World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:9, No:11, 2015

Towards the Use of Software Product Metrics as an Indicator for Measuring Mobile Applications Power Consumption

Ching Kin Keong, Koh Tieng Wei, Abdul Azim Abd. Ghani, Khaironi Yatim Sharif

Open Science Index, Computer and Information Engineering Vol:9, No:11, 2015 publications.waset.org/10002793.pdf

Abstract**—**Maintaining factory default battery endurance rate over time in supporting huge amount of running applications on energy-restricted mobile devices has created a new challenge for mobile applications developer. While delivering customers' unlimited expectations, developers are barely aware of efficient use of energy from the application itself. Thus, developers need a set of valid energy consumption indicators in assisting them to develop energy saving applications. In this paper, we present a few software product metrics that can be used as an indicator to measure energy consumption of Android-based mobile applications in the early of design stage. In particular, Trepn Profiler (Power profiling tool for Qualcomm processor) has used to collect the data of mobile application power consumption, and then analyzed for the 23 software metrics in this preliminary study. The results show that McCabe cyclomatic complexity, number of parameters, nested block depth, number of methods, weighted methods per class, number of classes, total lines of code and method lines have direct relationship with power consumption of mobile application.

Keywords**—**Battery endurance, software metrics, mobile application, power consumption.

I. Introduction

SINCE most of the mobile applications that introduced to the market place are highly consume energy due to the high usage of processing power, energy efficiency of mobile application is an important concern for energy restricted embedded system. While resource constraint has creating a limitation for diverse and complex functions execution on mobile application [5], several research works are done to optimize the power usage from hardware perspective such as processor idleness to reduce power consumption. However, such improvement does not be sufficient by them since poorly written applications can cruelly drain the extra battery power over a long period of time. In the past, software developers were concentrating on standard software quality characteristics such as maintainability and reliability of the software rather than focus on power consumption of the application that was designed and delivered. The possible reason that can explain this phenomenon is lacking of available technique and approach from the software engineering society to support their need. As a result, power hungry applications were developed and delivered to the market place [1]. In fact, these type applications can drain 30 to 40% of a mobile device's battery [2]. There are some studies have pay attention on the power consumption issue of software applications, however, they are mainly focus on source code-based analysis. Although this type of power consumption analysis and related estimation methods provide similar results to the actual power consumption of mobile devices, there are often too late for rework and it is language-dependent [30]. On the other hand, based on our observation, some of the software product metrics can be used for power consumption analysis instead of using source-code based power consumption analysis. In this paper, these software product metrics were identified and analyzed to suggest the new opportunity for measuring power consumption of mobile applications.

Organization of the paper is arranged as the following. Section II describes related works on power consumption reduction and estimation techniques. Section III describes our case study preparation procedure, data collection process and results analysis. Concluding remark and further work are discussed in Section IV.

Ching Kin Keong is a postgraduate student in the Faculty of Computer Science and Information Technology.

Koh Tieng Wei is a senior lecturer in the Department of Software Engineering and Information Systems under the Faculty of Computer Science and Information Technology, University Putra Malaysia (Corresponding Author; Tel: +603-89471799; fax: +603-89466576; e-mail: twkoh@upm.edu.my).

Abdul Azim Abd. Ghani is a professor in software engineering, and he is also with the Faculty of Computer Science and Information Technology (e-mail: azim@upm.edu.my).

Khaironi Yatim Sharif is a is a senior lecturer in the Department of Software Engineering and Information Systems under the Faculty of Computer Science and Information Technology, University Putra Malaysia (e-mail: khaironi@upm.edu.my).

II. Related Works

Most of the previous research works found in the literature were presenting approaches that can minimize energy consumption by investigating on executable application instead of during the design stage. In [22], designs of Dynamic Voltage and Frequency Scaling (DVFS) [27], [35], [37] and offloading cloud computing [16], [21], [31], [38], [39] are among the most popular power reduction research focus. Ma et al. [35] was reported that DVFS power control mechanism can use to adjust frequency and voltage according to the system state of a device. Mobile devices can dynamic adjust the CPU and DSP frequency. Liang et al. [37] established a table-based DVFS mechanism for frame decoding. They exploit the frame decoding complexity to minimize the power consumption of a processor, and adopt the runtime information of the hardware performance counters to evaluate the complexity of the decoding process to reduce energy consumption for 9 to 17 percent. Kyosuke et al. [36] proposed another power reduction method with the consideration of performance in Android terminals. They dynamically adjust CPU clock frequency at runtime by gaining feedback information from applications. They claim that the drawing framework save more energy than methods without dynamic CPU clock frequency adjustment. Silvén and Jyrkkä [15] presented the key differences between the implementation of computing solutions used in mobile communications equipment are found at the chip level: mobile devices low leakage silicon technology and lower clock frequency are used but they are essentially the same as those in personal and mainframe computers.

Kong et al. [38] show that by offloading some partition to the cloud can help in reducing energy of mobile devices. They proposed a dynamic computation offloading framework that can partition the Android application into two parts, local phone and server at the compile stage automatically. The partition that sends to server will be executed on the cloud and execution time on the mobile device will be reduced. Pan et al. [39] presented Learning-on-Cloud (LoC) policy to exploit cloud computing for power management. They offload sophisticated learning engines from local devices to the cloud with the least amount of communication data to reduce runtime overhead. That is, all learning data from many devices and with one thousand devices are connected to the cloud, the LoC agent is able to converge within a few iterations. They also claim that learning-based policies have less latency penalty. Chen et al. [4] proposed a new energy consumption model for cloud computing and can be integrated into Cloud systems to observe energy consumption and help in static or dynamic system-level improvement by measure energy consumption in Cloud environment with different run time task. Papageorgiou et al. [33] claim that by determine the time intervals between the logic of Web service response caching and Web service invocations can support the minimization of energy consumption of mobile Web service-based applications.

