Sustainable Computing: Informatics and Systems 21 (2019) 28–46

Contents lists available at ScienceDirect

Sustainable Computing: Informatics and Systems

journal homepage: www.elsevier.com/locate/suscom

Towards energy aware object-oriented development of android applications☆

Hareem Sahar*, Abdul A. Bangash, Mirza O. Beg

National University of Computer and Emerging Sciences, Islamabad, Pakistan

ARTICLE INFO

Article history:
Received 30 May 2018
Received in revised form 25 August 2018
Accepted 20 October 2018
Available online 19 November 2018

Keywords:
Energy consumption
Mining software repositories
Software metrics
Sustainable software
Static analysis
Energy-aware development

ABSTRACT

Energy consumption has become a concern for developers due to the increasing complexity of applications that are to run on devices with limited battery power. Developers want to develop energy efficient applications however existing tools do not bridge the gap between understanding where energy is consumed and suggesting how the code can be modified in order to reduce energy consumption. A generalized method to relate software structure with its energy consumption is hence desirable. Previous attempts to relate change in object-oriented structure to its effects on energy consumption have been inconclusive. In this paper, we proposed a methodology to relate software structural information represented as metrics to energy consumption. Employing our methodology we empirically validated three Object Oriented (OO) metric suites; the Abreus Metrics (MOOD), Chidamber and Kemerer (CK) metrics and Martin's package metric suite and determine their relationship with energy consumption. Our results show that software structural metrics can be reliably related to energy consumption behavior of programs using a total of 63 releases from seven open-source iteratively developed android applications.

1. Introduction

Mobile technology has become pervasive and the ever-increasing complexity of smartphone applications places heavy demand on the otherwise limited battery power. The demand as well as development of computationally intensive smartphone applications has increased manyfold 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 energy use. However developers lack knowledge of factors that affect energy consumption of software [3] and are at the same time inhibited by limited tool support within IDEs for optimizing energy consumption of applications [4]. On-line energy-aware development tools that can assist developers during application development by providing valuable energy insights are hence needed.

An application of our work is a plugin for Android Studio [1] a commonly used IDE for mobile application development. The plugin can be used during the development phase to measure the change in energy between any two versions of code. The plugin employs correlations between metrics determined by our work to determine the change in energy consumption. The EnSights plugin shown in Fig. 1 can be run on an application under development to provide energy insights as changes are made to the code. The user can drill down to see which metrics are responsible for the depicted energy change.

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 the 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 plug-ins that can be directly integrated into the development environments. The availability of such tools within IDEs

☆ This journal paper is an extension of Abdul A. Bangash, H. Sahar, and M. O. Beg, "A methodology for relating software structure with energy consumption," in 17th IEEE International Working Conference on Source Code Analysis and Manipulation. IEEE, 2017.

* Corresponding author. Tel.: +923245214496.
E-mail addresses: hareemesahar@gmail.com (H. Sahar), abdul.ali@nu.edu.pk (A.A. Bangash), omer.beg@nu.edu.pk (M.O. Beg).

[1] https://github.com/AbdulAli/EnSights/

https://doi.org/10.1016/j.suscom.2018.10.005
2210-5379/

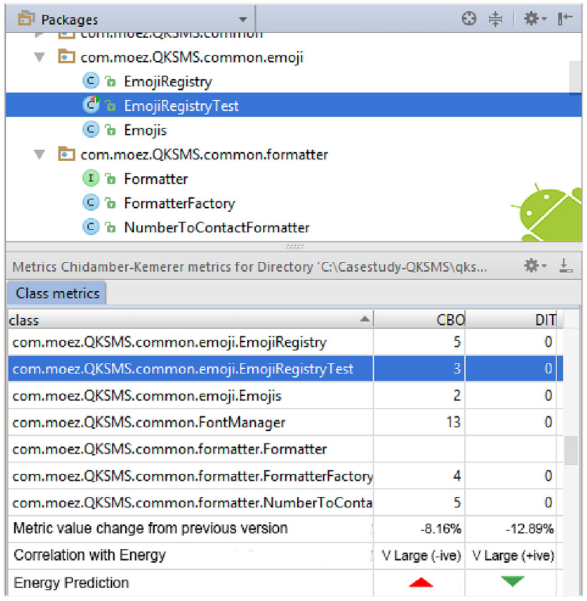

Fig. 1. Demonstration of the working of the EnSights plugin that utilizes the correlations determined by our work.

will allow developers to not only diagnose energy issues in their software but at the same time map them to their cause.

Previously, similar studies examining the effect of structural changes on power consumption were performed by [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 erringly, the study attempted to correlate power consumption of software with whole program metrics instead of considering execution path relevant metrics.

We believe that energy consumption depends on the path that a particular test-case follows during execution. Hence, the structure of the whole program cannot be related with energy consumption of a specific execution path/trace. For instance, consider Fig. 2a which shows metric *D*(*Distance from Main Sequence*) from the Martin metrics suite for the entire QKSMS application. Earlier studies([5] and [6]) used same *D*(*Distance from Main Sequence*) metric values for all tests without considering their execution path. However, in reality, each test passes through a different path and activates different components during its execution. This can be confirmed by looking at Figs. A1 and A2, which show the execution trace of two tests of QKSMS and the corresponding components activated by each test. The metrics relevant to execution path of both tests are plotted in Fig. 2b along with the metrics of remaining three tests of QKSMS. The metrics of each of the five tests are represented through separate lines and can be seen to have sufficient variability among them.

Our work addresses limitations of [5] by correlating energy consumption of tests only with metrics relevant to test execution structure (i.e. only activated components) instead of considering structural information of the whole program that we also refer to as overall structure. To ensure that variability exists in structure of software we related structural information to energy consumption across several versions of seven different applications. We evaluate our technique using two additional metrics suites, Fernando Brito-e-Abreu's MOOD metrics suite [9] and Martin's Package metrics suite [10] in addition to the CK 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 63 releases of seven open-source android applications.

In order to examine how structural information relates to the energy consumption behavior of android applications, we extract several releases of seven open-source iteratively developed android applications from the Github repository out of which three have been previously studied [11]. QKSMS is a messaging app, BeHe ExploreR is a web browser, PDFCreator is an application to create and edit PDF files, Pijaret is a text encryption/decryption tool, LibreTorrent is a torrent client application, QR Scanner is an android QR code scanner and Simple Music Player is an audio player application. We extract structural information of the case studies using MetricsReloaded[2], 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 prior work 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. Around 1500 test runs were performed and energy measurements were collected using PowerTutor tool.[3] After collecting all the relevant data we applied *Spearman* ρ*-rank* correlation and identify that half of the CK and Martin metrics 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 estimate energy changes even before programs are built. A possible application of our correlation results is proposed in Fig. 1, a developer while refactoring class level structure can gain some useful insights about energy profile of application being developed.

In particular, we investigate the correlation between software structural metrics and energy consumption by exploring the following research questions:

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

Presently the available techniques correlate energy consumption with complete structure of application. We are interested in comparing the results of previous technique [6] when evaluated on our case studies with those of our own.

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

As software energy consumption varies as per test case execution, so does the structure through which it executes. We are interested to extract each test case relevant structure for the application to see how exercised components' structural information correlate with energy consumption.

RQ3: *Which structural metrics strongly correlate with software energy consumption?*

The results are discussed to reason about the correlation found, if any, from the performed experiments. Every metric in a metrics suite has different characteristics. There is a need to identify

[2] https://plugins.jetbrains.com/plugin/93-metricsreloaded
[3] http://ziyang.eecs.umich.edu/projects/powertutor/

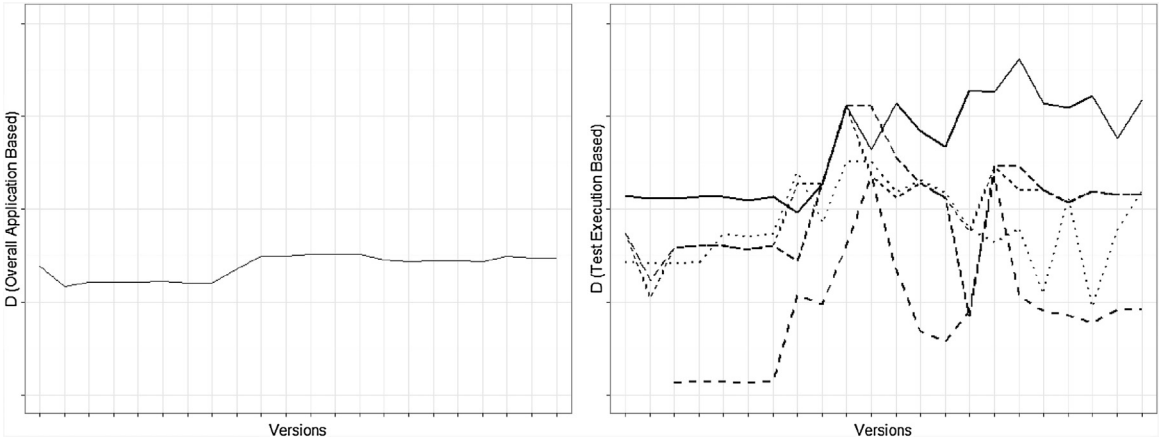

Fig. 2. D(Distance from Main Sequence) values from Martin metrics suite for 24 versions of QKSMS case study(a) without considering execution path of tests and (b) while considering execution path of five tests.

the characteristics which cause strong correlation value between metric and energy consumption.

RQ4: *What are the possible reasons for the observed correlations?* Here we try to reason about the observed correlations that have been discussed while answering previous research question.

This work in this paper builds upon a previously conducted study to relate software structure with energy consumption [11]. We extend previous work by including seven case studies each having at least three version as a result of which we claim that the reliability of our results has improved. We also present a thorough analysis of our results in this paper by trying to reason about the observed correlations. Not only this, we have automated our methodology and developed a tool that provides energy insights to the developers during the development phase. The tool estimates change in energy consumption of software application after a significant change in structure of the application's source code. The remaining paper is structured as follows: In Section 2, we present review of relevant literature. In Section 3, we present a methodology to relate software structure to energy consumption on the basis of test execution path. Section 4 contains details of our experimental setup. An analysis of the experiments we performed on several versions of seven case studies is described in Section 5. Section 6 discusses the threats to the validity of our study and Section 7 concludes the paper.

2. Related work

2.1. 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 [12] and has grown in importance because of the popularity of limited battery devices such as smartphones. Energy performance has also become a serious concern for application software developers [13] and several tools have been developed for mobile energy measurement. [14] 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 its system calls on FSM transitions. Exploiting the same technique, authors develop a tool [15] that provides fine grained information related to energy consumption of Android applications. A general decision making framework was also proposed [16] to help developers select energy efficient libraries in their applications which use Java Collection API. Trepn Profiler [17] is an energy profiler developed by Qualcomm, for android devices. unfortunately, it only works on Snapdragon MDP developer devices. In this study we employ a tool called PowerTutor for energy measurements since PowerTutor is open-source and 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.

2.2. Software metrics

Researchers have long sought to analyze the impact of change in software structure on quality of code development [18]. Metrics such as Chidamber and Kemerer(CK) metric suite [7], Martin's package metric suite [10] and MOOD metric suite [9] have been developed to assess design quality of software [19]. These metrics are briefly explained in Table 1. Evaluations of these metric suites were conducted by [20] and [21,22] that indicate the effectiveness of these metrics. Furthering their work [23] 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, [19] empirically evaluated two other object-oriented metric suites MOOD and QMOOD metrics in addition to CK. Their results show that MOOD and QMOOD metric suites do not have ability similar to CK to predict fault-proneness of software developed by agile process. [24] and [25] 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.

