Is well-tested code more energy efficient?

Adel Noureddine, Matias Martinez, Houssam Kanso, Noëlle Bru

June 2022

Is well-tested code more energy efficient?

Adel NOUREDDINE [a,1], Matias MARTINEZ [b], Houssam KANSO [a] and Noelle BRU [c]
[a] *Universite de Pau et des Pays de l'Adour, E2S UPPA, LIUPPA, Anglet, France*
[b] *Université Polytechnique Hauts-de-France, LAMIH UMR, CNRS 8201, Valenciennes, France*
[c] *Universite de Pau et des Pays de l'Adour, E2S UPPA, CNRS, LMAP, Anglet, France*

Abstract. Software testing plays an important role in building quality software and improving maintainability. However, there are no research studies to analyze its impact on energy efficiency. In this paper, we conduct a preliminary study on the impact of unit tests and code coverage on the energy consumption of software. Our empirical study analyzes the energy consumption of multiple JSON libraries and the relation of their energy efficiency to test metrics. Although our study has limitations in the size of the data set, we found that there are hints for a positive correlation between line coverage and energy consumption.

Keywords. Software Energy, Software Testing, Power Consumption, Code Coverage

1. Introduction

Software energy consumption is a major concern for software developers, practitioners [8] and architects [2]. An important issue is the lack of tools to monitor software energy, and limited knowledge in understanding the factors impacting the energy consumption of software [10]. In particular, the authors of [10] note the *lack of knowledge on how to write, maintain, and evolve energy-efficient software*. The authors also discussed the state of the art of energy-aware software testing, and found that few studies propose energy-aware software testing techniques. These techniques offer new approaches to reduce the energy consumption of test suites [7], including in Android [6], or detecting energy bugs through software tests [1].

Current approaches allow software developers to monitor the energy consumption of their devices' architectures [3], their applications [4], and within the source code [9], thus allowing to detect energy hotspots. With such tools and in-depth software energy knowledge, developers can detect and improve their software. However, the technical and psychological scalability of these approaches (such as resistance from developers to adopt new energy-aware coding behaviors, and the pressure of project deadlines and release) limits their effectiveness, as developers report a lack of proper tools and knowledge as shown in [10]. We argue that leveraging existing, more accepted, and adopted approaches in software development, to guide developers in writing energy-aware software is needed. In particular, we argue for leveraging software testing for energy efficiency.

[1] Corresponding Author: Adel Noureddine, Universite de Pau et des Pays de l'Adour, E2S UPPA, LIUPPA, Anglet, France; E-mail: adel.noureddine@univ-pau.fr

Therefore, guiding software developers in writing better quality code, through software testing, might help in achieving energy optimizations and gains in software with little additional investment to practitioners and current development practices. The advantages of software testing are well known in terms of improving software quality and maintainability, and reducing bugs. However, the impact of tests quality on the energy efficiency of software is not well understood or studied. In this experiment, we analyze if software written with unit tests and having good code coverage (along with other test metrics) is more energy-efficient than software with no unit tests or low code coverage.

In this paper, we focus on the energy consumption of applications from the same domain, all capable of doing a same functionality: parsing JSON files. We measure the energy consumption and capture test metrics from 14 Java JSON libraries studied by [5], giving as input a set of well-formed JSON files. We execute the file parsing functionality from all of them given as input the same data set of well-formed JSON files collected by [5]. This allows us to fairly compare the measurement of the energy consumption for executing that functionality. We focus on JSON libraries because JSON is one of the most used data representation format. For instance, GSON, one of the lib analyzed, is used by more than 331K projects hosted on Github. In the context of Android, energy consumption of applications is an important problem (as mobile users care about energy consumption and the battery life of their devices). Many applications use JSON format for storage of data, and can also receive the result on API call in that format.

Our initial results show that applications having a good test suite, expressed in popular metrics such as coverage, does not equate in having optimal power consumption. The remainder of the paper is as follows: we detail our empirical methodology in Section 2, then outline our experimental results in Section 3. We analyze and discuss our findings in Section 4. We present the limitations and threats to validity in Section 5. Finally, we conclude in Section 6.

