

**CS 540 - Advanced Software  
Engineering  
University of Illinois at Chicago  
(Spring 2020)**

**Project Report on  
Empirical Evaluation of EARMO Toolset**

**Team - 04**

**Apurva Raghunath  
Arvind Ganesan  
Nandana Shimoga Prasad  
Suhan Nath**

# **Contents**

1. Introduction
  - 1.1. Objective
  - 1.2. Background
  - 1.3. Refactoring
2. System Requirements
3. Implementation
  - 3.1. Generating Jar from dexjar tool
  - 3.2. Configuring EARMO
  - 3.3. Apps and their Categories
4. Results
  - 4.1. Graph with distribution of antipatterns detected by EARMO across 60 apps for  $n=2$
  - 4.2. Distribution of antipatterns across each set of 20 apps
  - 4.3. Distribution of metaheuristics across 60 apps
  - 4.4. Apps with positive refactored energy increment
  - 4.5. Fitness Report
5. Automating the EARMO runs & processing the results
  - 5.1. Automating EARMO runs
  - 5.2. Processing the results
  - 5.3. Manual steps
6. Findings
7. Threats to validity
8. Scope for improvement
9. References

# **1. INTRODUCTION**

## **1.1 Objective**

The goal of EARMO is to improve the energy consumption of Android apps. EARMO refactors an app's source code in order to improve its energy consumption. In addition, EARMO can detect anti-patterns in an app's source code. The work carried out in this project is to evaluate EARMO on about 60 publicly available open source android apps. The apps that were utilized, their energy consumption after refactoring and the antipatterns that were found are discussed in further sections.

## **1.2 Background**

We began our study by looking at various literature that dealt with Android energy profiling and energy efficiency of mobile applications and how these compared to EARMO.

This section describes few of the tools that measures the energy consumption of android apps:

### **Power Estimation Tool For Android (PETrA)**

PETrA is a software based tool which can be used to measure the energy consumption of Android apps[4]. A method level estimation is done by PETrA. The tool is compatible with Android version 5.0 or higher. PETrA first installs the app, cleanses the app cache and reset the Android tool which is needed for creating a test environment. It then exercises the app using either the Android MONKEY tool or an Android MONKEY Runner Script, which generates a pseudo-random stream of user events, system-level events in order to create stress on the app which is under analysis. PETrA then computes the energy consumed by previous methods by collecting the data. To communicate with hardware components, PETrA uses the following components: dmtracedump[7], Batterystats[8], and Systrace[9].

PETRA has been studied on a set of 54 mobile applications. We tried to execute the tool with the instructions provided as in the paper.

Giving an APK for profiling and a Monkey script to run the various scenarios, the tool reported an error 'Number of trials has exceeded'. On digging a bit into the source code we hit a roadblock and decided to contact the author regarding the issue. The author responded back saying that he was aware of the existing issue and he was working on a newer version to fix it. Hence, this tool could not be used.

## **Method Level Energy Estimation Technique for Android Development(MELTA)**

MELTA is a probabilistic model which measures the energy level estimation for android apps[3]. It measures energy consumed by each module of the respective applications, which is a non-trivial factor for developers to create energy efficient codes.

Using Android studio tools which includes dmtracedump and BatteryHistorian, MELTA extracts the relevant energy consumption and execution traces. It maps the application power profile onto corresponding execution traces to determine the method level energy accurately. MELTA was evaluated on 15 open source apps and the results were compared to the Trepn Profiler.

To understand the working of MELTA as a tool, the author was contacted. However, there was no response. The unavailability of MELTA as a tool thus made us explore other tools for estimating the energy consumption of apps.

## **Energy Measurements as a Service for Mobile Applications (EMaaS)**

EMaaS is a peer-to-peer cloud based system that provides energy measurements as a service for mobile applications[6]. This system combines the estimates from an energy model with hardware based measurements. The system is accessible in the form of a software as service to the community of mobile software researchers. The energy model used in this approach was built on various profile details generated by different hardware profilers.

EMaaS being an off-the-shelf service provider does not help us in evaluating the energy of apps. EMaaS evaluates the APK and outputs the

energy consumption values but does not provide us an in depth analysis of energy profiling factors like anti patterns, code smells. Hence, EMaaS was not considered for energy improvements of the apps.

## **Leafactor: Improving Energy Efficiency of Android Apps via Automatic Refactoring**

Leafactor analyzes the code statically, transforms it, to implement Android-specific, energy optimizations[5]. It uses two engines in its system: one to handle Java files and another to handle XML files. The authors have analyzed 140 open-source Android apps using Leafactor. The optimizations supported by the tool include: DrawAllocation, WakeLock, Recycle, ObsoleteLayoutParam, ViewHolder. The major drawback for our project is that this approach does not present energy information and just produces an output of a number of refactorings that can be done.

## **Energy-Aware Refactoring Approach for Mobile Apps (EARMO)**

The energy consumption of mobile apps is a trending topic and researchers are actively investigating the role of coding practices on energy consumption. Recent studies suggest that design choices can conflict with energy consumption. Therefore, it is important to take into account energy consumption when evolving the design of a mobile app.

EARMO, a novel anti-pattern correction approach that accounts for energy consumption when refactoring mobile anti-patterns was thus proposed[1]. EARMO is evaluated using three multiobjective search-based algorithms on 20 publicly available android apps. The obtained results show that EARMO can generate refactoring recommendations in less than a minute, and remove a median of 84 percent of anti-patterns[1]. The authors of EARMO evaluated their approach on a preliminary study of 14 apps and obtained the refactoring coefficients  $\delta(EC(r_i))$  for each of the antipattern types they support [Figure 1].

EARMO consists of four steps. The first step consists of estimating the energy consumption of an app, running a defined scenario. In the second step, an abstract representation of the mobile's app. design, i.e.,

code meta-model is built. In the third step, the code metamodel is visited to search for anti-pattern occurrences.

The solutions produced by the earmo claims to meet the following objectives: 1) remove a maximum number of anti-patterns in the system, and 2) improve the energy consumption of the code design. The metaheuristics run in EARMO are non-deterministic. In order to get confidence in the results, the EARMO tool has to be run on the apk several times. In this project, for an apk, a separate set of 2 and a set of 3 runs are made by the EARMO tool[2].

| Refactoring Type                        | $\delta EC$ (ratio) |
|---|---------------------|
| Collapse hierarchy                      | 0.0056              |
| Inline class                            | -0.0315             |
| Inline private getters and setters      | -0.0237             |
| Introduce parameter object              | 0.0047              |
| Move method                             | -0.0020             |
| Move resource request to visible method | -0.0412             |
| Replace HashMap with ArrayMap           | -0.0160             |
| Replace Inheritance with delegation     | -0.0067             |

**Figure 1. Deltas of Energy Consumption by Refactoring Type**

### 1.3 Refactoring

The continuous addition of new functionalities or poor design choices i.e., anti-patterns can lead to design decay. Software maintainers thus use a software maintenance activity to transform the structure of the code without modifying its behaviour. This transformation is referred to as refactoring. For refactoring, identification of places in code to be refactored is crucial. However, this is time consuming and burdensome. This is due to the reason that anti-patterns can have different impacts on the software design. One solution to this problem is automated refactoring as a combinatorial optimization problem.

EARMO is an automated-refactoring approach for refactoring mobile apps while controlling for energy consumption. It targets two categories of antipatterns:

(i) anti-patterns that stem from common Object oriented design pitfalls i.e., Blob, Lazy Class, Long-parameter list, Refused Bequest, and Speculative Generality

(ii) anti-patterns that affect resource usages and in the Android documentation i.e., Binding Resources too early, Hash-Map usage, and Private getters and setters.

