GreenAdvisor: A Tool for Analyzing the Impact of Software Evolution on Energy Consumption

Karan Aggarwal, Abram Hindle, and Eleni Stroulia
Department of Computing Science
University of Alberta
Edmonton, Canada
{kaggarwa, abram.hindle, stroulia}@ualberta.ca

***Abstract*—Change-impact analysis, namely "identifying the potential consequences of a change" is an important and well studied problem in software evolution. Any change may potentially affect an application's behaviour, performance, and energy consumption profile. Our previous work demonstrated that changes to the system-call profile of an application correlated with changes to the application's energy-consumption profile. This paper evaluates and describes *GreenAdvisor*, a first of its kind tool that systematically records and analyzes an application's system calls to predict whether the energy-consumption profile of an application has changed. The *GreenAdvisor* tool was distributed to numerous software teams, whose members were surveyed about their experience using *GreenAdvisor* while developing Android applications to examine the energy-consumption impact of selected commits from the teams' projects. *GreenAdvisor* was evaluated against commits of these teams' projects. The two studies confirm the usefulness of our tool in assisting developers analyze and understand the energy-consumption profile changes of a new version. Based on our study findings, we constructed an improved prediction model to forecast the direction of the change, when a change in the energy-consumption profile is anticipated. This work can potentially be extremely useful to developers who currently have no similar tools.**

***Index Terms*—Software energy consumption, energy efficiency, software tools, application software**

I. Introduction

Software undergoes continuous change through evolutionary software-development processes, and maintenance activities such as bug fixes and patches. 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 is little tool support to help developers with this challenging task.

This problem manifests itself as a great challenge in the context of mobile applications. According to Pew [1], as of May 2013, 63% of adult smartphone owners (in North America) use their phones to go online and 34% of smartphone Internet users go online primarily using their phones (and not a more "traditional" device such as a desktop or laptop computer). These trends have 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. Despite its importance however, developers are generally unable [2], [3] to estimate the energy consumption of their applications, and predict how it may be impacted by code changes.

Some research effort has been devoted to the examination of factors affecting the energy consumption of mobile applications and constructing models to estimate smartphone energy consumption [4], [5], [6]. Most previous work has focused on helping end users track the energy consumption of their smartphone [7], [8], [9], but relatively little attention has been paid to the effect of code changes on an application's energy-consumption profile. Most notably, Hindle [10] proposed *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 software should be measured, and investigates the correlation of energy consumption across software versions 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.

Applications invoke *system-calls (syscalls)* in order to use resources and services mediated by the operating system (OS). Examples of such resources and services include hardware devices, network communication, intra-application communication, and peripherals. To access the peripherals that typically consume quite substantial energy, the application would use system-calls to request access to a peripheral from the OS and invoke its operations. Thus the system-call log, a record of system-calls invoked by the application, serves as a signature of an application's resource usage. In our previous work [11], the green-mining methodology 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, it can potentially be very useful since developers have been found to be generally unaware of the factors affecting energy consumption of their applications [3].

This paper introduces *GreenAdvisor*, a tool developed to operationalize this *Rule of Thumb* [11], in the context of evolutionary software development by comparing the system-call logs of subsequent commits and predicting how the new version's energy-consumption profile will compare to that of the previous version. We conducted a user study to evaluate *GreenAdvisor* against the projects of several software teams in order to measure its perceived usefulness, while at the same time probing the developers' understanding of the factors that affect the energy consumption of their applications. The user study examines whether the users feel that the tool is useful on two fronts: (a) the tool's ability to predict change in energy-consumption profile with change in code, and (b) its ability to locate the code segment responsible for the change. Furthermore, we refined the original *Rule of Thumb* model [11] 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.

The rest of this paper is organized as follows: this work in the context of related work is described in Section II; the tool is described in Section III. Our evaluation methodology is described in Section IV; our findings are reported in Section V and discussed in Section VI; and our conclusions are summarized in Section VIII.

II. Related Work

This section reviews the relevant literature regarding power-model construction, mining software repositories, and energy discussions online.