2. Hypothesis and Methodology

In this section, we detail our research questions and our experimental setup and empirical methodology for software energy measurements and software testing.

2.1. Research Questions

The research question that guides our research is: *Is there a correlation between energy consumption of an application and the quality of its test suite?*

To answer the research question, we set the following hypothesis:

- Null hypothesis $H0$: there is no correlation between a metric that represents test quality and energy consumption of the application of the test.
- Alternative hypothesis $H1$: there is a correlation between a metric that represents test quality and energy consumption.

2.2. Measuring the Energy Consumption of Applications

To address our research question, we first measure the energy consumption of executing an application, and then collect the metrics related to the test cases. Finally, we correlate energy consumption and those metrics.

2.2.1. Requirements on the evaluation dataset

We study applications that implement a particular functionality F. This allows us to fairly compare the energy across different implementations of F, and to remove the potential threats of comparing two different functionalities and/or applications with different energy requirements. Note that we measure energy consumption of each implementation of F using the same input values I.

2.2.2. Measuring Energy Consumption

We call *test workload* to the execution of F from an application A given I as input. We measure the energy consumption of the workloads using the power tool called PowerJoular [2], which uses Linux kernel's Intel RAPL through the powercap interface [3]. In order to minimize the impact of any noise during the measurement of energy of F, (caused by e.g., system states or background services), we run each test workload in a loop for 200 times and measure its energy consumption. We report the energy consumption per loop by dividing the total energy by the number of loops.

2.2.3. Collecting Test metrics

We use SonarQube[4] and JaCoCo[5] to collect test and coverage metrics for the applications under evaluation. We collected the following metrics: branch coverage, SonarQube coverage[6], line coverage, lines to cover, uncovered conditions, uncovered lines, and the number of tests.

2.2.4. Computing Correlation

We correlate the energy results with the test metrics using the Pearson correlation method. We also compute the p-value, which allows us to reject or accept our Null hypothesis.

2.2.5. Experiment Infrastructure

We run our experiments on a Dell Precision 5520 laptop with an Intel Core i7-7820HQ processor, running Fedora 34 with Linux kernel 5.11. We compile and run the Java libraries using openJDK 11.

2.3. Evaluation Dataset

In this paper, we focus on a single functionality F: parsing a JSON file for disk for creating a representation of it in RAM. We focus on applications written in Java as it is one of the most used languages in open-source development [7].

For our experiments, we use a set of 14 Java JSON libraries that implement this functionality. Those libraries, listed in Table 1, were previously studied by [5]. That work

[2] https://gitlab.com/joular/powerjoular

[3] https://www.kernel.org/doc/html/latest/power/powercap/powercap.html

[4] SonarQube: `https://www.sonarqube.org/`

[5] JaCoCo: `https://www.eclemma.org/jacoco/`

[6] The SonarQube coverage is a mix of Line coverage and Condition coverage: `https://docs.sonarqube.org/latest/user-guide/metric-definitions/`.

[7] Stats of popularity of programming languages: `https://githut.info/`

considers 20 libraries, however, we could not analyze 6 of them for different reasons, including: *a)* Unavailable source code, which is required by our experiment to compute the metrics from the tests ([5] only uses their binary JARs). *b)* Build failing due to unavailable dependencies. *c)* Failure on the computation of test metrics using JaCoCo tool.

As input data I, we use the publicly available dataset of JSON files provided by [5], conformed by 152 well-formed JSON files. Given this data, our experiment measures the energy that each JSON library requires to parse all those JSON files. This allows us to fairly compare the energy consumption of the 14 libraries by doing a single functionality (JSON file parsing) on the same input data (152 files).

3. Experimental Results

In this section, we present the results of our experimental study and the correlation analysis between energy consumption and testing metrics.

3.1. Energy Consumption

Table 1 outlines the energy consumption of each JSON library, along with the execution time, and the average power consumption of the CPU.