## **2. SYSTEM REQUIREMENTS**

Initially, the EARMO tool was run on a local machine with 8GB ram with HDD. However, the runtime was quite high for runs of  $n=2$  value. To mitigate the issue of slowness, several cloud VM instances with the following configurations were spawned.

- 12GB RAM
- 5GB SSD space
- Java Standard Edition Development Kit 1.8
- Dex2Jar 2.0 tool
- Tools required : Earmo Tool[2]

## **3. IMPLEMENTATION**

For the project, 60 apps were chosen from fdroid[12]. The range of years of apk published is from 2015 to 2017. During the selection of the apps, the category of the app was not considered. The complete list of apps can be seen in Table 1.

### **3.1 Generating Jar using dex2jar tool**

The earmo tool requires creating a .jar file from the android apk file. As a result, the tool dex2jar was used in this project to produce a .jar file for the given .apk file. The tool can be found in the link: <https://sourceforge.net/projects/dex2jar/>. The command to generate a jar file for the apk is given below.

**Windows: d2j-dex2jar.bat filename.apk**

**Ubuntu : sh d2j-dex2jar.sh filename.apk**

Sometimes, the dex2jar may not be able to successfully generate a .jar file. In that case, the apk was not considered and an alternative app was identified.

### 3.2 Configuring EARMO

To run EARMO, the RefactoringStandarStudyAndroid.prop file has to be used to set the parameters that are needed. The file contains different parameters to execute RefactoringStandarStudyAndroid.Jar. Some of the parameters configured in this project are as follows

- **pathProjecttoAnalyze:** set the absolute path of the generated .jar file
- **outputDirectory:** The output directory to store the results of JMetal framework
- **generateFromSourceCode:** set to **2** for analyzing .jar files.
- Specify the type of anti-patterns to detect. It is a comma-separated list of antipattern names.

**detectedAntipatterns:**

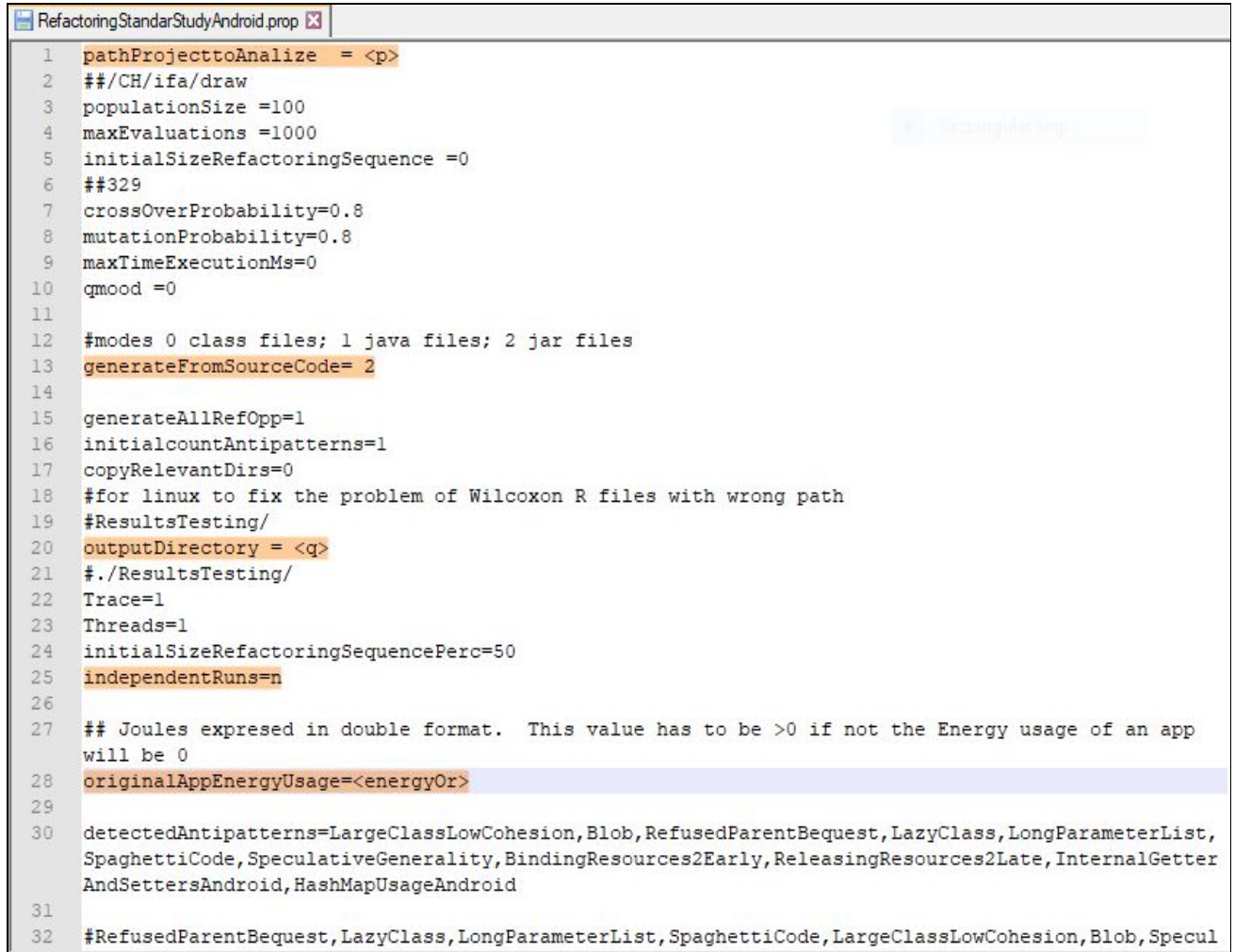
LargeClassLowCohesion,Blob,RefusedParentBequest,LazyClass,LongParameterList,SpaghettiCode,SpeculativeGenerality,BindingResources2Early,ReleasingResources2Late,InternalGetterAndSettersAndroid,HashMapUsageAndroid

- **independentRuns=n**, where n is the number of independent execution runs you want to perform. Since EARMO uses metaheuristics techniques which generate different solutions every time it is executed, we had to perform several runs in order to have statistical confidence in the results. In this project runs of value 2 and 3 were carried out to analyze each of the 60 apps
- **androidEnergyDeltas:** path to the deltas.txt energy consumption file in the earmo's root folder.  
Example: **androidEnergyDeltas=deltas.txt**
- **androidEnergyDeltas:** To set the initial energy consumed by the android application. with  $E_0$ 's path. Since hardware profilers and other software tools were not available to get the exact app's energy, different



values of  $E_0$  with one of the values provided in the author's paper were tried out.

A sample prop file is shown in the below figure.



```
1 pathProjecttoAnalyze = <p>
2 ##/CH/ifa/draw
3 populationSize =100
4 maxEvaluations =1000
5 initialSizeRefactoringSequence =0
6 ##329
7 crossOverProbability=0.8
8 mutationProbability=0.8
9 maxTimeExecutionMs=0
10 qmood =0
11
12 #modes 0 class files; 1 java files; 2 jar files
13 generateFromSourceCode= 2
14
15 generateAllRefOpp=1
16 initialcountAntipatterns=1
17 copyRelevantDirs=0
18 #for linux to fix the problem of Wilcoxon R files with wrong path
19 #ResultsTesting/
20 outputDirectory = <q>
21 #./ResultsTesting/
22 Trace=1
23 Threads=1
24 initialSizeRefactoringSequencePerc=50
25 independentRuns=n
26
27 ## Joules expressed in double format. This value has to be >0 if not the Energy usage of an app
28 will be 0
29 originalAppEnergyUsage=<energyOr>
30
31 detectedAntipatterns=LargeClassLowCohesion,Blob,RefusedParentBequest,LazyClass,LongParameterList,
32 SpaghettiCode,SpeculativeGenerality,BindingResources2Early,ReleasingResources2Late,InternalGetter
AndSettersAndroid,HashMapUsageAndroid
31
32 #RefusedParentBequest,LazyClass,LongParameterList,SpaghettiCode,LargeClassLowCohesion,Blob,Specul
```

