Empirical Evaluation of Two Best-Practices for Energy-Efficient Software Development

Procaccianti, G.; Fernandez, H.J.; Lago, P.

published in
Journal of Systems and Software
2016

DOI (link to publisher)
10.1016/j.jss.2016.02.035

document version
Early version, also known as pre-print

Empirical Evaluation of Two Best Practices for Energy-Efficient Software Development

Giuseppe Procaccianti*, Hector Fernandez, Patricia Lago

VU University Amsterdam, De Boelelaan 1081a, 1081 HV, Amsterdam, The Netherlands.

Abstract

Background. Energy efficiency is an increasingly important property of software. A large number of empirical studies have been conducted on the topic. However, current state-of-the-Art does not provide empirically-validated guidelines for developing energy-efficient software.

Aim. This study aims at assessing the impact, in terms of energy savings, of best practices for achieving software energy efficiency, elicited from previous work. By doing so, it identifies which resources are affected by the practices and the possible trade-offs with energy consumption.

Method. We performed an empirical experiment in a controlled environment, where we applied two different Green Software practices to two software applications, namely query optimization in MySQL Server and usage of "sleep" instruction in the Apache web server. We then performed a comparison of the energy consumption at system-level and at resource-level, before and after applying the practice.

Results. Our results show that both practices are effective in improving software energy efficiency, reducing consumption up to 25%. We observe that after applying the practices, resource usage is more energy-proportional i.e. increasing CPU usage increases energy consumption in an almost linear way. We also provide our reflections on empirical experimentation in software energy efficiency.

Conclusions. Our contribution shows that significant improvements in software energy efficiency can be gained by applying best practices during design

*Corresponding author

Email addresses: g.procaccianti@vu.nl (Giuseppe Procaccianti), hector.fernandez@vu.nl (Hector Fernandez), p.lago@vu.nl (Patricia Lago)

Preprint accepted for publication in Journal of Systems and Software February 24, 2016

and development. Future work will be devoted to further validate best practices, and to improve their reusability.

Keywords: software engineering, best practices, energy efficiency, empirical experimentation

---

1. Introduction

The energy impact of software has been recognized as significant with respect to the overall energy consumption of its execution environment [1, 2]. Many researchers have been working on sophisticated software power models [3, 4] able to estimate and predict the energy consumption of software applications through different parameters. In spite of this effort, no reusable information is available for practitioners and developers to create energy-efficient software applications. A step in this direction has been made by Larsson et al. [5] from Intel Corp., which provided a number of guidelines and best practices for creating energy-efficient software. However, little to no validation has been performed on those practices, and their effectiveness in terms of energy consumption has not been precisely quantified.

To understand how software can impact on energy consumption, consider the following example[1]: after launch, the popular Youtube video of the "Gangnam Style" song reached a record amount of visualizations during the first year after its publication – roughly 1.7 billion. The amount of energy used by Google to transfer 1MB across the Internet (as reported by the company on their website[2]) is 0.01kWh (a rough average), and displaying it uses 0.002kWh (depending on the destination device). Hence, the energy needed to stream and display the "Gangnam Style" video is 0.19 kWh. Multiplying this amount of energy by the 1.7 billion visualizations gives 312 GWh of total energy consumption, which is roughly the yearly energy demand of a city of 22.000 inhabitants (as an example, the city of Isernia, Italy, consumed 340 GWh of electricity in 2013 [6]).

However, this impressive amount of energy might hide huge wastes. A complex software architecture lies behind modern web applications and ser-

[1] `https://www.2degreesnetwork.com/groups/energy-carbon-management/resources/gangam-style-it-sustainable/`

[2] `http://www.google.com/green/bigpicture/`

vices (e.g. webservers, database servers, middleware) and countless instances are executed every second in physical and virtual environments. Even a tiny optimization on a single software application, on such a massive scale, could potentially lead to significant energy savings. For this reason, software architects and developers need to think about energy efficiency and a solid knowledge base is needed to provide guidance in building energy-efficient software.

