2017 IEEE 17th International Working Conference on Source Code Analysis and Manipulation

A Methodology for Relating Software Structure with Energy Consumption

Abdul A. Bangash*, Hareem Sahar† and Mirza O. Beg‡
Department of Computer Science
National University of Computer and Emerging Sciences, Islamabad, Pakistan
Email: *abdul.ali@nu.edu.pk , †hareem.sahar@nu.edu.pk , ‡omer.beg@nu.edu.pk

***Abstract*—With the widespread use of mobile devices relying on limited battery power, the burden of optimizing applications for energy has shifted towards the application developers. In their quest to develop energy efficient applications, developers face the hurdle of measuring the effect of software change on energy consumption. A naive solution to this problem would be to have an exhaustive suite of test cases that are executed upon every change to measure their effect on energy consumption. This method is inefficient and also suffers from environment dependent inconsistencies. A more generalized method would be to relate software structural metrics with its energy consumption behavior. Previous attempts to relate change in object-oriented metrics to their effects on energy consumption have been inconclusive. We observe that structural information is global and executed tests are rarely comprehensive in their coverage, this approach is prone to errors. In this paper, we present a methodology to relate software energy consumption with software structural metrics considering the test case execution traces. Furthermore, we demonstrate that software structural metrics can be reliably related to energy consumption behavior of programs using several versions of three open-source iteratively developed android applications. We discover that by using our approach we are able to identify strong correlations between several software metrics and energy consumption behavior.**

***Keywords*-energy; energy consumption; mining software repositories; software metrics; sustainable-software; static-analysis**

I. Introduction

Mobile technology has become pervasive and in recent years with ever-increasing complexity of smartphone applications that place heavy demands on the otherwise limited battery power. The demand as well as development of computationally intensive smartphone applications has increased manifold but at the same time users have become more conscious about battery consumption. Applications that consume more power necessitate frequent cellphone recharges that severely affect the usability of mobile devices [1]. According to a survey, the primary reason of poor reviews of an application is heavy battery usage [2]. These factors are forcing the developers to develop applications that are optimized for power consumption. However, developers lack knowledge of factors that affect energy consumption of software [3]. Tools such as [4] that enable on-line energy aware development can assist developers during application development by providing valuable energy insights.

With the high level goal of on-line energy aware development in mind, we study the impact of software structural changes on energy consumption. Specifically we are interested in identifying those aspects of software structure that can be correlated to its energy consumption behavior reliably. Once identified, this information can help in building better software energy estimation models to be used in plugins that can be directly integrated into the development environments.

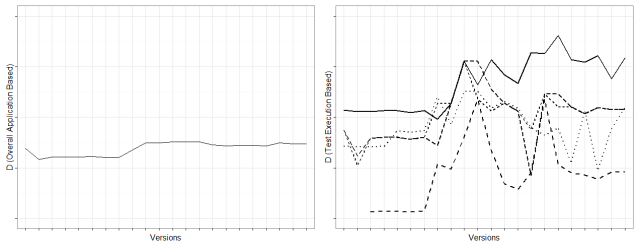

Fig. 1. (a) D(Distance from Main Sequence) values from Martin metrics suite for entire application and (b) for five separate test case executions of 24 versions of QKSMS case study

Previously similar studies examining the effect of structural changes on power consumption were performed by Hindle [5], [6] using extended CK metrics [7], line of code(LOC) and churn measures [8]. The results show limited evidence of correlation between the software structure and power consumption. This could be due to two reasons; firstly, there was not sufficient variation in structural metrics of the case studies used or perhaps, more erringly, the study attempted to correlate power consumption of software with whole-program metrics. We believe that energy consumption depends on test execution path and the structural information of the whole program cannot be correlated with energy consumption of a specific test execution trace. For instance, consider Fig. 1, which shows metric *D*(*Distance from Main Sequence*) from the Martin metric suite for the entire application and the corresponding values of five different test cases of an application. It is clear how a specific metric value varies with the change in execution path trace of each test case.

Our work addresses this limitation by relating software energy consumption with the metrics relevant to test execution path. To ensure that sufficient variability exists in structure of the software we related structural information to energy consumption across several versions of three different applications. We evaluate our technique using two additional metric suites, Fernando Brito-e-Abreu's MOOD metric suite [9] and Martin's Package metric suite [10] in addition to the CK

2470-6892/17 $31.00 © 2017 IEEE
DOI 10.1109/SCAM.2017.18

metrics [7]. To the best of our knowledge this is the first study examining the relationship between execution path structure and energy consumption of software.

The main contributions of this paper are:

- A methodology that considers execution path structure in order to correlate structural metrics with energy consumption.
- An empirical evaluation of the impact of structural changes on software energy consumption on multiple applications and their various use-cases.
- A successful demonstration of correlation between energy consumption and three object oriented metric suites using several versions of three open-source android applications.

In order to examine how structural information relates to energy consumption behavior of android applications, we extract several releases of three open-source iteratively developed android applications from the Github repository; QKSMS is a messaging app, BeHe ExploreR is an android web browser and PDFCreator is an application to create and edit PDF files. We extract structural information of the case studies using MetricsReloaded[1], an android tool that extracts the structural properties of android applications from source code. In order to measure energy consumption of application use-cases several tests were executed multiple times. In contrast to [5], we gather energy measurements on an android smartphone rather than using an emulator or any additional hardware equipment since accurate power profiling tools are now available. More than 1300 test runs were performed to collect energy consumption information. After collecting all the relevant data we applied *Spearman ρ-rank* correlation [11] and identify that all CK and Martin metrics (except LCOM) and two MOOD metrics are correlated to the energy consumption of software with varying degrees of strength. Our approach can be used to create development tools that can quickly evaluate energy changes even before programs are built.

The rest of the paper is structured as follows: In Section II, we provide a brief review of relevant literature. In Section III, we present a methodology to relate software structure to energy consumption on the basis of test execution path. Section IV contains details of our experimental setup. An analysis of the experiments we performed on several versions of three case studies is described in Section V. Section VI discusses the threats to the validity of our study and Section VII concludes the paper.