Figure 2. RefactoringStandarStudyAndroid.prop file

### 3.3 Apps and their Categories

Following is the list of the 60 apps along their category and description, that were used for evaluation. Also, the count of categories of apps chosen is given in Figure 3

| APP NAME               | CATEGORY       | DESCRIPTION   |
|------------------------|----------------|---|
| Mycontacts             | Utilities      | It lets the user make direct phone calls straight from the widget itself without launching the dialer app first   |
| Uberspot               | Games          | Port of the 2048 game by Gabriele Cirulli   |
| Geometerplus           | Utilities      | An Ebook reader   |
| List My Apps           | Utilities      | Compiles a list of apps installed on the device   |
| SoundManager           | Utilities      | Exposes all the audio volume controls, provides quick access to ringmode and vibration settings, and allows user to set timers  |
| Scrabble               | Games          | Scrabble game   |
| Speedo                 | Utilities      | Measures user speed with GPS  |
| Sasabus                | Utilities      | SASAbus is the first application to consult the bus schedule of SASA (Städtischer Autobus Service AG) for all lines of the city of Bozen, Meran and Leifers (South Tyrol) |
| Ringdroid              | Utilities      | An application for editing and creating user own ringtones, alarms, and notification sounds   |
| Saymytexts             | Utilities      | An app that reads out loud the SMS user receive while a headset is plugged or a bluetooth is connected  |
| Minilens               | Utilities      | A Free Puzzle Platform Game   |
| Hostseditor            | Utilities      | An app to view and freely edit the /system/etc/hosts file on your device  |
| Unix Time Clock Widget | Widget & Theme | Adds a widget that displays Unix Epoch  |

|                       |                |  |
|-----------------------|----------------|--|
| Applications Info     | Utilities      | Monitor all available information about all installed applications or packages on a device   |
| Fdroid privileged     | Utilities      | It can make use of system privileges or permissions to install, update and remove applications on its own  |
| Amdroid               | Media          | It allows user to listen to media collection from anywhere phone has signal  |
| Android token         | Utilities      | Turning a mobile phone into a One Time Password (OTP) generation device which can be used in the place of hardware tokens                        |
| Apollo                | Media          | The music app that comes bundled with CyanogenMod ROM v10  |
| Binuaral beats        | Media          | Binaural Beats application helps user to relax and meditate  |
| Bitclock 16           | Widget & Theme | A clock with a 16 bit theme  |
| Boilr                 | Utilities      | Monitor Bitcoin, cryptocurrencies, cryptoassets, futures and options, and trigger alarms according to user settings. 90+ exchanges and counting. |
| Catalog               | Utilities      | Book cataloguing tool  |
| Character recognition | Utilities      | OCR software based on Tesseract library to perform character recognition on images selected from the gallery or captured from the camera.        |
| Cryptfs password      | Utilities      | This app lets user changes the Android disk encryption password  |
| Daily Money           | Utilities      | Record user daily expense, income, asset and liability, Show and count the detailsExport/Import to CSV Pie and time chart of balance             |

|                  |           |   |
|------------------|-----------|---|
| FBreader         | Utilities | An e-book reader. Features include the ability to stock up on books from online OPDS libraries like Project Gutenberg straight from the app   |
| Flashback        | Utilities | When a user receives a call the app displays the history of user interactions with the caller   |
| Free speech      | Utilities | Free Speech is a sound board for people who have problems speaking. When the user presses a button, a sound is played. Users can use recorded sounds, record new sounds, or generate sounds using the text-to-speech (TTS) built-into Android |
| HN               | News      | View articles in ViewText, Google, or user system browser, upvote stories Collapse  |
| Tryton           | Utilities | Client for the Tryton ERP system. It lets user access and edit enterprise information (accounting, invoice, production, stocks, and more) from user device  |
| ShoLi            | Utilities | Simple tool to edit shopping lists, and then to be a support for checking on those very lists   |
| SimpleDO         | Utilities | An app to track and manage todo items   |
| AlarmClock       | Utilities | An application that requires that user solve a math problem to deactivate the alarm   |
| AttendanceViewer | Utilities | It visualizes the attendance in a streamlined manner, along with additional details for each subject  |
| Calculator       | Utilities | Simple calculator   |
| KindMind         | Health    | An app that suggests kind actions that can be taken after the feelings & need have been identified  |

|                    |                |   |
|--------------------|----------------|---|
| LesserPad          | Utilities      | Simple memo pad that stores user text in a file on the SD card  |
| STK Addon          | Games          | The free and open-source cross-platform racing game and lists addons and displays some information and their images |
| Summation          | Utilities      | It adds values from a list of items. Each item has a label and a number and multiple lists are supported            |
| SW Journal         | Health         | It is a minimalistic workout tracker designed for those who know what they want and who are concentrated on results |
| TickMate           | Utilities      | It is basically a one bit journal. For each day, user can specify whether something occurred or not                 |
| ToDo               | Utilities      | Categorize and prioritize checklists, assign due dates and alarms   |
| Bodhi Timer        | Utilities      | An elegant minimalist count-down timer  |
| Easy Token         | Widget & Theme | It is a RSA SecurID compatible software authenticator with advanced usability features                              |
| Expr Eval          | Utilities      | It is an application that solves math expressions   |
| Simple light       | Utilities      | Use the device's camera to calculate proper exposure  |
| SimplyDo           | Utilities      | It is a simple shopping/TODO/task list manager  |
| WordPower MadeEasy | Education      | Vocabulary hundreds of word meanings chosen carefully from various SAT, GRE and GMAT course materials               |
| Little Sir Echo    | Utilities      | Listens for SMS/MMS package notifications, sets a user-defined timer to remind                                      |

|                    |           |  |
|--------------------|-----------|--|
| NF Card            | Utilities | Read contactless ic cards using the NFC sensor   |
| Preference Manager | Utilities | It allows user to edit these preferences in a simple and easy way  |
| BirthDroid         | Utilities | Birthdroid lets user keep track of birthdays via a widget or the app itself  |
| Easy RSS           | News      | An Android client for RSS services compatible with Google Reader API, such as the open source self-hosted project FreshRSS |
| Fake Dawn          | Utilities | Alarm clock that gradually increases brightness and sound volume to lead user out of deep sleep and wake user up gently    |
| Hangar             | Utilities | An app that provides a customizable shortcut launcher  |
| MPDroid            | Media     | An MPD client which supports streaming   |
| Sudowars           | Games     | Sudoku app to play Sudoku in multiplayer mode over Bluetooth against another person  |
| MatrixCalc         | Utilities | Square matrix calculator with support for addition, subtraction, multiplication, determinants, and inverses                |
| Cardgamescore      | Games     | An application to keep track of scores for games   |

