Saving Energy on Mobile Devices by Refactoring

Marion Gottschalk[1], Jan Jelschen[2], Andreas Winter[2]

Abstract

Energy-efficiency is an import topic in information and communication technology and has to be considered for mobile devices, in particular. On the one hand, the environment should be protected by consuming less energy. On the other hand, users are also interested in more functionality of their mobile devices on hardware and software side, and at the same time, longer battery durations are expected. This paper shows an approach to save energy on the application level on mobile devices. This approach includes the definition, detection, and restructuring of energy-inefficient code parts. Energy savings are validated by different energy measurement techniques.

1. Motivation

Mobile devices and the mobile communication network are big energy consumer of information and communication technologies (ICT) in Germany [1] and their rising sales volume in the last years shows their significance in ICT [2]. Meanwhile, the energy consumption of mobile devices and the mobile network amounts to 12.9 PJ in Germany per year. This corresponds to the complete energy production of the atomic power plant Emsland [3] within four months. This amount shows that energy saving for mobile devices is also important to reduce the energy consumption in Germany, and hence, to protect the environment. In addition, the scope of mobile devices is increasing, due to new requirements, more powerful processors, and a broad variety of different apps provided on mobile devices. All these functionalities have an influence on battery duration and battery lifetime. This paper focuses on supporting developers to save energy on application level and does not consider the OS, e.g. selecting the most energy efficient network connection.

Therefore, an approach for saving energy on mobile devices on application level is described. Software evolution techniques, such as reverse engineering and reengineering, are used. *Reverse engineering* describes approaches to create abstract views of software systems to enable efficient code analyses. *Software reengineering* is a process to improve the software quality by reconstituting it in a new form without changing its functionality [4]. In this work, the energy consumption of existing Android apps [5] is changed by code analysis and transformation, which do not change apps' behavior. The process of code analysis and transformation is called *refactoring* in this work. The approach starts with defining energy-inefficient code parts called *energy code smells*. For these, a reengineering process [6] is described and validated by different, software-based energy measurement techniques [7]. Aim of this work is the collection and validation of energy code smells and corresponding refactorings which are presented in form of a catalog.

This paper is structured as follows: Firstly, the reengineering and the evaluation process to show energy savings by refactoring is described in Section 2. Secondly, in Section 3 validated energy code smells are described, in which the energy code smell *Data Transfer* (c.f. Section 3.1) and its restructuring are presented in more detail. Thirdly, measurement results of some energy code smells are summarized in Section 4. Finally, Section 5 concludes this paper with a summary and an outlook.

---

[1] OFFIS e.V., 26121 Oldenburg, Germany, gottschalk@offis.de, Division Energy, Architecture Engineering and Interoperability

[2] University of Oldenburg, 26121 Oldenburg, Germany, {jelschen, winter}@se.uni-oldenburg.de, Department of Computing Science, Software Engineering

2. Basic Techniques

Energy saving on mobile devices by refactoring needs two fundamental steps. Firstly, a *reengineering process* must be defined to perform the bad smell detection and the code restructuring. Secondly, the benefit (energy saving) for the refactoring must be shown by an *evaluation* which includes measurement techniques, checked smartphones and apps, and settings during the measurements.

2.1. Reengineering Process

The process to generate more energy-efficient code is depicted in Figure 1. It describes the platform-independent process from an energy-inefficient app to a more efficient one. The process starts with reverse engineering for the Java code of an app. In Step 1, the code is parsed into an abstract representation which conforms to a Java meta-model [8] and it is stored into a central repository to execute efficient analyzes. Android apps are parsed according the Java meta-model [5]. In the Steps 2 and 3, the refactoring by analyzing and restructuring the abstracted code is done and saved. The analysis is done by static code analysis which has proved itself in software evolution, to identify energy code smells. Restructure changes the abstract view without changing the intended functionality. In the last Step 4, the abstract view is parsed back into code (Android app) which can be compiled and executed.