II. Related Work

A. Energy Measurement

Energy consumption is the measure of effort in order to perform a task whereas power is the measure of instantaneous rate of energy consumption. For short running programs, where task completion time is limited, energy is used as a measure of performance and for long running systems, power is the appropriate measure [12]. Software energy consumption is a non-functional performance quality measure and has grown in importance because of the popularity of limited battery devices such as smartphones. A number of studies have been carried out to measure the impact of design patterns [13], [14], [15], method inlining [16] and algorithms [17] on energy consumption. Pathak et al. [18] proposed a system-call based power modeling approach that uses a finite-state-machine(FSM) to model power states and transitions. The energy of various system calls was calculated with a low error rate after instrumenting an application and mapping it's system calls on FSM transitions . Exploiting the same technique, authors develop a tool [19] that provides fine grained information related to energy consumption of Android applications. A general decision making framework was also proposed [20] to help developers select energy efficient libraries in their applications which use Java Collection API. Trepn Profiler[2] is a power profiler developed by Qualcomm for android devices but unfortunately, it only works on Snapdragon MDP developer devices. In this study we employ a tool called PowerTutor [21] for energy measurements since Power Tutor is open-source and is easily configurable for many android devices. PowerTutor calculates the application level energy consumption of major android phone components including CPU, LCD and WiFi with a high level of accuracy.

B. Software Metrics

Researchers have long sought to analyze the impact of change in software structure on quality of code development [6]. Metrics such as Chidamber and Kemerer(CK) metric suite [7], Martins package metric suite [10] and MOOD metric suite [9] have been used to characterize software structure. These metrics are described in Table I. Evaluations of these metric suites were conducted by Basili et al. [22] and Briand et al. [23], [24] that indicate the effectiveness of these metrics. Furthering their work Gyimothy et al. [25] proved that these metrics are good code quality predictors. He empirically validated the object-oriented(OO) CK metrics in terms of their ability to predict faults using linear and logistic regression. In another work, Olague et al. [26] empirically evaluated two other object-oriented metric suites MOOD and QMOOD metrics in addition to CK. Their results show that these metric suites do not have a similar ability to predict fault-proneness of software developed by agile process. Fioravanti [27] and Yu et al. [28] conducted evaluations for the same purpose and presented similar results. Our work use CK, MOOD and Martin in an entirely different context as we attempt to correlate these metrics to energy consumption of software.

C. Green Mining

Green mining is a process that aims to analyze power consumption behavior from the information extracted from various software repositories. To address the problems faced by researchers in discovering power usage statistics of software, Hindle et al. [29] introduced a regression test framework

[1]https://plugins.jetbrains.com/plugin/93-metricsreloaded

[2]https://developer.qualcomm.com/software/trepn-power-profiler

TABLE I
A BRIEF DESCRIPTION OF CK, MARTIN AND MOOD METRICS

Chidamber and Kemerer (CK) Metric Suite - Class Level Metrics [7]	
Metric Name Value	**Definition**
Weighted Methods per Class (WMC)	Aggregate of complexities of methods in a class. When complexities are equal, number of methods defined in each class
Depth of Inheritance Tree (DIT)	Number of acestors of a class
Number of Immediate sub-classes of a Class (NOC)	Number of direct descendants of a class
Coupling between Object (CBO)	Number of classes coupled to a specific class
Lack of Cohesion on Methods (LCOM)	Number of pairs of member functions with shared instance variables subtracted by the total pairs of member functions without shared instance variables.
Response For a Class (RFC)	Number of methods of a specific class plus number of other class methods called by the methods of this class.
Martins Package Metric Suite - Package Level Metrics [7]	
Metric Name Value	**Definition**
Efferent Couplings (Ce)	Classes in the package that depend on the other packages. It is an indicator of the packages independence.
Afferent Couplings (Ca)	Number of other packages that depend upon classes within the package. It is an indicator of the packages responsibility
Abstractness (A)	Ratio of the number of abstract classes (and interfaces) in the analyzed package to the total number of classes in the analyzed package.
Instability (I)	Ratio of efferent coupling (Ce) to total coupling (Ce + Ca) such that I = Ce / (Ce + Ca).
Distance from the Main Sequence (D)	Perpendicular distance of a package from,the idealized line A + I = 1.
Fernando Britoe Abreus MOOD Metric Suite - Application Level Metrics [6]	
Metric Name Value	**Definition**
Method Hiding Factor(MHF) and Attribute Hiding Factor (AHF)	These metrics measure how properties like variables and methods are encapsulated in a class. A private property is completely hidden. Encapsulation is calculated with respect to other classes.
Polymorphism Factor (PF)	Number of actual method overrides divided by the maximum number of possible method overrides.
Coupling Factor (CF)	Number of actual couplings among classes in relation to the maximum number of possible couplings.
Method Inheritance Factor (MIF)	Number of inherited methods divided by total methods available in class.
Attribute Inheritance Factor (AIF)	Number of inherited attributes divided by total attributes available in class. A class that inherits large number of methods/attributes from its ancestor class has very high MIF/AIF.

called Green Miner. It is a hardware based framework that runs several tests repeatedly for a software to measure its energy consumption. Aggarwal et al. [30] analyze the pattern of change in system calls of application and based on their results suggest a rule-of-thumb that any change in system calls has an impact on the application's energy consumption profile. In a later work [31] they propose a technique that predicts the energy consumption profile of an application using system calls of a software. Hasan et al. [32] and Pinto et al. [33] studies the energy behavior of Java data-structures with the latter focusing on thread-safe implementations.

Energy consumption of software has also been related to its structure in literature. Hindle [6], [5] proposed a Green Mining experimental methodology to relate change in object-oriented metrics with power consumption. In contrast to previous work, instead of considering the object-oriented metrics of the overall software structure, our work only considers metrics relevant to execution path.

III. METHODOLOGY

In this section we present a novel methodology for correlating energy consumption of software to structural information relevant to test execution traces. We also present an implementation of the proposed methodology in the context of our case studies in the next section.

The steps to carry out a study in which energy measurements can be correlated to structural information relevant to the test execution traces are enumerated as follows:

- Choose multiple products and build its versions
- Identify test cases and configure test bed
- Specify measurement procedure and gather energy measurements
- Specify granularity level of structural information
- Gather structural information and execution traces
- Filter structural information based on execution traces
- Relate filtered structural information to energy measurements

Choose multiple products and build its versions: The first step is to choose multiple iteratively developed products on which the study will be conducted and build all its available versions. The purpose of selecting multiple products is to ensure variation among the structural metric values so that a generalizable correlation model can be derived between metrics and energy. Building applications is a constant issue in repository mining studies because of the evolving requirements of a system, as well as dependencies. This is the most challenging and time consuming task because if we are unable to build an application, instrumentation is not possible and hence we can not gather execution traces or energy profiles. It is also important to make sure that the source code of the selected versions of the product can be deployed on experimental hardware.