**Table 1. List of apps chosen for evaluation**

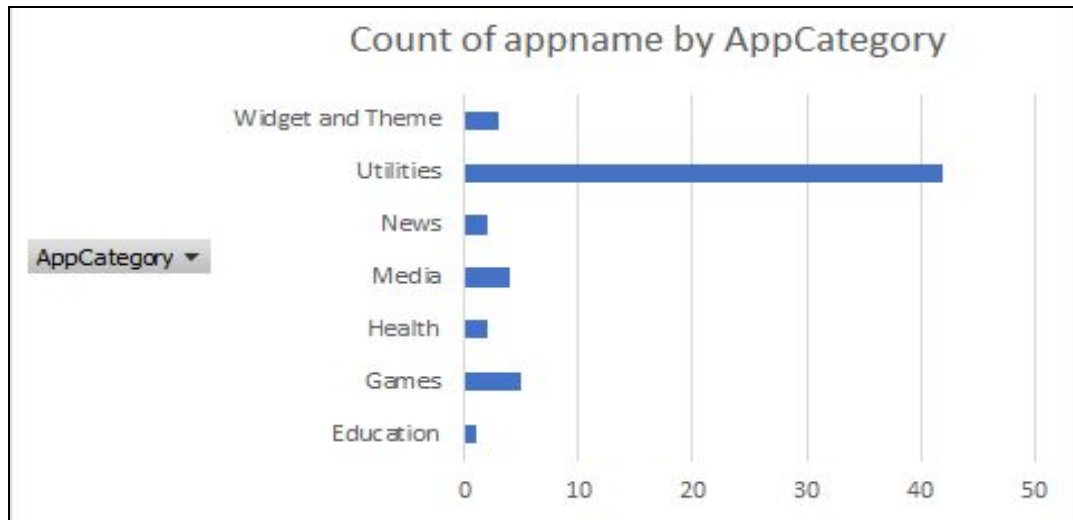


Figure 3. Count of apps in each category

## 4. RESULTS

On closer observations of values for runs of  $n=2$  and  $n=3$ , the values were extremely similar for both the number of runs. Hence, the results of  $n=2$  only are shown in subsequent graphs.

### 4.1 Graph with distribution of antipatterns detected by EARMO across 60 apps for $n=2$ .

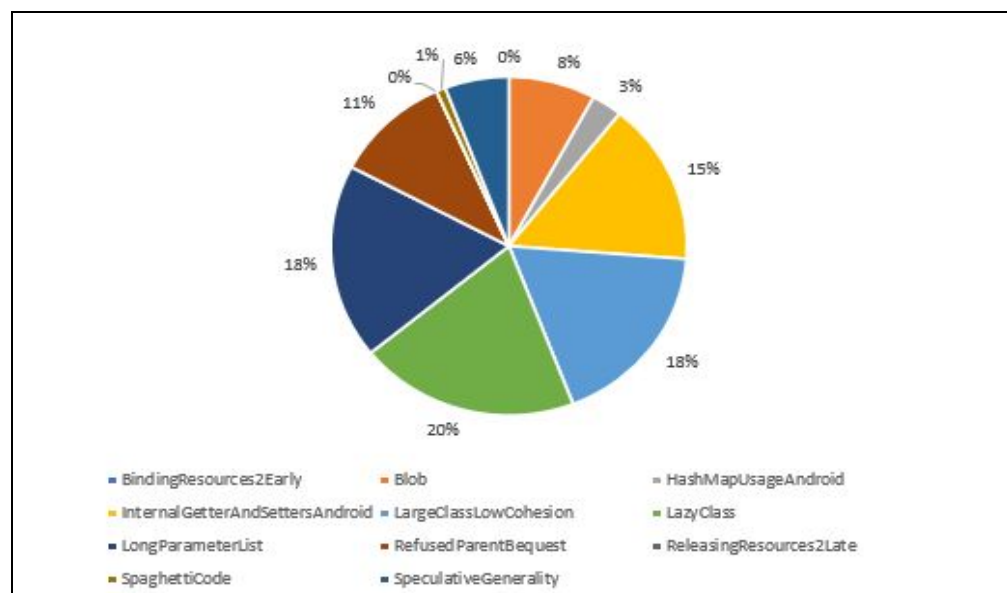


Figure 4. Distribution of antipattern across 60 apps

The below table provides the list of some anti-patterns detected by EARMO:

| Anti-Patterns               | Description   | Refactoring Strategy of EARMO   |
|-----------------------------|---|---|
| LazyClass                   | Small classes with low complexity that do not justify their existence in the system   | Inline class (IC). Move the attributes and methods of the LC to another class in the system.  |
| Long-parameter list         | A class with one or more methods having a long list of parameters, specially when two or more methods are sharing a long list of parameters that are semantically connected | Introduce parameter object (IPO). Extract a new class with the long list of parameters and replace the method signature by a reference to the new object created. Then access to this parameters through the parameter object |
| Speculative Generality      | There is an abstract class created to anticipate further features, but it is only extended by one class adding extra complexity to the design                               | Collapse hierarchy (CH). Move the attributes and methods of the child class to the parent and remove the abstract modifier  |
| Refused Bequest             | A subclass uses only a very limited functionality of the parent class   | Replace inheritance with delegation (RIWD). Remove the inheritance from the RB class and replace it with delegation through using an object instance of the parent class  |
| Binding Resources too early | Refers to the initialization of high-energy-consumption components of the device, e.g., GPS, Wi-Fi before they can be used  | Move resource request to visible method (MRM). Move the method calls that initialize the devices to a suitable Android event  |



|                             |  |  |
|-----------------------------|--|--|
| HashMap usage               | From API 19, Android platform provides ArrayMap which is an enhanced version of the standard Java HashMap data structure in terms of memory usage. it can effectively reduce the growth of the size of these arrays when used in maps holding up to hundreds of items    | Replace HashMap with ArrayMap (RHA). Import ArrayMap and replace HashMap declarations with ArrayMap data structure         |
| Blob                        | A large class that absorbs most of the functionality of the system with very low cohesion between its constituents   | Move method (MM). Move the methods that does not seem to fit in the Blob class abstraction to more appropriate classes     |
| Private getters and setters | Use of private getters and setters to access a field in a class decreasing the performance of the app because of simple inlining of Android virtual machine that translates this call to a virtual method called, which is up to 7 times slower than direct field access | Inline private getters and setters (IGS). Inline the private methods and replace the method calls with direct field access |

**Table 2. List of anti-patterns detected by EARMO**

## **4.2 Distribution of antipatterns across each set of 20 apps**

We can observe from the graphs that the top 3 predominant antipatterns are LongParameterList, LazyClass & InternalGettersAndSettersAndroid. As per android coding principles, avoiding InternalGettersAndSettersAndroid has an impact on energy as well as program execution time and developers can take care of it.