2.3. Green mining

Green mining is a process that aims to analyze power consumption behavior from the information extracted from various software repositories [6]. To address the problems faced by researchers in discovering power usage statistics of software, [26] introduced a regression test framework called Green Miner. It is a hardware based framework that runs several tests repeatedly for a software to measure its energy consumption. [27] analyzes the pattern of change in system calls and based on their results suggest a rule-

Table 1
A brief description of CK, Martin and MOOD Metrics

Metric Name Value	Definition
Chidamber and Kemerer (CK) Metric Suite - Class Level Metrics [7]	
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 ancestors of aclass
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.
Martin's Package Metric Suite - Package Level Metrics [10]	
Metric Name Value	Definition
Efferent Couplings (Ce)	Classes in the package that depend on the other packages. It is an indicator of the package's independence.
Afferent Couplings (Ca)	Number of other packages that depend upon classes within the package. It is an indicator of the package's 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 Brito e Abreu's MOOD Metric Suite - Application Level Metrics [9]	
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.

of-thumb that any change in system calls has an impact on the application's energy consumption profile. In a later work [28] they propose a technique that predicts the energy consumption profile of an application using system calls. Several research works investigate whether there is an energy-efficient approach for Android app development or not [29]. More specifically [30] and [31] studies the energy behavior of Java data-structures with the latter focusing on thread-safe implementations. The concurrency control mechanisms have also been examined by [32] and [33]. Researchers have also built tools that examine the energy consumption of alternative implementations of commonly used programming constructs [34]. Energy consumption of software has also been related to its overall structure in literature. [6,5] proposed a Green Mining experimental methodology to relate change in object-oriented metrics with power consumption. In contrast, instead of considering the object-oriented metrics of whole software application, our work only considers metrics relevant to execution path. The Ensights tool we developed provides insights to developers based on change in metrics values between two versions.

3. 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 trace are shown in Fig. 3 as well as enumerated below:

- 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 we test different scenarios of usage of application. Each distinct functionality maps to a separate test case whose energy is measured by executing the test on actual hardware i.e. android device. For energy measurements an existing tool named PowerTutor was used which runs on android device and logs the energy measurements when a test case runs. 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 sure that these test cases are comprehensive in terms of use-case functionality so that

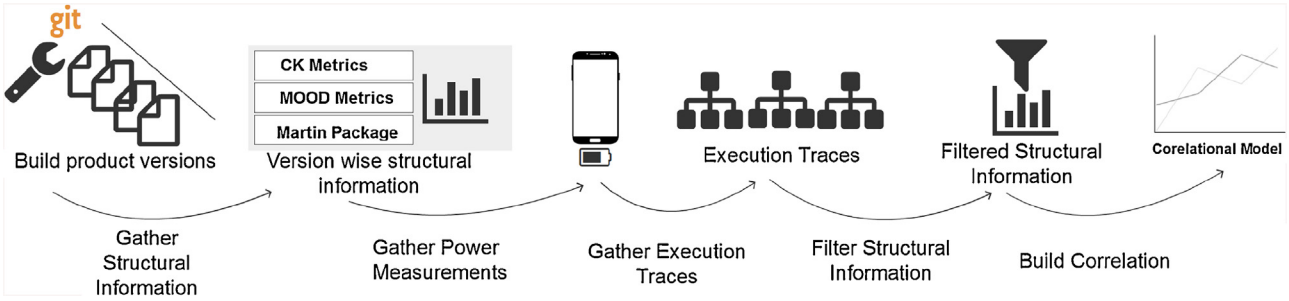

Fig. 3. Steps involved in proposed methodology.

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 as 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 such as closing background applications 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 the class level affect energy consumption of an application then class granularity will be chosen whereas if changes at line level have to be correlated with energy then LOC granularity is selected. Selecting particular granularity helps us ensure that the metrics of interest 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.

Algorithm 1. GATHER EXECUTION TRACES

```
Input: applicationVersion, testCase
Output: executionTrace
1 versionWithTracerCode = generateVersionWithTracerCode(applicationVersion);
// Gather execution traces
2 loggedTraces = executeVersion(testCase);
// Discarding re-occuring traces
3 executionTrace = fetchUniqueTraces(loggedTraces);
```

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: filteredStructuralMetrics[][]: filtered metrics for each
        application version
/* Gather structural information                                    */
1  for i ← 1 to N do
2      v[i] ← buildApplicationVersion(appVer[i])
3      sm[i] ← extractStructuralMetrics(appVer[i])
       /* Gather execution traces                                   */
4      for j ← 1 to M do
5          vt ← generateVerWithTracerCode(appVer[i]);
6          loggedTraces = execute(vt, test[j]);
           // Discarding re-occuring traces
7          et[i][j] ← extractUniqueTraces(loggedTraces, sm[], ...);
8      end
9  end
   // Filter Metric values based on execution traces
10 for i ← 1 to N do
11     for j ← 1 to M do
12         filteredStructuralMetrics[i] ← filterStructuralMetrics(sm[i],
           et[i][j]);
13     end
14 end
```

Algorithm 3. 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
1  for i ← 1 to M do
2      components[i] = executeTestCase(test[i],sm[i])
       // Components exercised through this test case
3      for j ← 1 to components.size do
4          for k ← 1 to N do
5              if components[j] ∈ granularityLevel[k] then
6                  executionTraces[i][j].add(components[j])
7              end
8          end
9      end
10 end
```

Gather structural information and execution traces: In this step, structural information is extracted from all versions of the selected software. Afterwards, the execution paths of the test cases are logged as they help in filtering test case specific structural information. 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 used to identify the static components of code that are traversed when a specific test case is executed. Each version's code needs 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 have an effect on the energy consumption values of tests because they were calculated prior to instrumenting the code. After instrumenting the code we run a test to collect the log of components that the test passes through. Having knowledge of those components allows us to filter the test path specific metrics from the metrics of entire application. The procedure for gathering execution traces is elaborated in Algorithm 1. The algorithm takes a test case as input and finds the path that the test traverses (similar to Fig. A1) for a particular version of application under study. This is done through code instrumentation(at class and package level) and creating log of each component that the test passes through as it executes. Although structural information and execution traces can be gathered manually, however, we automate these steps by developing a tracing tool and use it for collecting traces. Our tool (shown in Fig. 1) is able to log test execution traces of Java based android applications on class and package granularity level [4].

Filter structural information based on execution traces: As mentioned earlier, the collected execution traces and the considered structural information both need to be at the same granularity level. This helps in filtering out structural information that is related to the execution trace and hence we avoid building an unrealistic correlation model that correlates energy consumption of a specific test case to complete product's structural information. In order to make sure our models are realistic, we filter the structural information by cropping out irrelevant chunks of the structure which are not part of this test execution.

In our study, we perform this step through our automated tracing tool as well. The tool employs Algorithm 2 for generating the filtered structural information. It works by identifying components of code (packages and classes) which are exercised by a specific test case when it runs. Since some of the components may be exercised more than once so we eliminate repeated occurrences of various components using Algorithm 3. Once unique traces have been identified using Algorithm 2, the metrics of components that are part of the trace are added up and the remaining metrics are not used. We refer to these metrics as filtered metrics.

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. 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* and *Kendall-tau* test to determine the required correlations.

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

4. 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 which is based on green mining experimental guidelines [6] instead of traditional experiment guidelines [35] for conducting software engineering experiments.

Choose multiple products and build its versions: For the purpose of our experiment we chose multiple releases of seven products. First is QKSMS - a messaging application, BeHe ExploreR - an android phone web browser, PDF Creator - an application for converting text and image documents to PDF, Pijaret - a text encryption/decryption tool, Libre Torrent - a libtorrent client for downloading torrents, QR Scanner - a privacy friendly QR-Code/Barcode decoding application and Simple Music Player - application for running audio files. The choice of applications was challenging because we needed to instrument the application code to extract execution traces and hence applications whose source code was available and could also 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 three releases, 2) application is iteratively developed having development span of a year or more, 3) application has multiple functionalities i.e. at least more than one scenario of application usage. 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 five versions because the remaining versions showed no variation in terms of metrics and energy possibly because of minor changes done by developers in those versions. Six versions of Pijaret were selected starting from August 29, 2016 to September 22, 2016. For Pijaret we were able to compile and build all six versions. Four versions of Libre Torrent were selected starting from October 18, 2016 to March 21, 2017. For Libre Torrent we were able to compile and build all four versions but could deploy only two versions on test bed. Three versions of Simple Music Player were selected starting from June 05, 2016 to April 08, 2017. Four versions of QR Scanner were selected starting from November 10, 2016 to December 27, 2016 and we were able to compile and build all four versions. 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 scenario 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. The user 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. For

Table 2
Specifications of Android Device used in Experiments

Device Model	Samsung Galaxy S6 (G920F)
Chipset	Exynos 7420 Octa
CPU Platform	Octa-core (4x2.1 GHz Cortex-A57 and 4x1.5 GHz Cortex-A53)
GPU	Mali-T760MP8
Screen Size	5.1 inches
Screen Resolution	1440 x 2560 pixels, 16:9 ratio (577 ppi density)
Wi-Fi	802.11 a/b/g/n/ac

Pijaret we execute one test in which the user enters a 1041 character long text and encrypts it with a key, now the encrypted text is decrypted back by the same key. For Libre Torrent a single test case is executed in which a famous torrent file is used to download from test bed. As configuration, downloading and uploading limit is set to unlimited and as soon as the download exceeded 5Mb storage, the downloading is stopped. For QR Scanner a test case is executed in which four android barcode format images are scanned sequentially. In Simple Music Player test case an audio track is played and the track is skipped manually two to three times, when the track is about to end it is paused and the test case terminates. 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 whole application metrics are related to a test without considering execution trace as done previously [6].

The hardware to execute test cases is Galaxy S6 (specifications given in Table 2) running Android 6.1 Marshmallow and for measuring energy consumption PowerTutor 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 at a constant value of 10% while test cases are executed so that minimum energy is consumed by display. Fig. 10 and 11 illustrate that the energy consumption of each test case per version of QKSMS is different. While plotting these graphs we exclude QKSMS's version 2.6.1 because it is an outlier exhibiting higher energy consumption than the other versions. Similarly the graph for BeHe ExploreR browsing test is shown in Fig. 12 and two tests of PDF Creator are plotted in Figs. 13 and 14.

Specify measurement procedure and gather energy measurements: For energy consumption measurement we use PowerTutor. Although PowerTutor'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 [26,36]. Each test case execution is performed by manually 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. We waited for a few minutes before beginning a new test and cleared cache in the meanwhile. After performing this step, we end up with more than 1400 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, a 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. Once the instrumented code is deployed each test is re-run 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 execution traces extracted in previous step. The objective is to exclude irrelevant structural elements of code that were not exercised during test execution and considering metrics of only those parts of code which are exercised by a certain test case. We do this by eliminating irrelevant metrics from all the gathered metrics. In Appendix A we show the filtered CK and Martin metric values for each test case of all case studies including QKSMS,BeHe ExploreR, PDF Creator, Pijaret, QR Scanner, Libre Torrent and Simple Music Player.

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 which is independent variable and energy consumption which is the dependent variable. We employ *Spearman's* ρ and *Kendall-tau* correlation to determine the correlations that exist in our data. *Spearman* and *Kendall-tau* are well-known statistical tests used to determine the dependency of one variable on another and hence using them was a natural choice as we want to determine how energy values are related to metrics values. Both work 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 provided in the next section.

5. Case study results

In this section we present an analysis of results of the case studies used in our experimentation. Our case-studies are seven selected applications and we have collected the metrics values and energy measurements for different tests of these case studies. In order to examine the relationship between metrics and energy we employ *Spearman's* ρ*-rank* and *Kendall's* correlation coefficient. These tests are commonly used to examine if correlations exists among variables. Studies [19] and [6] similar to ours also used these tests and this confirms the correctness of choice of statistical test. We report the results of correlation between energy and metrics while addressing each of our research questions in detail.

RQ1:Can energy consumption of software be related to structural information of whole application?

Our first research question investigates the relationship between overall structure of software and its energy consumption. Answering this research question helps us to reason why energy consumption cannot be related to the overall structure of an application; and also allows us to establish our findings in context with previously conducted studies [5]. The structural information of software in our experiments is represented by three existing object-oriented metric suites and the mean distribution of each metric in these suites is shown in Figs. 4–6 (also see Appendix A for remaining Figures). We formulate 17 null hypothesis (one for each metric in the three suites) to answer RQ1. The null-hypothesis of each test is that ρ or τ is zero which means that no correlation exists between energy and a metric under study. The null-hypotheses is rejected if p-value is less than $\alpha = 0.05$.