Others energy reduction techniques include network communication [13], [14], [24], [25], adaptation [5], [17], context awareness [11], [12], display [19], [26], resource scheduling [9], [18] and platform [29]. Harjula et al. [32] claimed that it is necessary to reduce the need for time consuming measurements with real-life networks and devices to facilitate designing energy-efficient networking solutions. They presented an advanced (e-Aware) that makes a difference between media transfer for maximizing the accuracy and signaling to estimate how application layer protocol properties influence the energy consumption of mobile devices. They are measuring energy in two perspective which are in 3G (WCDMA) and WLAN (802.11) networks. By using the device-specific coefficients, the model is fined-tuned for different devices. Besides, transport layer protocol is also being used to minimize energy consumption. Kravets and Krishnan [20] claim that power usage can be reduced through transport layer protocol. They choose the short period of time to turn off the communication device and suspend communication by using their design and implementation of innovative transport level protocol. The important task of deciding when to restart communication, and queuing data for future delivery during periods of communication suspension has been managed.

Chen et al. [5] claimed that battery endurance is one of the most significant user experiences for mobile devices. The limitations on mobile devices have restricts the functional design of hardware architecture and applications. Thus, they proposed an Anole framework. It is a framework that uses a set of APIs and adaptation policies to create an energy adaptation layer to change application and system state dynamically based on the energy status and the user expectation. They use the concept of adaptation to add an energy adaptation layer by providing a set of APIs and policies on top of their previous study when some events is occur through the energy profiling that they have made[23].

Brandolese [30] claim that previous software performance estimation approaches (instruction-level simulation and static-time source characterization) are either accurate but slow or flexible but independent. Therefore, they propose a hybrid approach that combines the strength of two approaches to make a fully automatic method to estimate the C program execution time and energy used. Bornholt et al. [7] created power modelling tools that can be uses to estimate power draw based on previous measured correlations between metric and power by using utilization metric. Z.X. Liao et al. [28] claimed that by using Temporal-based Apps Predictor (TAP) one can determine the apps usage of mobile phone to reduce energy consumption. Thompson et al. [34] presented that application developers will only be able to measure energy consumption characteristics of a design after implementation due to multiple layers of abstractions and middleware problem. They proposed a model-driven methodology for accurately matching the power consumption of smartphone application architectures to fix this issue. They use the System Power Optimization Tool (SPOT) to automate power consumption emulation code generation and simplify analysis of power consumption early in the lifecycle of mobile applications. Zhang et al. [10] had proposed an automated power model construction technique that uses built-in battery voltage sensors and knowledge of battery discharge behavior to monitor power consumption while explicitly controlling the power management and activity states of six components: CPU, LCD, Wi-Fi, cellular interfaces, GPS, and audio. Flinn and Satyanarayanan [8] show that applications can dynamically change their behavior to conserve energy. They demonstrated the collaborative relationship between the OS and application can be used to meet user-specified goals for battery duration. It is able to select the correct tradeoff between energy conservation and application quality by monitoring energy supply and demand. The results show that this approach can meet goals that extend battery life up to 30%. Some researchers create power modelling tools to estimate power consumption.

It is reasonably to conclude that most of the previous

Open Science Index, Computer and Information Engineering Vol:9, No:11, 2015 publications.waset.org/10002793.pdf

Open Science Index, Computer and Information Engineering Vol:9, No:11, 2015 publications.waset.org/10002793.pdf

inventions are focus on hardware, network and communication protocol. Our direction in this topic obviously differs significantly. That is, we find an opportunity from software engineering perspective by investigating and proposing a set of valid power consumption indicator which should be available early in the application design phase. A power consumption estimation model can be derived from these indicators to assist software engineer to predict the energy usage in design phase and deliver better quality software product to the market place.

III. Case Study Preparation and Setup

Our case study is focusing on Android mobile applications. Android operating system is a world known mobile OS. There are over 1,300,000 mobile applications have been developed and place in Google play store. It is the largest free mobile applications platform compare to other mobile platform such as Apple Store and Windows store. Mobile applications are built from 4 components which are activities, services, content provider and broadcast receiver. An activity in a mobile application represents a single screen with a user interface, it allow user to interact with the phone to do something with just touching on the screen like take a video, send a file via Bluetooth. Each activity is independent with other activity. Service is a component that without an interface. Services run in the background and provide continuous operations such as playing music and connecting to database. Content provider is one of the main building blocks of Android mobile applications. The main function of content provider is providing content to applications or sharing data among other applications. Broadcast receiver is another component that uses to broadcast announcements without a user interface such as E-mail notification.

A. Open Source Mobile Application Selection

There are 1474 open source applications available in Fdroid. The FDroid repository is an installable catalogue of Free and Open Source Software (FOSS) applications for the Android platform. The applications in Fdroid are also available in Google Play Store [3]. We select randomly six mobile applications from the list of applications in the FDroid repository [6]. That is, random table was used to get the number of page and number of position of open source applications. The random number tables are composed of the digits from 0 through 9, with approximately equal frequency of occurrence. On each digits are printed in blocks of five columns and blocks of five rows. The six applications being selected are "Did I?", “Coin Flip”, “Currency Converter”, “Battery level”, “HydroMemo” and “Applocker”.

B. Process of Empirical Study

The relationship of software product metrics and power consumption of mobile applications is investigated empirically. These software product metrics include McCabe cyclomatic complexity, number of parameters, nested block depth, afferent coupling, efferent coupling, instability, abstractness, normalized distance, depth of inheritance tree, weighted methods per class, number of children, number of overridden methods, lack of cohesion of methods, number of attributes, number of static attributes, number of methods, number of static methods, specialization index, number of classes, number of interfaces, number of packages, total lines of code and method lines of code.