The aim of this work is assessing the impact of two best practices for energy-efficient software development on energy consumption. We want to answer the following research questions: 1. what is the energy impact of each practice, 2. what are the main factors causing such impact (i.e. system resources).

Our work follows the guidelines for empirical experimentation in software engineering provided by Wohlin et al. [7] and Basili et al. [8]. For the purpose of this experimentation, we selected two best practices for energy-efficient software development. We elicited those practices inspired by academic literature and industry [9, 10, 11, 12, 13, 5, 14] and collected them in a wiki[3] to share them with academics and practitioners. Such practices were chosen because they are applicable to widely-used and well-known open source software applications (specifically, we selected the Apache WebServer and the MySQL Database Server) that will serve as test cases. These applications were executed in a controlled environment (the Software Energy Footprint Lab, SEFLab [15]). During the experiment, we gathered two types of data: power consumption (both of the execution environment as a whole and of the single hardware components) and usage ratio of the different system resources. Then, we performed hypothesis testing on the data to answer our research questions. Besides assessing the energy impact of each practice, we elicited, for each test case implementation, the factors that we identified as most relevant for energy consumption purposes. From this comparison, we extracted meaningful information to further define the complex relationship between software and energy.

This paper is organized as follows: Section 2 presents our vision for research in software energy efficiency, along with an overview of previous empirical studies on the energy consumption of software applications; Section 3

[3] `https://wiki.cs.vu.nl/green_software/index.php/Best_practices_for_energy_efficient_software`, last visited on December 5th, 2015.

introduces the aim of our contribution and the research questions that drive our research. In Section 4 we present our study design, in terms of subjects, objects, dependent/independent variables and instrumentation. In Section 5 we describe how the experiment was executed. In Section 6 we present our experimental results for each practice and hypothesis testing. In Section 7 we answer our research questions and discuss the implications of our findings. In Section 8 we discuss the validity aspects and possible threats arising from our experiment design and execution. In Section 9 we draw conclusions and outline our future research efforts.

2. Background

2.1. Research Vision

The present contribution falls in the scope of *Green Software* research. *Green software* can be characterized in two ways: (1) the software code being "green", i.e. with a reduced environmental impact, typically in terms of energy consumption, independently of its purpose (as in *green IT*, or *greening of IT*) or (2) the software purpose being "green", improving the environmental impact of the supported processes in their usage context (also denoted as greening by IT [16]). In particular, our contribution is focused on the energy efficiency of software applications, hence we refer to Green Software as in (1). For this reason, in the remainder of this paper the terms "Green Software" and "Energy-efficient software" will be used interchangeably.

Modern technologies (e.g. distributed systems, mobile computation, virtualization and cloud computing) make the relationship between hardware and software more complex, especially in terms of energy use. For this reason, we consider energy efficiency an *emergent* property of software systems in use: the complexity of hardware-software interactions and multiple software layers create an environment that we are currently unable to deterministically describe. Consequently, our research makes use of an *inductive* approach, i.e. we build knowledge on software energy efficiency by gathering and analyzing empirical evidence [17].

Such evidence, obviously, needs to be contextualized in order to provide a consistent body of knowledge. Specifically, the hardware execution environment plays a very important role. For example, as we will show in our analysis of related work in the next section, performing experimentation on mobile devices, servers, or embedded systems results in a variety of different insights – which can often be contradictory. This is why, aside from simply

assessing the impact of best practices, in our research we also try to explain the mechanisms behind such impact, by measuring and analyzing context-specific data. In this paper, we select a small number of best practices and applications as subjects – a decision that clearly poses an issue of generalization to our results (see Section 8) – but we focus on precision of measurements and rigorous data analysis techniques with the goal of increasing the internal validity of our experimentation.

