Predicting The Energy Consumption Level of Java Classes in Android Apps: An Exploratory Analysis

Emanuele Iannone
SeSa Lab - University of Salerno, Italy
eiannone@unisa.it

Manuel De Stefano
SeSa Lab - University of Salerno, Italy
madestefano@unisa.it

Fabiano Pecorelli
Tampere University, Finland
fabiano.pecorelli@tuni.fi

Andrea De Lucia
SeSa Lab - University of Salerno, Italy
adelucia@unisa.it

ABSTRACT

Mobile applications usage has considerably increased since the last decade. Successful apps need to make the users feel comfortable while using them, thus demanding high-quality design and implementation. One of the most influencing factors for user experience is battery consumption, which should have the minimum possible impact on the battery. The current body of knowledge on energy consumption measurement only reports approaches relying on complex instrumentation or stressing the application with many test scenarios, thus making it hard to measure energy consumption in practice. In this work, we explore the performance of machine learning to predict the energy consumption level of Java classes in Android apps, leveraging only a set of structural properties extracted via source code analysis, without requiring any hardware measurements tools or executing the app at all. The preliminary results show the poor performance of learning-based estimation models, likely caused by (1) an insufficient amount of training data, (2) a limited feature set, and (3) an inappropriate way to label the dependent variable. The paper concludes by presenting the limitations of the experimented models and the possible strategies to address them.

CCS CONCEPTS

• **Software and its engineering → Software maintenance tools**.

KEYWORDS

Energy Consumption, Machine Learning, Android Apps

ACM Reference Format:
Emanuele Iannone, Manuel De Stefano, Fabiano Pecorelli, and Andrea De Lucia. 2022. Predicting The Energy Consumption Level of Java Classes in Android Apps: An Exploratory Analysis. In *IEEE/ACM 9th International Conference on Mobile Software Engineering and Systems (MOBILESoft '22), May 17–24, 2022, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/1122445.1122456

MOBILESoft '22, May 17–24, 2022, Pittsburgh, PA, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Mobile applications (a.k.a. apps) have changed our lives and habits, and their use has been increasing at a very high pace [6, 31]. Users need to be comfortable with them in terms of functional and non-functional aspects, and one of the main aspects influencing the user's satisfaction is energy consumption [39], that is one of the main reasons a user uninstalls an application [42].

In this regard, previous studies have shown that this is one of the main motivations that could lead users to uninstall an application. Therefore, measuring the consumption of mobile apps is a key factor for their success. In the last decade, several techniques have been proposed in this respect. On the one hand, there exists purely hardware-based solutions [9, 16], which rely on special instrumentation to directly measure the amount of energy drained from the battery when the app is exercised with specific test scenarios. On the other hand, model- or software-based solutions [8, 12, 27, 32, 41] leverages the APIs offered by the device, which compute an estimate of the energy consumed using the sensors installed.

Although these solutions have proven reliable and precise, they need to run realistic test cases that stress the different parts of the code, which are not easy to define. Moreover, hardware-based solutions require costly equipment, which can also be difficult to set up and run. In this respect, the current state of the practice lacks *lightweight* and *rapid* measurements, which can be run without any particular requirements and in a short amount of time. Such techniques may address both the developers' and users' need to have early feedback on the app energy consumption level.

In this paper, we investigate the effectiveness of predicting the *energy consumption level* of classes in mobile apps using of machine learning algorithms that leverage *structural properties*—i.e., code metrics and smells—extracted via source code analysis. In this respect, previous works have shown correlations between object-oriented code metrics [15] and different types of code smells [25, 30], making them a suitable option for predicting the consumption of the components of an app. This investigation is carried on the dataset by Palomba et al. [30], made of 620 Java classes belonging to 60 open-source Android apps. The preliminary results show that our machine learning-based estimation models failed, in most cases, to assign the right energy consumption level. Future work should provide additional insights on the efficiency of such energy estimation models, such as by (1) building a dataset with additional training data, (2) identifying new features affecting the energy consumption, and (3) exploring new strategies for making more meaningful and interpretable predictions.