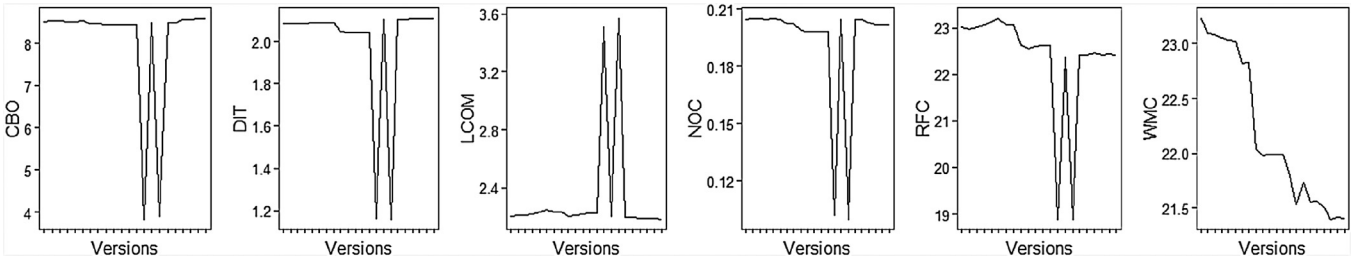

Fig. 4. Mean CK Metrics distribution for twenty-two versions of QKSMS case study. The X axes represent the case study version. The Y axes represent the corresponding metric value for each version calculated using MetricsReloaded.

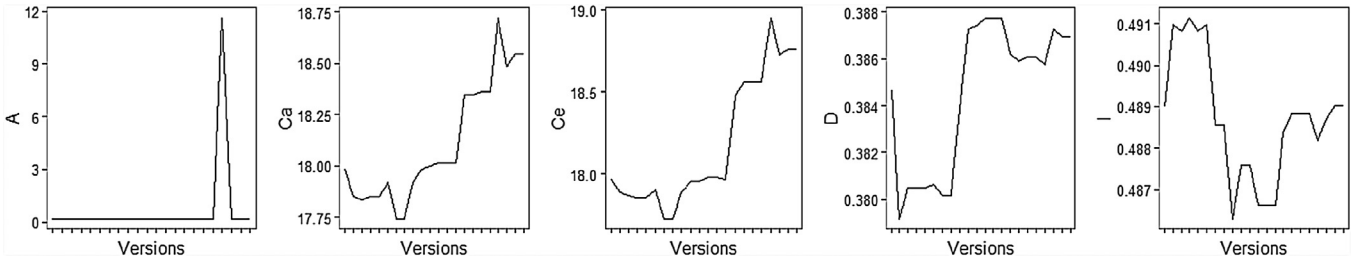

Fig. 5. Mean Martin Metrics distribution for twenty-two versions of QKSMS case study. The X axes represent the case study version. The Y axes represent the corresponding metric value for each version calculated using MetricsReloaded.

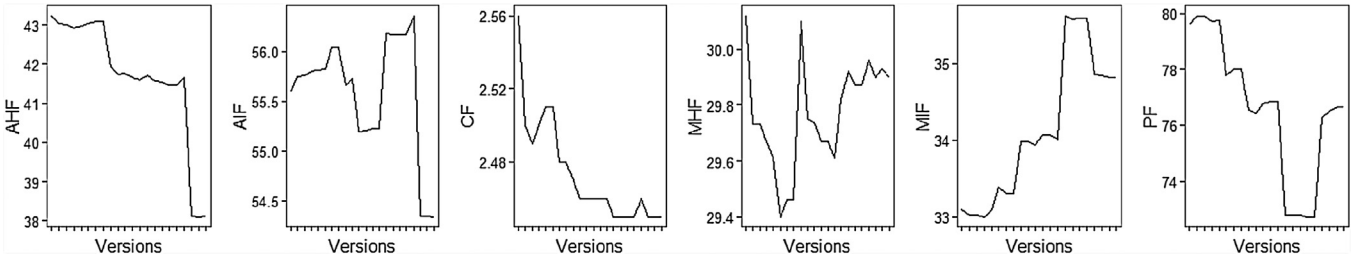

Fig. 6. Mean MOOD Metrics distribution for twenty-two versions of QKSMS case study, The X axes represent the case study version. The Y axes represent the corresponding metric value for each version calculated using MetricsReloaded.

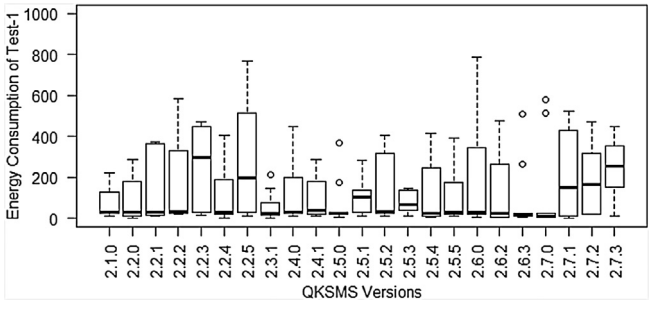

Fig. 7. Energy consumption of idle screen test - QKSMS case study. The X axes represent versions of application and Y axes represent corresponding energy consumption value of 10 runs of test.

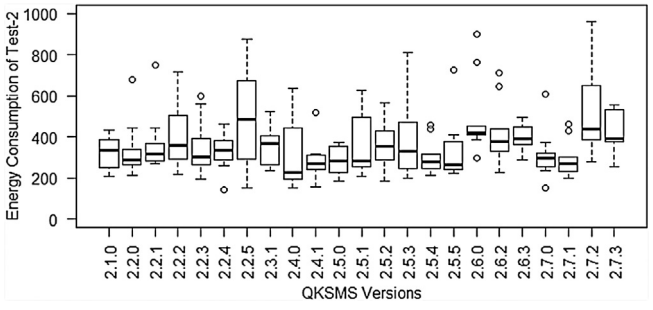

Fig. 8. Energy consumption of sending 5 character text message - QKSMS case study. The X axes represent versions of application and Y axes represent corresponding energy consumption value of 10 runs of test.

We analyzed the collected data and found that the energy values of tests have little variability with mean centered in different ranges for each case study(300 to 400mJ for QKSMS and 4500mJ for BeHeExploRer) as shown in Figs. 7–11 and Fig. 12 respectively. The mean energy consumption changes across versions but there is no clear trend, see Figs. 7–14 . 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 metrics values from the seven case-studies and relate them with energy consumption of tests to compute the *Spearman's ρ-rank* correlation. The third column of Table 3 shows the results of applying *Spearman's ρ-rank* and *Kendall-tau* to the overall metrics of the seven applications combined. After p-value correction through Bonferroni method we found that half of CK and Martin metrics of overall application correlate with energy consumption values. However, we argue that these correlations are not only small in magnitude ($[-0.5, +0.5]$) but are probably also inaccurate because the metrics may not be covering the entire code with uniformity.

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 all the case studies, one value of each MOOD metric is generated as shown in Fig. 6.

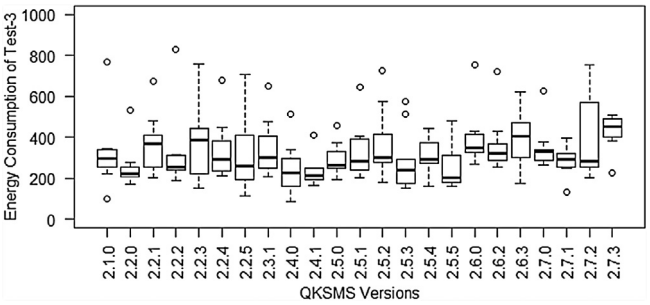

Fig. 9. Energy consumption of sending 200 character text message - QKSMS case study. The X axes represent versions of application and Y axes represent corresponding energy consumption value of 10 runs of test.

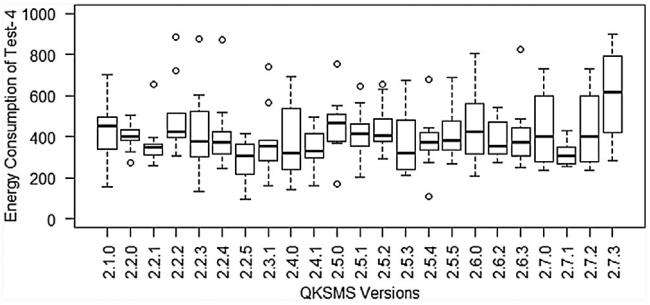

Fig. 10. Energy consumption of sending multimedia message - QKSMS case study. The X axes represent versions of application and Y axes represent corresponding energy consumption value of 10 runs of test.

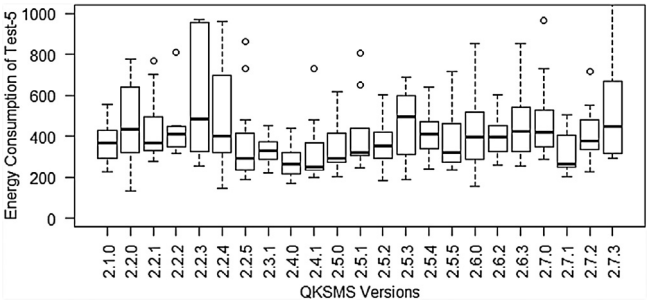

Fig. 11. Energy consumption of sending 5 character text message to 5 contacts - QKSMS case study. The X axes represent versions of application and Y axes represent corresponding energy consumption values of 10 runs of test.

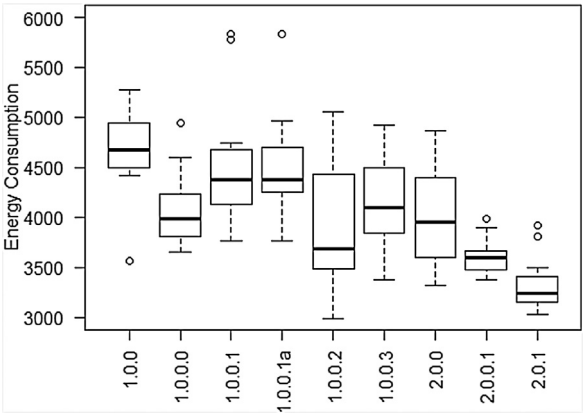

Fig. 12. Energy consumption of browsing test - BeHe ExploeR case study. The X axes represent versions of application and Y axes represent corresponding energy consumption values of 10 runs of test.

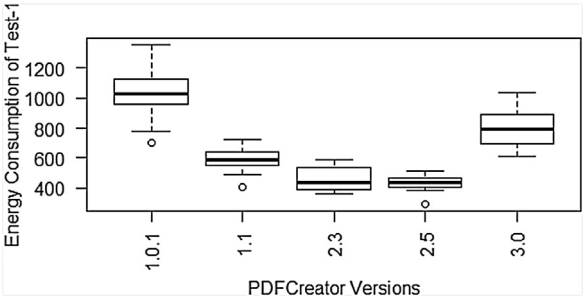

Fig. 13. Energy consumption of creating pdf from image - PDF Creator case study. The X axes represent versions of application and Y axes represent corresponding energy consumption values of 10 runs of test.

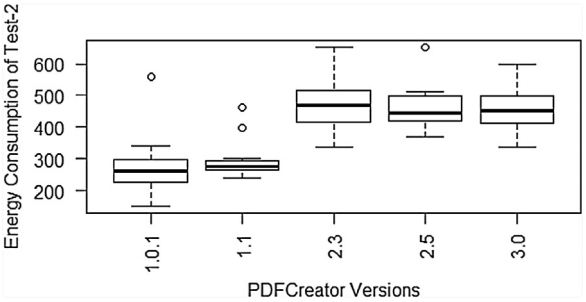

Fig. 14. Energy consumption of creating pdf from text - PDF Creator case study. The X axes represent versions of application and Y axes represent corresponding energy consumption values of 10 runs of test.

Table 3
Spearman-rho and Kendall-tau Correlation for CK, Martin and MOOD Metrics of overall application structure.

		Spearman-rho		Kendall-tau	
		ρ	p-val	tau	p-val
CK	CBO	−0.4837624	**0.0004291**	−0.2997141	**0.002599**
	DIT	0.5097778	**0.000183**	0.3615772	**0.0003141**
	LCOM	−0.3797873	0.007112	−0.2189863	0.02828
	NOC	0.119002	0.4154	0.1064794	0.2969
	RFC	−0.3933247	0.005176	−0.3337626	**0.0007472**
	WMC	−0.4969762	**0.0002807**	−0.4020516	**4.88E−05**
Martin	A	−0.4461062	**0.001315**	−0.3313156	**0.001182**
	Ca	−0.5068124	**0.0002024**	−0.295516	**0.003345**
	Ce	−0.3472066	0.01451	−0.219178	0.02827
	D	0.4249399	**0.002339**	0.2887165	**0.003914**
	I	0.3862846	0.006116	0.2911819	0.2911819
MOOD	AHF	−0.06159304	0.4649	−0.05905772	0.3026
	AIF	0.3646337	**7.56E−06**	0.2433295	**2.30E−05**
	CF	0.1846911	0.02723	0.1207936	0.04174
	MHF	0.1529232	0.06825	0.1198706	0.0373
	MIF	−0.1887149	0.02399	−0.1220346	0.03397
	PF	0.2340898	**0.004894**	0.1541888	**0.007669**

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

The second research question aims to investigate whether any correlation between metrics and energy exists if we only consider the metric values relevant to test execution path. For answering this question we use the structural information of the execution trace that is collected using Algorithm 2 and classified as *filtered metrics*. We formulate one null hypothesis for each CK and Martin metric which assumes the non-existence of correlation between a metric and corresponding energy values. In total we had 11 hypothesis and to evaluate each hypothesis, *Spearman's* ρ*-rank* and *Kendall-tau* correlation is applied to the filtered CK and Martin metrics of all versions and their respective energy consumption values. The results of our analysis are reported in the Table 4.