Identify test cases and configure test bed: In this step, selected product's functionalities are identified because test cases are different scenarios of usage of application. Each distinct functionality maps to a separate test case whose energy is then measured by executing it on actual hardware. The energy consumed by a test can be due to CPU consumption, WiFi, GPS or LCD, so energy of each component involved in test execution is to be measured. For example, if an application is performing a CPU intensive task like searching, CPU energy must be measured whereas for a task like browsing WiFi energy must also be considered. It is also important to make

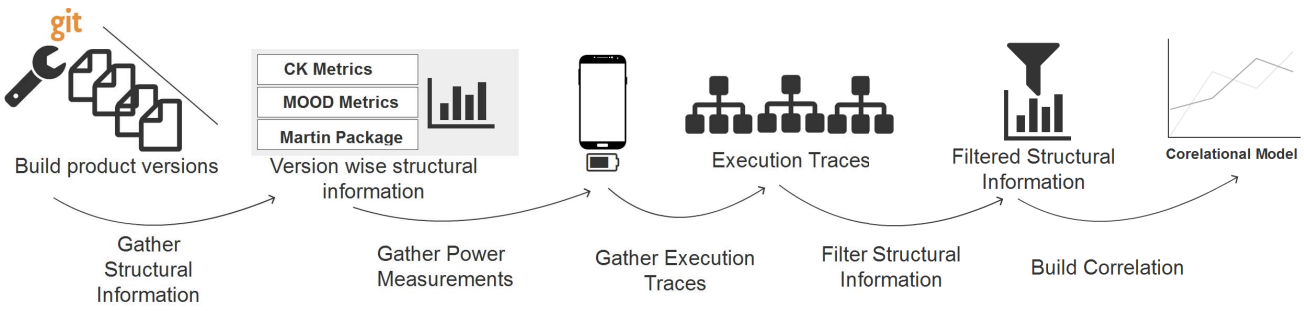

Fig. 2. Proposed Methodology

sure that these test cases are comprehensive in terms of use-case functionality so that entire application is covered by exercising these tests. Furthermore, defining test cases on the basis of code coverage criterion such as structural, decision, modified decision/condition or pair-wise coverage can be an additional step which is out of the scope of this paper. After the identification of test cases the operating system and hardware needs to be specified on which the test cases will be executed. Test case energy consumption is highly dependent on these factors. Moreover since mobile applications have highly variable consumption that can vary across each test execution hence necessary precautions should be taken to exclude background noise during test case execution for purposes of accuracy.

Specify measurement procedure and gather energy measurements: In this step we decide how energy consumption of software is to be measured and the software/hardware tools required to accurately measure energy. In order to reduce error each test case should be executed multiple times on each version of the software to collect average energy consumption information. This step can be performed manually. However, automated test execution using scripts can make the task of running tests multiple times easier.

Specify granularity level of structural information: In this step the granularity level at which software structural information is going to be extracted and analyzed has to be decided. Granularity could be at project, class, package, method or line of code level (LOC). For example, if one wants to analyze how changes at class level affect energy consumption of application then class granularity will be chosen whereas if changes at line level have to be correlated with energy then LOC granularity is selected. The purpose of selecting granularity is to make sure that the metrics with which we want to find correlation and test execution traces are at same level. After selecting granularity, it needs to be decided how structural information at specified granularity can be extracted. For instance, if one has selected class level granularity, *cohesion* and *coupling* would provide interesting structural information to be considered. This information can be extracted using existing CK metrics suite.

Gather structural information and execution traces: In this step, structural information is extracted from all versions of the selected software. It should be made sure that one is able to build selected products' versions so that test case specific

Algorithm 1: EXTRACT UNIQUE TRACES

Input: $sm[]$: software structural metrics
$test[]$: M test cases
granularityLevel[]: the specified value can include enum(CLASS,PACKAGE,METHOD)
Output: executionTraces[][]: executionTraces of tests

```
for i ← 1 to M do
    components[i] = executeTestCase(test[i],sm[i])
    // Components exercised through this test case
    for j ← 1 to components.size do
        for k ← 1 to N do
            if components[j] ∈ granularityLevel[k] then
                executionTraces[i][j].add(components[j])
            end
        end
    end
end
```

structural information can be gathered by instrumenting the code. The purpose of this step is to track and log the execution paths of the test cases when they are executed. These execution paths are going to help us identify the static components of code that are traversed when a specific test case is executed. Each version's code need to be instrumented in order to gather execution traces at the selected granularity level. For instance, if the chosen granularity is class then the code has to be instrumented in such a way that when a test case is executed, all the classes involved in the execution are logged. It is to be noted that instrumentation does not affect energy consumption of tests in any way. It only allows us to filter the test path specific metrics.

The procedure for gathering execution traces is elaborated in Algorithm 1. Although both steps can be performed manually, we automate these steps by developing a tracing tool and use it for collecting traces. Our tool is able to log test execution traces of Java based android applications on class and package granularity level[3].

Filter structural information based on execution traces: As mentioned earlier, the collected execution traces and the considered structural information both need to be on the same granularity level. This helps in filtering out structural information that is related to the execution path trace and

[3]https://github.com/AbdulAli/ExecutionTraceTracker

Algorithm 2: GENERATE FILTERED METRICS

Input: $appVer[]$: all versions of application
$test[]$: the test cases
N: number of total versions of application
M: number of test cases
L: number of runs per test case

Output: $energy[][][]$: energy measurements
filteredStructuralMetrics[][]: filtered metrics for each application version

```
/* Initialize energy measurements */
initialize(energy[][][])
/* Gather structural information */
for i ← 1 to N do
    v[i] ← buildApplicationVersion(appVer[i])
    sm[i] ← extractStructuralMetrics(appVer[i])
    for j ← 1 to M do
        for k ← 1 to L do
            energy[i][j][k] ←
              calculateEnergy(appVer[i],test[j])
        end
    end
    /* Gather execution traces */
    for j ← 1 to M do
        vt ← generateVerWithTracerCode(appVer[i]);
        loggedTraces = execute(vt, test[j]);
        // Discarding re-occuring traces
        et[i][j] ← extractUniqueTraces(loggedTraces,
          sm[], ...);
    end
end
// Filter Metric values based on execution traces
for i ← 1 to N do
    for j ← 1 to M do
        filteredStructuralMetrics[i] ←
          filterStructuralMetrics(sm[i], et[i][j]);
    end
end
```

hence we avoid building an unrealistic correlation model that correlates energy consumption of a specific test case to complete product's structural information. Instead, we filter the structural information by cropping out irrelevant chunk of the structure which is not part of this test execution. This step identifies parts of code which are exercised by a specific test case. In our study, we perform this step through our automated tracing tool as well. Algorithm 2 describes the procedure for generating the filtered structural information using this tool.

Relate filtered structural information to energy measurements: The final step is to aggregate the collected data meaningfully and determine the relationship between filtered structural information and energy measurements of software. Recall that the filtered structural information is the structure of a certain test execution trace. This step can be performed using different statistical tests and machine learning algorithms like linear regression, logistic regression, decision trees, neural networks etc. However in our study, we use *Spearman's ρ-rank* test to determine the required correlations.