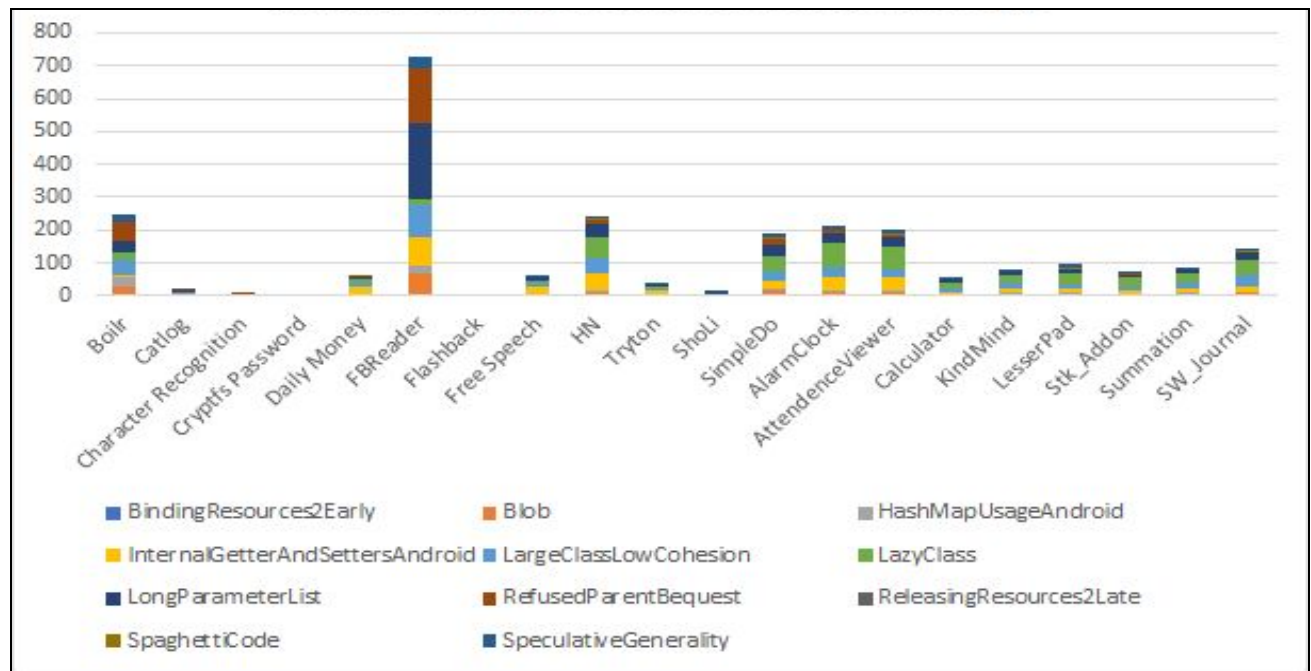


Figure 5. Anti patterns distribution across 20 apps

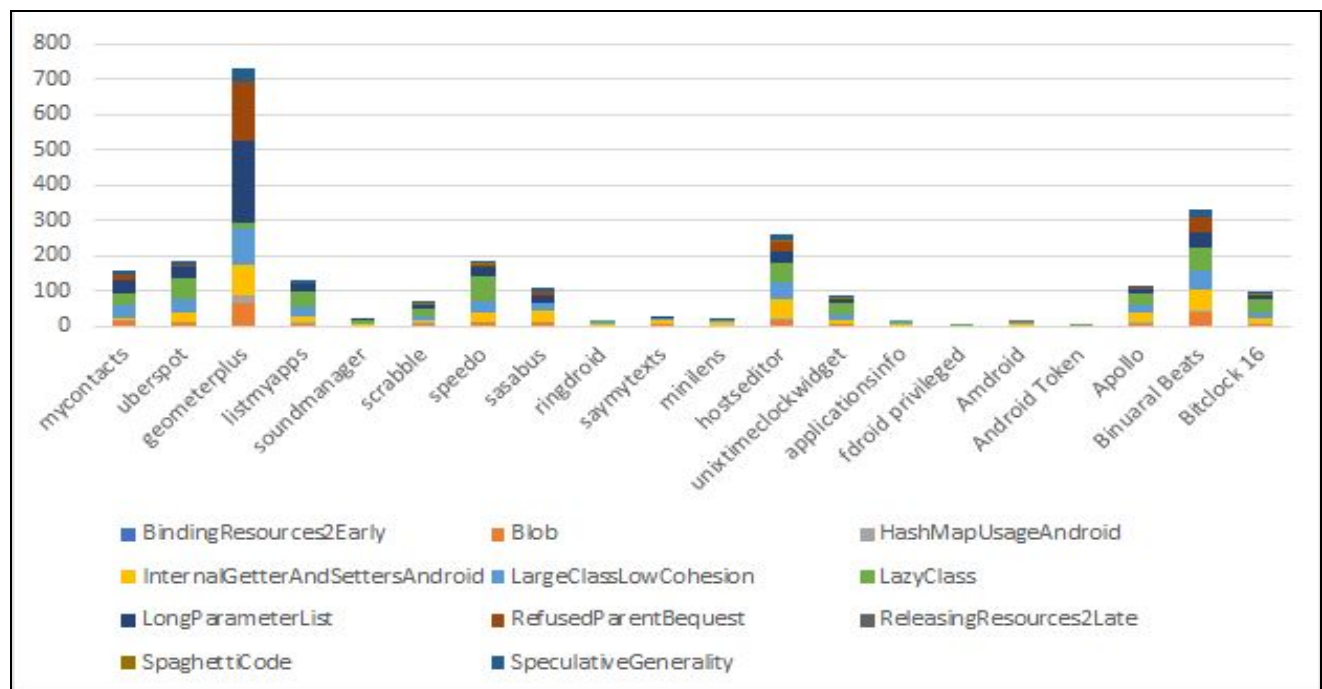


Figure 6. Anti patterns distribution across next set of 20 apps

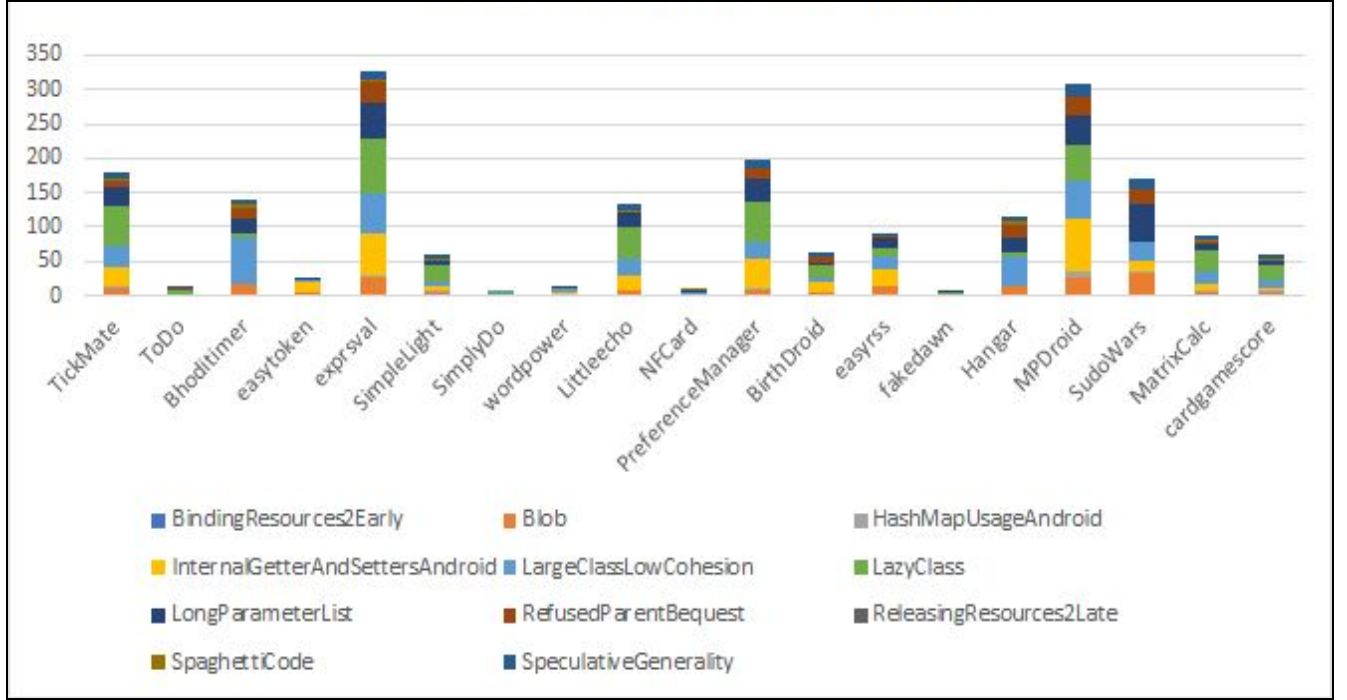


Figure 7. Anti patterns distribution across the final set of 20 apps

### 4.3 Distribution of the Metaheuristics across 60 apps

EARMO comprises 3 metaheuristics namely MOCcell, NSGA-II and SPEA2. Each of them offer refactoring suggestions as well as energy consumed after refactoring. The energy after refactoring is given by Equation 1. In our project, 3 different values of  $E_0$  were used : 21.28127, 20 and 25. From the graphs, it can be seen that the energy after refactoring the apps is negative. This is impossible since the app cannot provide energy to the smartphone. Upon inspection, we found that EARMO was trying to refactor the android v4/v7 compat libraries which ideally should not have been done by the EARMO. This may be due to the fact that the libraries were not included as .jar and was directly added as code in the app.