Table 4
Spearman-rho and Kendall-tau Correlation for CK and Martin Metrics of Execution Based Structure

		Spearman-rho		Kendall-tau	
		ρ	p-val	tau	p-val
CK	CBO	−0.4464019	**2.30E−08**	−0.3152238	**2.48E−08**
	DIT	0.2169767	0.009242	0.1304124	0.0214
	LCOM	0.00939736	0.9113	0.05378986	0.3423
	NOC	−0.0137847	0.8702	0.01276904	0.8241
	RFC	−0.1306113	0.12	−0.1094513	0.0527
	WMC	−0.3148168	**0.0001284**	−0.2395228	**2.24E−05**
Martin	A	−0.279396	**0.0007266**	−0.1912342	**0.0007882**
	Ca	−0.4932322	**3.89E−10**	−0.3539978	**4.21E−10**
	Ce	−0.4084669	**4.10E−07**	−0.2885587	**3.57E−07**
	D	−0.1973148	0.01817	−0.148146	0.008967
	I	0.4574313	**9.28E−09**	0.3272197	**7.71E−09**

Interestingly enough, all the metrics in CK and Martin metric suite correlate to energy consumption with high level of statistical significance except *LCOM*, *NOC* and *RFC*, all of which are class level metrics. However after applying p-value correction for multiple hypotheses, DIT was also discarded because of insignificant correlation. A moderate correlation is observed between energy and CK's *CBO*(ρ = −0.7) as well as *WMC*(ρ = 0.7) even after p-value correction. Similarly, Ca(ρ = −0.6) and Ce(ρ = −0.5) in Martin's package suite also exhibit moderate correlations. We believe 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 and eliminate redundancies.

MOOD metrics are at the application granularity level and therefore independent of the execution trace and hence irrelevant for this question. However the results of MOOD metrics are provided in answer to RQ1.

RQ3:Which structural metrics significantly correlate with software energy consumption?

In answer to this question we reason about the metrics that display high correlations and those that do not correlate at all with energy consumption of software applications under study. The correlation coefficients denoted by ρ and τ are 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 or τ, 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→ *trivial*, 0.1 − 0.3→ *minor*, 0.3 − 0.5→ *moderate*, 0.5 − 0.7→ *large*, 0.7 − 0.9→ *very large*, and 0.9 − 1→ *almost perfect*) [37].

Although almost all of the metrics correlated with energy as shown in Table 4 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) and WMC(Weighted Method Complexity) have moderately negative correlation with energy (ρ = −0.4464019 and ρ = −0.3148168). This means that a decrease in CBO and WMC value negatively affects the energy of an application. In Martin's Package metrics Ca (Afferent couplings) is the number of other packages that depend upon 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 negative correlations with energy having ρ values of −0.4932322, −0.4084669 and −0.279396 respectively whereas I(Instability) has a positive correlation (ρ = 0.4574313).

RQ4:What are the possible reasons for the observed correlations?

The energy consumption is indirectly but moderately dependent on coupling between objects (CBO) because if coupling of a class decreases, it means that most of the work is done only by the class itself hence number of currently used variables will also increase due to increased functionality and current state will become difficult to maintain. All of this will cost high energy consumption during context switching due to complex state [7]. Number of children (NOC) of a class is indirectly dependent on energy consumption because lesser number of children means less coupling between objects, since CBO is indirectly related to energy NOC is also indirectly related to energy. However NOC does not show significant correlation with energy unlike CBO. Energy consumption also has direct dependence on Depth of Inheritance (DIT) because if depth of inheritance of a class increases, the object of the lowest class will take a lot of memory because it will have to instantiate all the variables of classes above it and due to high memory usage it will consume high energy when accessed [7].

Abstractness (A) is a package level metric and affects energy consumption of the application significantly. It has an inverse effect on energy, when value of a package decreases, package becomes more concrete and changes related to package will have a large impact on the project. Instability (I) directly affects energy consumption of the application and has a moderate effect on the energy. If the value of instability increases this means that changes in package will have a more effect on the application. Distance (D) has direct effect on energy consumption of the application because when the value of distance increases it will mean that the package has become unbalanced for the application and changes in package will have a high impact on the application. Attribute inheritance factor (AIF) directly has a minor effect on energy consumption of an application, because increase in AIF means that the inheritance between the classes in the project has increased and as we have discussed earlier inheritance directly effects energy, so increase in AIF will also increase the energy consumption. We only consider the aforementioned metrics while building estimation models because of their significant correlations with energy. The metrics that do not correlate significantly may lead to misleading estimates and are hence dropped from the remaining discussion.

5.1. 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 TC4 QKSMS's v2.5.2 with v2.7.3 in Table 5 and BeHe ExploreR's v1.0.0.2 with v2.0.1 in Table 6 . In these figures, only those metrics are included which show a correlation with energy in Table 4 and the aforementioned versions were chosen because they are distant in time. ρ column shows the magnitude of correlation between certain metrics and energy i.e., how much the energy is affected by a one-unit change in the corresponding metric. It is 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 5 clearly shows that majority metrics predict that overall energy will increase in v2.7.3 of QKSMS which can be confirmed from Fig. 10 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 6 predict an overall decrease in energy of BeHe ExploreR's later version. Fig. 12 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

Table 5
Correlation Results providing Energy Insights between Two Versions of QKSMS for Sending Multimedia Message

QKSMS 2.5.2	
Metric	**Value**
CBO	16.230
WMC	37.615
A	0.136
Ca	29.696
Ce	28.956
I	0.485
AIF	0.552
PF	0.768

QKSMS 2.7.3 (Energy Insights)

Metric	Value	Energy	Impact	ρ	Change(%)
CBO	14.906	⇑	-ive	V Large	−8.16%
WMC	39.145	⇓	-ive	Moderate	+4.06%
A	0.150	⇓	-ive	Large	+10.81%
Ca	25.097	⇑	-ive	Large	−15.48%
Ce	26.258	⇑	-ive	Large	−9.32%
I	0.506	⇑	+ive	Moderate	+4.47%
AIF	0.556	⇑	+ive	Minor	+0.74%
PF	0.796	⇑	+ive	Minor	+3.66%
Overall Energy		**Increase**			

Table 6
Correlation Results providing Energy Insights between Two Versions of BEHE Explorer for Browsing Test

QKSMS 2.5.2	
Metric	**Value**
CBO	6.5
WMC	15.083
A	0
Ca	6
Ce	8
I	0.58
AIF	0.936
PF	1

QKSMS 2.7.3 (Energy Insights)

Metric	Value	Energy	Impact	ρ	Change(%)
CBO	5.5	⇑	-ive	V Large	−15.38%
WMC	22.833	⇓	-ive	Moderate	+51.38%
A	0.023	⇓	-ive	Large	+0.02%
Ca	12.333	⇓	-ive	Large	+105.56%
Ce	16	⇓	-ive	Large	+100%
I	0.7	⇑	+ive	Moderate	+20.69%
AIF	0.818	⇓	+ive	Minor	−12.61%
PF	2.190	⇑	+ive	Minor	+119%
Overall Energy		**Decrease**			

through static analysis and by using our approach developers can gain energy insights about their application structure during development. For instance, a developer will be able to gain insight about increase in energy consumption of QKSMS version 2.7.3 based on the current and previous version's structural information.

We acknowledge the fact that our energy estimation models may not provide realistic estimates for projects of all types and sizes. To obtain more precise and accurate estimates we need to build better energy estimation models since a single model may not suffice. An estimation model built from a large set of diverse projects using multiple metrics would provide more accurate insights.

6. Threats to validity

There are four levels of validity threats that we discuss following common guidelines provided in [38].

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 PowerTutor 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 contexts 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 called *Spearman's ρ-rank* for determining correlations. Spearman test is used to examine the relationship between two variable and has also been previously used by researchers for similar studies [5]. The conclusion validity however is affected by the fact that we did not take care of the energy spent in APIs. There are two ways in which API energy consumption can be accounted for. One is to include the API code and the respective metrics in methodology and the other is to filter the API calls and their respective energy. Both tasks are challenging and are not currently being handled by the methodology.

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. To measure this effect we re-ran some of the experiments using the Green Mining methodology [5] and used well-known tools, MetricReloaded and PowerTutor.

External validity External validity affects our ability to generalize the results of our experiments beyond the case study. We performed more than 1400 execution to test 12 different scenarios on 63 versions of seven 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 medium complexity because we believe that none of our case studies was highly complex. 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.

Reliability validity The theory behind experiment is sufficiently clear and hypotheses are well defined however one general threat is low number of samples which may reduce ability to reveal true patterns in data. To counteract threat to reliability of experiment we ran each test multiple times. To ensure that our experiment can be replicated we chose applications that are publicly available on repositories. We also attempt to provide all necessary details to replicate our study [5] and hence we hope the reliability of study is reasonable.

[5] https://github.com/Hareem-E-Sahar/Replication-Package-

7. Conclusion

The limited battery life of mobile phones has led to the increased demand for developing energy efficient android applications. Developers need energy estimation models that estimate the change in energy consumption of software with change in software structure. We present a novel methodology for relating software change to software energy consumption on the basis of execution structure. The proposed methodology can be used to perform experiments for developing energy estimation models for smart phone applications. We apply our 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 of Android applications. However the methodology presented in this paper can be used to conduct similar investigations on case studies of varying levels of complexity.

Appendix A. Test Case Execution of QKSMS

Figs. A1 and A2

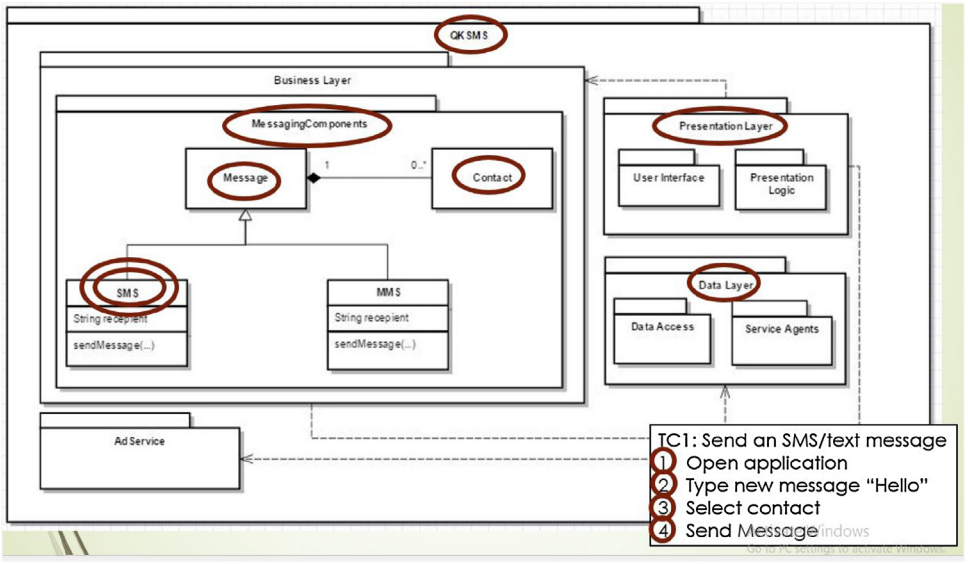

Fig. A1. Test Case Execution Structure of QKSMS Scenario - Sending an SMS Text Message. The test activates components in Presentation Layer, Data Layer and Business Layer.