IV. EXPERIMENTAL SETUP

In this section we describe our experimental setup whose objective is to determine how structural changes in software affect software energy consumption. For setting up our experiment we follow the methodology proposed in the previous section.

Choose multiple products and build its versions: For the purpose of our experiment we chose multiple version of three products. First is QKSMS - a messaging application, BeHe ExploreR - an android phone web browser and PDF Creator - an application for converting text and image documents to PDF. The choice of applications was challenging because we needed to instrument the application code to extract execution traces and hence the applications that could be successfully built were considered. We choose these case studies on the basis of following factors: 1) application is open-source and has at least five releases, 2) application is iteratively developed having development span of a year or more, 3) application has multiple functionalities. All these android applications are open-source, Java based and meet the aforementioned criteria. We select all 24 versions of QKSMS that are available on Github starting from revision 2.1.0 on September 1, 2015 with 76,051 LOC to revision 2.7.3 on September 5, 2016 with 1,08,827 LOC. We were able to successfully compile 22 versions for structural analysis. Nine versions of BeHe ExploreR are selected starting from revision 1.0.0 on Feb 11, 2016 with 1,432 LOC to revision 2.0.1 on Jan 20, 2017 with 6,435 LOC. For BeHe ExploreR we were able to compile and build 8 versions. 13 versions of PDF Creator were selected starting from August 6, 2016 with 9,010 LOC to March 16, 2017 with 10,871 LOC. Out of these we used only 5 versions because the remaining versions showed very little variation. The selected versions are built using gradle android Studio.

Identify test cases and configure test bed: We design a set of test cases covering the important use case scenarios of applications under study. For QKSMS we test 5 scenarios, the first is an idle screen test case while second, third and fifth scenario tests the functionality of sending text-messages 5 characters long, 200 characters long and a 5 character message sent to five contacts. The fourth scenarios tests the functionality of sending multi-media message of size 100Kb. For BeHe ExploreR case study, one test case is executed which measures energy consumption of accessing a webpage. For executing this test, the user opens google homepage, types the term *"Reddit Software Engineering"* and executes search. Then clicks on the first link of Reddit from retrieved searches and scrolls till the end of displayed webpage. For PDF Creator we execute two tests. In the first test an image is selected, cropped and effects are applied on it. The resulting image is saved as PDF. In second test 50 characters are typed and converted into PDF. These tests only cover frequent use-case scenarios of an application and by no means intend to cover the application comprehensively. It is also important to note that since we relate metrics on the basis of execution

trace to energy consumption of a test, results would not be biased regardless of the coverage of test case because only the structure of executed part is considered. However, we believe that incorrect correlations may be observed if overall application metrics are related to a test without considering execution trace as has been done previously [6].

The hardware to execute test cases is Galaxy S6 running Android 6.1 Marshmallow and for measuring energy consumption PowerTutor [21] is used. Each test case is run ten times for each version of the software. During all these runs background applications are killed, screen-savers and automatic updates are also turned off. The screen brightness level remains constant while test cases are executed. Fig. 8 and 9 illustrate that the energy consumption of each test case per version of QKSMS is different. We exclude QKSMS's version 2.6.1 while plotting these graphs because it is an outlier exhibiting higher energy consumption than other versions. Similarly the graph for BeHe ExploreR browsing test is shown in Fig. 10. We have not included the remaining energy consumption results due to space limitations.

Specify measurement procedure and gather energy measurements: For energy consumption measurement we use PowerTutor tool [21]. Although Power Tutor's power model was built for a specific set of android smartphones, however its power model parameters can easily be configured to make it compatible with other smartphone devices like Samsung Galaxy S6. For each version of all products, we compile and deploy it on test-bed hardware. To accurately measure energy, we perform 10 runs of each test case and record average energy consumption as done previously by [29], [34]. Each test case execution is performed manually by hand interacting with the application to record energy measurement of each run. Tests were executed multiple times to minimize the inaccuracies of measurement and to get reliable results. After performing this step, we end up with more than 1300 energy measurement tests. All the energy measurements are in mJ.

Specify granularity level of structural information: In this study we are interested in finding how changes at class, package and application level affect an application's energy. This structural information is provided by the object-oriented CK, Martin and MOOD suites already. CK metrics are at class level, Martin metrics are at package level and MOOD suite provide application level information.

Gather structural information and execution traces: The selected metrics are extracted using MetricsReloaded [35] plugin available for android projects. The source code of each version is instrumented at package and class level with the help of our tracing tool to collect test execution traces. The purpose of instrumentation is to log execution traces when a specific test case is executed. After instrumentation the code of each version is rebuilt and deployed on test hardware. After instrumented code deployment, each test is re-run once again for each version to record its execution trace.

Filter structural information based on execution traces: Finally we filter the previously gathered structural information of the overall product on the basis of extracted execution traces in the previous step. The objective is to consider metrics of only that part of code which is exercised by a certain test case. The filtered CK and Martin metric values for five test cases of QKSMS are plotted in Fig. 4 and Fig. 5 respectively, where they represent package and class level metric values as per test case execution trace. Whereas for a single test case, the filtered metric values of both metric suites for BeHe ExploreR are plotted in Fig. 6 and 7.

Relate filtered structural information to energy measurements: We perform a correlation analysis on data collected for each case study to determine a relation between software metrics and energy consumption. We employ *Spearman's* ρ correlation to determine the correlations that exist in our data. *Spearman* is a well-known statistical test used to determine the dependency of one variable on another. Spearman works by counting agreements and disagreements in ranks between samples and are suitable for data that is not normally distributed as it is in our case.

The results of this analysis are given in the next section.

V. Case Study Results

In this section we present an analysis of the results of case studies used in our experimentation. Our case-studies comprise three applications and we employ *Spearman's* ρ*-rank* correlation coefficient to examine if correlations exist in our collected data. We report the results of correlation between energy and metrics while addressing each of our research questions in detail.

RQ1:Why can't the energy consumption of software be related to the overall structure of an application?

Our first research question investigates the relationship between the overall structure of software and its energy consumption. Answering this research question allows us to establish our findings in context with previously conducted studies using CK metrics [5]. The structural information of software in our experiments is represented by three existing object-oriented metric suites. We calculate Spearman's coefficient for each metric under investigation separately given as the ρ-value. The null-hypothesis of the test is that ρ is zero which means that no correlation exists between energy and metrics. We reject the null-hypotheses if p-value is less than $\alpha = 0.05$.