2.2. Related Work

A number of empirical experiments on software energy consumption have been conducted as Green Software became a popular research topic. In this section, we present those which are more related to our contribution, ordered by publication date, and summarize their findings. In Table 1 we give a more structured overview: we list the *purpose* of the study, the experimental *context* (e.g. on-line vs. off-line[7]), the *subjects* selected for the study and the *testbed* on which the energy measurements were performed (where applicable). As criteria for selection, we focused on the viewpoint of developers: hence, we selected studies analyzing the impact of programming techniques or practices on energy consumption, as well as studies that try to empirically characterize energy-intensive code elements.

1. Capra et al. [18] analyze the impact of application development environments over the energy efficiency of software applications. They propose a measure of the impact of application environments on the development process, called *framework entropy*, and evaluate it over a set of 63 open source applications. Hereby we list the main findings of this work:

    - *Finding 1.* A high framework entropy is beneficial for the energy efficiency of small and medium applications.
    - *Finding 2.* A high framework entropy is detrimental for the energy efficiency of large applications.
    - *Finding 3.* Different functional types of applications have different energy efficiency levels.
    - *Finding 4.* ERPs, text, image editors and games are less energy efficient than FTP clients and servers, and calendars.

2. Sahin et al. [19] investigate the energy impact of using software design patterns. They consider a set of 15 design patters and evaluate the energy

Study	Purpose	Context	Subjects	Testbed
[18]	Evaluate energy impact	Off-line single-object	Application Development Environments (63 open-source projects)	Server
[19]	Evaluate energy impact	Off-line single-object	Software Design Patterns (15 Design Patterns in 3 categories)	Embedded System
[20]	Evaluate energy impact	Off-line, multi-object variation	Algorithms and Programming Languages (8 Towers of Hanoi implementations)	Server
[21]	Trace the evolution of software energy consumption	Off-line, multi-object variation	3 products (Firefox, Vuze, rTorrent) in different versions and scenarios	Laptop PC
[22]	Evaluate energy impact	Off-line, multi-object variation	8 Distributed Programming Abstractions on 5 scenarios	Server-Client
[23]	Evaluate energy impact	Off-line single-object	4 web servers on a Web Application	Embedded System
[24]	Evaluate energy impact	Off-line single-object	2 best-practices on a mobile application	Smartphone
[25]	Evaluate energy impact	Off-line, multi-object variation	3 Thread Management Constructs on 8 different benchmarks	Server

Table 1: Summary of the related work.

Study	Purpose	Context	Subjects	Testbed
[26]	Evaluate energy impact	Off-line single-object	3 best practices for energy-efficient programming in Android	Smartphone
[27]	Identify most energy-greedy API calls	Off-line, case study	55 Android applications	Smartphone
[28]	Identify most used programming solutions for energy-efficient software	On-line, thematic analysis	325 questions and 558 answers on Stack Overflow about energy-efficient software	N/A
[29]	Evaluate energy impact	Off-line, multi-object variation	God Class refactoring on 2 open–source software applications	Desktop PC
[30]	Evaluate energy impact	Off-line, multi-object variation	Different Collection implementations on 7 Java Applications	Embedded System
[31]	Evaluate energy impact	Off-line, multi-object variation	6 refactorings on 197 applications	Embedded System
[32]	Evaluate energy impact	Off-line, multi-object variation	Presence of advertisements in 21 mobile applications	Smartphone

Table 2: Summary of the related work (cont'd).

consumption of a "proxy" application developed on purpose for the study. The application is evaluated in two versions, before and after applying the design pattern. Hereby we list the main findings of this work:

- *Finding 1.* The impact of applying a design pattern varies greatly, from less than 1% to more than 700%, among the considered patterns.
- *Finding 2.* The impact of design patterns is not consistent with respect to the pattern category (i.e. Creational, Structural, Behavioral [33]).
- *Finding 3.* The impact of design patterns cannot be predicted by looking at how it influences high-level design artifacts.