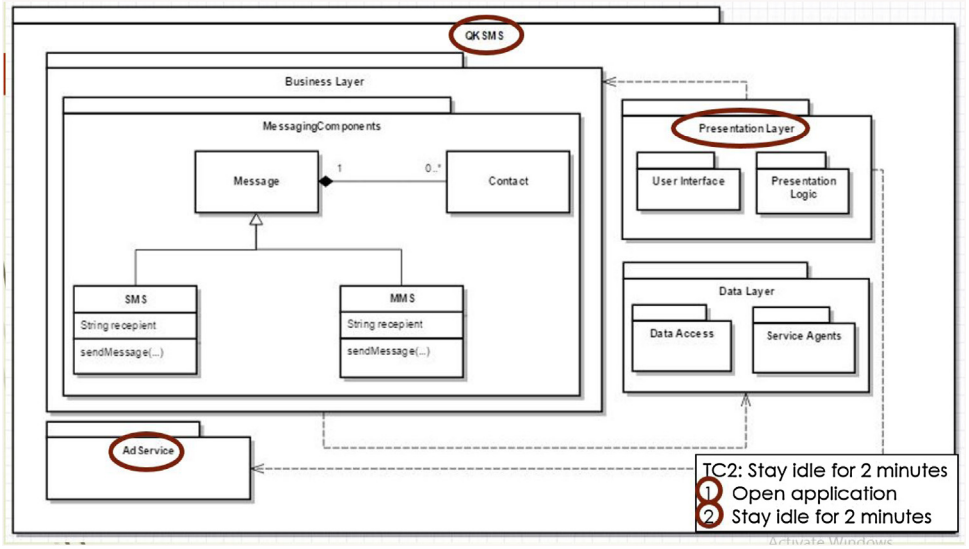

Fig. A2. Test Case Execution Structure of QKSMS Scenario - Staying Idle for 2 minutes. The test activates only Presentation Layer components and Advertisement Services.

Appendix B. Variation of metrics for the case studies

Figs. B1–B31

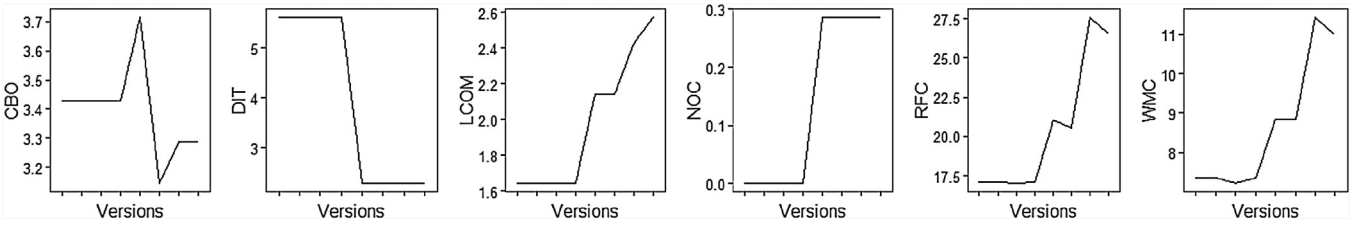

Fig. B1. Mean CK Metrics distribution for eight versions of BeHe Explorer case study. The X axes represent the case study version. The Y axes represent the corresponding metric value for each version calculated using MetricsReloaded.

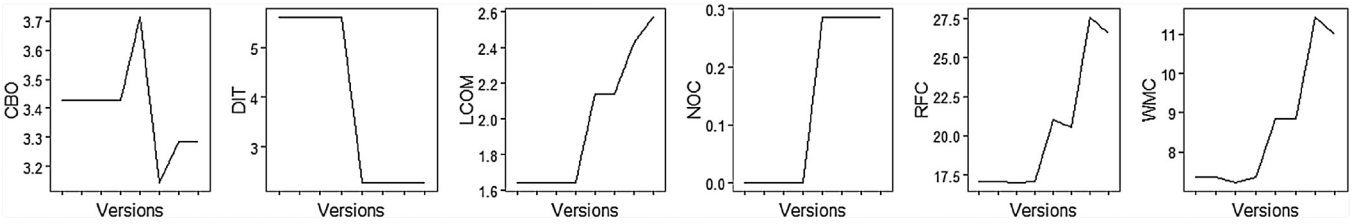

Fig. B2. Mean CK Metrics distribution for eight versions of BeHe Explorer case study. The X axes represent the case study version. The Y axes represent the corresponding metric value for each version calculated using MetricsReloaded.

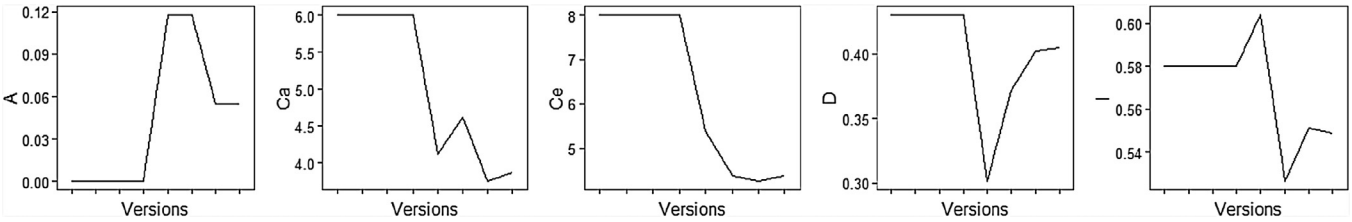

Fig. B3. Mean Martin Metrics distribution for eight versions of BeHe Explorer case study. The X axes represent the case study version. The Y axes represent the corresponding metric value for each version calculated using MetricsReloaded.

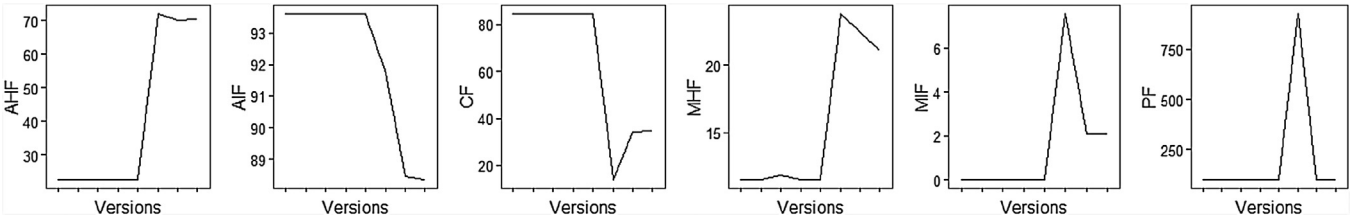

Fig. B4. Mean MOOD Metrics distribution for eight versions of BeHe ExploreR case study. The X axes represent the case study version. The Y axes represent the corresponding metric value for each version calculated using MetricsReloaded.

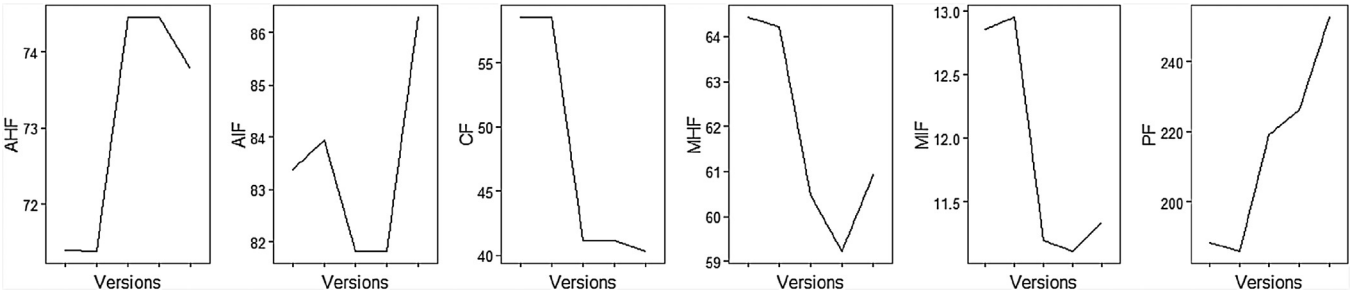

Fig. B5. Mean MOOD Metrics distribution for five versions of PDF Creator case study. The X axes represent the case study version. The Y axes represent the corresponding metric value for each version calculated using MetricsReloaded.

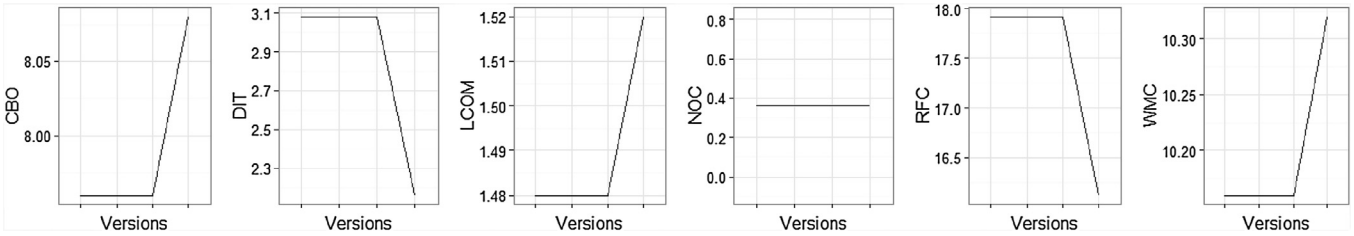

Fig. B6. Mean CK Metrics distribution for four versions of QR Scanner case study. The X axes represent the case study version. The Y axes represent the corresponding metric value for each version calculated using MetricsReloaded.

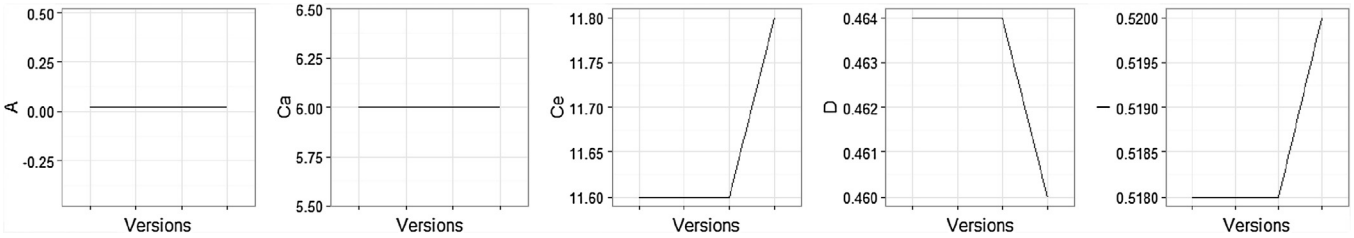

Fig. B7. Mean Martin Metrics distribution for four versions of QR Scanner case study. The X axes represent the case study version. The Y axes represent the corresponding metric value for each version calculated using MetricsReloaded.

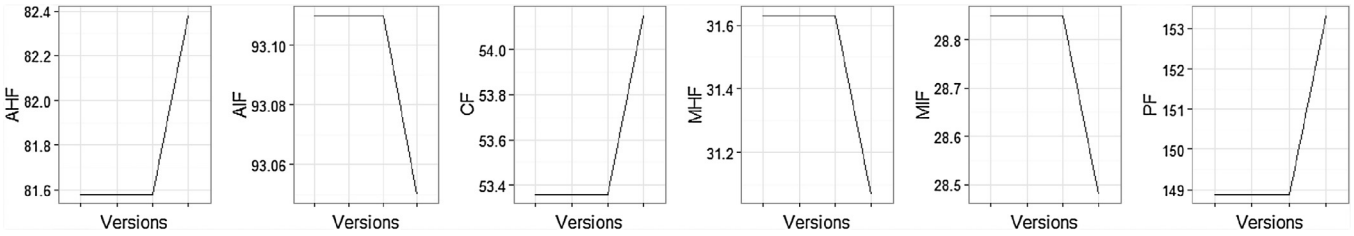

Fig. B8. Mean MOOD Metrics distribution for four versions of QR Scanner case study. The X axes represent the case study version. The Y axes represent the corresponding metric value for each version calculated using MetricsReloaded.