Table 1. Energy consumption for each JSON library workload

Library	Avg Power (Watts)	Energy (Joules)	Time (sec)	Standard deviation for Power
json-lib	46.87	225.43	4.81	5.29
sojo	51.76	76.34	1.475	8.87
flex-json	55.58	37.24	0.67	5.01
corn	53.02	36.32	0.685	4.66
mjson	52.64	26.32	0.5	8.54
jsonij	54.05	25.4	0.47	5.66
jsonutils	38.79	23.27	0.6	4.02
genson	54.88	17.29	0.315	5.39
fastjson	52.11	16.94	0.325	8.48
json-simple	34.79	14.44	0.415	4.17
json	34.08	13.8	0.405	6.41
gson	30.54	10.38	0.34	3.78
jackson-databind	29.35	10.13	0.345	3.72
cookjson	25.77	7.73	0.3	1.71

Our first observation is that energy consumption does not follow execution time. For instance, genson library consumed 17.29 joules on average per workload execution and took 315 milliseconds. In comparison, JSON library took more time (405 ms) and consumed 13.8 joules for processing the same JSON files. Observing the average power consumption during the experiment for each library sheds a light over their CPU usage and power consumption, as the average power can vary from 25.77 watts to 55.58 watts, a 30 watts difference on a laptop computer. Overall, the energy consumption of the mea-

Table 2. Metrics extracted from test cases and the energy consumption from each application.

Library	Branch coverage	Coverage	Line coverage	Uncovered lines
json-simple	50.5	55.7	58	336
mjson	61.9	67.2	71	314
corn	52.1	60.3	64.7	556
flex-json	69.7	72.6	74	445
sojo	95.5	96.7	97.2	82
jsonutils	61.5	67.6	71	879
json-lib	71.3	74.1	75.8	1181
genson	67.3	73.2	76.3	1234
jsonij	55.4	40.4	35.9	4090
cookjson	87.2	55.5	47.8	3406
json	84.4	34.7	25.2	6356
gson	79.1	34.9	27.3	10257
fastjson	78.4	83.9	87.6	3469
jackson-databind	70.3	75.4	78.2	7108

sured 14 libraries shows a big difference with energy varying from just 7.72 joules for cookJSON to 225.43 joules for JSON-lib, more than 2820% increase.

3.2. Test Metrics

Table 2 shows the metrics extracted from the test suites of our JSON libraries. Branch and line coverage varies greatly from as low as 50.5% and 25,2%, respectively, to as much as 95,5% and 97.2% respectively. However, the relation with energy consumption is not straightforward as the highest 2 libraries in branch coverage have the lowest and the 2nd highest energy consumption. Lines covered and uncovered also range from a few hundreds to around 28 thousand lines, and the number of tests varies from 3 tests to just below 5000 tests.

In the next section, we further analyze the results, and study and discuss the correlation of the test metrics with energy consumption.

4. Analysis and Discussions

We first plot the distribution of values of each test metric in Figure 1. We observe that a few libraries have a higher variation of values and are beyond the average range of other libraries. Instead of removing those outliers (as our dataset is limited in this preliminary study), we decide to analyze our data using logarithmic values. This logarithmic transformation allows to decrease the difference between the different values and limiting the impact of outliers while still maintaining the order of values, and it often produces a normal distribution of the studied metrics.

Using the logarithmic approach, we calculate, in Table 3, the Pearson's correlation coefficient between different metrics extracted from the test execution (e.g., coverage) and the average power consumed by parsing functionality. We apply a logarithmic transformation to the metrics' value for two reasons: 1) it often produces a normal distribu-