3. Noureddine et al. [20] analyze the energy impact of programming languages and algorithmic choices. The impact is evaluated through a low-level library called *PowerAPI* on 8 different implementations of the Towers of Hanoi program, varying the implementation language and the used algorithm (recursive vs. iterative). Hereby we list the main findings of this work:

   - *Finding 1.* The algorithm choice has a significant impact on energy consumption. The recursive algorithm is more energy-efficient than the iterative one.
   - *Finding 2.* The chosen programming language has a significant impact on energy consumption as well. The Java implementation is more energy efficient than the others, not considering compiler optimizations.
   - *Finding 3.* The impact of compiler optimizations is also relevant. Compiling the C++ implementation with the O2 compiler option increases energy efficiency significantly.

4. Hindle [21] investigates the impact of software change on power consumption, and the relationship with software metrics. Subjects are 3 applications: Firefox, Vuze, rTorrent. For each application a set of different versions and releases is selected. Hereby we list the main findings of this work:

   - *Finding 1.* Power consumption is not consistent among different versions.

- *Finding 2.* Performance evolutions can affect power consumption in multiple ways.
- *Finding 3.* No significant correlation was found between static OO-related software metrics (e.g. coupling, cohesion, fan-in/fan-out) and power consumption. Process-related metrics (e.g. added/removed lines, file churn) exhibit positive correlation with power consumption in a limited amount of cases.

5. Kwon and Tilevich [22] analyzed and evaluated the impact in terms of energy consumption of major Distributed Programming Abstractions (DPA) when developing communication mechanisms for mobile devices, such as RPC, RMI or SOAP. Authors implemented 8 versions of different benchmarks for middleware platforms, each version adopting a specific communication abstraction. Hereby we list the main findings of this work:

   - *Finding 1.* Binary-based DPAs (eg. raw sockets) are more energy efficient than XML-based ones, because of the smaller overhead in communication data.
   - *Finding 2.* Asynchronous DPA mechanisms have no additional energy costs.
   - *Finding 3.* Marshaling/unmarshaling consume more energy on network communication than on CPU processing. Serialization protocols are more energy-efficient in high-throughput networks.

6. Manotas et al. [23] investigated the energy impact of web servers in web applications. Using a specialized framework, authors executed a web application with three different web servers and analyzed the resulting energy consumption. Hereby we list the main findings of this work:

   - *Finding 1.* The energy consumption of a web application greatly varies depending on the chosen web server.
   - *Finding 2.* The variation depends on the specific feature of the web server. The same web server might be more energy efficient in a specific scenario (e.g. search) and very inefficient in others (e.g. statistics). Hence, the impact of each web server is not consistent.

7. Tonini et al. [24] evaluate the energy impact of two best-practices proposed by Google to improve performance in Android applications. The

practices can be summarized as: "use appropriate *for* syntax" and "avoid getters/setters". The two best practices were applied in different implementations of the same mobile application and tested upon three different smartphones. Hereby we list the main findings of this work:

- *Finding 1.* Both practices led to statistically significant improvements in both energy efficiency and performance of the tested application.
- *Finding 2.* Improvements span from 20 to 30% for both practices.
- *Finding 3.* Improvements are consistent among different devices.

8. Pinto et al. [25] analyzed and evaluated the impact in terms of energy consumption of 3 different thread management strategies, i.e. *explicit threading, thread pooling, work stealing*, applied on 8 different benchmarks. They also analyzed how energy consumption varies in relationship to the number of active threads. Hereby we list the main findings of this work:

   - *Finding 1.* Different thread management constructs have different impacts on energy consumption. For I/O-bound programs, *explicit threading* is the most energy-efficient, whereas *work stealing* is the least. For highly parallel benchmarks, the opposite holds.
   - *Finding 2.* Energy consumption typically increases as the number of threads increases, and then gradually decreases as the number of threads approaches the number of CPU cores.
   - *Finding 3.* Being faster is not synonymous of being greener. Sequential execution often leads to the least energy consumption, whereas parallel execution leads to improved energy/performance trade-off.

9. Li and Halfond [26] evaluate the impact of 3 best practices for Android application development extracted from the official Android developers community forum. The practices can be summarized as: bundle small HTTP requests, reduce memory usage and improve performance to decrease energy consumption. Authors developed three small software applications to test the impact of each practice. Hereby we list the main findings of this work:

   - *Finding 1.* Bundling small HTTP requests could save energy.