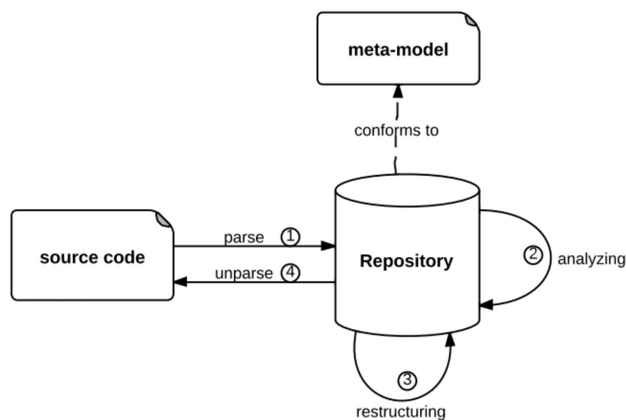

Figure 1: Reengineering Process [6]

The abstract view of apps' code is provided by TGraphs which can represent each programming language by nodes and edges. TGraphs are directed graphs, whose nodes and edges are typed, attributed, and ordered [9]. Thanks to the SOAMIG project [8], the tooling for the Java programming language is available and used for this work. This abstract view allows performing efficient analyses and transformations by static analyses and graph transformations. Therefore, GReQL (Graph Repository Querying Language) [8] and the JGraLab API [10] are used. GReQL can be used to extract static information and dependencies in source code represented as TGraphs. The JGraLab API implements the GReQL approach and provides means for graph manipulation. An example of the analysis and restructuring by JGraLab is shown in Section 3 for explaining the reengineering process in more detail.

2.2. Evaluation Process

After the refactoring, the energy improvement of refactored apps must be checked. Therefore, an evaluation process is created which includes the usage of different software-based *energy measurement techniques*, *hard-* and *software*, and the *measurement procedure* involving the duration and the mobile devices' settings during the energy measurement.

2.2.1. Energy Measurement Techniques. The approach uses three software-based energy measurement techniques: *file-based*, *delta-B*, and *energy profiles* [7]. These techniques are realized by the Android app Andromedar [11] which is used for validating the energy consumption of apps before and after refactoring. The *delta-B* technique calculates the energy consumption through the battery level, which can be read out with the Android BatteryManager API [12]. If the battery level changes, Andromedar calculates a new value of the current energy consumption by using the

BatteryManager API. The *system-file* technique uses an internal system file which is implemented by some mobile device vendors, only. It stores information about the current battery discharge and battery voltage. This information is used by Andromedar to calculate the energy consumption. *Energy Profiling* uses an internal XML file, which is created by the vendor or someone who has detailed information about the mobile devices' hardware components. This XML file gives information on the average power consumption of each hardware component built into the mobile device. In this case, the energy consumption is not measured directly, but the active time of all hardware components, i.e. Andromedar measures the time between switching components on and off. After measuring, the active time of each component in each particular state is multiplied with its average power, from which the energy consumption results.

2.2.2. Hardware. For validation, two mobile devices are used: aHTC One X [13] and a Samsung Galaxy S4 [14]. The HTC One X is used for the validation of all energy refactorings and the Samsung Galaxy S4 is only used for validating the *Backlight* energy refactoring (cf. Section 3.2). Both devices use the Android OS which is necessary to apply the reengineering process with the Java meta-model and the energy measurements by Andromedar. The HTC One X uses Android 4.1.1 with HTC Sense 4+. It has a 4.7 inch HD, super LCD 2 screen with a resolution of 1280x720 pixels. The Samsung Galaxy S4 uses Android 4.2.2 with TouchWiz. The screen is a 5 inch Full HD Super AMOLED screen with a resolution of 1920x1080 pixels. This information is listed to give an overview about considered components during the energy measurements in Section 4.