All tests were performed on a Google Nexus 7 with a 1.5Ghz Qualcomm Quad-Core CPU, 2GB memory and 32GB Storage that running the default installation of Android 4.4 (KitKat). Fig. 1 summarizes the process of empirical study.

Open source mobile applications were downloaded from Fdroid and installed into Google Nexus 7 to identify all available functions. We analyzed the source code being used in the mobile applications for each functions in Eclipse IDE. In particular, we study all classes and methods in the source code of the applications. For each method in a class file, a break point is added in the first row of the method and run the apps in debug mode to trace the function. This is to identify which methods will be involved while running specific function. Fig. 2 shows the example of identified classes and methods that are involved in start apps function.

C. Software Product Metrics

The software product metrics were captured using the metric plugin in Eclipse. In order to capture software metric of particular function, all unrelated source code will be commented or deleted while capturing software metric. Fig. 3 shows the product metrics of start application function.

Based on "Object-Oriented Metrics, measures of Complexity" by Brian Henderson-Sellers, number of classes indicates the total number of classes in the selected scope. Number of children shows us the total number of direct subclasses of a class. A class implementing an interface counts as a direct child of that interface. Number of Interfaces means the total number of interfaces in the selected scope. Depth of Inheritance Tree (DIT) is the distance from class Object in the inheritance hierarchy. While number of Overridden Methods (NORM) shows the total number of methods in the selected scope that are overridden from an ancestor class. Number of methods (NOM) refers to the total number of methods defined in the selected scope. Number of Fields shows the total number of fields defined in the selected scope. Total lines of code will counts non-blank and non-comment lines in a compilation unit. Method Lines of Code (MLOC) will calculates and sum non-blank and non-comment lines inside method bodies. Specialization index indicate the average of the specialization index, defined as NORM * DIT / NOM. McCabe Cyclomatic Complexity is counting the number of flows through a piece of code. Weighted Methods per Class (WMC) is the summation of the McCabe Cyclomatic Complexity for all methods in a class. Lack of Cohesion of Methods (LCOM*) is a measure for the Cohesiveness of a class. It use the Henderson-Sellers method to calculate. If

Open Science Index, Computer and Information Engineering Vol:9, No:11, 2015 publications.waset.org/10002793.pdf

(m(A) is the number of methods accessing an attribute A, calculate the average of m(A) for all attributes, subtract the number of methods m and divide the result by (1-m). A low value indicates a cohesive class and a value close to 1 indicates a lack of cohesion and suggests the class might better be split into a number of subclasses. Robert Martin defines metric in the coupling perspective in "OO Design Quality Metrics, An Analysis of Dependencies". Afferent Coupling (Ca) shows the number of classes outside a package that depend on classes inside the package. Efferent Coupling (Ce) calculates the number of classes inside a package that depend on classes outside the package. Instability (I) use the formula Ce / (Ca + Ce). Abstractness (A) refers to the number of abstract classes (and interfaces) divided by the total number of types in a package and Normalized Distance from Main Sequence calculate by | A + I - 1 |, this number should be small, close to zero for good packaging design.