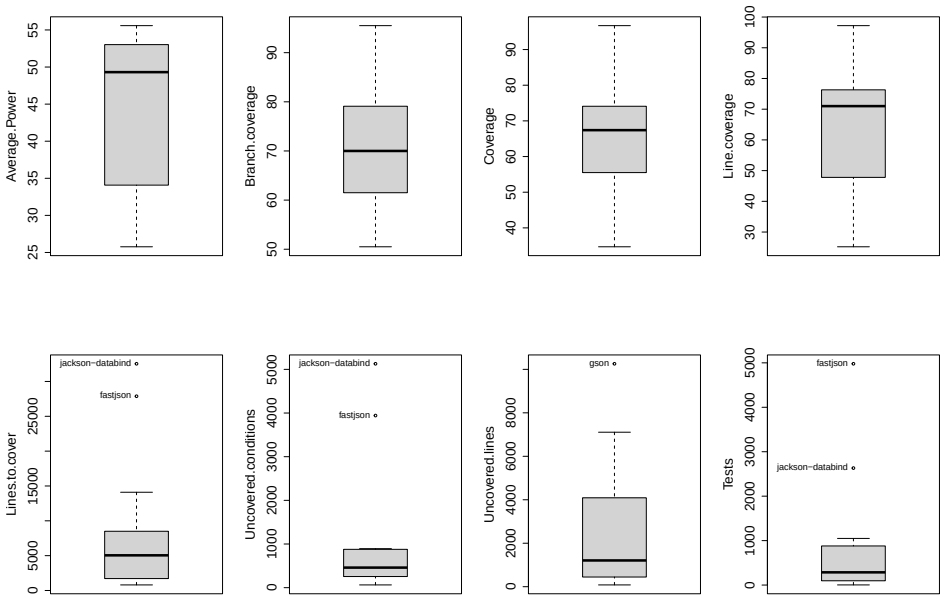

Figure 1. Distribution of values for each metric

Table 3. Pearson's correlation coefficient between test metrics and average power consumption, using Pearson algorithm on logarithmic values. Last column shows the p-values.

Metrics from tests	Correlation coefficient	P-value
Branch coverage	-0.27	0.342
SonaQube Coverage	0.42	0.133
Line coverage	0.46	0.095
Lines to cover	-0.36	0.211
Uncovered conditions	-0.018	0.95
Uncovered lines	-0.53	0.049
Number of tests	-0.044	0.881

tion, and 2) it allows us to decrease the difference between the values, limiting the impact of outliers, and, consequently, avoiding the need of removing them.

We observe that two test metrics exhibit a moderate correlation value with an acceptable P-value: line coverage with 0.46 correlation and a P-value of 0.095 (above the 0.05 range, but within the 0.1 range); and uncovered lines which has a negative correlation at -0.53 and a good P-value at 0.049 (below the 0.5 significance range).

Figure 2 shows the correlation with Line coverage where we note that, if we exclude the 3 out-of-range values, we might have better correlations. We also found the same trend for the correlation of Uncovered line, which has 2 out-of-range values. A detailed plot of both correlations (in Figure 2 and 3), shows that we might have better correlations if not for 2 or 3 out-of-range values.

We finally plot a principal component analysis (PCA) graph in Figure 4, which allows us to synthesize and understand the importance of each metric and to explain the

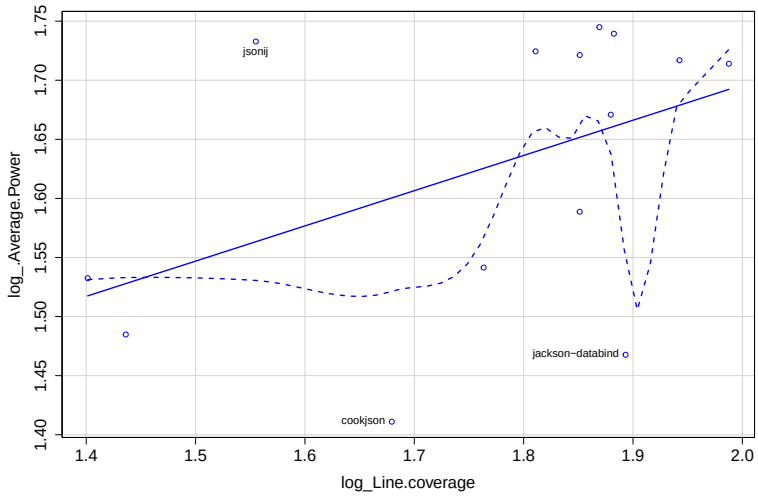

Figure 2. Line coverage correlation plot with average power consumption. The straight line is the least squares fitting, while the dashed line is the smoothing model on our datset.