2.2.3. Applications. For validation, two Android apps are chosen: *GpsPrint* [15] and *TreeGenerator*. GpsPrint is a free Android app with about 1642 LOC. It locates the position of a mobile device and seeks via Internet for an address. TreeGenerator is a self-written Android app for this validation with about 345 LOC. Each second it displays another name of a tree, its type, and a picture of it. Both apps display advertisements at the bottom of the screen. GpsPrint is used for the energy refactorings: *Third-Party Advertisement* and *Binding Resources Too Early* (paragraph on Section 3). TreeGenerator is used for the refactorings: *Third-Party Advertisement*, *Statement Change*, *Backlight* and *Data Transfer* (paragraph on Section 3).

2.2.4. Measurement Procedure. The evaluation process for each energy refactoring is the same; hence, a short overview about the process is given. For each energy refactoring, ten energy measurements are done before and after the refactoring and an average is created to compare the results. Each energy measurement has a duration of two hours. To guarantee correct, comparable, and reproducible results, the mobile devices' settings should be the same during the different energy measurements. This includes *screen*, *notification*, *application*, *wireless and network*, and *gesture* settings.

Screen: The screen is permanently on but on lowest brightness because apps only run when the screen is active, and the lowest brightness is chosen to reduce and to normalize the energy consumption of the screen. Auto rotation of the screen is turned off, because the rotation sometimes interrupts the GPS connection. This was observed by first measurements with GpsPrint (2.2.3).

Notification: Automatic notifications are set off, such as email client, play store, calendar, etc. This is necessary to make comparable energy measurements which are not interrupted by individual notifications, such as emails or updates.

Application: All background apps, such as HTC Services, Google Services, 7digital, etc. are set off, so that only the tested app is running and energy consumption is not affected by further apps. Also, the power saver of the OS is turned off.

Wireless and network: The modules mobile data, blue-tooth, GPS, NFC, and Wi-Fi are set to off, if they are not needed.

Gesture: Additional gestures, such as the three finger gestures, are turned off to reduce the error rate when energy measurements are started, e.g. unwanted touches on the screen which makes repeatable measurements difficult, so the user interface is not viewed.

Miscellaneous: The HTC does not have a SIM card installed, with the result that the HTC seeks for it, permanently. Additional measurements checks the influence of the missing SIM card, and it shows that the influence is very small [16]. Also, no further apps are installed apart from the standard apps, the tested app, and Andromedar.

3. Energy Refactoring Catalog

The energy refactoring catalog includes a list of five energy code smells which are validated: *Third-Party Advertising*, *Binding Resources Too Early*, *Statement Change*, *Backlight*, and *Data Transfer* [16]. This paper describes one of the energy code smells in more detail, *Data Transfer*, and introduces further five which are described in more detail in [16] or [19]. The aim is to demonstrate possible areas of wasting energy by bad programming and strategies to improve apps' energy behavior.

The detailed description of *Data Transfer* follows a template which is developed in [16]. The template includes: a name, a definition, a motivation, constraints, an example, an analysis, a restructuring, and an evaluation. Thus, a consistent presentation of the energy code smells is ensured. A part of the template is derived from the description of code smells by Fowler et al. [17].

3.1. Energy Code Smell: Data Transfer

Description: Data Transfer refers to loading data from a server via network instead of reading pre-fetched data from the app's storage [18].

Motivation: Many apps use data, such as images, videos, and sound effects. Programmers can decide whether these data are stored on the apps' storage or on an external server. The measurements by [18] show that the usage of an external server needs more energy than the local one. Additionally, Android apps offer a further possibility to access data. Data are stored on an external server and are loaded into apps' cache during their runtime. While the cache is not cleared, the data do not have to be loaded again from the server. Hence, three variants exist: (1) data are loaded from server (every time), (2) data are stored on the mobile device, and (3) data are stored in the cache. These variants are considered in the following description and evaluation.