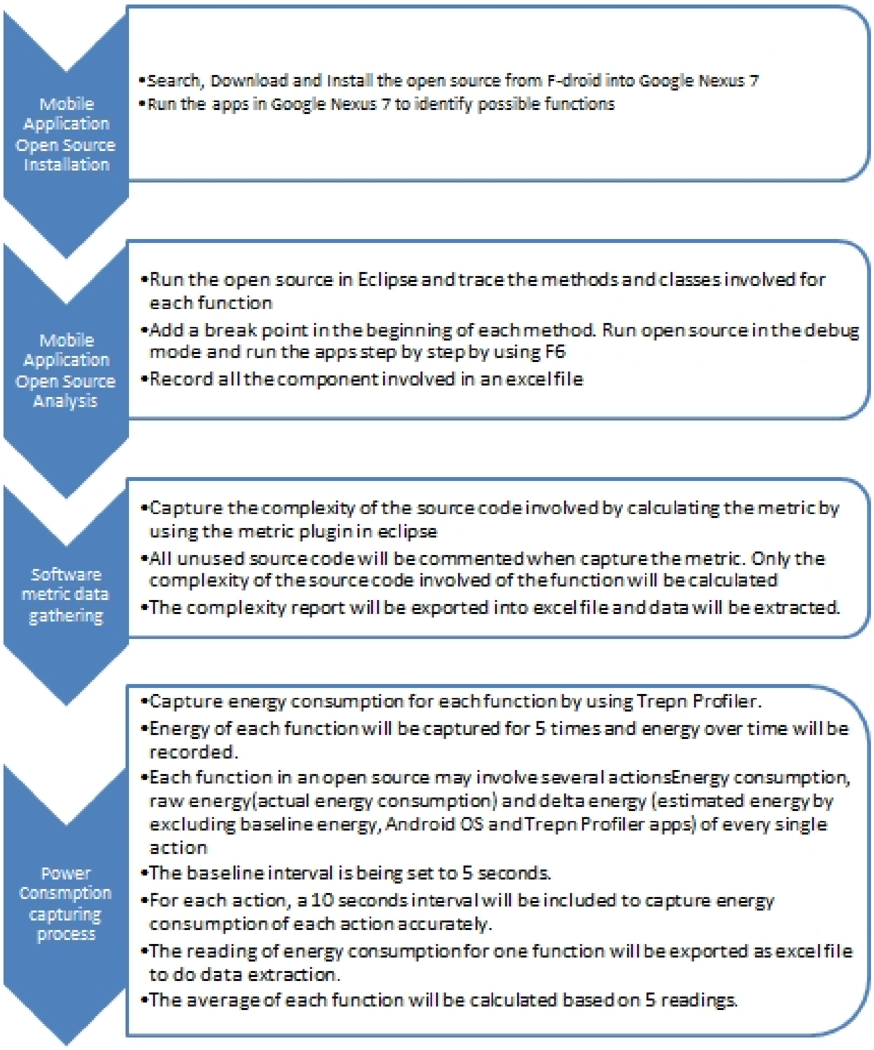

Fig. 1 Process of empirical study

Apps Name	Did I?			
Function Name	start apps			
Class	Main	FragmentHabits	DatabaseHelper	ViewPagerAdapterHabit
Method	onTabSelected	onCreateView	DatabaseHelper	ViewPagerAdapterHabit
	onCreate	onResume	getHabitDao	getCount
	onCreateOptionsMenu	loadUI	getDayDao	instantiateItem
		getHelper		loadQuestion
				isViewFromObject

Fig. 2 Classes and methods of start application function

Open Science Index, Computer and Information Engineering Vol:9, No:11, 2015 publications.waset.org/10002793.pdf

Metrics	avg	Total
McCabe Cyclomatic Complexity	2	48
Number of Parameters	1.5	36
Nested Block Depth	1.417	34
Afferent Coupling	0.166666667	4
Efferent Coupling	0.166666667	4
Instability	0.020833333	0.5
Abstractness	0	0
Normalized Distance	0.020833333	0.5
Depth of Inheritance Tree	3.25	13
Weighted methods per Class	12	48
Number of Children	0	0
Number of Overridden Methods	0.75	3
Lack of Cohesion of Methods	0.742	2.967
Number of Attributes	7	28
Number of Static Attributes	1.5	6
Number of Methods	6	24
Number of Static Methods	0	0
Specialization Index	0.446	1.783
Number of Classes	0.166666667	4
Number of Interfaces	0	0
Number of Packages	0	0
Total Lines of Code	15.79166667	379
Method Lines of Code	7.292	175

Fig. 3 Software product metrics for start application function "Did I"

C. Power Consumption Data Collection and Analysis

Power consumption of mobile devices can be collected by using energy Trepn profiler. It is an on-target power and performance profiling tool for mobile devices for Qualcomm processor. Before testing the application, Bluetooth, Wi-Fi, 3G will be turned off and the device is set to airplane mode to avoid extra power consumption being capture by energy profiler. The baseline interval is being set to 5 seconds to capture the Android OS power consumption and Trepn Profiler application power consumption in order to estimate application consumption accurately. Battery Power [µW] (Raw), Battery Power [µW] (Delta) and time have being collected. Battery powers are measure in unit microwatt (µW) and time measure in unit milliseconds (ms).

- *Battery Power [µW] (Raw)* **-** Power consumption in unit microwatt that have not been processed, actual power consumption
- *Battery Power [µW] (Delta)* - Power consumption in unit microwatt that have been processed after eliminated Android OS and Trepn Profiler apps power consumption, estimated apps power consumption

We use 10 seconds as an interval to separate each reading to capture power consumption accurately for every reading. That is, we will wait for 10 seconds after doing an action (a click or swipe) before performing next action. Fig. 4 shows the 10 seconds interval overlay. For each function, 5 readings will be capture to calculate the average. The data captured will be store in the device's SD card in the form of a CSV file. Fig. 5 shows one of the exported excel file of Trepn profiler for view about function in mobile application named Coin Flip. Time in first column represents start time in milliseconds; battery status shows 0 if not charging and shows 1 if charging. Time in third column represents end time in milliseconds.

D. Average Power Consumption and Software Product Metrics Data Collection

As we discuss in Section III, the application is installed in Google Nexus 7. We first launch the selected application to list down all available functions. Trepn Profiler will be launch and we will run all selected application in Trepn Profiler. By executing and monitoring the power usage of the application, we collect Battery Power [µW] (Raw), Battery Power [µW] (Delta) with a 10 seconds interval for each action performed to get a more accurate reading as in Fig. 5 from time 2831ms to 11773ms. Reading will be 0 in these 10 seconds. Power consumption for this function will start in 11866ms to 13679ms. We expect the power consumption for 5 readings to be consistent as the source code and function are the same. We then rearrange the 5 readings in the format as Fig. 5 with overall represent overall power consumption in µW. Applications represents applications power consumption in µW and time used represent time frame in milliseconds. Power of each reading consumed will be sum up, and time taken is being calculated by using (end time of action performed – start time of action performed) formula. Fig. 6 shows the raw power that we extract from exported excel document. We sum up the raw power as in Fig. 7 for each of the particular action. Then, we calculate the average of power and total up the power used for each function. Fig. 8 shows the average of power used and Fig. 9 shows the total average usage of power consumption. After power consumption is being captured as shown in Table I, we map each of these power consumption reading with the software product metrics gather from the Eclipse plugin as shown in Table II for the functions F1 to F9. We then identify which metrics have high relationship with the mobile application power consumption based on statistical analysis.