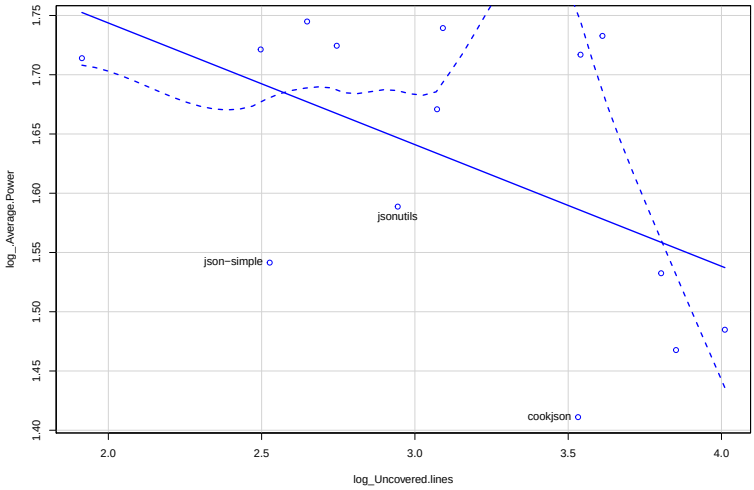

Figure 3. Uncovered lines

variability of the libraries. Each arrow represents a metric variable: if the arrows are close to each other, then we conclude that a strong correlation exists, while opposite arrows means a negative correlation, and orthogonal means no correlations. We observe that the average power is drawn closer to line coverage, *i.e.*, a possible correlation exists, while the average power is in near-perfect opposite of uncovered lines, *i.e.*, a possible strong negative correlation exists.

Our statistical analysis, although on a small dataset, sheds the light on potential positive correlation between line coverage and power consumption, and potential negative correlation between uncovered lines and power consumption. That is, the higher lines we

cover in our test, the higher the average power consumption of the program might be. In contrast, the higher the uncovered lines are, the lower the average power consumption.

Now, we proceed to accept or reject our hypothesis (see Section 2.1) using the p-values from Table 3. Considering a significance level $\alpha = 0.05$, all the p-values related to test metrics are greater than α, which means that we are not able to reject the Null hypothesis. On the contrary, the p-values from the correlation between the metric Uncovered lines is smaller than $\alpha = 0.05$, which means we can reject the Null hypothesis for that metric.

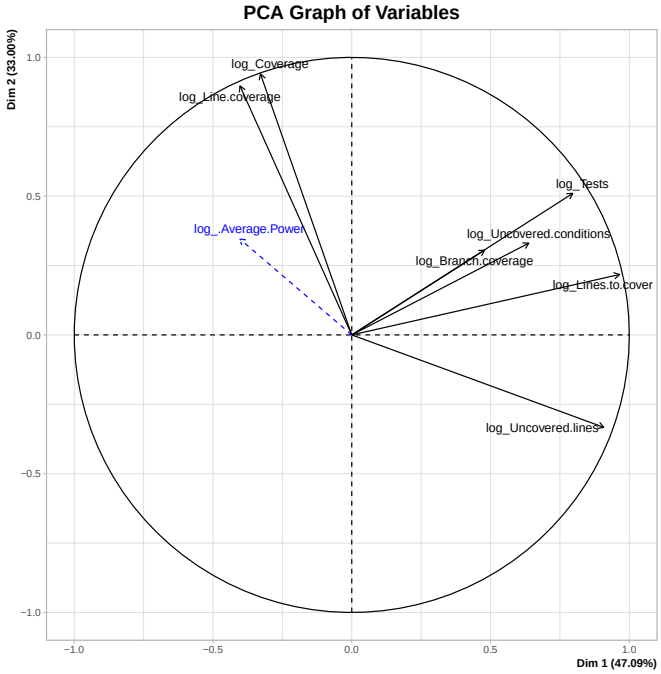

Figure 4. Principal component analysis (PCA) of our dataset, using logarithmic values

Response to the RQ: Is there a correlation between energy consumption of an application and the quality of its test suite?

Our experiment shows that there is a moderate positive correlation between Line coverage and power consumption of the parsing functionality of JSON libraries. However, we do not have enough evidence at the level $\alpha = 0.05$ to conclude that there is a linear relationship in the population of Java JSON libraries between coverage and power consumption.

The main takeaway of this research is that applications having a good test suite, expressed in popular metrics such as coverage, do not necessarily exhibit optimal power consumption. Test metrics give users confidence about the software of the application. However, energy consumption metrics are still hidden for most developers and consumers of open-source software, such as the JSON libraries evaluated in this experiment. This means that an application could be, for instance, well tested in terms of having high

coverage but, at the same time, it could not be efficient in terms of energy consumption. The results of this experiment do call to expose non-functional factors such as energy consumption in open-source software.