Constraints: The variants (2) and (3) request storage on the mobile devices while an app is installed. If memory size of the data is big, users will get a problem when installing many data-intensive apps. Variant (1) needs less storage on mobile devices, but requests high data traffic to load all data each time. If the data access of apps should be changed, many changes must be done. At first, the data must be collected to upload them to a server (for variant (1) and (3)) or to download and integrate them into an app (for variant (2)). For all variants, different Android APIs exist which must be known to detect this energy code smell.

Example: Example source code (TreeGenerator) showing the "*Data Transfer*" bad smell is demonstrated in Figure 2. It shows on the left side the code for loading data from server (1), and on the right side data is read from the memory card (2). The code on the left side loads data from the server in line 8 and 11 by the method `image(url, applicationCache, deviceCache, size, alternativeFile)`. If the Boolean `applicationCache` on the left side is changed, the usage of app's cache will be possible (3). The code on the right side calls the image files from the storage in line 8 and 12 by the method `getDrawable(file)` which contains the path to the image on the storage.

```
1  @Override
2  public void run() {
3     i = (int) (Math.random() * 51);
4     AQuery aq = new AQuery(pic);
5     //tree list
6     if(i == 1) {
7        value.setText("Nikko-Tanne");
8        aq.id(pic).image("http://
              mgottschalk.eu/img/bilder/
              nikko.jpg", false, false, 200,
               R.drawable.ic_launcher);
9     } else if(i == 2){
10       value.setText("Riesen-Tanne");
11       aq.id(pic).image("http://
              mgottschalk.eu/img/bilder/
              riesentanne.jpg", false, false
              , 200, R.drawable.ic_launcher)
              ;
12    }
13    [...]
14 }
```

```
1  @Override
2  public void run() {
3     i = (int) (Math.random() * 51);
4     Resources res = getResources();
5     //tree list
6     if(i == 1) {
7        value.setText("Nikko-Tanne");
8        Drawable picture = res.getDrawable
              (R.drawable.nikko);
9        pic.setImageDrawable(picture);
10    } else if(i == 2){
11       value.setText("Riesen-Tanne");
12       Drawable picture = res.getDrawable
              (R.drawable.riesentanne);
13       pic.setImageDrawable(picture);
14    }
15    [...]
16 }
```

Figure 2: Example for Data Transfer [16]

Analysis: Apps must be detected which use the `image` method of the `AQuery` API to know which part within the app must be changed. Therefore, a GReQL query is used. The query seeks for a class (`AQuery`) and an access (`image()`) vertex within the TGraph which is created by parsing the code from Figure 2 (left). These vertices must be a part of the same block and must be called in a specific order.

```
1  from cache : V{frontend.java.Literal},  block : V{frontend.java.Block}, image : V{
       frontend.java.Access}, aquery : V{frontend.java.Class}
2  with cache.value = "false" and aquery.name = "AQuery" and
       cache <--{frontend.java.HasOperand} ... block ->frontend.java.BlockContainsStatement
       ... aquery  and image <--{frontend.java.HasOperand} ... block
3  report cache
4  end
```

Figure 3: Query for DataTransfer [16]

Figure 3 shows the GReQL query (Section 2) which is used to detect this energy code smell. The query consists of three parts: `from`, `with`, and `report`. In the `from` part vertices and edges are defined which are needed to detect the energy code smell, e.g. the vertex `cache` of the type `Literal` must exist to detect the attribute `applicationCache` of the `image` method (Figure 2). The second part is the `with` part, which includes the content of the query. Firstly, the known values of the vertices are checked (in line 2: `cache.value = "false"` and `aquery.name = "AQuery"`). Secondly, the link between these vertices is checked by `cache <--{frontend.java.HasOperand} ... block`. This means, that the vertex `cache` is an operand of a further vertex which is connected via further edges with the `block` vertex. The last part is the `report` part which defines the vertex which will be returned.