Fig. 4 Ten seconds interval overlay

E. Data Analysis

Every single action that interacts with mobile phone consumes power. Some action may have high power usage; some action may have low power usage. During the mapping process, we categorize all the functions in these 2 different categories. Actions that involve high power usage, which have significant spark when capturing power consumption are start application that involved method startActivity (Intent), create new page and create component like dialog. Actions that involve low power usage, which have no significant spark when capturing power consumption are a single button click and select an item. After mapping power consumption with all the metrics based on these 2 categories, we analyze using SPSS statistical tool. There are 21 functions having high

Open Science Index, Computer and Information Engineering Vol:9, No:11, 2015 publications.waset.org/10002793.pdf

power usage and 9 functions having low power usage. For category of high power usage, the correlate bivariate shows that McCabe cyclomatic complexity, number of parameters, nested block depth, number of methods and weighted methods per class, number of classes, total lines of code and method lines of code are having significant relationship with the application power consumption at the 0.05 level of confidence. The values of Pearson correlation for these three metrics are 0.662, 0.581, 0.695, 0.662, 0.707, 0.526, 0.585 and 0.526. For the number of methods that involved during the debug process also show the result is significant and having 0.712 for Pearson correlation.

For category of low power usage, the correlate bivariate shows that only the number of classes involved during debug process is significant and the value of Pearson correlation is 0.699. Fig. 10 shows the bivariate correlation between Object-oriented metrics and power consumption. Fig. 11 shows the bivariate correlation between OO design quality metrics and power consumption.

Time [ms]	Battery Status	Time [ms]	Battery Power [uW] (Raw)	Battery Power [uW] (Delta)
34	0	742	931000	0
839	0	1667	2092000	920400
1762	0	2728	1616000	444400
2831	0	3747	961000	0
3843	0	4704	952000	0
4801	0	5717	978000	0
5813	0	6776	982000	0
6835	0	7761	952000	0
7845	0	8765	990000	0
8858	0	9770	1041000	0
9865	0	10751	935000	0
10845	0	11773	965000	0
11866	0	12773	1490000	318400
12870	0	13679	1274000	102400
13774	0	14689	1168000	0
14783	0	15701	1150000	0
15794	0	15809	1066000	0
16703	0	16708	1066000	0
16811	0	16818	1108000	0
17717	0	17721	1108000	0
17818	0	17821	1092000	0
18689	0	18694	1092000	0
18793	0	18821	1091000	0
19598	0	19704	1091000	0
19699	0	19806	1116000	0
20617	0	20721	1116000	0
20717	0	20825	1066000	0
21623	0	21727	1066000	0
21723	0	21829	1096000	0

Fig. 5 Trepn profiler power consumption data

Raw Power	Overall(μW)	Apps(μW)	Start time	End Time	Overall(μW)	Apps(μW)	Start time	End Time
1	1549000	304380	21832	25854	1255000	10380	32819	35855
	1457000	212380			1540000	295380		
	1312000	67380			1292000	47380		
	1304000	59380						
2	1325000	154660	16179	20199	1536000	365660	27158	28166
	1612000	441660						
	1213000	42660						
	1256000	85660						
3	1702000	455200	19680	23712	1569000	322200	32695	33661
	1489000	242200						
	1326000	79200						
	1384000	137200						
4	1922000	663520	19381	22134	1563000	304520	29433	30216
	1379000	120520						
	1292000	33520						
5	1826000	530460	18439	22334	1381000	85460	29569	31324
	1280000	0			1569000	273460		
	1411000	115460						
	1333000	37460						

Fig. 6 Raw power consumption of 5 readings for edit habit function

Open Science Index, Computer and Information Engineering Vol:9, No:11, 2015 publications.waset.org/10002793.pdf

Sum of power	Overall(µW)	Apps(µW)	Time Used(ms)	Overall(µW)	Apps(µW)	Time Used(ms)
1	5622000	643520	4022	4087000	353140	3036
2	5406000	724640	4020	1536000	365660	1008
3	5901000	913800	4032	1569000	322200	966
4	4593000	817560	2753	1563000	304520	783
5	5850000	683380	3895	2950000	358920	1755

Fig. 7 Sum of power consumption for one function

Average of power	Overall(µW)	Apps(µW)	Time Used(ms)	Overall(µW)	Apps(µW)	Time Used(ms)
	5474400	756580	3744.4	2341000	340888	1509.6

Fig. 8 Average power consumption for one function

Sum of average of power	Overall(µW)	Apps(µW)	Time Used
	7815400	1097468	5254

Fig. 9 Sum of average power consumption for one function

TABLE I
AVERAGE POWER CONSUMPTION FOR EACH FUNCTION

Function	Overall (µW)	Apps (µW)
F1:Did I start	2337600.00	1133962.20
F2:Did I Answer question	1332600.00	52260.60
F3:Did I set alarm	4222200.00	789370.80
F4:Did I edit habit	7815400.00	1097468.00
F5:Did I add habit	5854200.00	1024560.00
F6:Did I view edit habit page	1833800.00	580836.00
F7:Did I delete habit	2743400.00	236703.20
F8:Did I view progress	1291800.00	85756.00
F9:Did I view habit	1276600.00	126127.60

IV. CONCLUSION AND FUTURE WORK

In this paper, we present a set of possible indicators that can be used to measure power consumption of mobile application. From the result, we can summarize that number of methods has the highest bivariate correlation with mobile application's power consumption. Other software product metrics such as McCabe cyclomatic complexity, number of parameters, nested block depth, number of methods and weighted methods per class, number of classes, total lines of code and method lines of code are possible to be an valid indicator to measure mobile applications' power consumption for high power usage functions. Our future work will be generating a power consumption estimation model to estimate the power consumption of Android mobile applications.