To summarize, this paper (1) presents an empirical comparison of four different machine learners having the goal to predict the energy consumption of Java classes in Android apps by leveraging code smells and code metrics, and (2) provides an online appendix [18] containing all the data and scripts employed to foster replications and further experiments built upon this study.

2 RELATED WORK

Energy consumption measurement of mobile apps has been the subject of several studies. In recent years the research has been focused on envisioning tools and mechanisms to measure the battery drain reliably and efficiently. On the one hand, there are hardware-based solutions such as PowerScope [9] and GreenMiner [16], which rely on circuits and physical measurement tools (e.g., a multimeter); on the other hand, there exists software solutions that require little or no equipment at all, e.g., PowerBooter [41], eLens [12], or PETrA [8]. Almost all approaches calculate the consumption with direct measurements on the device's battery or via sensor data APIs. However, there exist alternative ways to this purpose, such as the one proposed by Gupta et al. [11], who built a linear regression model to estimate the battery consumption of Windows Phone apps based on the `.dll` files called. Similarly, Aggarwal et al. [1] relied on the number of system calls and the number of times they were changed to predict the energy consumption of two Android apps.

Several empirical studies have shown other factors that negatively or positively affect energy consumption. The way design patterns are implemented [36], the choice of certain data structures [13], and the presence of specific types of code smells [5, 14, 25, 30] may largely decrease the energy efficiency of mobile apps. Conversely, there is also evidence of the beneficial effects of good programming practices, such as query optimization [34], the use of lock-free data structures [17], and an appropriate management of resources [21]. Violations to such practices have been branded as Android-specific code smells by Reimann et al. [35]. Other choices, such as calls to specific Android API methods [22], screen color palettes [23], and sorting algorithms [4] have a variable impact, depending on how they are applied.

Our contribution. This work increases the granularity level of such predictions by providing estimates at the class-level of Android apps. We rely on software metrics and characteristics that can be computed via the sole static code analysis of the Java classes that constitute an app. We also compare the performance of different machine learning algorithms.

3 PRELIMINARY EVALUATION

The *goal* of the study is to predict the energy consumption level of Java classes in the *context* of Android apps with the *purpose* of making rapid and lightweight estimates by using structural properties of the source code. Specifically, we analyzed the performance of different machine learning algorithms on making multi-class classification, i.e., predicting which is the overall energy consumption level, at the *class* granularity level, irrespective of the belonging Android app. We opted for a classification as its predictions provide more *meaningful* and *interpretable* results than a regression model, which would estimate the consumption using a continuous value (measured in Joule). Based on these considerations, we asked:

RQ. *How good is a machine-learning-based classification model in predicting the energy consumption level of Java class in Android apps?*

3.1 Context

The *context* of the study consists of 60 real-world open-source Android apps whose energy consumption was measured in the work by Palomba et al. [30] These apps differs in terms of size and scope and come from the F-Droid repository.[1] The dataset contains the estimated energy consumption of methods of 620 public classes belonging to the 60 apps. The estimates were done with PETrA [8], a fully software-based energy measurement tool that stress each method with multiple runs of randomly generated test cases[2] to profile the energy consumption, measured in Joule, according to the device's battery sensors.[3] The method-level consumption values of all the repeated runs had been aggregated using the mean since no outlier was found in the measurements distributions.

In addition, the dataset reports the number of occurrences of 19 types of code smells affecting each class. Specifically, five of which are regular object-oriented smells that the authors detected using the DECOR detection rules [24], while the remaining 14 are specific to the Android context—as defined in the catalog[4] of Reimann et al. [35]—and detected using the tool aDoctor [29].