We analyzed the collected data and found that the energy values of tests have little variability with mean centered around 300 to 400 mJ for each test as shown in Fig. 8 through 10. The mean energy consumption changes across versions but there is no clear trend except in PDF Creator where energy consumption decreases in the later versions. Also little variation is observed in metrics values of case studies other than QKSMS. Hence for the purpose of answering this question we combined the set of metric values from the three case-studies and relate them with energy consumption of tests to compute the *Spearman's* ρ*-rank* correlation. The third column of Table II shows the results of applying *Spearman's* ρ*-rank* to the overall metrics of the three applications combined. After p-value correction through Bonferroni method we found that

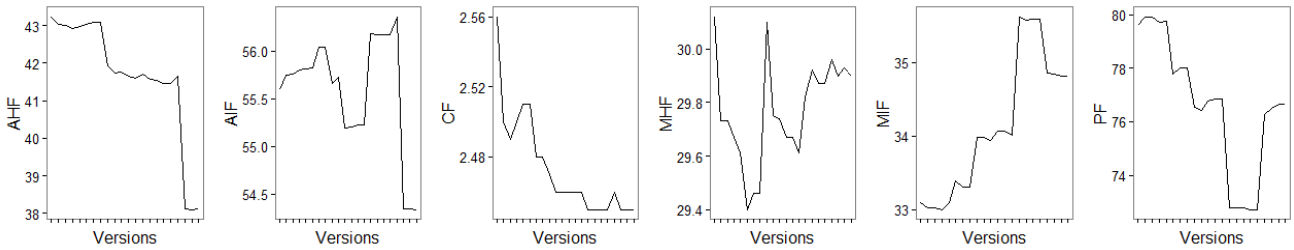

Fig. 3. Mean MOOD Metrics distribution for twenty-two versions of QKSMS case study

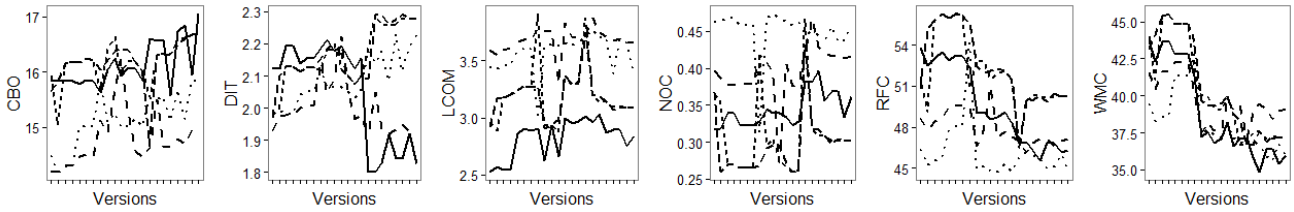

Fig. 4. Mean Filtered CK Metrics distribution of five test cases for 22 versions of QKSMS case study on the basis of execution structure

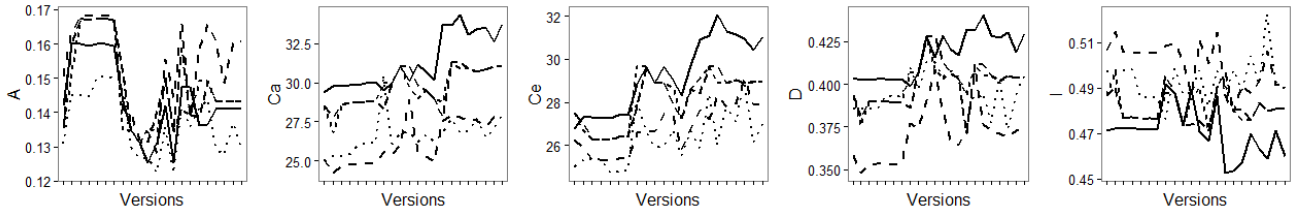

Fig. 5. Mean Filtered Martin Metrics of five test cases for 22 versions of QKSMS case study on the basis of execution structure

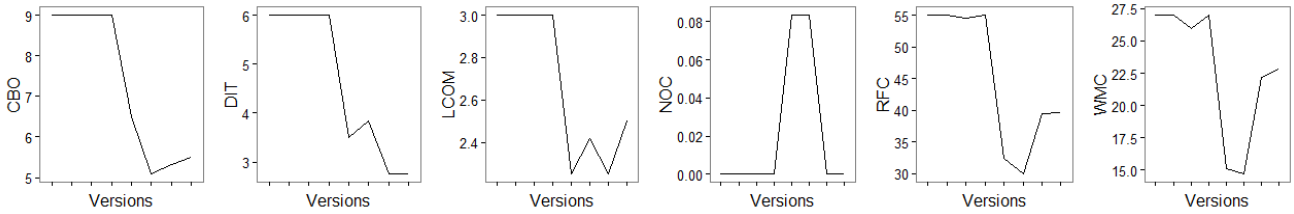

Fig. 6. Mean Filtered CK Metrics for eight versions of BeHe ExploreR case study on the basis of execution structure

almost half of CK metrics and few Martin metrics of overall application correlate with energy consumption values. In contrast to CK and Martin metrics which are computed at class and package level respectively, MOOD metrics are computed at the application level. For each version of QKSMS one value of each MOOD metric is generated as shown in the Fig. 3. After applying the correlation test and p-value correction on the MOOD metric suite only AIF $(p = 0.2546)$ seems to be slightly correlated to energy. However, we argue that these correlations are not only small in magnitude $([-0.4, +0.4])$ but are probably also inaccurate because the metrics may not be covering the entire code with uniformity.

RQ2:How does structural information relevant to test execution trace relate to energy consumption of software applications?

The second research question aims to determine whether any correlation between metrics and energy exists if we only consider the metric values relevant to test execution trace. For answering this question we use the structural information of the execution trace that is collected using algorithm 1 and classified as *filtered metrics*.

TABLE II
SPEARMAN'S-RHO RANK CORRELATION FOR OVERALL METRICS AND TEST STRUCTURE SPECIFIC METRICS

		Overall Structure		Test Execution Structure	
		ρ	p-value	ρ	p-value
CK	CBO	-0.1698	0.0604	-0.7163	**2.20E-16**
	DIT	0.4026	**3.80E-06**	0.7028	**2.20E-16**
	LCOM	-0.3881	**9.19E-06**	-0.1596	0.0720
	NOC	0.1180	0.1937	-0.5134	**5.77E-10**
	RFC	-0.3187	**0.0003278**	-0.1899	0.03178
	WMC	-0.4359	**4.67E-07**	-0.1899	0.0318
Martin	A	-0.4034	**3.73E-06**	-0.6626	**2.20E-16**
	Ca	-0.1203	0.1851	-0.6874	**2.20E-16**
	Ce	-0.1247	0.1695	-0.6639	**2.20E-16**
	D	0.1656	0.0671	0.2755	**0.001644**
	I	0.3784	**1.59E-05**	0.4397	**2.07E-07**
MOOD	AHF	-0.1385	0.1267	Applicable on overall structure	
	AIF	0.2546	**0.004488**		
	CF	0.1468084	0.1052		
	MHF	0.0643	0.4801		
	MIF	-0.1765	0.0508		
	PF	0.2007	0.0260		