TABLE II
SOFTWARE PRODUCT METRICS FOR FUNCTION 1 TO FUNCTION 9

Metric/Function	**F1**	**F2**	**F3**	**F4**	**F5**	**Metric/Function**	**F6**	**F7**	**F8**	**F9**
Afferent Coupling	4	0	0	5	5	McCabe Cyclomatic Complexity	19	17	63	61
Efferent Coupling	4	0	0	3	3	Number of Parameters	19	19	50	38
Instability	0.5	0	0	0.375	0.375	Nested Block Depth	18	17	47	41
Abstractness	0	0	0	0	0	Depth of Inheritance Tree	13	13	15	15
Normalized Distance	0.5	0	0	0.625	0.625	Weighted methods per Class	19	17	63	61
McCabe Cyclomatic Complexity	48	20	4	23	29	Number of Children	0	0	0	0
Number of Parameters	36	10	8	22	23	Number of Overridden Methods	1	1	3	2
Nested Block Depth	34	9	3	23	27	Lack of Cohesion of Methods	0.8	1	2.238	2.422
Depth of Inheritance Tree	13	2	6	16	16	Number of Attributes	21	21	34	35
Weighted methods per Class	48	20	4	23	29	Number of Static Attributes	4	4	7	7
Number of Children	0	0	0	0	0	Number of Methods	12	10	32	27
Number of Overridden Methods	3	0	0	2	2	Number of Static Methods	0	0	0	0
Lack of Cohesion of Methods	2.967	0.955	0	1.867	1.822	Specialization Index	1.5	1.5	0.844	0.65
Number of Attributes	28	11	16	21	21	Number of Classes	4	4	5	5
Number of Static Attributes	6	0	2	5	5	Number of Interfaces	0	0	0	0
Number of Methods	24	5	3	14	15	Number of Packages	0	0	0	0
Number of Static Methods	0	0	0	1	1	Total Lines of Code	229	230	483	425
Specialization Index	1.783	0	0	4.5	4.5	Method Lines of Code	105	119	226	186
Number of Classes	4	1	1	5	5	Afferent Coupling	6	6	3	3
Number of Interfaces	0	0	0	0	0	Efferent Coupling	2	2	5	5
Number of Packages	0	0	0	0	0	Instability	0.25	0.25	0.625	0.625
Total Lines of Code	379	126	105	279	305	Abstractness	0	0	0	0
Method Lines of Code	175	70	16	133	155	Normalized Distance	0.75	0.75	0.375	0.375

Open Science Index, Computer and Information Engineering Vol:9, No:11, 2015 publications.waset.org/10002793.pdf

		Overall power consumption	Apps power consumption
McCabe Cyclomatic Complexity	Pearson Correlation	0.031	.662**
	Sig. (2-tailed)	0.894	0.001
	N	21	21
Number of Parameters	Pearson Correlation	0.018	.581**
	Sig. (2-tailed)	0.938	0.006
	N	21	21
Nested Block Depth	Pearson Correlation	0.113	.695**
	Sig. (2-tailed)	0.625	0
	N	21	21
Depth of Inheritance Tree	Pearson Correlation	0.186	0.13
	Sig. (2-tailed)	0.418	0.574
	N	21	21
Weighted methods per Class	Pearson Correlation	0.033	.662**
	Sig. (2-tailed)	0.888	0.001
	N	21	21
Number of Children	Pearson Correlation	.[b]	.[b]
	Sig. (2-tailed)	.	.
	N	21	21
Number of Overridden Methods	Pearson Correlation	0.179	.442*
	Sig. (2-tailed)	0.439	0.045
	N	21	21
Lack of Cohesion of Methods	Pearson Correlation	0.001	.440*
	Sig. (2-tailed)	0.997	0.046
	N	21	21
Number of Attributes	Pearson Correlation	-0.078	0.353
	Sig. (2-tailed)	0.737	0.116
	N	21	21
Number of Static Attributes	Pearson Correlation	-0.124	0.325
	Sig. (2-tailed)	0.592	0.15
	N	21	21
Number of Methods	Pearson Correlation	0.124	.707**
	Sig. (2-tailed)	0.593	0
	N	21	21
Number of Static Methods	Pearson Correlation	-0.152	0.086
	Sig. (2-tailed)	0.511	0.71
	N	21	21
Specialization Index	Pearson Correlation	0.231	-0.017
	Sig. (2-tailed)	0.314	0.941
	N	21	21
Number of Classes	Pearson Correlation	0.303	.526*
	Sig. (2-tailed)	0.182	0.014
	N	21	21
Number of Interfaces	Pearson Correlation	-0.084	0.207
	Sig. (2-tailed)	0.717	0.367
	N	21	21
Number of Packages	Pearson Correlation	.[b]	.[b]
	Sig. (2-tailed)	.	.
	N	21	21
Total Lines of Code	Pearson Correlation	0.031	.585**
	Sig. (2-tailed)	0.895	0.005
	N	21	21
Method Lines of Code	Pearson Correlation	-0.001	.526*
	Sig. (2-tailed)	0.996	0.014
	N	21	21
Class	Pearson Correlation	0.068	0.282
	Sig. (2-tailed)	0.77	0.216
	N	21	21
Method	Pearson Correlation	0.089	.712**
	Sig. (2-tailed)	0.701	0
	N	21	21
Overall power consumption	Pearson Correlation	1	.591**
	Sig. (2-tailed)		0.005
	N	21	21
Apps power consumption	Pearson Correlation	.591**	1
	Sig. (2-tailed)	0.005	
	N	21	21
Time used	Pearson Correlation	.963**	0.422
	Sig. (2-tailed)	0	0.057
	N	21	21

Fig. 10 SPSS bivariate correlation result between OO metrics and power consumption

Open Science Index, Computer and Information Engineering Vol:9, No:11, 2015 publications.waset.org/10002793.pdf

