benjamin Scholbrock, and Gokhan Memik. Into the Wild: Studying Real User Activity Patterns to Guide Power Optimizations for Mobile Architectures. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 42, pages 168–178, New York, NY, USA, 2009. ACM.

[38] Richard W. Stevens and Stephen A. Rago. *Advanced Programming in the UNIX(R) Environment (2nd Edition)*. Addison-Wesley Professional, 2005.

[39] Claas Wilke, Sebastian Richly, S Gotz, Christian Piechnick, and Uwe Aßmann. Energy consumption and efficiency in mobile applications: A user feedback study. In *Green Computing and Communications (GreenCom), 2013 IEEE and Internet of Things (iThings/CPSCom), IEEE International Conference on and IEEE Cyber, Physical and Social Computing*, pages 134–141. IEEE, 2013.

[40] Chenlei Zhang. The impact of user choice and software change on energy consumption. Master's thesis, University of Alberta, 2013.

[41] Chenlei Zhang, Abram Hindle, and Daniel M German. The impact of user choice on energy consumption. *Software, IEEE*, 31(3):69–75, 2014.

[42] Lide Zhang, Birjodh Tiwana, Zhiyun Qian, Zhaoguang Wang, Robert P. Dick, Zhuoqing Morley Mao, and Lei Yang. Accurate Online Power Estimation and Automatic Battery Behavior Based Power Model Generation for Smartphones. In *CODES/ISSS '10*, pages 105–114, New York, NY, USA, 2010. ACM.

Appendix A

User Study Materials

This appendix contains the supporting materials used during the empirical evaluation (user studies) of the *GreenAdvisor* tool. Appendix A.1 has the protocol followed during the study. Appendix A.2 has the letter of initial contact that was distributed among the participants, and lists the goals and protocol followed during the study. Appendix A.3 has the consent form that was distributed among the participants, and lists the details and procedure to be followed during the study. Appendix A.4 contains the user study feedback questionnaire that was used by participants to evaluate the *GreenAdvisor* tool.

The consent forms and the supporting material was distributed and collected by Gregory Burlet (M.Sc. student at University of Alberta) as there was a possibility of conflict of interest with I being appointed as a Teaching Assistant for the course during *Winter'15* term. I had access to the forms and data only after the final grades were posted for the students.

A.1 Protocol

The researchers will present the study to the CMPUT 301 students during labs. The labs are, in fact, led by the researchers who are experts in the tools used.
They will introduce themselves as graduate students in Computing Science at the University of Alberta working with Dr. Abram Hindle and they will invite the students to participate in the study.
They will explain:

- The goal of the study is to analyze the energy consumption changes with the application code changes and also to better detect the syntactical errors in the software code using the tool UnnaturalCode . You will be using the same set of tools as all students in the class, the only difference would be that the data of your usage of the development/communication tools and analytics services will be collected for further analysis towards my research.
- Your participation is voluntary.
    1. Data collected will not be linked to the students' name (data will be anonymized).
    2. The instructor and teaching assistants of the course will not know who is participating and who is not.
    3. Participants may withdraw at any time up to the end of the course term by emailing gburlet@ualberta.ca.
- We hope that the results of this research can have a positive impact on the way we develop applications, by factoring in the applications energy consumption, as well as help in better tools for detection of syntactical errors in the code. We also hope this experience to be used for future offerings of the course.

A.2 Letter of Initial Contact

Energy consumption and code syntax error localization in Undergraduate Software Engineering Courses

Correspondence to solicit participation.

Dear CMPUT 301 student,
I am a Masters student in Computing Science at the University of Alberta working with Dr. Abram Hindle. I would like to invite you to participate in my study “Energy consumption and code syntax error localization in Undergraduate Software Engineering Courses”.

There are two goals of the study:

- Analyze the energy aspect of the code developed during the class project.
- Use the code developed during the class project for improving our tool UnnaturalCode to

find the exact location of the bugs.

This would help the us understand the processes behind the energy profile and test the usefulness of our tool for identifying the changes in energy profile. The study would also help us in improving our tool UnnaturalCode, which could be used by developers for exactly locating the bugs.

The class would function same as it would had without the study, only difference being that data of your course projects will be collected for further analysis towards my research. You would be asked to fill a voluntary feedback form on ease of use of the tool used during class project as well.

Note that:

- Your participation is voluntary.
- Data collected will not be linked to your name (data will be anonymized).
- The instructor and TAs of the course will not know if you are participating or not.
- You may withdraw your participation at any time up to the end of the course project by emailing myself at gburlet@ualberta.ca. Your data will not be used for study, if you chose to do so.
- You will not get any specific benefits from the study except probable knowledge of energy profile of my application during the course project.
- We don’t think there are any potential risks involved in the study. However, if any such risks arise, you will be informed and you may discontinue from the study.

I would appreciate if you can participate in my study. I believe the results of this research can have a positive impact on the way we develop software in a team environment.