The described dataset was used to train and test several prediction models, built using three learning algorithms: *Multinomial Logistic Regression* (MLR) [2], *Support Vector Machine* (SVM) [7], and *Decision Tree* (DT) [3], as they are widely adopted in classification tasks. We involved a *random multi-class classifier*—which assigns a random energy consumption class to a given data point—as comparison baseline for the models' performances. We exploited the implementation of these models provided by the `TidyModels`[5] package for the `R` programming language.

3.2 Study Design

In the following we describe how we arranged the experimental process, starting from the selection of the dependent variable and the independent variables, to the definition of the entire training and testing pipeline, alongside the metrics used to evaluate the models' performances.

Dependent Variable. Since our study focuses on predicting the energy consumption at the *class* granularity level, we brought the method-level energy measurements—contained in the input dataset—to the class level by aggregating the consumption of all the methods belonging to a given class. This choice was driven by the fact that our goal was to investigate whether static factors can effectively predict energy consumption. Unfortunately, most of the static factors that have been associated with energy consumption in the past are all defined at a class-level granularity, hence we had to aggregate method-level observations to be able to build our

[1] F-Droid website: https://f-droid.org/
[2] Generated with Monkey: https://developer.android.com/studio/test/monkey
[3] Collected with BatteryStats: https://tinyurl.com/batterystats
[4] The smells catalog:https://martinbrylski.github.io/android_smells/
[5] `TidyModels` website: http://tidymodels.org

models. More details are reported in Section 3.4. We took into consideration three ways of aggregation: mean, sum, and maximum of the methods of a class. We applied a set of pre-processing steps to prepare the data for the classification task. We first normalized the dependent variable (for each aggregation) using the log transformation [19] to make its distribution more similar to a Gaussian one. Then, we mapped the normalized values to their percentiles and assign an energy consumption level based on the quartiles of the distribution (Q_1, Q_2, Q_3), so that each bin contains the same amount of classes. Specifically: $[min..Q_1[\rightarrow$ *'Negligible'*, $[Q_1..Q_2[\rightarrow$ *'Low'*, $[Q_2..Q_3[\rightarrow$ *'Moderate'*, and $[Q_3..max] \rightarrow$ *'High'*. We chose classification over regression since the target of our measurement is the end users, who will find more useful and interpretable an information on the overall consumption level of the application (just like it is done for house appliances) rather than the absolute amount of Joule consumed.

Independent Variables. The occurrences of the 19 code smells appearing in the dataset were treated as *independent variables* for our models, as some of them have been shown to be positively correlated with the energy consumption—i.e., the higher the number of instances, the higher the battery drain. This is the case of *Internal Setter*, *Leaking Thread*, and *Member Ignoring Method* Android-specific code smells, which can affect the consumption up to 87 times more than non-smelly classes.

Once we had collected the smell data, we mined additional metrics concerning structural code properties—such as size, cohesion, coupling, and complexity. These metrics were computed via inspection of the Abstract Syntax Trees (ASTs) of all the 620 Java classes found in the dataset. Specifically, we used an our own script (available in our online appendix [18]), to extract 22 different metrics, which we considered as additional *independent variables* of the models. We suspect that these kind of metrics may give additional guidance to our prediction models, as hinted by Hindle [15]. For example, long or complex classes may affect energy consumption since they requires executing more code or more complex programming constructs. To summarize, we involved 41 different independent variables, whose descriptions are all reported in our online appendix due to space limitations [18].