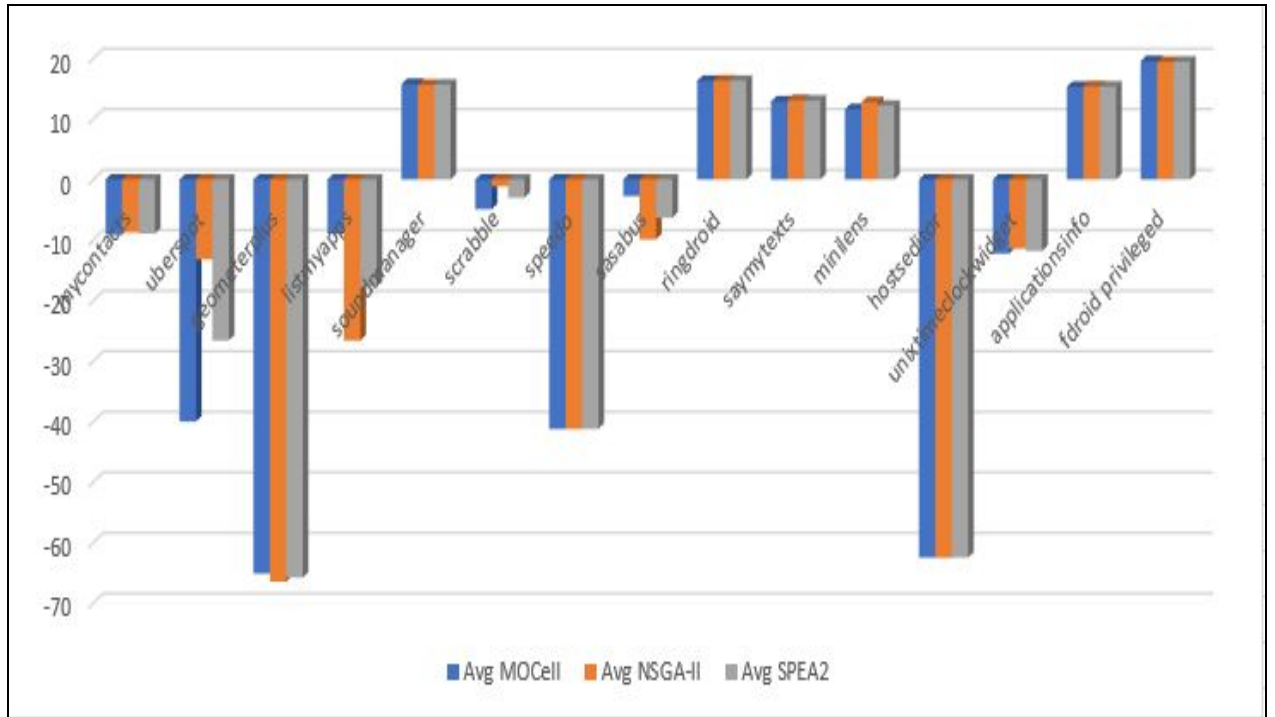
$$EC(a) = E_0 + \sum_{i=1}^n E_0 \times \delta EC(r_i)$$

Equation 1. Energy consumption of an app after refactoring process.

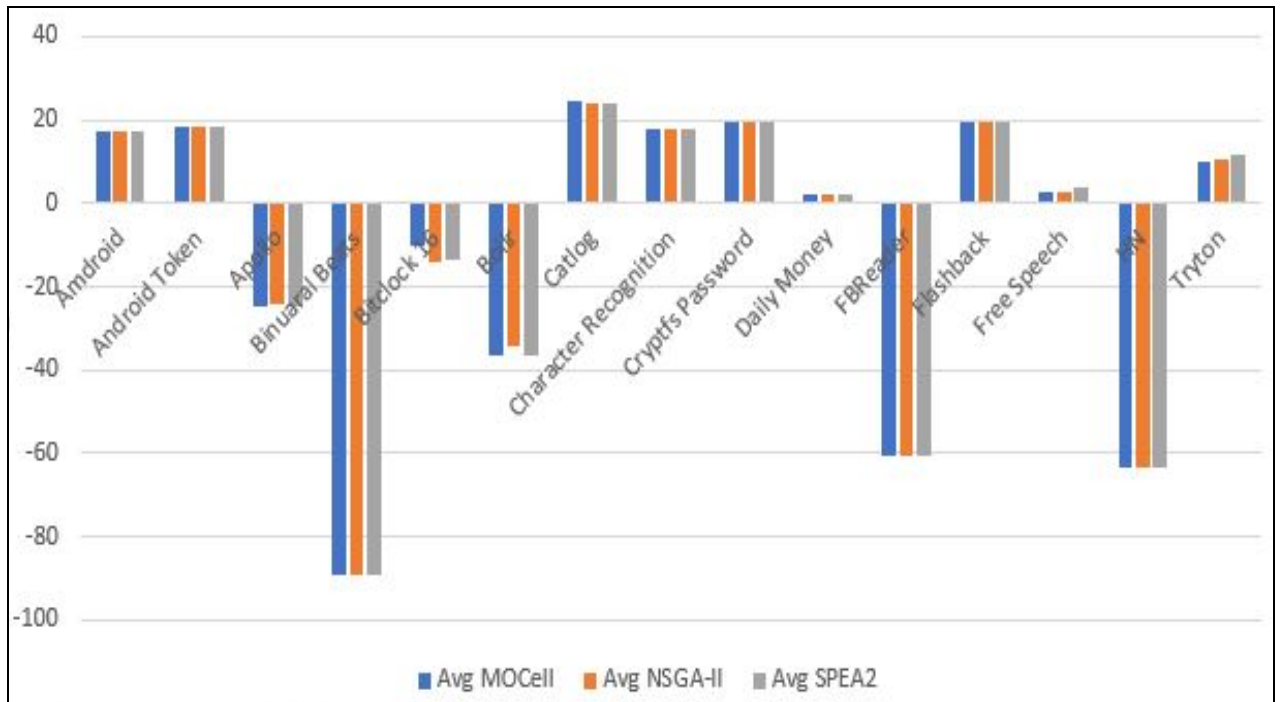
Since the number of refactorings are high and due to the fact that refactoring coefficients have negative values(Figure 8), the energy consumed after refactoring goes negative. The energy increment (EI) is given by the formula below (Equation 2) where  $EC(a)$  is the energy consumption of an app  $a$  and  $EC(a) \geq 0$ . As this is a change-in-percentage computation, initially  $EC(a) = E_0$  and with Equation 1,  $E_0$  does not impact the final value of  $EI(a)$ .