A. Power Modeling

A number of models have been proposed for modelling the energy consumption of devices, based on their hardware components or their instruction sets. The general model-construction methodology involves running sample applications on the devices under examination and using the collected records to infer models of the devices' energy consumption. These studies have focused on profiling individual components of the software like CPU, disk, network, peripherals, screen. This has led many researchers to building component profilers to predict the energy consumption of their software [8], [4], [12], [5], [9], [13], [14].

A hybrid was proposed by Pathak *et al.* [6], by applying system-call tracing to model the energy consumption of smart-phone applications. They concluded that previous models, built by correlating component utilization with energy consumption, are not accurate. They proposed the construction of finite state machine (FSM), based on modeling energy states and system calls for each component, which they integrated to produce a single smartphone FSM that can estimate the energy consumption of an application.

Hao *et al.* [15] built an energy-consumption model, eCalc, for estimating CPU energy consumption at the program and method levels. eLens [16] is an extension of eCalc that uses other hardware components, besides the CPU, to profile energy at multiple levels. Li *et al.* [17] extended it by profiling bytecode instructions, and estimating energy consumption of each line of source code with high accuracy. However, their studies relied on quite device-specific models and focused on end users rather than assisting development environments.

B. Mining Software Repositories

The MSR methodology – *i.e.,* the use of statistical and data mining techniques to discover interesting information from software repositories [18] that can be leveraged in future decisions – has also been applied to understanding software energy-consumption profiles. Gupta *et al.* [19] studied the energy consumption of a Windows phone, using a combination of power traces and execution logs to build power models. They also successfully utilised this data to detect energy patterns that could predict buggy modules.

Hindle [10] described the *Green Mining* methodology for collecting energy-consumption data over multiple application versions. Based on energy-consumption measurements and software metrics, they examined correlations between software changes and energy consumption over multiple application versions. Hindle *et al.* [20] also created the *Green Miner* test bed, a dedicated hardware infrastructure to measure the energy consumption of Android applications. 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 the phone requirements. This test bed was used for measuring the energy consumption of the projects examined in this study.

Our previous work [11] proposed a simple model, based on system-call counts, to predict changes in the energy-consumption profile of the applications. A simple rule was proposed by relating the energy-consumption profile to the system-call profile. This *Rule of Thumb* states that significant changes in system-call profiles lead to significant changes in energy consumption. This helps developers develop an intuition about code changes inducing energy-consumption changes, which otherwise might be difficult to detect in the absence of energy-consumption measuring instrumentation.

The case studies presented in this paper are based on the Green Mining methodology, correlating code changes with system-call profile changes and changes in energy-consumption.

C. Online Discussions about Energy

Online discussion forums are potentially a rich source of information relevant to understanding software energy consumption. Pinto *et al.* [3] analysed a dataset of more than 300 questions, and 800 users on the Stackoverflow for the energy-consumption related discussions among developers. They found that developers lack the necessary tools and knowledge for energy-aware development.

Pathak *et al.* [21] analysed four online mobile user forums for discussions on bugs that led the applications to consume high amounts of energy. Wilke *et al.* [22] analysed energy-related user comments on Google play applications. They found that energy inefficiency negatively impacts the user rating of the applications. They also discovered that free and paid-for applications had similar energy-efficiency related complaints, evidently demonstrating a general obliviousness to energy-efficiency driven development among application

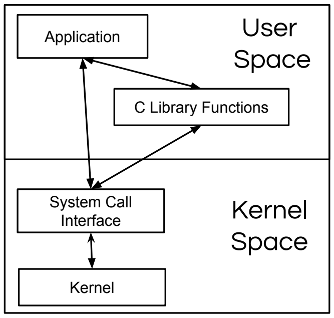


Fig. 1. This diagram shows how applications, C library functions, system-calls, and the kernel interact with each other.

developers. Hence, developers are realizing the importance of their applications' energy efficiency, but are unable to follow energy-efficiency driven development for the lack of tools.

III. *GreenAdvisor*

The *GreenAdvisor* tool is based on our previous findings [11] that measured changes in the system-call profile, and used these changes in system-call counts to predict the changes in the application's energy consumption.