Machine Learning Pipeline. After collecting the dependent and independent variables, we configured the machine learners for predicting the energy consumption. We evaluated the four learning algorithms on the same dataset using a 10-fold cross-validation strategy [38], in which the dataset is divided into 10 random subsets of approximately equal size (folds). The training and testing phases were repeated 10 times. Iteratively, nine folds were used to train the models, while the remaining one was used as the test set, ensuring that in each iteration a different permutation of folds is used. Before running the training phase of each iteration, we applied a feature selection strategy to remove those predictors (1) showing near-zero variance (i.e., predictors having the same value in almost all data points), and (2) having large absolute correlations (in this case 0.6)—using the Spearman's rank correlation coefficient [37]—with other predictors. An example of near-zero variance predictor is one that, for 1000 samples, has two distinct values and 999 of them are a single value. These steps were accomplished to remove those variables that would have given poor or no information to the models, and mitigate the effect of multi-collinearity [28]. Then, we took into account the tuning of models' hyper-parameters by running the Grid Search algorithm. Internally, it relies on another cross validation to systematically search the best combination of hyper-parameters, so that a given error measure is minimized—here we selected the F-measure [33]. Finally, we trained the models on the test set using the optimal hyper-parameter configuration, and run the predictions on the test set—without considering the predictors dropped after the feature selection done previously.

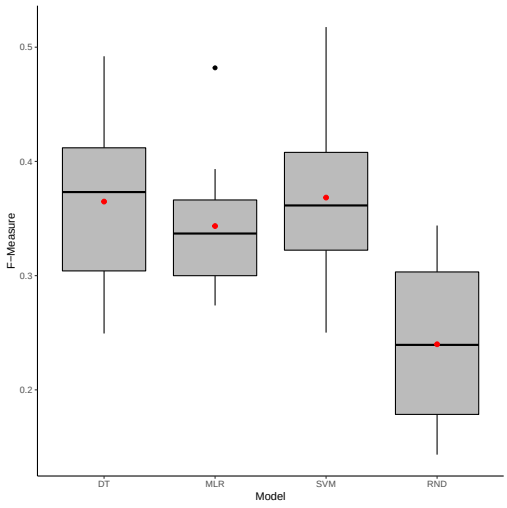

Figure 1: Box plots representing the F-measure values obtained during the 10-fold cross validation of the four classifiers selected, in the aggregation by sum case. The straight horizontal lines in the boxes indicate the median, while the red dots the mean.

Evaluation Metrics. The prediction results were used to obtain the *confusion matrix*, reporting the outcomes for each energy consumption level (i.e., 4×4 matrix). For each class c, we computed *Precision*, *Recall*, and *F-measure* performance metrics [33] for the multi-class classification task, obtaining their *one-vs-all* version (a.k.a., *macro* measures), i.e., when treating c as the positive class and the remaining ones as a single negative class. Section 3.3 discusses the achieved results only in terms of F-measure distributions obtained by the 10-fold cross validation on the sum aggregation, leaving the other results in our online appendix [18]. The F-measure distributions are represented via box plots and compared using the Friedman test [10] with the Nemenyi post-hoc test [26], and reported the results using MCB plots (Multiple comparisons with the best) [20]. The Friedman test is a non-parametric equivalent to ANOVA which does not assume any distribution of the data. It allows us to assess the significance of the different outcomes achieved across multiple test attempts (in our case, datasets) when using different treatments (in our case, the various models). We set the significance level to $\alpha = 0.05$ [10]. All the study materials can be found on the online appendix [18].

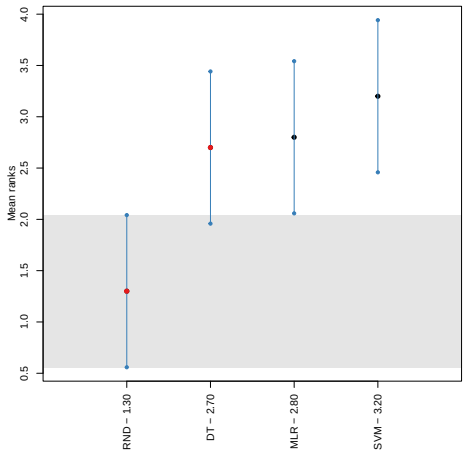

Figure 2: Nemenyi test results for statistical significance between the F-measure values obtained during cross validation of the four classifiers (using the sum aggregation). The results are presented by mean of MCB plots [20].