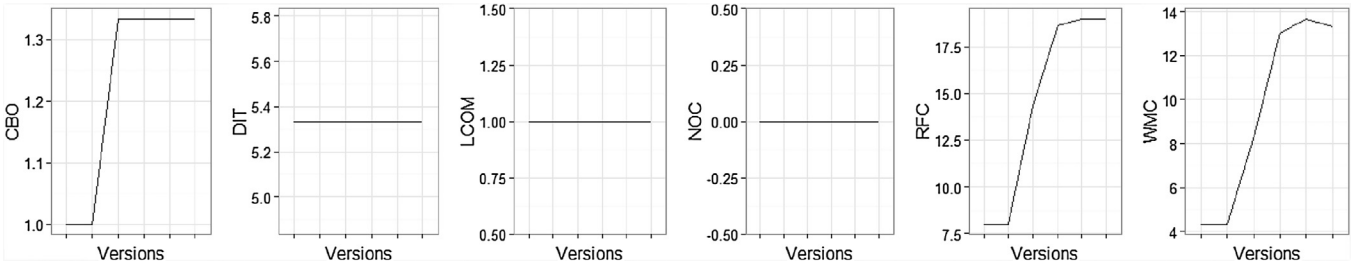

Fig. B9. Mean CK Metrics distribution for six versions of Pijaret case study. The X axes represent the case study version. The Y axes represent the corresponding metric value for each version calculated using MetricsReloaded.

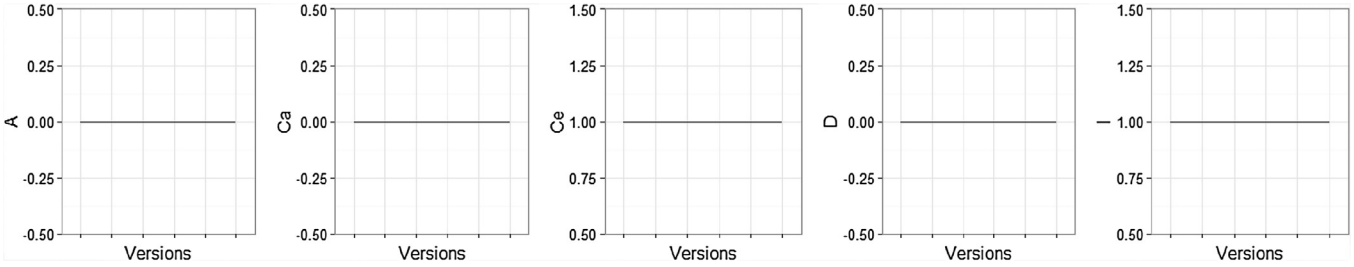

Fig. B10. Mean Martin Metrics distribution for six versions of Pijaret case study. The X axes represent the case study version. The Y axes represent the corresponding metric value for each version calculated using MetricsReloaded.

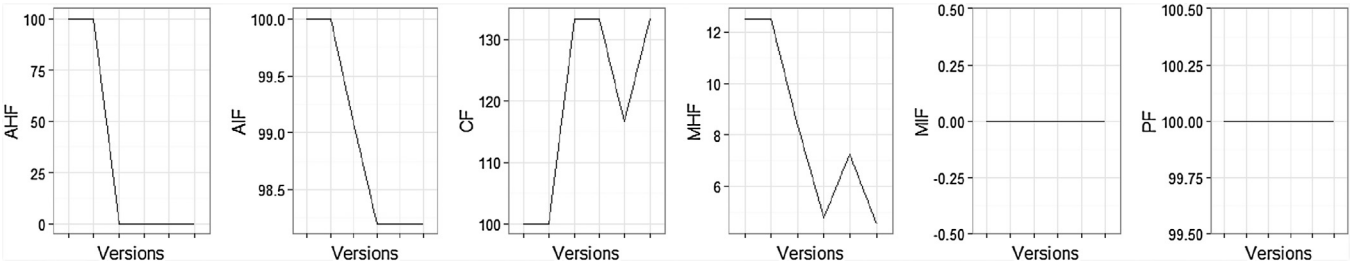

Fig. B11. Mean MOOD Metrics distribution for six versions of Pijaret case study. The X axes represent the case study version. The Y axes represent the corresponding metric value for each version calculated using MetricsReloaded.

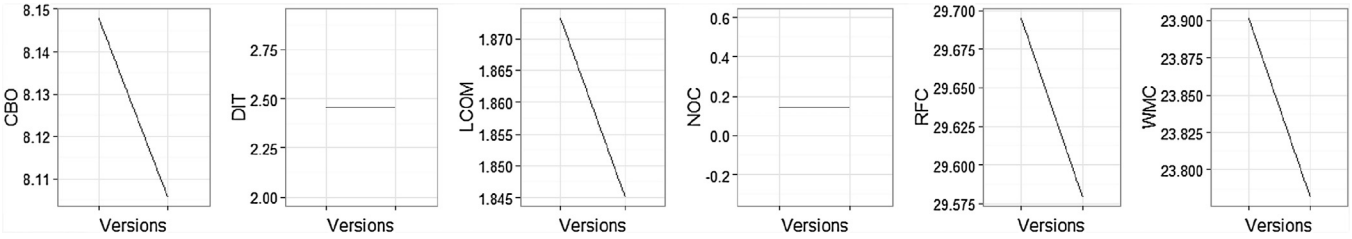

Fig. B12. Mean CK Metrics distribution for two versions of Libre Torrent case study. The X axes represent the case study version. The Y axes represent the corresponding metric value for each version calculated using MetricsReloaded.

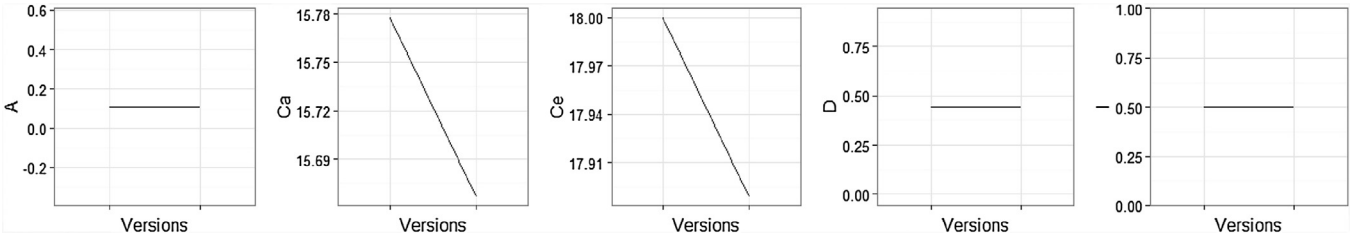

Fig. B13. Mean Martin Metrics distribution for two versions of Libre Torrent case study. The X axes represent the case study version. The Y axes represent the corresponding metric value for each version calculated using MetricsReloaded.

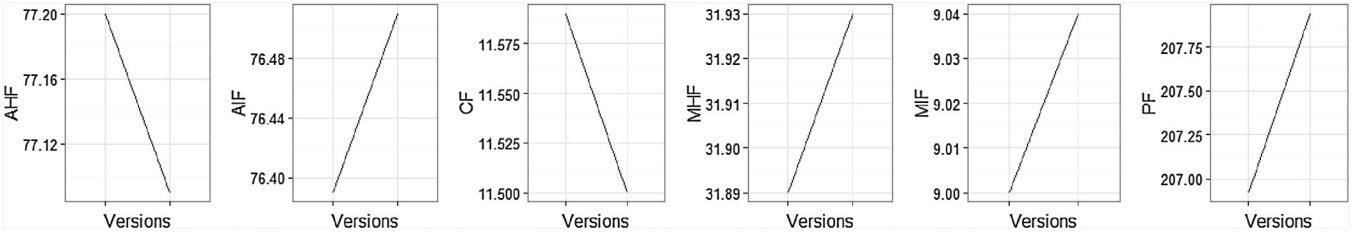

Fig. B14. Mean MOOD Metrics distribution for two versions of Libre Torrent case study. The X axes represent the case study version. The Y axes represent the corresponding metric value for each version calculated using MetricsReloaded.

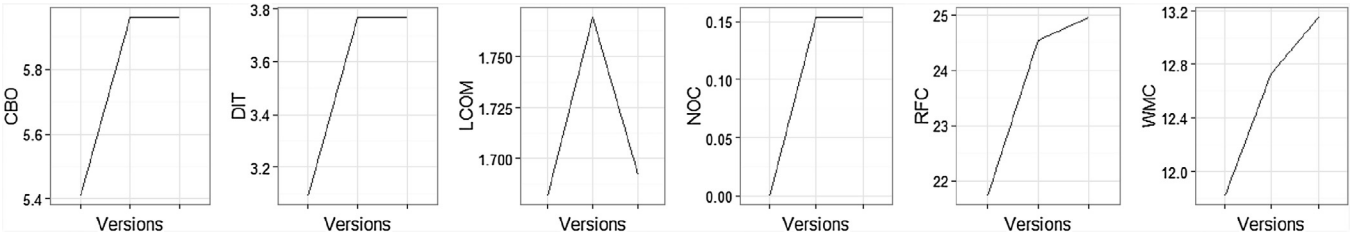

Fig. B15. Mean CK Metrics distribution for three versions of Simple Music Player case study. The X axes represent the case study version. The Y axes represent the corresponding metric value for each version calculated using MetricsReloaded.

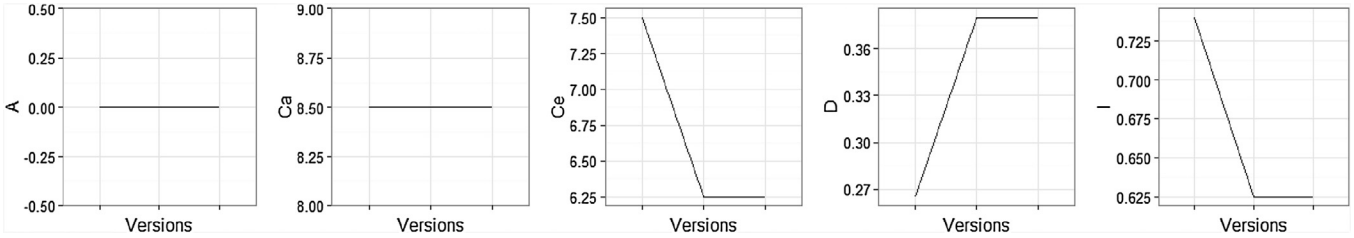

Fig. B16. Mean Martin Metrics distribution for three versions of Simple Music Player case study. The X axes represent the case study version. The Y axes represent the corresponding metric value for each version calculated using MetricsReloaded.

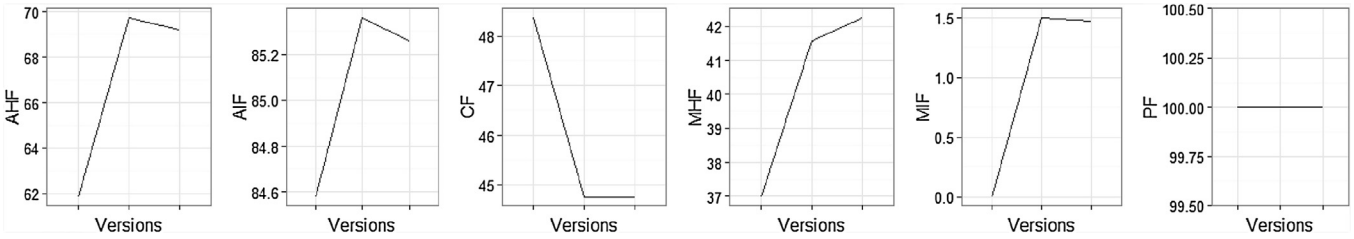

Fig. B17. Mean MOOD Metrics distribution for three versions of Simple Music Player case study. The X axes represent the case study version. The Y axes represent the corresponding metric value for each version calculated using MetricsReloaded.

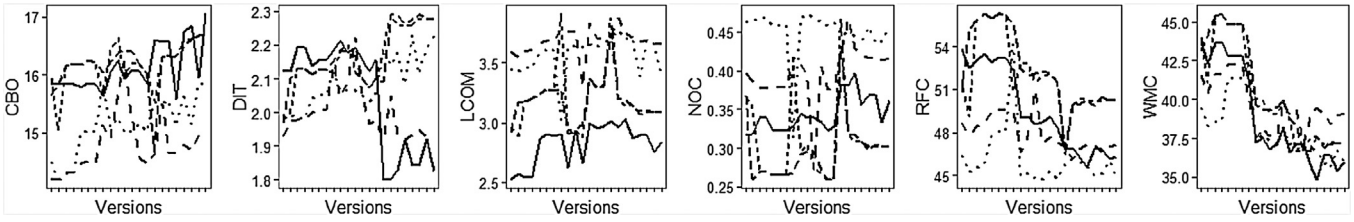

Fig. B18. Mean Filtered CK Metrics distribution of five test cases for 22 versions of QKSMS case study on the basis of execution structure. The X axes represent case study versions and Y axes represent corresponding metric value for each version after filtering based on execution structure.