$$EI(a) = \frac{EC(a') - EC(a)}{EC(a)} \times 100.$$

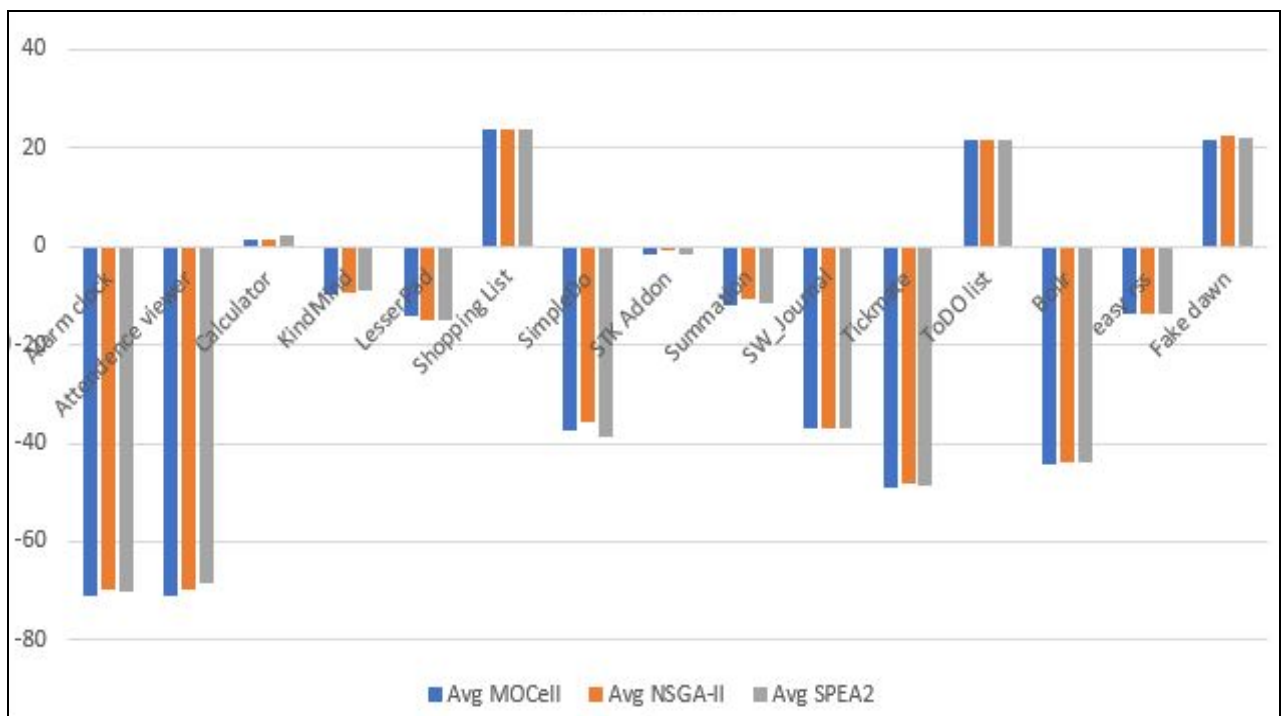
**Equation 2. Energy increment of an app after refactoring process.**



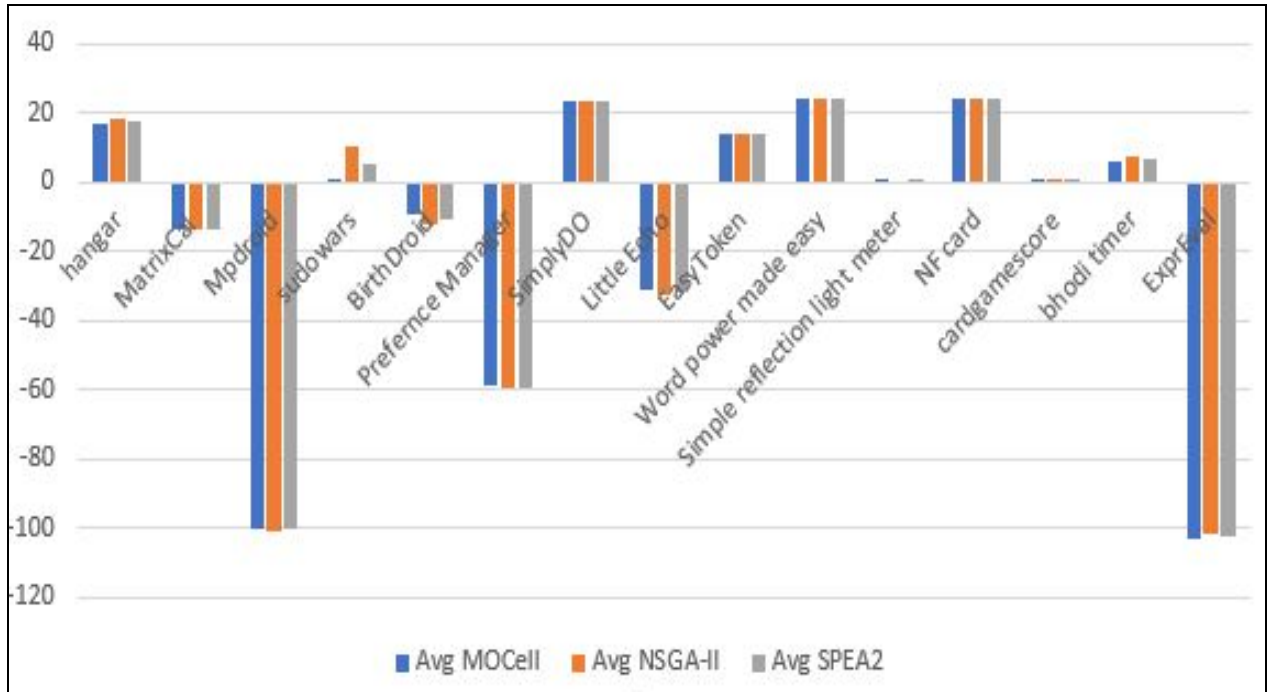
**Figure 8. Distribution of MOCcell, NSGA-II and SPEA2 across 15 apps with  $E_0 = 21.28127$**



**Figure 9. Distribution of MOCell, NSGA-II and SPEA2 across 15 apps with  $E_0=20$**



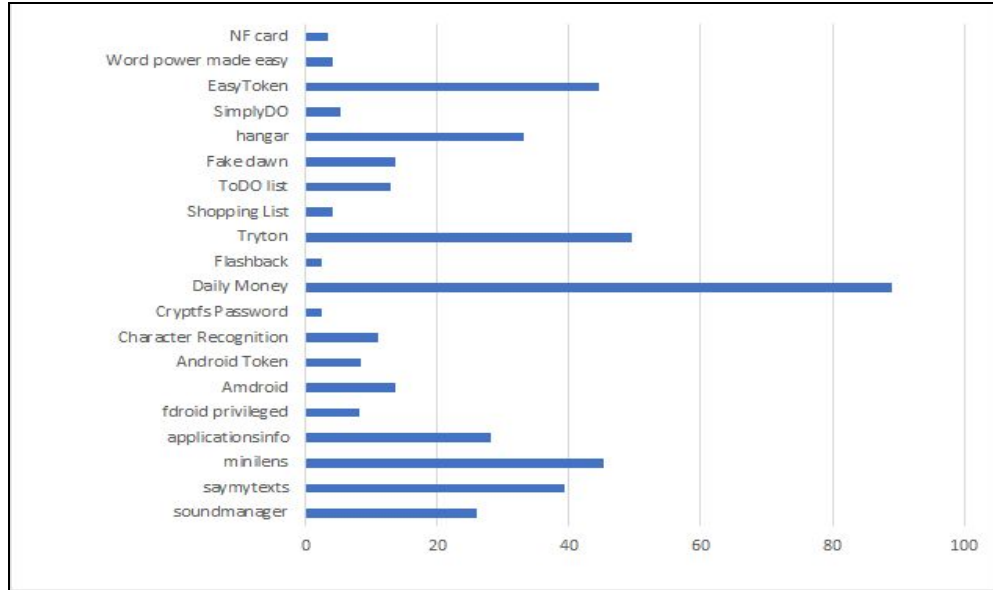
**Figure 10. Distribution of MOCell, NSGA-II and SPEA2 across 15 apps with  $E_0=25$**



**Figure 11. Distribution of MOCeII, NSGA-II and SPEA2 across 15 apps with  $E_0=25$**

#### 4.4 Apps with positive refactored energy increment

For the following 20 apps, android.support libraries were not refactored. In the case of DailyMoney, there are 57 antipatterns, 37 for Tryton and 27 for EasyToken. The details of the count of antipatterns of each app can be found in our github repo[10]. The antipatterns detected were only for the app's source code. As per EARMO, negative values of EI are desired. In the chart below, absolute value was taken for better representation. For the rest of the apps, an interesting find is that the support libraries were refactored which is not desired and causes a negative value in energy after refactoring.