A. *Background*

System-calls provide an API for user-space applications to access the services, abstractions and devices managed by the OS kernel and other OS components. System-calls are functions provided by the OS kernel that user processes invoke to manipulate or access resources [11].

Typically system-calls are provided for communicating with the hardware (for example, accessing the hard disk), creating and executing new processes, managing memory use, sending (receiving) data to (from) other processes, and receiving event notifications [23]. Figure 1 presents the relationships between user applications, C library functions, system-calls, and the OS kernel [23]. System-calls are typically invoked through a library, such as the C standard library, graphics libraries, network libraries, inter-process communication (IPC) libraries, and occasionally by the application itself. System-call logs can be generated with tools like `strace` [1].

Different versions of an application are expected to invoke different system calls at run time, if the application behaviour has changed from the previous version. In our previous work [11], system-call profiles were related with energy-consumption profiles. This method can help developers predict changes in the energy profile of their applications, without any special instrumentation. While examining multiple versions of two open-source Android applications – `firefox` and `calculator` in the study, it was found that system-call profile exhibits instability similar to the energy-consumption measurements. The variance per system-call averaged across the versions is shown in Figure 2. Some system-calls are less stable and have higher variance than others. Thus system-call counts should be measured multiple times to get a stable average. In energy-consumption research, it is very important to address non-determinism of the environment by executing multiple tests [10].

Linear and logistic regression models were built using these measurements to estimate energy consumption and to classify versions as low/high energy consuming versions. The main contribution of that work was a *Rule of Thumb* that states – *"If the system-call profile changes significantly from the previous version, it is probable that the application's energy consumption has changed as well"* [11]. Using this rule, if any system-call changes statistically significantly, then the application's energy consumption is also likely to have changed statistically significantly. A pairwise Student's *t*-test established the statistical significance of difference between the energy consumption or system-call counts of two commit-versions. Energy-consumption measurements of the same system tend to be normal and parametric, thus the *t*-test is appropriate.

In order to establish the usefulness of the *Rule of Thumb*, we examined four metrics: precision, recall, specificity, and F-measure (F_1):

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

SS is the number of times that the changes in energy consumption and system-call counts are significant. NS is the number of times the change in energy consumption is significant, while the system-call profile remains the same. SN is the number of times the system-call profile changes significantly, but the energy consumption does not. Finally, NN is the number of times that neither change is significant.

All four metrics range from 0 to 1, with 0 being the worst and 1 being the best possible value. High precision indicates that the significant change in system-call counts leads to a significant change in power consumption. High recall indicates that the cases where significant energy consumption changes were observed, co-occur with the cases with significant change in system-call counts. High specificity indicates that our *Rule of Thumb* generates few false positives: the cases where energy consumption change was not significant were not marked as significant. F_1, the weighted mean of precision and recall, is a measure of accuracy; the higher the F_1 the more balanced and accurate the model is.

The precision, recall and F_1 values were much better than random guess but not very high, while specificity was quite high. As most of the commits do not produce significant energy-consumption changes, high specificity indicates the *Rule of Thumb* model works in the majority of the cases [11]. It is much easier for developers to apply, as system-call counts can be easily traced using freely available tools like `strace`, rather than using expensive hardware-based energy-consumption instrumentation [20]. We use this *Rule of Thumb* model from