- *Finding 2.* Higher memory usage only slightly increases the average energy consumption of each access.
- *Finding 3.* Avoiding references to the array length for each loop iteration can save energy.
- *Finding 4.* Directly accessing fields instead of accessing them through methods can save energy. The reason is that the virtual methods that are used to access field values are expensive operations.
- *Finding 5.* Static invocation appears to be more energy-efficient in Android.

10. Linares-Vásquez et al. [27] aim at identifying whether some API calls are more energy-consuming than others, and if sequences of API calls (patterns) repeat themselves frequently, causing anomalies in energy consumption. The study analyzed the execution traces of 55 Android applications, looking for the most energy-greedy Android API calls. Hereby we list the main findings of this work:

    - *Finding 1.* APIs related to *GUI & Image Manipulation* and *Database* are the most energy-consuming.
    - *Finding 2.* Using getters and setters when accessing internal class fields causes high energy consumption. This finding is coherent with the previous study, and creates a trade-off between information hiding and energy efficiency.
    - *Finding 3.* Refreshing application views and widgets causes high energy consumption.

11. Pinto et al. [28] mined the StackOverflow platform to find the most common problems regarding energy efficiency, their causes, and the most recommended programming solutions for energy-efficient software. The study found a total of 325 questions and 558 answers from more than 800 software developers. Hereby we list the main findings of this work:

    - *Finding 1.* There are misconceptions about software energy consumption, like confusion between power and energy and the correlation between energy and performance.
    - *Finding 2.* The major causes for energy consumption according to developers are: unnecessary resource usage, hidden background activities, excessive synchronization.

- *Finding 3.* Among the most suggested solutions, those matching with the scientific state-of-the-art were: reduce I/O to a minimum, buffer I/O commands, avoid polling, use efficient data structures.

12. Perez-Castillo and Piattini [29] investigated the energy impact of refactoring "God Class" anti-patterns. They used a tool to detect and refactor multiple occurrences of these anti-patterns in two open source software applications written in Java. They also analyzed the architectural impact of the refactoring by extracting relevant software metrics. Hereby we list the main findings of this work:

    - *Finding 1.* The resulting applications after refactoring were more maintainable, but more energy consuming (power consumption and execution time increased).
    - *Finding 2.* The increase in power consumption and execution time was most likely due to a dramatic increase in message exchange after the refactoring (up to 14 times).
    - *Finding 3.* A quality trade-off between design time (maintainability) and run time (energy consumption) is introduced.

13. Sahin et al. [31] also investigate the energy impact of commonly used refactorings. Authors describe an empirical experiment conducted applying 6 different refactorings to 9 Java applications. Refactorings were chosen with respect to their popularity among developers using the Eclipse IDE. Hereby we list the main findings of this work:

    - *Finding 1.* In almost 30% of the cases, refactorings caused a significant impact over energy consumption.
    - *Finding 2.* The energy impact was not consistent: both negative and positive impacts were determined across different applications and JVM versions.
    - *Finding 3.* Execution time has a moderate correlation with energy consumption, but it is not an accurate predictor.

14. Gui et al. [32] performed a study to assess the hidden costs of mobile ads for developers. They selected 21 apps from the Google Play Store and evaluated the impact of ads, by comparing original versions against instrumented versions obtained via refactoring of the Google Mobile Ads

API calls. Then they analyzed the impact of ads with respect to energy consumption and resource usage (CPU, memory and bandwidth). Hereby we list the main findings of this work:

- *Finding 1.* The energy impact of ads is quite substantial (15% on average). The impact on other resources is even more significant, especially in terms of data traffic (up to 97% increase).
- *Finding 2.* Ads–related changes are a significant part of mobile app maintenance (1 out of 4 releases, in 50% of the apps).
- *Finding 3.* Ads are frequently mentioned in user complaints.