3.3 Results

Figure 1 shows boxplots reporting the F-measure values achieved by *Decision Tree* (DT), *Multinomial Logistic Regression* (MLR), *Support Vector Machine* (SVM), and the *Random Classifier* (RND), respectively. We report only the case of aggregation by sum, since it is the one performing better than the others, however similar. According to the figure, the classification performance appears to be pretty low, regardless of the classifier, with an F-measure ranging between 0.30 and 0.40. When turning to the comparison between the classifiers, SVM appears to perform slightly better than the other two while the Random Classifier achieves a clearly lower f-measure, thus indicating that, poor as it is, the performance of the proposed classification technique is still better than a simple random classification. These observations are also confirmed by the results of Nemenyi test, as reported in Figure 2, in which SVM and MLR appear to perform significantly better than both DT and the Random Classifier. However, given the preliminary nature of the study, we can not draw certain conclusions.

The poor performances might be the consequence of several choices. Firstly, the dataset we exploited is made of only a scarce amount of data points (the Java classes), limiting what the models learned from them. This urges the need for richer datasets having a large amount of energy measurements, to enable machine learning-based energy estimation. Secondly, we considered a limited set of features, motivated by the availability of tools able to compute them in short time and without running or building the apps. As a matter of fact, the literature showed the existence of other static properties correlated with the energy consumption of apps, that should be involved in future work. Finally, the performance may be influenced by the splitting criterion adopted to assign each class to the corresponding energy consumption level. The splitting was performed by dividing the variable in quartiles according to its distribution, hence being strongly related to other observation in the dataset. More details about this aspect are reported in Section 3.4.

3.4 Limitations

One of the study's major limitations lies in the calculation of our dependent variable. While the experimental dataset provides method-level measurements for the energy consumption, we aggregated such measures by computing, for each class, the mean, the sum, and the maximum of the energy consumption for all the methods in it contained. This choice could have biased our results since an aggregated measure may not represent the actual class energy consumption. We made this decision owing to the limited amount of method-level predictors that could be extracted from the source code alone—indeed, most code metrics and smells are defined at the whole class level, reducing the number of possible explanatory variables for the models. In the future, we plan to replicate the experiment by directly relying on a dataset containing class-level information about the dependent variable. It must be also considered that, for the sake of having as much training data as possible, we trained the models using an heterogeneous dataset, containing class instances coming from systems of different domains. This might have inevitably impacted the overall performance. In the follow-up studies, we plan to consider a more homogeneous dataset, allowing a within-project approach, whenever available.

Other possible imitations are related to the data set used to train the classifiers, since it did not include secondary features that might impact both dependent and some of the independent variables. In this preliminary work, we did not consider any confounding factors, restricting the interpretation of the selected structural properties. Moreover, the explainability of the models was neglected (e.g., variable importance) given the preliminary nature of the study. We plan to consider them in a complete evaluation.

Finally, the splitting criterion we applied to label the energy consumption values into the four levels strongly depends on the corresponding variable distribution in the dataset. Such a criterion may threaten the generalizability of the presented classification approach. Indeed, the labeling phase should be performed on a larger and more representative dataset to be applicable in practice. We suspect that using a different and more robust labeling criteria, may solve this limitation and provide "fairer" energy consumption levels. For instance, an ideal schema could be based on the *EU energy labeling schema*,[6] resambling Wilke et al's idea [40],where appliances are assigned to a letter (A, B, C, etc.) depending on their energy consumption (the closer to A, the lower).

4 CONCLUSION

This paper presents a mobile app's energy consumption estimation technique based on static properties (i.e., code metrics and smells). The proposed approach predicts the energy consumption level for Java class in Android applications. Results of a preliminary evaluation indicate an average F-measure of ≈ 0.4, suggesting that the technique still needs to be further improved to be considered for real applications. Our future research directions aim to replicate the study on a larger dataset and consider a broader set of predictors.