[1] Strace is part of systat http://sebastien.godard.pagesperso-orange.fr/

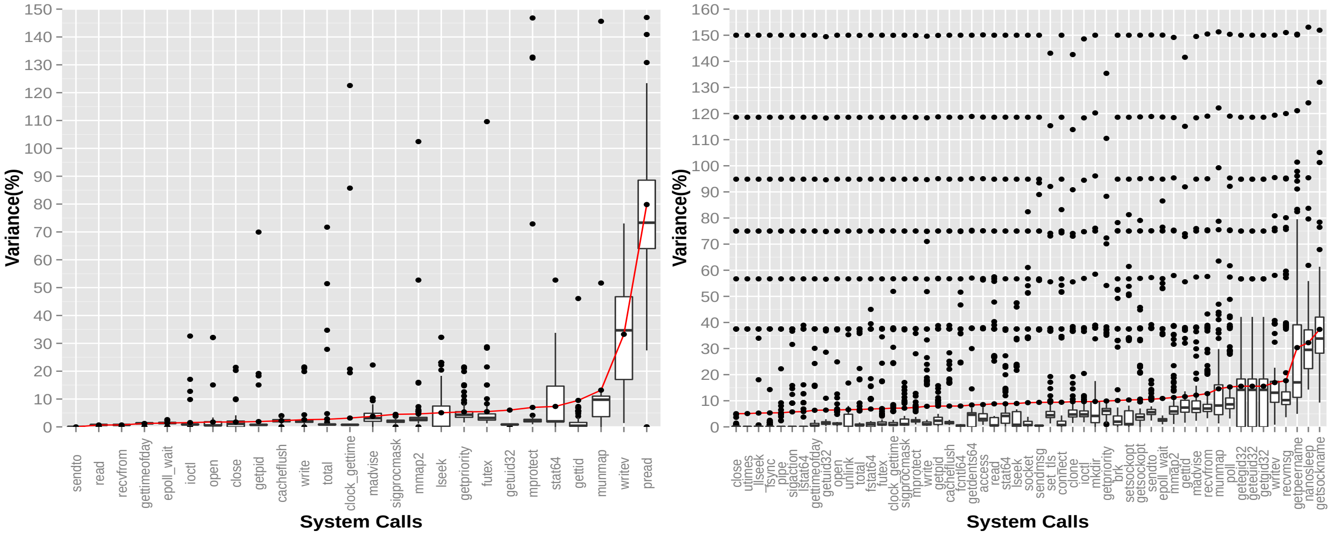


Fig. 2. **Variance as a percentage of mean value across 10 runs per system-call per version for the `Calculator` (left) and `Firefox` (right) applications [11]. The X axis refers to the system calls, while the Y axis refers to variance in percentage.**

our previous work [11] to build a tool called *GreenAdvisor* and evaluate *Rule of Thumb* in the light of new data collected from our survey described in latter section.

B. The GreenAdvisor Tool

The *GreenAdvisor* tool has been built to enable programmers to apply the *Rule of Thumb* model [11] 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* [11] to alert the application developer about a possible change in the energy-consumption profile of their application. By profiling the system-call traces between application versions, it alerts the developer about possible changes in energy-consumption profile and 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. It 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 using a Student's *t*-test in order to apply the *Rule of Thumb* .

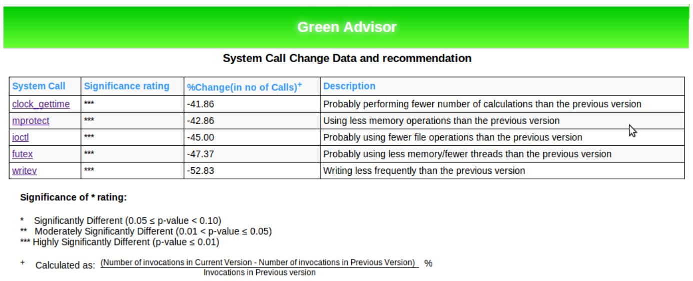
The *GreenAdvisor* generates a report as shown in Fig. 3, listing the system-calls that have changed significantly, and the source code in the `diff` between versions that potentially causes the functional changes in the application. For example, if calls to the `open` system-call increase significantly, the tool indicates to the developer that they might be performing a higher number of open file operations. The report also highlights the code changes that might be responsible for the change in the system-call profile using a bag-of-words model.

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 IO 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 by the authors, 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. 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` between the application code prior to and its code after the commit. If any of those identifiers are found in the `diff`, they are indicated in the report associated with the particular system-call. By indicating the changed code that might be behind the expected energy-consumption change, the tool can 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.

In summary, the *GreenAdvisor* workflow involves the following step sequence.