Restructuring: If the query detects such a code part, the restructuring will start. Due to the complexity (saving images locally, calling each image) of the changes, this restructuring must be done manually from variant (1) to (2). The restructuring from variant (1) to (3) can be done automatically with the JGraLab API. Therefore, nearly the same query can be used, only a further vertex for the Boolean `applicationCache` must be added which must be detected and changed to `true`.

Evaluation: Figure 4 shows the energy consumption of the TreeGenerator app in all variants. Variant (1) (data are loaded from server) consumes the most energy with 4232 J in two hours. The difference to variant (2) (data is stored locally) with a switched-off Wi-Fi module amounts 368 J. If the Wi-Fi module is not switched-off, the difference is very small which shows that the data traffic does not have a real influence on the switched on Wi-Fi modul. Variant (3) (data is stored in app's cache) also saves energy which amounts 224 J in two hours. The results of the other two energy measurement techniques (delta-B and energy profile) are summarized in Figure 5. They show the same trend such as the file-based technique.

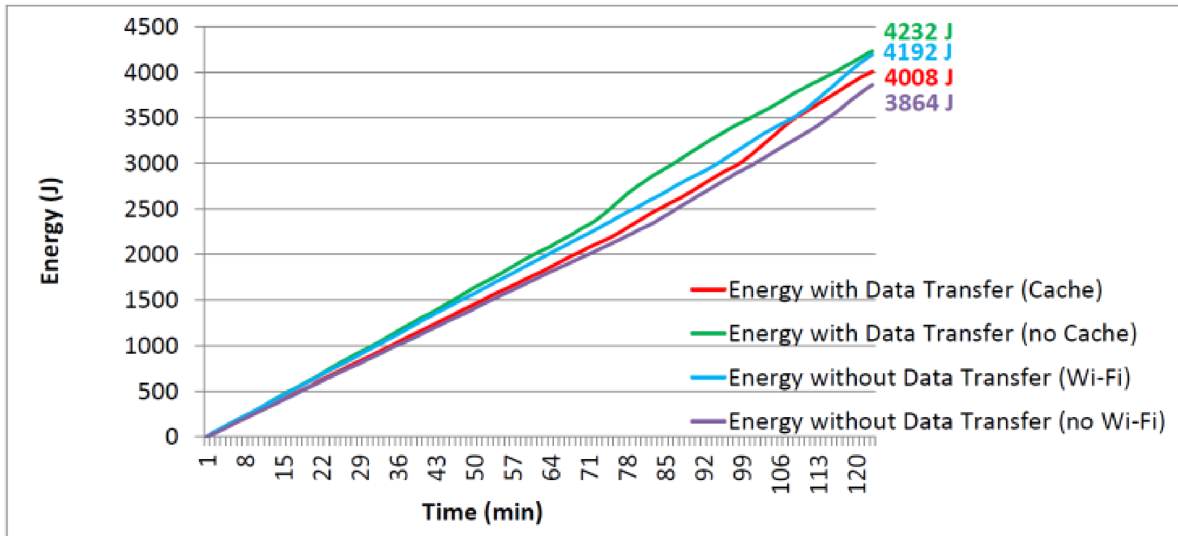

Figure 4: File-based Measurement [16]

3.2. Further Energy Code Smells

The following energy code smells are explained briefly to give an overview of further areas for saving energy. A more detailed descriptions of the first four can be found in [16]. However, their energy savings are summarized in Section 4. The last energy code smell is described in [19].

Third-Party Advertisings are integrated code parts within apps which display advertisements during operation. Thereby, advertisements do not have an influence on apps' functionality, but might consume energy through 3G or Wi-Fi connection [20]. If advertisements are deleted, programmers might have to change their business model, but the main functionality remains unaffected.

Binding Resources Too Early refers to hardware components, such as Wi-Fi and GPS, which are switched on by apps at an early stage when they are not yet needed by the app or user [21].

Statement Change describes alternative programming statements, such as `if` and `switch`, which can be substituted with each other, because they have the same functionality, but potential different energy consumptions [22].