[6] https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:32017R1369

REFERENCES

[1] Karan Aggarwal, Chenlei Zhang, Joshua Charles Campbell, Abram Hindle, and Eleni Stroulia. 2014. The Power of System Call Traces: Predicting the Software Energy Consumption Impact of Changes. In *Proceedings of 24th Annual International Conference on Computer Science and Software Engineering* (Markham, Ontario, Canada) *(CASCON '14)*. IBM Corp., USA, 219–233.

[2] Dankmar Böhning. 1992. Multinomial logistic regression algorithm. *Annals of the institute of Statistical Mathematics* 44, 1 (1992), 197–200.

[3] L. Breiman, J. Friedman, C.J. Stone, and R.A. Olshen. 1984. *Classification and Regression Trees*. Taylor & Francis. https://books.google.it/books?id=JwQx-WOmSyQC

[4] Christian Bunse, Hagen Höpfner, Essam Mansour, and Suman Roychoudhury. 2009. Exploring the Energy Consumption of Data Sorting Algorithms in Embedded and Mobile Environments. In *2009 Tenth International Conference on Mobile Data Management: Systems, Services and Middleware*. 600–607. https://doi.org/10.1109/MDM.2009.103

[5] Antonin Carette, Mehdi Adel Ait Younes, Geoffrey Hecht, Naouel Moha, and Romain Rouvoy. 2017. Investigating the energy impact of Android smells. In *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 115–126. https://doi.org/10.1109/SANER.2017.7884614

[6] L. Ceci. 2021. *Number of mobile app downloads worldwide from 2016 to 2021.* https://www.statista.com/statistics/271644/worldwide-free-and-paid-mobile-app-store-downloads/

[7] Corinna Cortes and Vladimir Vapnik. 1995. Support-Vector Networks. *Mach. Learn.* 20, 3 (Sept. 1995), 273–297. https://doi.org/10.1023/A:1022627411411

[8] Dario Di Nucci, Fabio Palomba, Antonio Prota, Annibale Panichella, Andy Zaidman, and Andrea De Lucia. 2017. Software-based energy profiling of Android apps: Simple, efficient and reliable?. In *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 103–114. https://doi.org/10.1109/SANER.2017.7884613

[9] J. Flinn and M. Satyanarayanan. 1999. PowerScope: a tool for profiling the energy usage of mobile applications. In *Proceedings WMCSA'99. Second IEEE Workshop on Mobile Computing Systems and Applications*. 2–10. https://doi.org/10.1109/MCSA.1999.749272

[10] Milton Friedman. 1940. A comparison of alternative tests of significance for the problem of m rankings. *The Annals of Mathematical Statistics* 11, 1 (1940), 86–92.

[11] Ashish Gupta, Thomas Zimmermann, Christian Bird, Nachiappan Nagappan, Thirumalesh Bhat, and Syed Emran. 2014. Mining Energy Traces to Aid in Software Development: An Empirical Case Study. In *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM '14)*. https://doi.org/10.1145/2652524.2652578

[12] Shuai Hao, Ding Li, William G. J. Halfond, and Ramesh Govindan. 2013. Estimating mobile application energy consumption using program analysis. In *2013 35th International Conference on Software Engineering (ICSE)*. 92–101. https://doi.org/10.1109/ICSE.2013.6606555

[13] Samir Hasan, Zachary King, Munawar Hafiz, Mohammed Sayagh, Bram Adams, and Abram Hindle. 2016. Energy Profiles of Java Collections Classes. In *Proceedings of the 38th International Conference on Software Engineering (ICSE '16)*. https://doi.org/10.1145/2884781.2884869

[14] Geoffrey Hecht, Omar Benomar, Romain Rouvoy, Naouel Moha, and Laurence Duchien. 2015. Tracking the Software Quality of Android Applications Along Their Evolution (T). In *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 236–247. https://doi.org/10.1109/ASE.2015.46