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



Green Advisor

System Call Change Data and recommendation

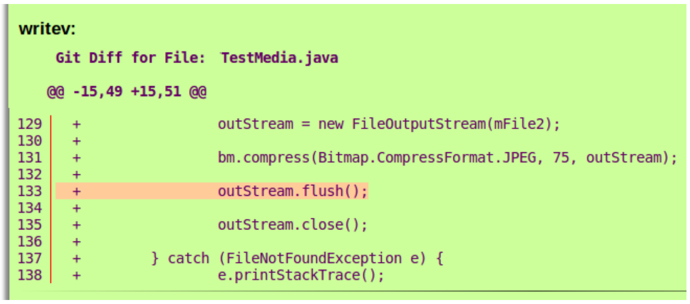
System Call	Significance rating	%Change(in no of Calls)+	Description
clock_gettime	***	-41.86	Probably performing fewer number of calculations than the previous version
mprotect	***	-42.86	Using less memory operations than the previous version
ioctl	***	-45.00	Probably using fewer file operations than 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 ($0.05 \le$ p-value < 0.10)
** Moderately Significantly Different ($0.01 <$ p-value ≤ 0.05)
*** Highly Significantly Different (p-value ≤ 0.01)

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

Fig. 3. A sample report generated by the *GreenAdvisor* tool indicating a change in system-call counts.