Backlight refers to the background color of an app. For different screen technologies (e.g. Super LCD and Super AMOLED) the energy consumption can vary for different background colors [23]. Selecting an appropriate background color may decrease the energy consumption.

Cloud Computing describes the usage of external services via Internet to execute high energy consuming apps instead of running them on mobile devices. Thus, energy is saved on mobile devices, and it allows further energy optimizations on servers [19].

4. Energy Measurement

For the most described energy code smells in Section 3, the three energy measurement techniques (Section 2.2) are applied. The energy code smell *Cloud Computing* is described in another work [19], hence the results are shown there. In Figure 5, the measurement results with and without energy code smells are depicted. In addition, the difference between the energy consumption before and after refactoring is shown. If the difference is negative, the energy consumption rises after refactoring, i.e. the inverse of the energy code smell will save energy. The file-based measurement technique is not available for the Samsung Galaxy S4.

Energy Code Smell	Measurement with Energy Code Smell			Measurement without Energy Code Smell			Difference (in %)		
	File Based	Energy Profiling	Delta-B	File Based	Energy Profiling	Delta-B	File Based	Energy Profiling	Delta-B
Third-Party Advertisement "GpsPrint"	6628 J	268 J	6405 J	5272 J	300 J	5144 J	20.5	-10.7	19.7
Third-Party Advertisement "TreeGenerator"	5310 J	308 J	4814 J	3681 J	269 J	3413 J	30.7	12.7	29.1
Binding Resources Too Early	11903 J	981 J	19566 J	11276 J	706 J	16032 J	5.3	28.0	18.1
Statement Change	3965 J	274 J	3768 J	3823 J	297 J	3382 J	3.5	-7.7	10.2
Backlight HTC	3681 J	269 J	3413 J	3796 J	272 J	3768 J	-3.0	-1.1	-2.9
Backlight S4	---	26152 J	10400 J	---	19184 J	4517 J	---	26.6	56.6
Data Transfer	4232 J	289 J	4064 J	3864 J	265 J	3497 J	8.6	8.3	14.0

Figure 5: Energy Measurement Results [16]

As you can see, the energy profiling has one major problem, which was detected during the validation. After a firmware update, the internal XML files used for realizing the energy profiling was also updated and the measurement results differ from the other two techniques [16]. This indicates that the vendor provides one internal XML file for the latest mobile devices, which are transferred to all devices which get the new firmware; independent on the build-in hardware (e.g. the new internal XML file has more processor states for the same mobile device after updating). Hence, the new power profile is adapted to the old one and the results of the energy consumption are manually calculated. With the adapted profile, the energy consumption for *Third-Party Advertisement* amounts 4461 J before refactoring and 4426 J after refactoring, a difference of 1 %. This small value results from active time of the Wi-Fi component because in both calculations the Wi-Fi is not switched off to execute the application behavior and the different traffic transfer does not have a real influence.

5. Conclusion

The definition and validation of energy code smells demonstrate an approach to save energy for mobile devices. All measurement results show that it is reasonable to perform energy refactorings on code to reduce energy consumption on mobile devices. The presented energy code smells are validated for Android apps. However, the definition of the energy code smells can also be applied on other platforms, such as Windows Phone and iOS, but has to respect hardware and operating

system specifics. These results show that saving energy by refactoring can support programmers to improve their existing apps and avoid energy code smells during developing new apps.

Next steps could be to extend the reengineering process to parse Windows Phone and iOS apps. Thereupon, the existing energy code smells could be validated for these platforms. Furthermore, hardware based energy measurement techniques could be used to confirm the software based measurement results. These would help to extend and to confirm the energy refactoring catalog in the mobile devices' area.

References

[1] L. Stobbe, N. Nisse, M. Proske, A. Middendorf, B. Schlomann, M. Friedewald, P. Georgieff, and T. Leimbach, "Abschätzung des Energiebedarfs der weiteren Entwicklung der Informationsgesellschaft,", Fraunhofer ISI, Fraunhofer IZM, Germany, 2008.