5. Limitations and Threats to Validity

As our paper presents a preliminary study, our experiment and analysis exhibit a few limitations and threats to its validity:

- All our analyzed libraries are part of a single application domain: JSON processing libraries. This task involves the CPU and memory access and is not representative of the entire scope of desktop and server applications.
- Our study is limited by the number of studied libraries (14 JSON libraries using 152 files, collected by [5]), and therefore a limited number of data points to perform statistical analysis and correlations. We also focus on a single functionality (parse JSON files) from 14 libraries. However, the energy consumption from other functionalities also from those libraries (*e.g.*, store a JSON file on disk) could follow other trends than those we present.
- We execute all our experiments in the same environment (a GNU/Linux laptop). Even though we limited the impact of system background services, our experiment was not conducted in a complete and fully controlled isolated environment. For instance, we did not factor variation of room temperature or CPU heating. To minimize that risk, we execute the experiment for each library 200 times, and report in this paper the average energy consumption.

6. Conclusion

In this paper, we performed a preliminary study of the impact of code coverage and test metrics on software energy. We compared the energy impact of the same workload on 14 Java JSON libraries, and analyzed the statistical correlation with testing metrics. Our initial results show that a positive correlation between line coverage and power consumption exists, and a negative one between uncovered lines and power consumption. However, the limitations of our study do not allow us to provide a definitive conclusion. In future work, we aim to further expand the application domains and the tested software in order to collect additional data to confirm or dispute our hypothesis and research questions.

References

[1] Abhijeet Banerjee, Lee Kee Chong, Sudipta Chattopadhyay, and Abhik Roychoudhury. Detecting energy bugs and hotspots in mobile apps. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 588–598, 2014.

[2] Rabih Bashroush, Eoin Woods, and Adel Noureddine. Data Center Energy Demand: What Got Us Here Won't Get Us There. {*IEEE*} *Software*, 33(2):18–21, 2016.

[3] Maxime Colmant, Romain Rouvoy, Mascha Kurpicz, Anita Sobe, Pascal Felber, and Lionel Seinturier. The next 700 cpu power models. *Journal of Systems and Software*, 144:382–396, 2018.

[4] Dario Di Nucci, Fabio Palomba, Antonio Prota, Annibale Panichella, Andy Zaidman, and Andrea De Lucia. Software-based energy profiling of android apps: Simple, efficient and reliable? In *2017 IEEE 24th international conference on software analysis, evolution and reengineering (SANER)*, pages 103–114. IEEE, 2017.

[5] Nicolas Harrand, Thomas Durieux, David Broman, and B. Baudry. The behavioral diversity of java json libraries. *ArXiv*, abs/2104.14323, 2021.

[6] Reyhaneh Jabbarvand, Alireza Sadeghi, Hamid Bagheri, and Sam Malek. Energy-aware test-suite minimization for android apps. In *Proceedings of the 25th International Symposium on Software Testing and Analysis*, pages 425–436, 2016.

[7] Ding Li, Yuchen Jin, Cagri Sahin, James Clause, and William GJ Halfond. Integrated energy-directed test suite optimization. In *Proceedings of the 2014 International Symposium on Software Testing and Analysis*, pages 339–350, 2014.

[8] Irene Manotas, Christian Bird, Rui Zhang, David Shepherd, Ciera Jaspan, Caitlin Sadowski, Lori Pollock, and James Clause. An empirical study of practitioners' perspectives on green software engineering. In *Proceedings of the 38th International Conference on Software Engineering*, ICSE '16, page 237–248, New York, NY, USA, 2016. Association for Computing Machinery.

[9] Adel Noureddine, Romain Rouvoy, and Lionel Seinturier. Monitoring energy hotspots in software. *Automated Software Engineering*, 22(3):291–332, 2015.

[10] Gustavo Pinto and Fernando Castor. Energy efficiency: A new concern for application software developers. *Commun. ACM*, 60(12):68–75, November 2017.