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();
```

Fig. 4. A code segment of `diff` highlighted by the *GreenAdvisor* tool indicating the likely source of the `writev` system-call.

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 code segments in the commit `diff` might be responsible for the change.
5) Finally, it generates a report of the changed system-calls and the code changes that might be responsible for these changes.

IV. The Case-Study Methodology

Using the *GreenAdvisor* tool, a user study was conducted with 13 student teams of 77 third-year University of Alberta students, enrolled in a software-engineering course. As part of their course deliverables 11 student teams developed geo-location-aware question-and-answer portal application while 2 teams developed a travel expense claims application. Using these projects, experiments were run to evaluate the prediction models. This section describes our data-collection method, our survey, and the experiments we ran on the 13 projects.

A. User Study on the GreenAdvisor Tool

In order to evaluate the *GreenAdvisor* tool, a voluntary survey was distributed to the course students. The user-study evaluates the tool on on two requirements: its ability to predict changes in the application energy-consumption profile with changes in code, and its ability to identify the code responsible for that change. 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 code inducing the energy-consumption change. Students could run the *GreenAdvisor* tool in the lab or at home on 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 it. 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 (seven teams) completed 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.

B. Expert Use of the Tool

In parallel with our survey of the student developer teams, distinct commits from each of the 13 projects (from teams that had consented) were selected manually to evaluate the *Rule of Thumb* proposed in [11].

a) Student projects: All of the 11 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 (milestones) 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 a corporate setting. The set of requirements included creating claims, submitting claims, approving claims, providing offline functionality, adding geolocation to the travel destination and so. They too had three course-deliverables.

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

b) Selection of commits: As each project has 100s of commits, it is infeasible to run experiments on all the commits. Even more importantly, 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 [24] suggests that estimating energy profiles based on sub-selecting commits does not heavily harm the accuracy 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.* [24]. Finally the commit messages were examined and, if any appeared important, the corresponding commits were chosen as well. Using this procedure, around ten commits per project were gathered. These commits were generally large ones, except the ones submitted as course deliverables. If a commit version that was submitted as a deliverable did not have 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 were selected after the manual inspection of commit messages.

c) Evaluating Rule of Thumb : The *Green Miner* test bed [20] was used for obtaining the energy-consumption measurements of the various versions (commits) of the applications. It includes four Galaxy Nexus phones, controlled by a Raspberry Pi that starts the tests, collects, and uploads the measurements to a centralized server. Each phone is also attached to an Arduino for measuring its energy consumption. The test bed was configured to run *jUnit* tests. The selected application versions were run to obtain their energy-consumption readings and their system-call profile counts. Multiple energy-consumption measurements recorded from *Green Miner* were used to establish if pairs of subsequent 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 application were built; each version was tested and its energy consumption was 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 to evaluate the *Rule of Thumb* model on each of these projects.

C. Improved Prediction Model

Rule of Thumb only predicts whether or not the application energy consumption has changed significantly. However, developers are more interested in the direction of energy-consumption change. They are more interested in knowing if their application's energy consumption has significantly increased or decreased. 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 confident in that their newly committed version is "safe" with respect to energy-consumption changes. 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 of the `calculator` and `firefox` applications from our previous work [11], 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:

- 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.

TABLE I
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

V. FINDINGS

A. The 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 77 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 the 7 teams, 4 agreed, while 3 disagreed, that tool was able to identify code associated with change in system-calls. Out of those three, one team was not able to see the associated code with the system-calls because those system-calls did not have 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, 5 teams reported that the tool was able to indicate energy-consumption change "regularly", while the other two reported that it worked only "sometimes".

Next, 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 of 7 identified commits (across seven projects) exhibited a change in the energy-consumption profile of the application. The teams that had reported that the tool was unable to predict energy-consumption changes, had identified commits that did not produce energy-consumption changes, and were hence unable to see any responsible code either.

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 how they could optimise the energy consumption of their applications, including minimal use of GPS, using dark colors, optimising local-storage update with the remote server — matching some of the suggestions of Pinto *et al.* [3]. 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.

In summary,

- 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; and
- 5 (30 students) out of 7 student groups (42 students) felt that the *GreenAdvisor* tool was able to indicate energy consumption changes regularly.

While some of the results are mixed, the tool was usable 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 the *GreenAdvisor* tool can indeed be useful to for actual developers.

B. Expert Use of the Tool

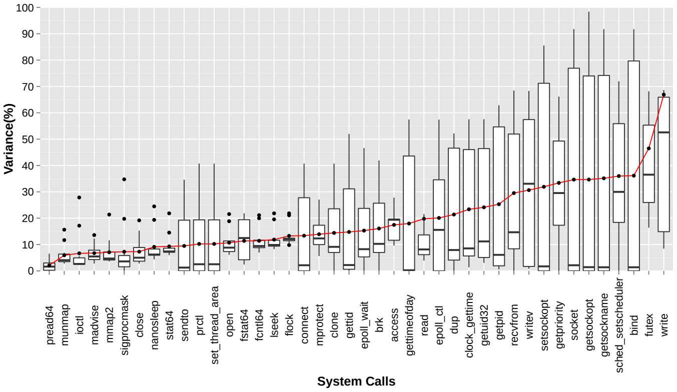