Spearman's ρ-rank correlation is then applied to the filtered CK and Martin metrics of all versions and their respective energy consumption. The results of our analysis are reported

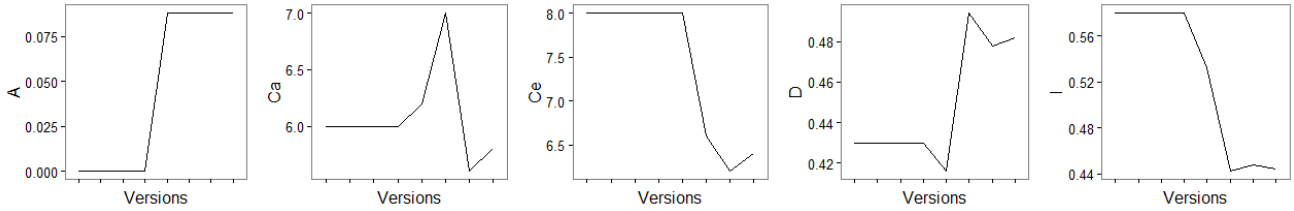

Fig. 7. Mean Filtered Martin Metrics distribution for eight versions of BeHe ExploreR case study on the basis of execution structure

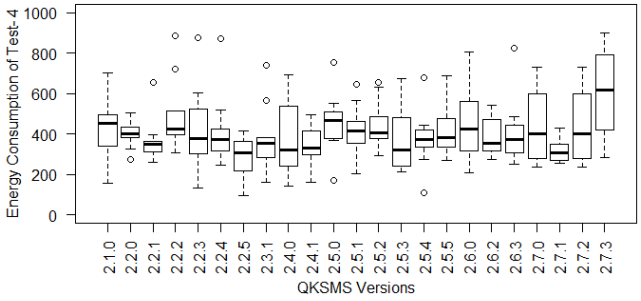

Fig. 8. Energy Consumption of Sending Multimedia Message

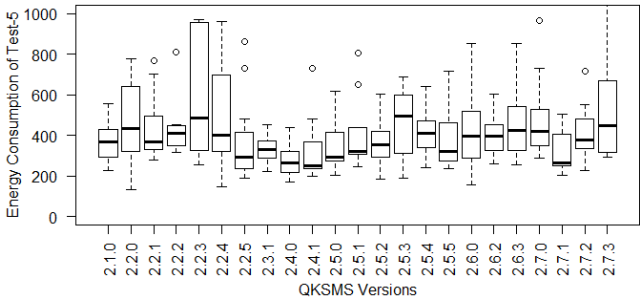

Fig. 9. Energy Consumption of Sending 5 Character Text Message to 5 Contacts

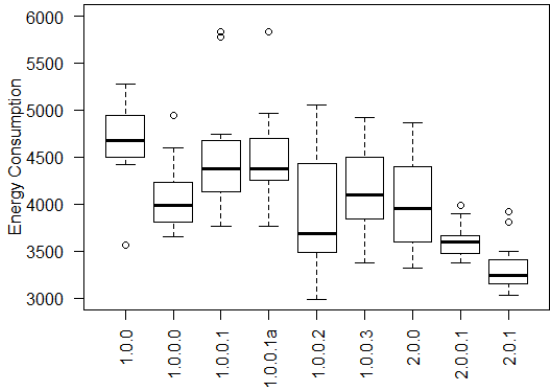

Fig. 10. Energy Consumption of Browsing Test

in the right most column of Table II.

Interestingly enough, all the metrics in CK and Martin metric suite correlate to energy consumption with high level of statistical significance except *Lack of Cohesion On Methods(LCOM)*, a class level metric. However after applying p-value correction for multiple hypotheses, WMC and RFC do not show significant correlation and were discarded. A strong correlation is observed between energy and CK's *CBO*($\rho = -0.7$) as well as *DIT*($\rho = 0.7$) even after p-value correction. Similarly, Ca($\rho = -0.6$) and Ce($\rho = -0.5$) in Martin's package suite also exhibit strong correlations. We believe

TABLE III
CORRELATION RESULTS PROVIDING ENERGY INSIGHTS BETWEEN TWO VERSIONS OF QKSMS FOR SENDING MULTIMEDIA MESSAGE

QKSMS 2.5.2	
Metric	**Value**
CBO	16.230
DIT	2.039
NOC	0.273
A	0.136
Ca	29.696
Ce	28.956
D	0.409
I	0.485
AIF	0.552

QKSMS 2.7.3 (Energy Insights)					
Metric	**Value**	**Energy**	ρ	**Impact**	**Change(%)**
CBO	14.906	⇑	-ive	V Large	-8.16%
DIT	1.934	⇓	+ive	V Large	-12.89%
NOC	0.416	⇓	-ive	Large	+52.55%
A	0.150	⇓	-ive	Large	+10.81%
Ca	25.097	⇑	-ive	Large	-15.48%
Ce	26.258	⇑	-ive	Large	-9.32%
D	0.358	⇓	+ive	Minor	-12.48%
I	0.506	⇑	+ive	Moderate	+4.47%
AIF	0.556	⇑	+ive	Minor	+0.74%
Overall Energy		**Increase**			

that the examined metrics are not totally independent and may capture redundant information. In future work we will address these dependencies between metrics to identify these redundancies.

MOOD metrics are at the application granularity level and therefore independent of the execution trace and hence irrelevant for this question.