		Overall power consumption	Apps power consumption
Afferent Coupling	Pearson Correlation	0.229	-0.059
	Sig. (2-tailed)	0.318	0.799
	N	21	21
Efferent Coupling	Pearson Correlation	0.162	0.331
	Sig. (2-tailed)	0.482	0.142
	N	21	21
Instability	Pearson Correlation	-0.192	0.147
	Sig. (2-tailed)	0.404	0.524
	N	21	21
Abstractness	Pearson Correlation	-0.084	0.207
	Sig. (2-tailed)	0.717	0.367
	N	21	21
Normalized Distance	Pearson Correlation	0.217	-0.005
	Sig. (2-tailed)	0.344	0.982
	N	21	21
Overall power consumption	Pearson Correlation	1	.591**
	Sig. (2-tailed)		0.005
	N	21	21
Apps power consumption	Pearson Correlation	.591**	1
	Sig. (2-tailed)	0.005	
	N	21	21

Fig. 11 SPSS bivariate correlation result between OO design quality metrics and power consumption

ACKNOWLEDGMENT

This research work has been funded by the Fundamental Research Grant Scheme (FRGS) under the Malaysian Ministry of Education (MOE) for the project no. 08-02-13-1360FR/5524441. The authors would like thank to the Research Management Centre of UPM and the MOE for their support and cooperation including students and other individuals who are either directly or indirectly involved in this project.

REFERENCES

[1] C. Sahin, F. Cayci, J. Clause, F. Kiamilev, L. Pollock, K. Winbladh. "Towards Power Reduction through Improved Software Design". In IEEE Energytech, pages 1 - 6, MAY 2012.

[2] S. Jha., Poorly written apps can sap 30 to 40% of a phone's juice, June 2011. CEO, Motorola Mobility, Bank of America Merrill Lynch 2011 Technology Conference.

[3] Mark D. Syer et al., Revisiting Prior Empirical Findings For Mobile Apps: An Empirical Case Study on the 15 Most Popular Open-Source Android Apps, In proceeding of: Proceedings of the 2013 Conference of the Center for Advanced Studies on Collaborative Research, 2013

[4] F.F Chen, J.G Schneider, Y. Yang, J. Grundy, Q. He. "An Energy Consumption Model and Analysis Tool for Cloud Computing Environments", in Proceedings of the 1st International Workshop on Green and Sustainable Software (GREENS'12), 2012, pp.45-50.

[5] H. Chen, B. Luo, W. Shi. "Anole: A Case for Energy-Aware Mobile Application Design", in Proceedings of the 41st International Conference on Parallel Processing Workshops (ICPPW'12), 2012, pp.232-238.

[6] Mark D. Syer, Meiyappan Nagappan, Ahmed E. Hassan, Bram Adams, Revisiting prior empirical findings for mobile apps: an empirical case study on the 15 most popular open-source Android apps, Proceedings of the 2013 Conference of the Center for Advanced Studies on Collaborative Research, November 18-20, 2013, Ontario, Canada

[7] J. Bornholt, T. Mytkowicz, K. S. McKinley. The Model Is Not Enough: Understanding Energy Consumption in Mobile Devices", in Posters Session of Hot Chips: A Symposium on High Performance Chips (HC24), 2012.

[8] J. Flinn, M. Satyanarayanan. "Energy-aware Adaptation for Mobile Applications", in Proceedings of 17th ACM Symposium on Operating Systems Principles (SOSP'99), pp.48-63.

[9] L. Luo, W. Wu, D. Di, F. Zhang, Y. Z. Yan, Y. K. Mao. "A Resource Scheduling Algorithm of Cloud Computing based on Energy Efficient Optimization Methods", in Proceedings of the IEEE International Green Computing Conference (IGCC'12), 2012, pp.1-6.

[10] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, L. Yang. "Accurate Online Power Estimation and Automatic Battery Behavior Based Power Model Generation for Smartphones", in Proceedings of the 8th IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS'10), 2010, pp.105-114.

[11] Young-Seol Lee; Sung-Bae Cho, "An Efficient Energy Management System for Android Phone Using Bayesian Networks," Distributed Computing Systems Workshops (ICDCSW), 2012 32nd International Conference on, vol., no., pp.102, 107, 18-21 June 2012.

[12] Donohoo, B. K.; Ohlsen, C.; Pasricha, S., "AURA: An application and user interaction aware middleware framework for energy optimization in mobile devices," Computer Design (ICCD), 2011 IEEE 29th International Conference on, vol., no., pp.168,174, 9-12 Oct. 2011

[13] N. Balasubramanian, A. Balasubramanian, A. Venkatramani. "Energy Consumption in Mobile Phones: A Measurement Study and Implications for Network Applications, in Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference (IMC'09), 2009, pp.280-293.

[14] O. Arnold, F. Ritchter, G. Fettweis, O. Blume. "Power Consumption Modelling of Different Base Station Types in Heterogeneous Cellular Networks", in the Proceedings of the Future Network and Mobile Summit (FNMS'10), 2010, pp.1-8.

[15] O. Silvén, K. Jyrkkä. "Observation on Power-Efficiency Trends in Mobile Communication Devices", LNCS 3553, 2005, pp.142-151.

[16] P. Bartalos, M. B. Blake. "Green Web Services: Modelling and Estimating Power Consumption of Web Services", in Proceedings of the IEEE 19th International Conference on Web Services (ICWS'12), 2012, pp.178-185.

[17] Jain, R.; Bose, J.; Arif, T., "Contextual adaptive user interface for Android devices," India Conference (INDICON), 2013 Annual IEEE, vol., no., pp.1,5, 13-15 Dec. 2013

[18] R. Yamini. "Power Management in Cloud Computing Using Green Algorithm", in Proceedings of the IEEE International Conference on Advances in Engineering, Science and Management (ICAESM'12), 2012, pp.128-133.