Fig. 5. **Variance as a percentage of mean value across 30 runs per system-call per version for all the applications considered. The X axis refers to the system-calls, while the Y axis refers to variance in percentage.**

TABLE II
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

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 has significantly changed. This prediction is verified by obtaining the actual energy-consumption measurements from the 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 is shown in Figure 5 by using 30 runs per commit-version of each application. As can be observed, the variation values are quite similar to the ones obtained before as shown in Figure 2.

Four 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 II. As can be observed, recall and specificity values are quite high, though the precision and F_1 scores are not that high, consistent with our previously obtained results [11].

These projects varied in their implementation, though the requirement specifications were exactly the same for 11 projects in the *Fall'14* term and 2 projects in the *Winter'15* term. Students were expected to create a questions-answer/travel-expense application according to these specifications. They had to use an Elasticsearch server provided by the instructors to store application's 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 and updating the user location 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 their application's energy consumption increased significantly. 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 implemented. 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 used dark screen colors, and exhibited lower energy consumption than the other projects, though it still had some commits with releative high energy consumption. 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.

C. *Improved prediction model*

We constructed logistic-regression models to predict the direction of change in energy consumption, *i.e.,* , increase or decrease, when the *Rule of Thumb* predicts significant energy-consumption change. We only used those system calls that 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 IV.

TABLE III
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

TABLE IV
SELECTED SYSTEM-CALLS WITH THEIR DESCRIPTIONS FROM THE IMPROVED PREDICTION MODEL

system-call	Description [25]
`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

The results are shown in Table III. The cases in the Table III 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, and tested on the projects dataset. In case `2`, the model was trained on the `firefox`, `calculator` and 50% randomly student projects, and tested on the remaining projects. In case `3`, the model was trained and tested on the combined dataset of all the student projects using 10-folds cross validation. We observe that when more student projects are used for training, the precision, specificity increases, resulting in 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. Since we use a test dataset to evaluate each logistic-regression model, metrics like likelihood ratio-test used to evaluate the models for the training data are not required. Hence, in order to determine the prediction accuracy of the model on our test dataset, the precision, recall, specificity, and F_1 metrics have been used. Using the *Rule of Thumb*, the commits can be separated into versions exhibiting significant energy-consumption changes or not. As the *Rule of Thumb* is highly accurate for predicting insignificant-change cases, the logistic regression model is applied on the cases where the *Rule of Thumb* predicts significant energy-consumption change. 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, we can be predict whether the energy consumption of the applications changed or not, and if it did, the direction of change.

VI. DISCUSSION

This paper introduces *GreenAdvisor*, a first of its kind of tool that can be used by developers to track the energy-consumption profile of their applications. The tool can predict 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 focuses on providing a perspective to users about the components or applications utilising 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, all without hardware instrumentation.

For the user study, students were asked to identify the code commits that they expected to have induced change in their application's energy consumption profile. The user study evaluated the tool regarding the students' perceived usefulness and ability to predict changes in the energy-consumption profile of their applications, and its ability to identify the code responsible for that change. The findings of the user study were overall positive, with most students agreeing to the tool's ability to predict change in energy-consumption profile. The tool correctly identified the no-change cases, even though the students thought otherwise. However, the tool was not able to identify the lines of source code that induced energy-consumption changes, according to the responses of three teams (out of seven teams) 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 energy-aware development, it is clear that students have only a rough idea of how their design and implementation decisions impact their application's energy profile. A similar observation was made by Pinto *et al.* [3] on *StackOverflow*, though their dataset included experienced developers rather than the undergraduate students considered in this study. This calls for an energy-aware development curriculum in the software engineering courses [2].

Using the *Rule of Thumb* model to predict the occurrence of significant changes in the application energy consumption, high specificity was observed across all 13 systems, as was observed across two larger Android applications in prior work [11]. 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 a 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, which additionally gives them a hint of the code that might be responsible for energy consumption change.

An improved model was developed using logistic regression to predict the direction of energy consumption change (increased or decreased) and evaluated. Though not exceptionally accurate, this model enhances the *Rule of Thumb* model [11] by predicting direction of the change, when there is a significant change. 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.

VII. Threats to validity

Internal validity is constrained by the choice of project requirements given that all projects in a term had the same requirements. The choice of commits selected for investigation of the *Rule of Thumb* model presents a validity construct. External validity is constrained by the test construction. It is possible that the tests that students wrote had limited coverage of the application's 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 [26]; some students are exceptional programmers, already employed in industry but many are not.