Sincerely,
Gregory Burlet (Masters Student, Computing Science, Study Coordinator)
Department of Computing Science,
University of Alberta,
Edmonton, AB, T6G 2R3
gburlet@ualberta.ca

A.3 Consent Form

Energy Consumption and Code Syntax Error Location in Undergraduate Software Engineering Courses

Informed Consent Form
I, ____, agree to participate in the study on "*Energy Consumption and Code Syntax Error Location in Undergraduate Software Engineering Courses*" conducted by Gregory Burlet(gburlet@ualberta.ca, Co-Investigator), Dr. Eleni Stroulia(stroulia@ualberta.ca, Co-Investigator) and Dr. Abram Hindle (abram.hindle@ualberta.ca, Principal Investigator).

In this study, I will be using the regular set of tools for software development in a team project as part of the class CMPUT 301. The **goal** of the study is to analyze the application code changes and its impact on the energy consumption of the application as well as location of bugs with tool UnnaturalCode. I will be using the same set of tools as all students in the class, the only difference would be that the data of my usage of the development/communication tools and analytics services will be collected for further analysis towards the investigator's research.
The data of my project's development (history of changes to codebase and work items, viewing of codebase, usage of analytics services) will be analyzed by the investigator for the purposes of the study. I will be asked to fill a voluntary feedback questionairre about the ease of use of tool at the end of the class.

I understand that this data will be stored for later analysis and that my name will not be associated with the data (the data will be anonymized). The collected consent forms, will be under lock-and-key in the researcher's office and will be only accessible to the researchers (Gregory Burlet and Dr. Stroulia) and to Dr. Hindle after the grading is completed for the course. Other research assistants may have access to the anonymized data for analysis purposes after that grading is completed.

I also understand that the instructor and teaching assistants of the course (CMPUT 301) are not aware of my participation (or non-participation) and that only the investigator has that information. The collected data will not be analyzed until after the course has finished and final grades have been submitted.
I am also aware that there are no specific benefits for me in the study except possible awareness about the energy profile of my application. I am also aware that there are no potential risks involved in the study. If any of such risks arise, I will be informed and may discontinue my participation.

I am aware that I am volunteering my participation in this study and may discontinue my participation at any time during the course term by email to Gregory Burlet gburlet@ualberta.ca.

Consent Statement:

I have read this form and the research study has been explained to me. I have been given the opportunity to ask questions and my questions have been answered. If I have additional questions, I have been told whom to contact. I agree to participate in the research study described above and will receive a copy of this consent form. I will receive a copy of this consent form after I sign it.

____   _______________
Participant's Name (printed) and Signature   Date

____
Name (printed) and Signature of Person Obtaining Consent   Date

Feel free to contact the Co-Investigator :

Gregory Burlet,
Department of Computing Science,
University of Alberta,
Edmonton, AB, T6G 2R3
gburlet@ualberta.ca

The plan for this study has been reviewed for its adherence to ethical guidelines by a Research Ethics Board at the University of Alberta. For questions regarding participant rights and ethical conduct of research, contact the Research Ethics Office at (780) 492-2615.
Please return this form only to me (gburlet@ualberta.ca)

A.4 User Study Feedback Questionnaire

Feedback Questionnaire:

Name:

This survey aims to understand the utility of the tool in understanding the energy consumption profile.

Goal: We want to understand how useful the tool is in helping developers recognize the effect of code changes on energy consumption. More specifically we are interesting in investigating the effectiveness of the tool in identifying the system calls associated with the energy-consumption change and the corresponding code changes.

1. Identify a code change that **you think** may cause significantly different system calls and, as a result, may change the energy-consumption of the application.

   Please write your code in format provided below:

   Previous version SHA(Commit ID):
   Changed Version SHA(Commit ID):

   PS: To check a repo's current commit ID, run: *git log --format="%H" -n 1*
   or check it on Github page of your repository in the commits section.

   Filename:
   Code snippet:

2. Please comment on the code segment and system calls identified **by the tool** to have changed significantly, energy consumption wise based on versions in previous Q1.

   Please write your code in format provided below:

   Previous version SHA(Commit ID):
   Changed Version SHA(Commit ID):

   PS: To check a repo's current commit ID, run: *git log --format="%H" -n 1*
   or check it on Github page of your repository in the commits section.

   Code displayed by the tool--
   System call:
   %Change in system call invocation(from the table in report generated by tool):
   Filename:
   Code snippet:

3. “The tool is able to identify the code associated with the change in system calls.” How much do you agree with the above statement?
__ Strongly Agree
__ Agree
__ Not Sure
__ Disagree
__ Strongly Disagree

4. In your opinion, how well does the tool predict whether a source-code change affects energy consumption?
Please tick one of the following:

__ Most of the time	[~80-100%]
__ Regularly	[~51-80%]
__ Sometimes	[~20-49%]
__ Occasionally	[~0-20%]

Please explain your answer; we would like to hear more about your experience with tool in order to improve its usefulness.