[15] Abram Hindle. 2012. Green mining: A methodology of relating software change to power consumption. In *2012 9th IEEE Working Conference on Mining Software Repositories (MSR)*. 78–87. https://doi.org/10.1109/MSR.2012.6224303

[16] Abram Hindle. 2012. Green mining: Investigating power consumption across versions. In *2012 34th International Conference on Software Engineering (ICSE)*. 1301–1304. https://doi.org/10.1109/ICSE.2012.6227094

[17] Nicholas Hunt, Paramjit Singh Sandhu, and Luis Ceze. 2011. Characterizing the Performance and Energy Efficiency of Lock-Free Data Structures. In *2011 15th Workshop on Interaction between Compilers and Computer Architectures*. 63–70. https://doi.org/10.1109/INTERACT.2011.13

[18] Emanuele Iannone, Manuel De Stefano, Fabiano Pecorelli, and Andrea De Lucia. 2022. Online Appendix. tinyurl.com/predicting-mobilesoft22. Accessed: 2022-02-09.

[19] Oliver N Keene. 1995. The log transformation is special. *Statistics in medicine* 14, 8 (1995), 811–819.

[20] Alex J Koning, Philip Hans Franses, Michele Hibon, and Herman O Stekler. 2005. The M3 competition: Statistical tests of the results. *International Journal of Forecasting* 21, 3 (2005), 397–409.

[21] Ding Li and William G. J. Halfond. 2014. An Investigation into Energy-Saving Programming Practices for Android Smartphone App Development. In *Proceedings of the 3rd International Workshop on Green and Sustainable Software* (Hyderabad, India) *(GREENS 2014)*. 46–53. https://doi.org/10.1145/2593743.2593750

[22] Mario Linares-Vásquez, Gabriele Bavota, Carlos Bernal-Cárdenas, Rocco Oliveto, Massimiliano Di Penta, and Denys Poshyvanyk. 2014. Mining Energy-Greedy API Usage Patterns in Android Apps: An Empirical Study. In *Proceedings of the 11th Working Conference on Mining Software Repositories* (Hyderabad, India) *(MSR 2014)*. 2–11. https://doi.org/10.1145/2597073.2597085

[23] Mario Linares-Vásquez, Gabriele Bavota, Carlos Eduardo Bernal Cárdenas, Rocco Oliveto, Massimiliano Di Penta, and Denys Poshyvanyk. 2015. Optimizing Energy Consumption of GUIs in Android Apps: A Multi-Objective Approach. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering* (Bergamo, Italy) *(ESEC/FSE 2015)*. 143–154. https://doi.org/10.1145/2786805.2786847

[24] Naouel Moha, Yann-Gael Gueheneuc, Laurence Duchien, and Anne-Francoise Le Meur. 2010. DECOR: A Method for the Specification and Detection of Code and Design Smells. *IEEE Transactions on Software Engineering* 36, 1 (2010), 20–36. https://doi.org/10.1109/TSE.2009.50

[25] Rodrigo Morales, Rubén Saborido, Foutse Khomh, Francisco Chicano, and Giuliano Antoniol. 2018. EARMO: An Energy-Aware Refactoring Approach for Mobile Apps. *IEEE Transactions on Software Engineering* 44, 12 (2018), 1176–1206. https://doi.org/10.1109/TSE.2017.2757486

[26] Peter Bjorn Nemenyi. 1963. *Distribution-free multiple comparisons*. Princeton University.

[27] Adel Noureddine, Aurelien Bourdon, Romain Rouvoy, and Lionel Seinturier. 2012. Runtime monitoring of software energy hotspots. In *2012 Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*. 160–169. https://doi.org/10.1145/2351676.2351699

[28] Robert M O'brien. 2007. A caution regarding rules of thumb for variance inflation factors. *Quality & quantity* 41, 5 (2007), 673–690.