VIII. Conclusions

A first of its kind, developer-centric system-call profile based tool *GreenAdvisor* has been introduced, that predicts changes in the energy-consumption profile of applications and corresponding changes in their code base. The *GreenAdvisor* tool uses the application's system-call profile to warn developers about possible changes in the application's energy consumption profile by using *Rule of Thumb* introduced in our previous study [11]. *GreenAdvisor* also indicates the possible code that might be behind that change from pre-defined code for a set of system-calls using a *bag-of-words* model.

GreenAdvisor was evaluated using 77 students (13 teams) in an undergraduate class. A short survey asked the students to use the tool for commits they expected to have introduced energy-consumption changes. The survey results were positive, though the students felt that the tool's ability to highlight the change inducing lines of code could be improved. Based on the survey results we concur with Pang *et al.* [2] that energy aware software development should be included in the software engineering courses.

GreenAdvisor and its *Rule of Thumb* model were evaluated through a series of experiments indicating that they can accurately find commits with changed energy-consumption profile, and they can determine the direction of the change. These experiments further confirm the generality and usefulness of the *Rule of Thumb* model on much more number of applications than our previous work. The *Rule of Thumb* exhibits high specificity and recall, though lower precision. This demonstrates the usefulness of the *Rule of Thumb*, since most of the code commits do not exhibit any significant changes in the energy profile of applications. In fact, in our survey, even though students identified commits that they thought exhibited different energy consumption, our model correctly recognized that this was not the case. Hence, in the absence of any instrumentation, *Rule of Thumb* allows the developers to better track the energy-consumption profiles of their applications. The 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. It is available freely for download with instructions for use here: https://github.com/kaggarwal/GreenAdvisor

GreenAdvisor helps developers to determine energy-consumption changing commits by aggregating a profile of system-call counts over many versions and applying an improved *Rule of Thumb* model. In order to improve the prediction, two open problems demand additional research effort: first, building a general model of system-calls that can work on all types of applications; and second, automatically associating API and common patterns to system calls for detection of energy change inducing code.

IX. Acknowledgements

The authors would like to thank NSERC and IBM Canada for the generous support.

References

[1] P. Research, "Smartphone Ownership 2013," http://www.pewinternet.org/2013/06/05/smartphone-ownership-2013/, 2013.

[2] C. Pang, A. Hindle, B. Adams, and A. E. Hassan, "What do programmers know about the energy consumption of software?" *PeerJ PrePrints*, vol. 3, 2015.

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

[4] S. Gurumurthi, A. Sivasubramaniam, M. J. Irwin, N. Vijaykrishnan, M. Kandemir, T. Li, and L. K. John, "Using Complete Machine Simulation for Software Power Estimation: The SoftWatt Approach," in *Proceedings of the 8th International Symposium on High-Performance Computer Architecture*, ser. HPCA '02, 2002.

[5] A. Shye, B. Scholbrock, and G. 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*, ser. MICRO 42. New York, NY, USA: ACM, 2009, pp. 168–178.

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

[7] A. Pathak, Y. C. Hu, and M. Zhang, "Where is the Energy Spent inside My App?: Fine Grained Energy Accounting on Smartphones with Eprof," in *Proceedings of the 7th ACM european conference on Computer Systems*, ser. EuroSys '12. New York, NY, USA: ACM, 2012, pp. 29–42.

[8] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, and L. Yang, "Accurate Online Power Estimation and Automatic Battery Behavior Based Power Model Generation for Smartphones," in *CODES/ISSS '10*. New York, NY, USA: ACM, 2010, pp. 105–114.

[9] R. Mittal, A. Kansal, and R. Chandra, "Empowering Developers to Estimate App Energy Consumption," in *Proceedings of the 18th annual international conference on Mobile computing and networking*, ser. Mobicom '12. New York, NY, USA: ACM, 2012, pp. 317–328.

[10] A. Hindle, "Green Mining: A Methodology of Relating Software Change to Power Consumption," in *MSR*, 2012, pp. 78–87.

[11] K. Aggarwal, C. Zhang, J. C. Campbell, A. Hindle, and E. Stroulia, "The power of system call traces: Predicting the software energy consumption impact of changes," in *Press of the 2014 Conference of the Center for Advanced Studies on Collaborative Research, IBM Corp*, 2014.

[12] A. Carroll and G. Heiser, "An Analysis of Power Consumption in a Smartphone," in *Proceedings of the 2010 USENIX conference on USENIX annual technical conference*, ser. USENIXATC'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 21–21.

[13] D. Li, S. Hao, J. Gui, and W. G. Halfond, "An empirical study of the energy consumption of android applications," in *Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on*. IEEE, 2014, pp. 121–130.

[14] M. Linares-Vásquez, G. Bavota, C. Bernal-Cárdenas, R. Oliveto, M. Di Penta, and D. Poshyvanyk, "Mining energy-greedy api usage patterns in android apps: an empirical study," in *Proceedings of the 11th Working Conference on Mining Software Repositories*. ACM, 2014, pp. 2–11.

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

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

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

[19] A. Gupta, T. Zimmermann, C. Bird, N. Naggapan, T. Bhat, and S. Emran, "Detecting Energy Patterns in Software Development ," Microsoft Research, Tech. Rep. MSR-TR-2011-106, 2011.

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

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

[22] C. Wilke, S. Richly, S. Gotz, C. Piechnick, and U. Aßmann, "Energy consumption and efficiency in mobile applications: A user feedback study," in *Green Computing and Communications (GreenCom), 2013 IEEE and Internet of Things (iThings/CPSCom), IEEE International Conference on and IEEE Cyber, Physical and Social Computing*. IEEE, 2013, pp. 134–141.

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

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

[25] The Linux man-pages project, "Linux Man Pages Online," http://man7.org/linux/man-pages/, 2013.

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