**Figure 12. Energy Increment as percentage**

## 4.5 Fitness Report

A sample fitness report of one execution of MOCell algorithm is shown below:

```

FitnessReport-MOCell-0-p50 - Notepad
File Edit Format View Help
Number of classes in the original system: 121
Number of refactoring opp. 554
Original anti-pattern count
9 solutions for LargeClassLowCohesion in in 667 ms.
4 solutions for Blob in in 11 ms.
1 solutions for RefusedParentBequest in in 20 ms.
22 solutions for LazyClass in in 12 ms.
7 solutions for LongParameterList in in 16 ms.
1 solutions for SpaghettiCode in in 28 ms.
6 solutions for SpeculativeGenerality in in 8 ms.
0 solutions for BindingResources2Early in in 10 ms.
0 solutions for ReleasingResources2Late in in 5 ms.
3 solutions for InternalGetterAndSettersAndroid in in 6 ms.
3 solutions for HashMapUsageAndroid in in 7 ms.
OriginalFitness: 0.9628099173553719
fitness:-0.996694214876033 1.8451424249998574 total Nodes:239 ref.success:239 ref.failed:0
Num. of evaluations: 1000
Best solution defect count
7 solutions for LargeClassLowCohesion in results-MOCell-0-p50 in 402 ms.
1 solutions for Blob in results-MOCell-0-p50 in 6 ms.
0 solutions for RefusedParentBequest in results-MOCell-0-p50 in 12 ms.
0 solutions for LazyClass in results-MOCell-0-p50 in 4 ms.
1 solutions for LongParameterList in results-MOCell-0-p50 in 9 ms.
0 solutions for SpaghettiCode in results-MOCell-0-p50 in 3 ms.
2 solutions for SpeculativeGenerality in results-MOCell-0-p50 in 7 ms.
0 solutions for BindingResources2Early in results-MOCell-0-p50 in 3 ms.
0 solutions for ReleasingResources2Late in results-MOCell-0-p50 in 3 ms.
0 solutions for InternalGetterAndSettersAndroid in results-MOCell-0-p50 in 3 ms.
0 solutions for HashMapUsageAndroid in results-MOCell-0-p50 in 2 ms.
Sequence:491, 81, 71, 199, 520, 303, 473, 439, 277, 114, 466, 38, 217, 158, 287, 44, 241, 48, 210
, 68, 94, 17, 396, 399, 193, 115, 146, 30, 156, 413, 160, 427, 10, 74, 132, 53, 463, 424, 166, 23

```

**Figure 13. FitnessReport for MOCell**

If the parameters are correctly set, this file will be created in the root folder of the app on which EARMO runs. The important line in the fitness report is:

**fitness:-0.996694214876033    1.8451424249998574    total    Nodes:239**  
**ref.success:239 ref.failed:0**

The second real number after the space is the estimated energy consumption of the app in Joules after applying the refactoring sequence. **total Nodes** is the number of refactorings in the sequence.

## **5. AUTOMATING EARMO RUNS & PROCESSING THE RESULTS**

### **5.1 Automating EARMO runs**

A NodeJS script was written that takes inputs as the apk,  $E_0$  and number of runs. The Jar is created for the inputted apk file, prop file is modified and EARMO refactoring script is run.

### **5.2 Processing the Results**

A python script was written to read the generated files for anti-patterns and create a consolidated \*.csv file.

### **5.3 Manual Steps**

Calculating the average values for MOCell, NSGA-II and SPEA2 and generating the graphs were done manually.

The source code can be found in our github repository[10]

## **6. FINDINGS**

During the analysis of results after every run, the energy after refactoring turned out to be negative in some cases and positive in few. The reason has to do with the refactoring count and the refactoring coefficients value. In Equation 1,  $E_0$  value inside



the summation is constant and can be moved outside the summation. Then,  $E_0$  is a common term in the first and second operand of the formula. Upon checking various refactoring sequences manually, we found that the refactoring sequence plays a large role in determining the energy after refactoring. Because the formula tends to be

$E_0(1 + \sum_{i=1}^n \delta(r_i))$  no matter how small and large the value of  $E_0$  is, the result depends on the refactoring type and their coefficients. This aspect was not covered in the authors' paper.

## **7. THREATS TO VALIDITY**

Some of the threats to validity concerning the evaluation of EARMO on android apps are as follows:

### **A. Construct validity threats**

- Due to the possible mistakes in the detection of anti-patterns when applying refactorings, there might be a concern in relation theory and observation. The tool results does not guarantee that all possible anti-patterns are detected and that all those detected are true anti-patterns[1].
- Manual effort is needed to validate the outcome of the refactorings performed in the source code to ensure that the output values correspond to the changes performed.

### **B. Threats to internal validity**

- This threat concerns the selection of anti-patterns, tools and analysis methods.
- EARMO evaluation uses Monkeyrunner to communicate with apps through simulated signals rather than signals triggered through real sensors on mobile devices. This can be considered as not realistic and thus rises a need for the use of a more realistic measurement.
- Energy after refactoring could have been more accurate if a hardware profilers or software based profiler that provides the app's energy were available.

## **8. SCOPE FOR IMPROVEMENT**

In this project hardware profilers were not used. A hardware profiler provides higher accuracy of app's energy by analyzing the voltage during the app's usage. Using a hardware profiler and running EARMO and a comparison with a software based approach can reveal more insights. Also, manually refactoring the app and verifying the results of EARMO would give an idea about the accuracy of EARMO.

## 9. REFERENCES

- [1] R. Morales, R. Saborido, F. Khomh, F. Chicano and G. Antoniol, "EARMO: An Energy-Aware Refactoring Approach for Mobile Apps," in IEEE Transactions on Software Engineering, vol. 44, no. 12, pp. 1176-1206, 1 Dec. 2018, doi: 10.1109/TSE.2017.2757486.
- [2] EARMO implementation wiki - <https://github.com/moar82/EARMO/wiki>
- [3] M. U. Farooq, S. U. Rehman Khan and M. O. Beg, "MELTA: A Method Level Energy Estimation Technique for Android Development," 2019 International Conference on Innovative Computing (ICIC), Lahore, Pakistan, 2019, pp. 1-10, doi: 10.1109/ICIC48496.2019.8966712.
- [4] Di Nucci, Dario & Palomba, Fabio & Prota, Antonio & Panichella, Annibale & Zaidman, Andy & Lucia, Andrea. (2017). PETrA: A Software-Based Tool for Estimating the Energy Profile of Android Applications. 10.1109/ICSE-C.2017.18.
- [5] L. Cruz, R. Abreu and J. Rouvignac, "Leafactor: Improving Energy Efficiency of Android Apps via Automatic Refactoring," 2017 IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems (MOBILESoft), Buenos Aires, 2017, pp. 205-206, doi: 10.1109/MOBILESoft.2017.21.
- [6] L. Cruz and R. Abreu, "EMaaS: Energy Measurements as a Service for Mobile Applications," 2019 IEEE/ACM 41st International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER), Montreal, QC, Canada, 2019, pp. 101-104, doi: 10.1109/ICSE-NIER.2019.00034.
- [7] <https://developer.android.com/studio/profile/traceview.html>
- [8] <https://developer.android.com/studio/profile/battery-historian.html>
- [9] <https://developer.android.com/studio/profile/systrace-commandline.html>
- [10] Earmo Automator - <https://github.com/suhan0694/earmo-node-shell>

[11] Earmo Results - <https://github.com/suhan0694/earmo-results-60-apps>

[12] <https://f-droid.org/>