[29] Fabio Palomba, Dario Di Nucci, Annibale Panichella, Andy Zaidman, and Andrea De Lucia. 2017. Lightweight detection of Android-specific code smells: The aDoctor project. In *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 487–491. https://doi.org/10.1109/SANER.2017.7884659

[30] Fabio Palomba, Dario Di Nucci, Annibale Panichella, Andy Zaidman, and Andrea De Lucia. 2019. On the impact of code smells on the energy consumption of mobile applications. *Information and Software Technology* 105 (2019), 43–55. https://doi.org/10.1016/j.infsof.2018.08.004

[31] Joe Parker. 2019. *10 years of growth of Mobile App Market.* https://www.knowband.com/blog/mobile-app/growth-of-mobile-app-market//

[32] Abhinav Pathak, Y. Charlie Hu, and Ming Zhang. 2012. Where is the Energy Spent inside My App? Fine Grained Energy Accounting on Smartphones with Eprof. In *Proceedings of the 7th ACM European Conference on Computer Systems* (Bern, Switzerland) *(EuroSys '12)*. 29–42. https://doi.org/10.1145/2168836.2168841

[33] David Powers and Ailab. 2011. Evaluation: From precision, recall and F-measure to ROC, informedness, markedness correlation. *J. Mach. Learn. Technol* 2 (01 2011), 2229–3981. https://doi.org/10.9735/2229-3981

[34] Giuseppe Procaccianti, Héctor Fernández, and Patricia Lago. 2016. Empirical evaluation of two best practices for energy-efficient software development. *Journal of Systems and Software* 117 (2016), 185–198. https://doi.org/10.1016/j.jss.2016.02.035

[35] Jan Reimann, M. Brylski, and U. Assmann. 2014. A Tool-Supported Quality Smell Catalogue For Android Developers. *Softwaretechnik-Trends* 34 (2014).

[36] Cagri Sahin, Furkan Cayci, Irene Lizeth Manotas Gutiérrez, James Clause, Fouad Kiamilev, Lori Pollock, and Kristina Winbladh. 2012. Initial Explorations on Design Pattern Energy Usage. In *Proceedings of the First International Workshop on Green and Sustainable Software* (Zurich, Switzerland) *(GREENS '12)*. IEEE Press, 55–61.

[37] C. Spearman. 1987. The Proof and Measurement of Association between Two Things. *The American Journal of Psychology* 100, 3/4 (1987), 441–471. http://www.jstor.org/stable/1422689

[38] Mervyn Stone. 1974. Cross-validatory choice and assessment of statistical predictions. *Journal of the royal statistical society: Series B (Methodological)* 36, 2 (1974), 111–133.

[39] Claas Wilke, Sebastian Richly, Sebastian Götz, Christian Piechnick, and Uwe Aßmann. 2013. Energy Consumption and Efficiency in Mobile Applications: A User Feedback Study. In *2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing*. 134–141. https://doi.org/10.1109/GreenCom-iThings-CPSCom.2013.45

[40] Claas Wilke, Sebastian Richly, Georg Püschel, Christian Piechnick, Sebastian Götz, and Uwe Aßmann. 2012. Energy labels for mobile applications. In *INFORMATIK 2012*, Ursula Goltz, Marcus Magnor, Hans-Jürgen Appelrath, Herbert K. Matthies, Wolf-Tilo Balke, and Lars Wolf (Eds.). Gesellschaft für Informatik e.V., Bonn, 412–426.

[41] Lide Zhang, Birjodh Tiwana, Robert P. Dick, Zhiyun Qian, Z. Morley Mao, Zhaoguang Wang, and Lei Yang. 2010. Accurate online power estimation and automatic battery behavior based power model generation for smartphones. In *2010 IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*. 105–114.

[42] Agustin Zuniga, Huber Flores, Eemil Lagerspetz, Petteri Nurmi, Sasu Tarkoma, Pan Hui, and Jukka Manner. 2019. Tortoise or hare? Quantifying the effects of performance on mobile app retention. In *The World Wide Web Conference*. 2517–2528.