[2] Statista, "Absatzprognosen für 2013: Smartphones verkaufen sich am besten," 2013. [Online]. Available: http://de.statista.com/themen/647/itk-.

[3] RWE Power AG, "Kernkraftwerk Emsland," 2013. [Online]. Available: https://www.rwe.com/we.

[4] J. Chikofsky and J. H. Cross, "Reverse Engineering and Design Recovery: A Taxonomy," *IEEE*, 1990.

[5] Android Developers, "Android, the world's most popular mobile platform," 2013. [Online]. Available: http://developer.android.com/about/index.html.

[6] M. Gottschalk, J. Josefiok, J. Jelschen, and A. Winter, "Removing Energy Code Smells with Reengineering Services," *Lect. Notes informatics, GI*, 2012.

[7] M. Josefiok, M. Schröder, and A. Winter, "An Energy Abstraction Layer for Mobile Computing Devices," *Oldenbg. Lect. Notes Softw. Eng.*, vol. 5, 2013.

[8] A. Fuhr, A. Winter, U. Erdmenger, T. Horn, U. Kaiser, V. Riediger, and W. Teppe, "Model-Driven Software-Migration - Process Model, Tool Support and Application," in *Migrating Legacy Applications: Challenges in Service Oriented Architecture and Cloud Computing Environment*, Hershey, PA: IGI Global, 2012.

[9] J. Ebert, V. Riediger, and A. Winter, "Graph Technology in Reverse Engineering, The TGraph Approach," in *10th Workshop Software Reengineering (WSR 2008)*, Bonn: GI, 2008, pp. 67–81.

[10] T. Horn and J. Ebert, "The GReTL Transformation Language," in *Theory and Practice of Model Transformations - 4th International Conferenc*, 2011.

[11] M. Schröder, "Erfassung des Energieverbrauchs von Android Apps," Carl von Ossietzky University, Diploma Thesis, 2013.

[12] Android Developers, "Monitoring the Battery Level and Charging State," 2013. [Online]. Available: http://developer.android.com/training/monitorin.

[13] HTC Corporation, "HTC One X," 2013. [Online]. Available: http://www.htc.com/uk/smartphon.

[14] Samsung, "Samsung Galaxy S4," 2013. [Online]. Available: http://galaxys4.samsung.de/tech.

[15] Robotmafia.org, "GpsPrint," 2012. [Online]. Available: https://play.google.com/st.

[16] M. Gottschalk, "Energy Refactorings," Carl von Ossietzky University. Masters' Thesis, 2013. Available: http://www.se.uni-oldenburg.de/documents/gottschalk-MA2013.pdf

[17] M. Fowler, K. Beck, W. Brant, W. Opdyke, and D. Roberts, *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, 2002, p. 431.

[18] A. Carroll and G. Heiser, "An Analysis of Power Consumption in a Smartphone," *USENIX Annu. Tech. Conf.*, 2010.

[19] V. Strokova, S. Sapegin, and A. Winter, "Cloud Computing for Mobile Devices," in *Proceedings of the 28th EnviroInfo 2014 Conference*, 2014.

[20] A. Pathak, Y. Charlie Hu, and M. Zhang, "Fine Grained Energy Accounting on Smartphones with Eprof," *EuroSys'12*, 2012.

[21] A. Pathak, Y. Charlie Hu, and M. Zhang, "Bootstrapping Energy Debugging on Smartphones: A First Look at Energy Bugs in Mobile Devices," *ACM*, 2011.

[22] "Energy Aware Software-Engineering and Development (EASED@BUIS)," 2013.

[23] X. Chen, Y. Chen, Z. Ma, and F. Fernandes, "How is Energy Consumed in Smartphone Display Applications?," *ACM*, 2013.