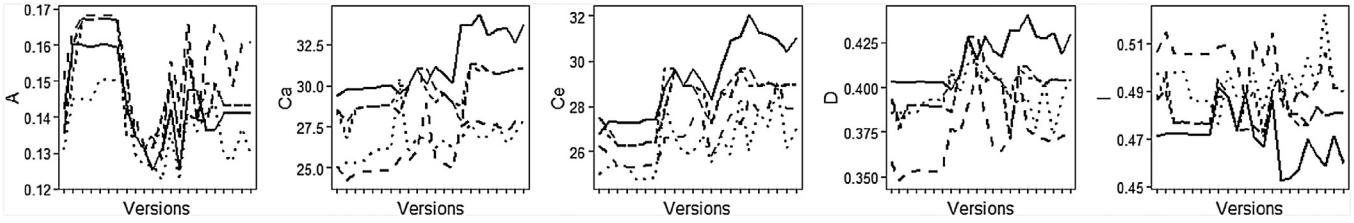

Fig. B19. Mean Filtered Martin Metrics of five test cases for 22 versions of QKSMS case study on the basis of execution structure. The X axes represent case study versions and Y axes represent corresponding metric value for each version after filtering based on execution structure.

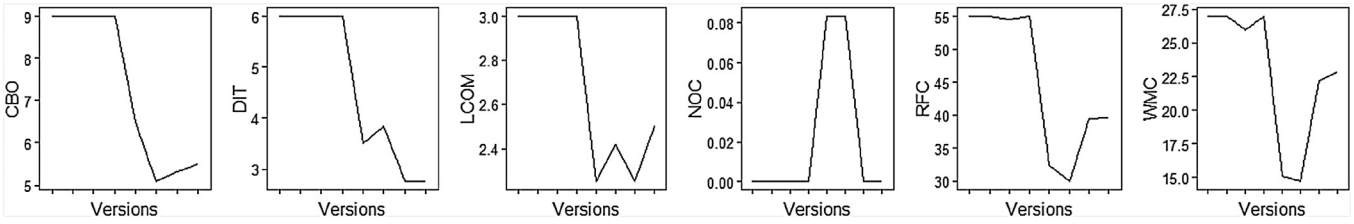

Fig. B20. Mean Filtered CK Metrics for eight versions of BeHe ExploreR case study on the basis of execution structure. The X axes represent case study versions and Y axes represent corresponding metric value for each version after filtering based on execution structure.

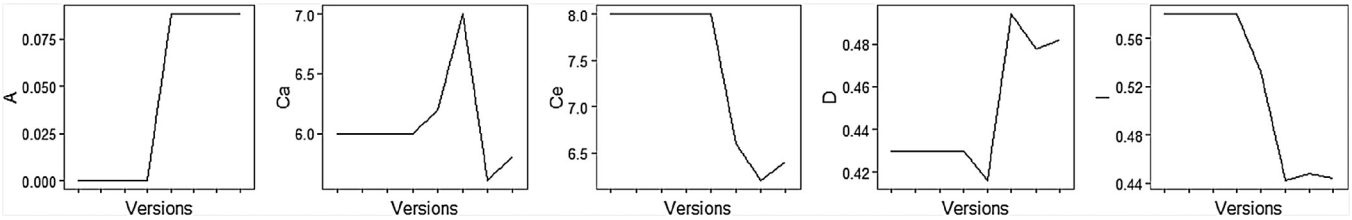

Fig. B21. Mean Filtered Martin Metrics distribution for eight versions of BeHe ExploreR case study on the basis of execution structure. The X axes represent case study versions and Y axes represent corresponding metric value for each version after filtering based on execution structure.

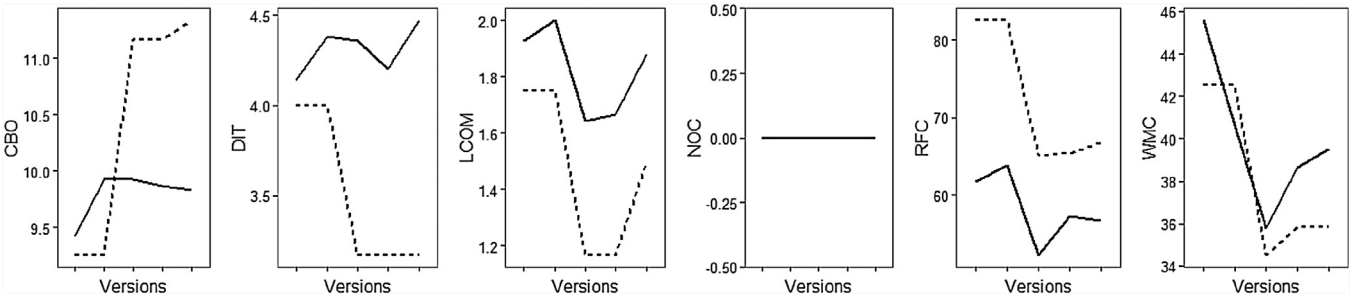

Fig. B22. Mean Filtered CK Metrics distribution of two test cases for five versions of PDF Creator case study on the basis of execution structure. The X axes represent case study versions and Y axes represent corresponding metric value for each version after filtering based on execution structure.

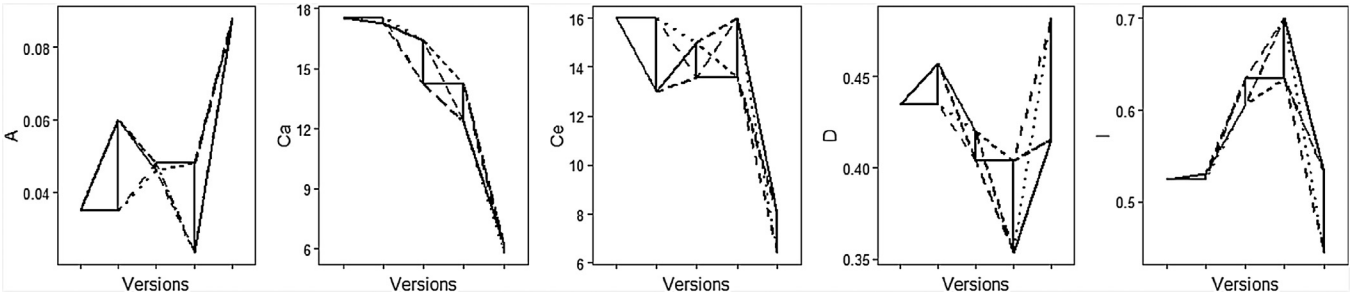

Fig. B23. Mean Filtered Martin Metrics distribution of two test cases for five versions of PDF Creator case study on the basis of execution structure. The X axes represent case study versions and Y axes represent corresponding metric value for each version after filtering based on execution structure.

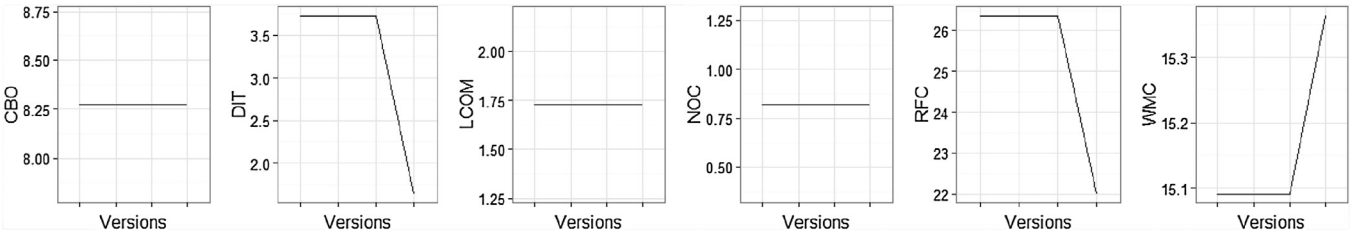

Fig. B24. Mean Filtered CK Metrics distribution of test case for four versions of QR Scanner case study on the basis of execution structure. The X axes represent case study versions and Y axes represent corresponding metric value for each version after filtering based on execution structure.

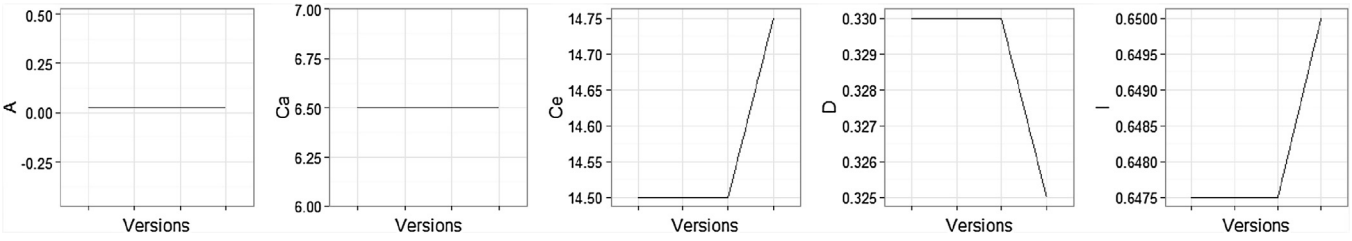

Fig. B25. Mean Filtered Martin Metrics of test case for four versions of QR Scanner case study on the basis of execution structure. The X axes represent case study versions and Y axes represent corresponding metric value for each version after filtering based on execution structure.

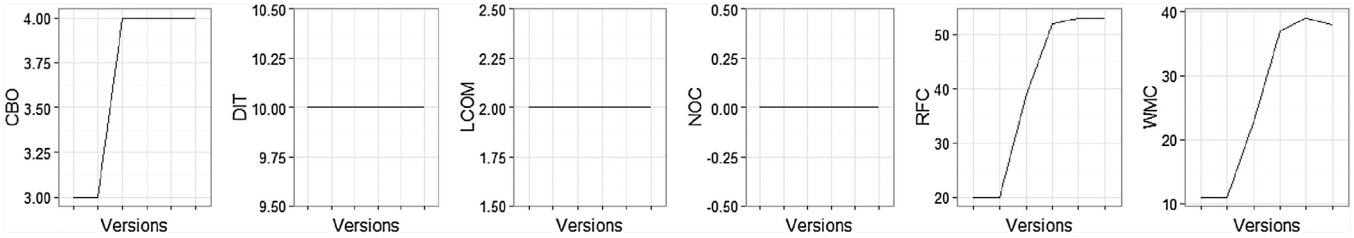

Fig. B26. Mean Filtered CK Metrics distribution of test case for six versions of Pijaret case study on the basis of execution structure. The X axes represent case study versions and Y axes represent corresponding metric value for each version after filtering based on execution structure.

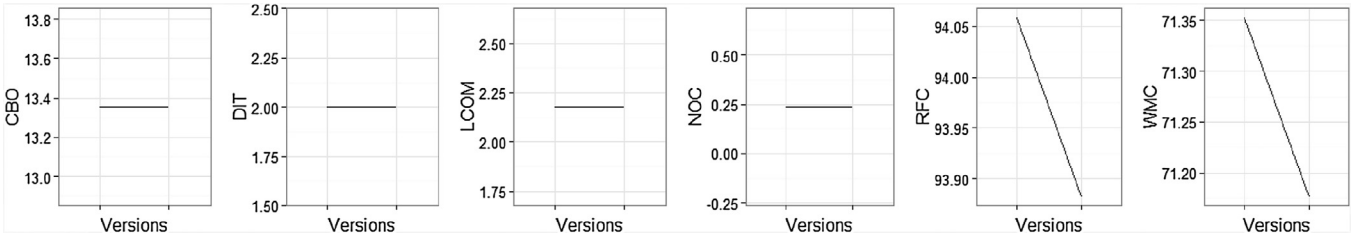

Fig. B27. Mean Filtered Martin Metrics of test case for six versions of Pijaret case study on the basis of execution structure. The X axes represent case study versions and Y axes represent corresponding metric value for each version after filtering based on execution structure.