[19] Tan Kiat Wee, Rajesh Krishna Balan, Adaptive display power management for OLED displays, Proceedings of the first ACM international workshop on Mobile gaming, August 13-13, 2012, Helsinki, Finland

[20] R. Kraves, P. Krishnan. "Application-driven Power Management for Mobile Communication", Journal of Wireless Network, Vol.6 Issue 4, July 2000, pp.263-277.

[21] S. A. Ahson, M. llyas, "Cloud Computing and Software Services: Theory and Techniques", CRC Press, Boca Raton, Florida, 2011.

[22] S. Gűrűn, R. Wolski, T. Sherwood, C. Krints. "Modelling, Predicting and Reducing Energy Consumption in Resource Restricted Computers", PhD Dissertation in Computer Science, University of California, Santa Barbara, 2007.

[23] T. Do, S. Rawshdeh, W, Shi. "pTop: A Process-level power Profiling Tool", in Proceedings of the 2nd Workshop on Power Aware Computing and Systems(HotPower'09), 2009.

[24] T. Pering, Y. Agarwal, R. Want. "CoolSpots: Reducing the power Consumption of wireless Mobile Devices with multiple Radio Interfaces", in Proceedings of the 4th International Conference on Mobile Systems, Applications and Services (MobiSys'06), 2006 pp.220-232.

[25] Y. Cui, X. Ma, H. Y. Wang, I. Stojmenonic, J. C.Liu. "A Survey of Energy Efficient Wireless Transmission and Modelling in Mobile Cloud Computing", Journal of Mobile Networks and Applications, Vol.18, Issue 1, Feb 2013, pp.148-155.

[26] Dongwon Kim; Nohyun Jung; Hojung Cha, "Content-centric display energy management for mobile devices," Design Automation Conference (DAC), 2014 51st ACM/EDAC/IEEE, pp.1,6, 1-5 June 2014

[27] Pathania, A.; Qing Jiao; Prakash, A.; Mitra, T., "Integrated CPU-GPU power management for 3D mobile games," Design Automation Conference (DAC), 2014 51st ACM/EDAC/IEEE, pp.1,6, 1-5 June 2014, doi: 10.1145/2593069.2593151

[28] Z.X. Liao, W.C. Peng, Y.C. Pan, P.R.Lei. "On Mining Mobile Apps Usage Behavior for Predicting Apps Usage in Smartphones", Proceedings of the 22nd ACM international conference on Conference on information & knowledge management, October 2013.

[29] Kadjo, D.; Ogras, U.; Ayoub, R.; Kishinevsky, M.; Gratz, P., "Towards

platform level power management in mobile systems," System-on-Chip Conference (SOCC), 2014 27th IEEE International, vol., no., pp.146,151, 2-5 Sept. 2014
[30] C. Brandolese, M.Politec. “Source-Level Estimation of Energy Consumption and Execution Time of Embedded Software”, Proceedings of the 11th EUROMICRO Conf. Digital System Design Architectures, Methods and Tools (DSD), pp. 115-123, 2008.
[31] Khairy, A.; Ammar, H.H.; Bahgat, R., "Smartphone Energizer: Extending Smartphone's battery life with smart offloading," Wireless Communications and Mobile Computing Conference (IWCMC), 2013 9th International, vol., no., pp.329,336, 1-5 July 2013
[32] E. Harjula, O. Kassinen, M. Ylianttila. "Consumption Model for Mobile Devices in 3G and WLAN Networks", in Proceedings of the 9th IEEE Consumer Communication and Networking Conference (CCNC'12), 2012, pp.532-537.
[33] A. Papageorgiou, U. Lampe, D. Schuller, R. Steinmetz, A. Bamis. "Invoking Web Services based on Energy Consumption Models:, in Proceedings of the IEEE 1st International Conference on Mobile Services(ICMS'12), 2102, pp.40-47
[34] C. Thompson, D. Schmidt, H. Tumer, J. White. "Analyzing Mobile Application Software Power Consumption Via Model-driven Engineering", in Proceedings of the 1st International Conference on Pervasive and Embedded Computing and Communication Systems(PECCS'11), 2011, pp.101-113
[35] Yi-Wei Ma; Jiann-Liang Chen; Ching-Hesign Chou; Shyue-Kung Lu, "A Power Saving Mechanism for Multimedia Streaming Services in Cloud Computing," Systems Journal, IEEE, vol.8, no.1, pp.219,224, March 2014
[36] Nagata, K.; Yamaguchi, S.; Ogawa, H., "A Power Saving Method with Consideration of Performance in Android Terminals," Ubiquitous Intelligence & Computing and 9th International Conference on Autonomic & Trusted Computing (UIC/ATC), 2012 9th International Conference on, vol., no., pp.578,585, 4-7 Sept. 2012
[37] Wen-Yew Liang; Ming-Feng Chang; Yen-Lin Chen; Chin-Feng Lai, "Energy efficient video decoding for the Android operating system," Consumer Electronics (ICCE), 2013 IEEE International Conference on, vol., no., pp.344,345, 11-14 Jan. 2013
[38] Gung-Yu Pan; Lai, B.-C.C.; Sheng-Yen Chen; Jing-Yang Jou, "A learning-on-cloud power management policy for smart devices," Computer-Aided Design (ICCAD), 2014 IEEE/ACM International Conference on, vol., no., pp.376,381, 2-6 Nov. 2014
[39] Deqian Kong; Tao Qi; Tan Yang; Yidong Cui, "A dynamic computation offloading framework for Android," Broadband Network & Multimedia Technology (IC-BNMT), 2013 5th IEEE International Conference on, vol., no., pp.134,138, 17-19 Nov. 2013

Open Science Index, Computer and Information Engineering Vol:9, No:11, 2015 publications.waset.org/10002793.pdf