RQ3:Which structural metrics strongly correlate with software energy consumption? In answer to this question we reason about the metrics that display strong correlations and those that do not correlate at all with energy consumption of software applications under study. The correlation coefficient denoted by ρ is measured on a scale of -1 to +1 and for all the tests the confidence $\alpha = 0.05$. We say that correlation is significant if *p-value* is less than 0.05. For significant correlations the larger the absolute value of ρ, the stronger the impact (positive or negative, depending upon the sign of the coefficient) on energy consumption. We measure the correlation on Hopkin's scale ($< 0.1 \rightarrow$ *trivial*, $0.1 - 0.3 \rightarrow$ *minor*, $0.3 - 0.5 \rightarrow$ *moderate*, $0.5 - 0.7 \rightarrow$ *large*, $0.7 - 0.9 \rightarrow$ *very large*, and $0.9 - 1 \rightarrow$ *almost perfect*) [36]. Although almost all of the metrics correlated with energy as shown in Table II but some of these correlations become insignificant after correcting for multiple hypotheses (Bonferroni corrected p-value=0.0045). The strength or impact of the remaining correlations also vary across the metrics suites. CK's CBO(Coupling between Objects) has a large negative correlation with energy ($\rho = -0.7163$) whereas DIT (Depth of Inheritance Tree) shows a large positive correlation ($\rho = 0.7028$). This means that a decrease in CBO value negatively affects the energy of an application whereas decrease in DIT has a positive effect on energy. DIT leads to the possibility of multiple superclass methods being called leading to a higher energy consumption. In Martin's Package metrics Ca (Afferent couplings) is the number of other packages that depend upon

TABLE IV
CORRELATION RESULTS PROVIDING ENERGY INSIGHTS BETWEEN TWO VERSIONS OF BEHE EXPLORER FOR BROWSING TEST

QKSMS 2.5.2	
Metric	**Value**
CBO	6.5
DIT	3.5
NOC	0.083
A	0
Ca	6
Ce	8
D	0.43
I	0.58
AIF	0.936

QKSMS 2.7.3 (Energy Insights)					
Metric	**Value**	**Energy**	ρ	**Impact**	**Change(%)**
CBO	5.5	⇑	-ive	V Large	-15.38%
DIT	2.75	⇓	+ive	V Large	-21.43%
NOC	0	⇑	-ive	Large	-100%
A	0.023	⇓	-ive	Large	+0.02%
Ca	12.333	⇓	-ive	Large	+105.56%
Ce	16	⇓	-ive	Large	+100%
D	0.353	⇓	+ive	Minor	-17.83%
I	0.7	⇑	+ive	Moderate	+20.69%
AIF	0.818	⇓	+ive	Minor	-12.61%
Overall Energy		**Decrease**			

classes within the package and is an indicator of the package's responsibility whereas Ce(Efferent couplings) is the count of classes in the package that depend on the other packages and is an indicator of the package's independence. Both these as well as A(Abstractness) show large negative correlations with ρ values of -0.69, -0.66 and -0.66 respectively.

A. Discussion

The results of our study show that correlation between metrics and energy consumption exist and metrics showing large correlations are good indicators of energy consumption behavior of an application. The energy insights that we derive from these correlations are demonstrated by comparing structural metric values of QKSMS's v2.5.2 with v2.7.3 in Table III and BeHe ExploreR's v1.0.0.2 with v2.0.1 in Table IV. In these figures, only those metrics are included which show a correlation with energy in Table II and the aforementioned versions were chosen because they are distant in time. Impact column shows the magnitude of correlation between certain metrics and energy measured on Hopkin's scale. The change column shows the percentage change in the value of a certain metric between the compared versions. A glimpse at Table III clearly shows that majority weighted metrics predict that overall energy will increase in v2.7.3 of QKSMS which can be confirmed from Fig. 8 that shows an increase from 449.8mJ to 604.4mJ as we move from v2.5.2 to v2.7.3. Similarly, majority weighted metrics in Table IV predict an overall decrease in energy of BeHe ExploreR's later version. Fig. 10 also confirms this fact because v1.0.0.2's median energy consumption is 3922.6mJ while v2.0.1's median consumption is 3331.9mJ. This shows that our correlation values are useful in providing information about energy through static analysis and by using our approach developers can gain energy insights about their application structure during development. For instance, if QKSMS's new version 2.7.3's test cases were not executed to measure energy consumption, a developer will still be able to gain an insight (based on the current and previous version's structural information) that the energy consumption will increase.

VI. Threats to Validity

There are three levels of validity threats that we discuss following common guidelines provided in [37].

Conclusion Validity The conclusion validity concerns issues that affect the ability to draw the correct conclusion from an experiment. It is highly dependent on the quality of data gathered, for instance, the reliability of energy measurements. We measure energy values using Power Tutor [21] at the application granularity level. To ensure consistency only a single person collected data for all the versions of an application. To counteract random irrelevancies that affect our measurement we killed all background applications and processes, turned off mobile data while executing the test cases of each version. The choice of statistical test is another important factor that affects validity of results. In this experiment we employ a well-known statistical method *Spearman's ρ-rank* for determining correlations. It has been previously used by researchers for similar studies [5].

Internal Validity Threats to internal validity concern selection of subject systems and analysis methods. The case studies we have selected are android applications. The issue with android applications is that they have a narrow scope, their maintenance span is very short and generally small changes are performed for every new release. It is therefore difficult to find case-studies for which multiple tests exercising different parts of application structure can be generated. In contrast to previous studies, we analyze the effect of software structural metrics on energy consumption while considering test case execution structure. However a weakness of work is that the energy measurements taken by PowerTutor may not be very accurate and could have been affected due to human error because tests were executed manually.

External Validity External validity affects our ability to generalize the results of our experiments beyond the case study. We executed more than 1300 runs of 8 different tests on 45 versions of three different applications manually and it took more than 30 days to collect energy measurements for all case studies. In addition, we re-executed each contexts separately for the instrumented versions of the applications to collect execution traces. This separate execution was performed to cancel out the effect of instrumented code on original structure of the application. The results of our experiment are valid for android applications of similar nature and complexity. However, external validity could be improved by evaluating more applications of varying complexity. In future work, we plan to evaluate our technique on larger and more diverse applications.

VII. Conclusion

This paper presents a novel methodology for relating software change to software energy consumption on the basis of execution structure. We apply the proposed methodology to conduct an evaluation of CK, MOOD and Martin Package metrics on several versions of three distinct open-source android applications; QKSMS, BeHe ExploreR and PDF Creator. We relate the metrics in these suites such as coupling between objects and weighted method calls to energy consumption and discover that all of CK and Martin metrics except one correlate with energy in a statistically significant way. As expected, the

MOOD metrics do not show large correlations with energy since they can only be computed at the application level. The results from our case-studies show the usefulness of these metric suites in assessing the energy consumption behavior of android applications developed iteratively. This is a key contribution of our research since previous studies were unable to establish clear correlations between software metrics and energy consumption behavior. Although this study is the first step towards building an energy model for online energy aware development, the methodology presented in this paper can be used to conduct similar investigations on case studies of varying levels of complexity.

REFERENCES

[1] C. Pang, A. Hindle, B. Adams, and A. E. Hassan, "What do programmers know about software energy consumption?" *IEEE Software*, vol. 33, no. 3, pp. 83–89, 2016.

[2] H. Khalid, E. Shihab, M. Nagappan, and A. E. Hassan, "What do mobile app users complain about?" *IEEE Software*, vol. 32, no. 3, pp. 70–77, May 2015.

[3] 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.