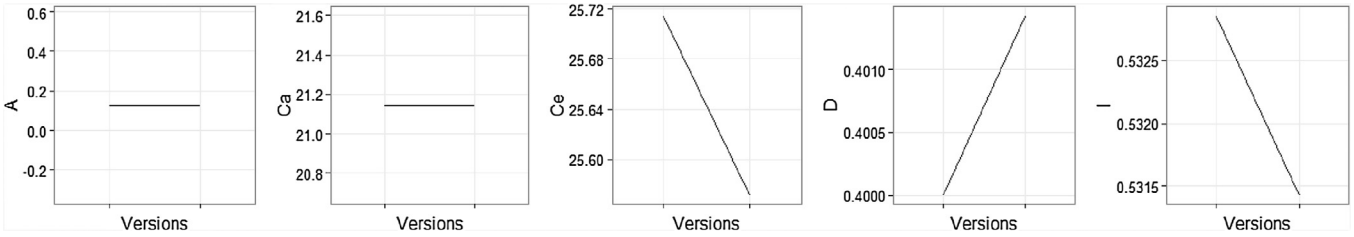

Fig. B28. Mean Filtered CK Metrics distribution of test case for two versions of Libre Torrent case study on the basis of execution structure. The X axes represent case study versions and Y axes represent corresponding metric value for each version after filtering based on execution structure.

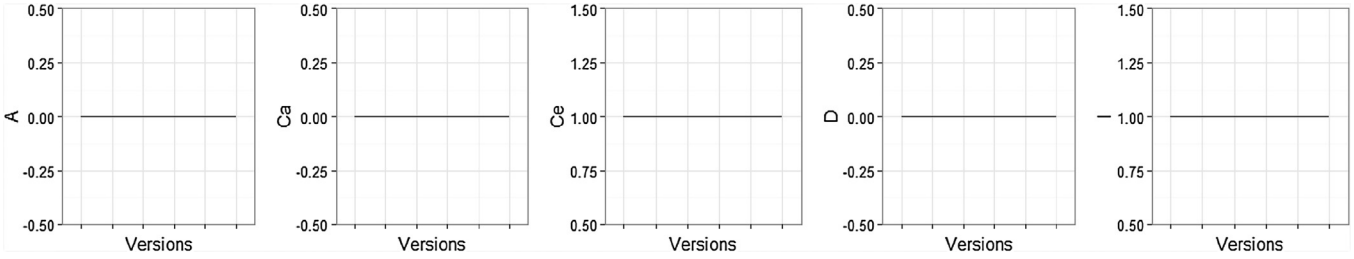

Fig. B29. Mean Filtered Martin Metrics of test case for two versions of Libre Torrent case study on the basis of execution structure. The X axes represent case study versions and Y axes represent corresponding metric value for each version after filtering.

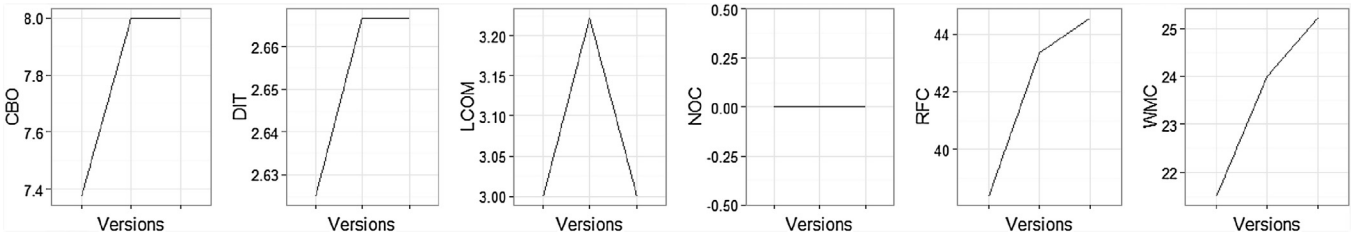

Fig. B30. Mean Filtered CK Metrics distribution of test case for three versions of Simple Music Player case study on the basis of execution structure. The X axes represent case study versions and Y axes represent corresponding metric value for each version after filtering based on execution structure.

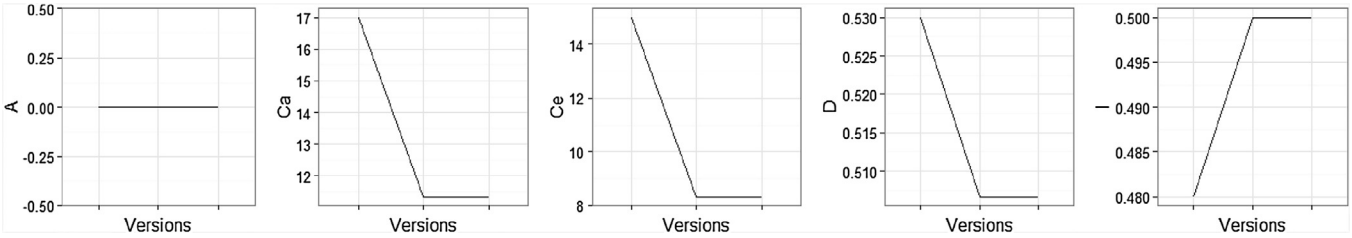

Fig. B31. Mean Filtered Martin Metrics of test case for three versions of Simple Music Player case study on the basis of execution structure. The X axes represent case study versions and Y axes represent corresponding metric value for each version after filtering based on execution structure.

References

[1] C. Pang, A. Hindle, B. Adams, A.E. Hassan, What do programmers know about software energy consumption? IEEE Softw. 33 (3) (2016) 83–89.
[2] H. Khalid, E. Shihab, M. Nagappan, A.E. Hassan, What do mobile app users complain about? IEEE Softw. 32 (3) (2015) 70–77, doi:10.1109/MS.2014.50.
[3] D. Li, S. Hao, J. Gui, W.G. Halfond, An empirical study of the energy consumption of android applications, in: in: Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on, IEEE, 2014, pp. 121–130.
[4] G. Pinto, F. Castor, Energy efficiency: a new concern for application software developers, CACM 60 (12) (2017) 68–75.
[5] A. Hindle, Green mining: a methodology of relating software change and configuration to power consumption, Empirical Softw. Eng. 20 (2) (2015) 374–409.
[6] A. Hindle, Abram, Green mining: A methodology of relating software change to power consumption, in: in: Proceedings of the 9th IEEE Working Conference on Mining Software Repositories, IEEE Press, 2012, pp. 78–87.
[7] S.R. Chidamber, C.F. Kemerer, A metrics suite for object oriented design, IEEE Transactions on Softw. Eng. 20 (6) (1994) 476–493.
[8] N. Nagappan, T. Ball, Use of relative code churn measures to predict system defect density, in: in: Software Engineering, 2005. ICSE 2005, Proceedings. 27th International Conference on, IEEE, 2005, pp. 284–292.
[9] F.B. e Abreu, W. Melo, Evaluating the impact of object-oriented design on software quality, in: in: Software Metrics Symposium, 1996, Proceedings of the 3rd International, IEEE, 1996, pp. 90–99.
[10] R.C. Martin, Agile software development: principles, patterns, and practices, Prentice Hall PTR, 2003.
[11] A.A. Bangash, H. Sahar, M.O. Beg, A methodology for relating software structure with energy consumption, in: in: Source Code Analysis and Manipulation (SCAM), 2017 IEEE 17th International Working Conference on, IEEE, 2017, pp. 111–120.
[12] A. Hindle, Green software engineering: the curse of methodology, in: in: IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER), Vol. 5, IEEE, 2016, pp. 46–55.
[13] C. Gustavo, Pinto, Energy efficiency:a new concern for application software developers, 2017.
[14] A. Pathak, Y.C. Hu, M. Zhang, P. Bahl, Y.-M. Wang, Fine-grained power modeling for smartphones using system call tracing, in: in: Proceedings of the sixth conference on Computer systems, ACM, 2011, pp. 153–168.
[15] A. Pathak, Y.C. Hu, M. Zhang, Where is the energy spent inside my app?: fine grained energy accounting on smartphones with eprof, in: in: Proceedings of the 7th ACM european conference on Computer Systems, ACM, 2012, pp. 29–42.
[16] I. Manotas, L. Pollock, J. Clause, Seeds: a software engineer's energy-optimization decision support framework, in: in: Proceedings of the 36th International Conference on Software Engineering, ACM, 2014, pp. 503–514.
[17] A. Bakker, Comparing energy profilers for android, in: in: 21st Twente Student Conference on IT, Vol. 21, 2014.
[18] G. Denaro, M. Pezzè, An empirical evaluation of fault-proneness models, in: in: Proceedings of the 24th International Conference on Software Engineering, ACM, 2002, pp. 241–251.
[19] H.M. Olague, L.H. Etzkorn, S. Gholston, 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 Trans. Softw. Eng. 33 (6) (2007) 402–419.
[20] V.R. Basili, L.C. Briand, W.L. Melo, A validation of object-oriented design metrics as quality indicators, IEEE Trans. Softw. Eng. 22 (10) (1996) 751–761.
[21] L.C. Briand, J. Wüst, J.W. Daly, D.V. Porter, Exploring the relationships between design measures and software quality in object-oriented systems, J. Syst. Softw. 51 (3) (2000) 245–273.
[22] L.C. Briand, J. Wüst, Empirical studies of quality models in object-oriented systems, Adv. Comput. 56 (2002) 97–166.
[23] T. Gyimothy, R. Ferenc, I. Siket, Empirical validation of object-oriented metrics on open source software for fault prediction, IEEE Trans. Softw. Eng. 31 (10) (2005) 897–910.
[24] F. Fioravanti, P. Nesi, A study on fault-proneness detection of object-oriented systems, in: in: Software Maintenance and Reengineering, 2001. Fifth European Conference on, IEEE, 2001, pp. 121–130.
[25] P. Yu, T. Systa, H. Muller, Predicting fault-proneness using oo metrics. an industrial case study, in: in: Software Maintenance and Reengineering, 2002, Proceedings. Sixth European Conference on, IEEE, 2002, pp. 99–107.
[26] A. Hindle, A. Wilson, K. Rasmussen, E.J. Barlow, J.C. Campbell, S. Romansky, Greenminer: A hardware based mining software repositories software energy consumption framework, in: in: Proceedings of the 11th Working Conference on Mining Software Repositories, ACM, 2014, pp. 12–21.
[27] K. Aggarwal, C. Zhang, J.C. Campbell, A. Hindle, E. Stroulia, The power of system call traces: Predicting the software energy consumption impact of changes, in: in: Proceedings of 24th Annual International Conference on Computer Science and Software Engineering, IBM Corp, 2014, pp. 219–233.
[28] K. Aggarwal, A. Hindle, E. Stroulia, Greenadvisor: A tool for analyzing the impact of software evolution on energy consumption, in: in: Software Maintenance and Evolution (ICSME), IEEE International Conference on, IEEE, 2015, pp. 311–320.
[29] W. Oliveira, R. Oliveira, F. Castor, A study on the energy consumption of android app development approaches, in: in: Proceedings of the 14th International Conference on Mining Software Repositories, IEEE Press, 2017, pp. 42–52.
[30] S. Hasan, Z. King, M. Hafiz, M. Sayagh, B. Adams, A. Hindle, Energy profiles of java collections classes, in: in: Proceedings of the 38th International Conference on Software Engineering, ACM, 2016, pp. 225–236.
[31] G. Pinto, K. Liu, F. Castor, Y. D. Liu, A comprehensive study on the energy efficiency of java thread-safe collections, Journal of Systems and Software.
[32] L.G. Lima, F. Soares-Neto, P. Lieuthier, F. Castor, G. Melfe, J.P. Fernandes, Haskell in green land: Analyzing the energy behavior of a purely functional language, in: in: Software Analysis, Evolution, and Reengineering (SANER), 2016 IEEE 23rd International Conference on, Vol. 1, IEEE, 2016, pp. 517–528.
[33] G. Pinto, F. Castor, Y.D. Liu, Understanding energy behaviors of thread management constructs, ACM SIGPLAN Notices 49 (10) (2014) 345–360.
[34] B. Fernandes, G. Pinto, F. Castor, Assisting non-specialist developers to build energy-efficient software, in: in: Proceedings of the 39th International Conference on Software Engineering Companion, IEEE Press, 2017, pp. 158–160.
[35] D.I. Sjoberg, B. Anda, E. Arisholm, T. Dyba, M. Jorgensen, A. Karahasanovic, E.F. Koren, M. Vokác, Conducting realistic experiments in software engineering, in: in: Empirical Software Engineering, 2002, Proceedings. 2002 International Symposium n, IEEE, 2002, pp. 17–26.
[36] A. Carroll, G. Heiser, et al., An analysis of power consumption in a smartphone, in: in: USENIX annual technical conference, Vol. 14, Boston, MA, 21–21, 2010.
[37] W.G. Hopkins, A new view of statistics, Will G. Hopkins, 1997.
[38] R.K. Yin, Case study research: Design and methods, Sage Publications, 2013.