[4] B. Fernandes, G. Pinto, and F. Castor, "Assisting non-specialist developers to build energy-efficient software," in *Proceedings of the 39th International Conference on Software Engineering Companion*. IEEE Press, 2017, pp. 158–160.

[5] A. Hindle, "Green mining: a methodology of relating software change and configuration to power consumption," *Empirical Software Engineering*, vol. 20, no. 2, pp. 374–409, 2015.

[6] A. Hindle, Abram, "Green mining: A methodology of relating software change to power consumption," in *Proceedings of the 9th IEEE Working Conference on Mining Software Repositories*. IEEE Press, 2012, pp. 78–87.

[7] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," *IEEE Transactions on software engineering*, vol. 20, no. 6, pp. 476–493, 1994.

[8] N. Nagappan and T. Ball, "Use of relative code churn measures to predict system defect density," in *Software Engineering, 2005. ICSE 2005. Proceedings. 27th International Conference on*. IEEE, 2005, pp. 284–292.

[9] J. Bansiya and C. G. Davis, "A hierarchical model for object-oriented design quality assessment," *IEEE Transactions on software engineering*, vol. 28, no. 1, pp. 4–17, 2002.

[10] R. C. Martin, *Agile software development: principles, patterns, and practices*. Prentice Hall PTR, 2003.

[11] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in software engineering*. Springer Science & Business Media, 2012.

[12] A. Hindle, "Green software engineering: the curse of methodology," in *IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, vol. 5. IEEE, 2016, pp. 46–55.

[13] C. Bunse, Z. Schwedenschanze, and S. Stiemer, "On the energy consumption of design patterns," in *Proceedings of the 2nd Workshop EASED@ BUIS Energy Aware Software-Engineering and Development*, 2013, pp. 7–8.

[14] A. Litke, K. Zotos, A. Chatzigeorgiou, and G. Stephanides, "Energy consumption analysis of design patterns," in *Proceedings of the International Conference on Machine Learning and Software Engineering*, 2005, pp. 86–90.

[15] C. Sahin, F. Cayci, I. L. M. Gutiérrez, J. Clause, F. Kiamilev, L. Pollock, and K. Winbladh, "Initial explorations on design pattern energy usage," in *Green and Sustainable Software (GREENS), 2012 First International Workshop on*. IEEE, 2012, pp. 55–61.

[16] W. G. da Silva, L. Brisolara, U. B. Corrêa, and L. Carro, "Evaluation of the impact of code refactoring on embedded software efficiency," in *Proceedings of the 1st Workshop de Sistemas Embarcados*, 2010, pp. 145–150.

[17] C. Bunse, H. Höpfner, S. Roychoudhury, and E. Mansour, "Choosing the" best" sorting algorithm for optimal energy consumption." in *ICSOFT (2)*, 2009, pp. 199–206.

[18] 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*. ACM, 2011, pp. 153–168.

[19] 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*. ACM, 2012, pp. 29–42.

[20] I. Manotas, L. Pollock, and J. Clause, "Seeds: a software engineer's energy-optimization decision support framework," in *Proceedings of the 36th International Conference on Software Engineering*. ACM, 2014, pp. 503–514.

[21] M. Dong and L. Zhong, "Self-constructive high-rate system energy modeling for battery-powered mobile systems," in *Proceedings of the 9th international conference on Mobile systems, applications, and services*. ACM, 2011, pp. 335–348.

[22] V. R. Basili, L. C. Briand, and W. L. Melo, "A validation of object-oriented design metrics as quality indicators," *IEEE Transactions on software engineering*, vol. 22, no. 10, pp. 751–761, 1996.

[23] L. C. Briand, J. Wüst, J. W. Daly, and D. V. Porter, "Exploring the relationships between design measures and software quality in object-oriented systems," *Journal of systems and software*, vol. 51, no. 3, pp. 245–273, 2000.

[24] L. C. Briand and J. Wüst, "Empirical studies of quality models in object-oriented systems," *Advances in computers*, vol. 56, pp. 97–166, 2002.

[25] T. Gyimothy, R. Ferenc, and I. Siket, "Empirical validation of object-oriented metrics on open source software for fault prediction," *IEEE Transactions on Software engineering*, vol. 31, no. 10, pp. 897–910, 2005.

[26] H. M. Olague, L. H. Etzkorn, S. Gholston, and S. Quattlebaum, "Empirical validation of three software metrics suites to predict fault-proneness of object-oriented classes developed using highly iterative or agile software development processes," *IEEE Transactions on software Engineering*, vol. 33, no. 6, pp. 402–419, 2007.

[27] F. Fioravanti and P. Nesi, "A study on fault-proneness detection of object-oriented systems," in *Software Maintenance and Reengineering, 2001. Fifth European Conference on*. IEEE, 2001, pp. 121–130.

[28] P. Yu, T. Systa, and H. Muller, "Predicting fault-proneness using oo metrics. an industrial case study," in *Software Maintenance and Reengineering, 2002. Proceedings. Sixth European Conference on*. IEEE, 2002, pp. 99–107.

[29] A. Hindle, A. Wilson, K. Rasmussen, E. J. Barlow, J. C. Campbell, and S. Romansky, "Greenminer: A hardware based mining software repositories software energy consumption framework," in *Proceedings of the 11th Working Conference on Mining Software Repositories*. ACM, 2014, pp. 12–21.

[30] 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 *Proceedings of 24th Annual International Conference on Computer Science and Software Engineering*. IBM Corp., 2014, pp. 219–233.

[31] K. Aggarwal, A. Hindle, and E. Stroulia, "Greenadvisor: A tool for analyzing the impact of software evolution on energy consumption," in *Software Maintenance and Evolution (ICSME), IEEE International Conference on*. IEEE, 2015, pp. 311–320.

[32] S. Hasan, Z. King, M. Hafiz, M. Sayagh, B. Adams, and A. Hindle, "Energy profiles of java collections classes," in *Proceedings of the 38th International Conference on Software Engineering*. ACM, 2016, pp. 225–236.

[33] G. Pinto, K. Liu, F. Castor, and Y. D. Liu, "A comprehensive study on the energy efficiency of javas thread-safe collections," in *Software Maintenance and Evolution (ICSME), 2016 IEEE International Conference on*. IEEE, 2016, pp. 20–31.

[34] A. Carroll, G. Heiser *et al.*, "An analysis of power consumption in a smartphone." in *USENIX annual technical conference*, vol. 14. Boston, MA, 2010, pp. 21–21.

[35] "Jetbrains plugin repository — metricsreloaded," https://plugins.jetbrains.com/plugin/93.

[36] W. G. Hopkins, *A new view of statistics*. Will G. Hopkins, 1997.

[37] R. K. Yin, *Case study research: Design and methods*. Sage publications, 2013.