Although our literature search was not conducted systematically, we reviewed a quite representative number of studies that allows us to make some considerations. As emerges from those findings, there are many preliminary insights and hypotheses about software energy efficiency. For example, it seems that large applications with many subsequent versions tend to be less energy-efficient than smaller ones. However, we also observe a certain degree of conflict and uncertainty. For example, some studies seem to show that high-level abstractions and languages are less energy-efficient than low-level ones, while others conclude the opposite. The impact of refactorings and code-level practices has also been addressed from multiple researchers. However, no single best practice or refactoring has been identified as consistently having a predictable impact over energy consumption. A possible reason for this is that there is no unified empirical approach to the problem. Many researchers focus on mobile applications (because in mobile environments, battery life is obviously a high priority), other focus on specific application domains (e.g. Information Systems) or technologies (DPAs). This leads to mostly *anecdotal evidence*, that is, applicable in specific contexts and cases. If our aim as researchers in the field is to provide sound reference and guidance to practitioners when building energy-efficient software, such evidence needs appropriate consolidation and contextualization. This strenghten and motivates our research vision (see 2.1).

3. Research Questions

In this section, we describe the goal of our experimental study inspired by the approach suggested by Basili et al. [8]. As described in Table 3, the aim of this work is the evaluation of the impact of a number of Green

Software practices, in terms of energy consumption, from the perspective of application developers, in the context of open-source software applications.

Table 3: Description of the experimentation goal [8].

Analyze	the impact of Green Software Practices
For the purpose of	evaluation
With respect to	Energy consumption
From the viewpoint of	Software Developers
In the context of	open-source software applications

The experimentation is driven by the following research questions (RQs):

RQ1: What is the impact of each practice in terms of energy consumption?

We answer this question by applying Green Software practices to already existing software applications and then analyzing the energy consumption values caused by the execution of the applications before and after the implementation of the practice. Assessing the impact of each practice helps developers in prioritizing them according to the pursued benefits. The impact (ΔE) is measured in watt-hours (Wh) and expresses the difference between the energy consumption at system level before (E_0) and after (E_1) applying the practice, namely:

$$\Delta E = E_1 - E_0$$

RQ2: Is the relationship between resources and power consumption affected by the application of each practice?

This RQ focuses on resource usage: every software application has a different usage pattern of system resources (CPU, RAM, HDDs, etc.). As shown in previous experiments [2], there is a relevant correlation between resource usage and energy consumption. Answering this RQ allows to identify which resource types drive the energy consumption and how the application of a practice can vary the usage pattern. This helps developers in monitoring the most appropriate resources, prioritize them and establish trade-offs between non-functional aspects (e.g. energy efficiency and performance). In addition, it allows to define whether an application has an *energy–proportional* behavior: in a perfectly *energy–proportional* system, power consumption is a linear

function of the resource usage utilization [34]. Achieving such condition allows to define a constant amount of energy per a certain workload, hence it gives a precise measure of energy efficiency. This is currently a challenge for IT systems, largely due to technological constraints; hence, a software–level technique that increases energy proportionality would be highly useful.

This RQ is answered by analyzing the correlation index among power consumption and the resource usage statistics before and after applying the practice. Power (and not energy) is used in this case as we are interested in analyzing the dynamic behavior of the resources. In particular, we define u_r as the usage ratio of a specific resource r, P as the power consumption at system level and P_r as the power consumption of the component responsible for providing a certain resource (e.g. the CPU for computational resources, the memory banks for the RAM, the motherboard for other I/O peripherals, etc.) . We also define:

- $cor(u_r, P)$ as the correlation index between u_r and P before applying the practice
- $cor(u'_r, P')$ as the correlation index between u_r and P after applying the practice
- $cor(u_r, P_r)$ as the correlation index between u_r and P_r before applying the practice
- $cor(u'_r, P'_r)$ as the correlation index between u_r and P_r after applying the practice

For our analysis we will make use of Spearman's rank correlation coefficient [35]: the possible values for this index are included in the interval $[-1, 1]$, where the endpoints of the interval indicate a perfectly linear function between two variables (hence, in our case, a perfectly power-proportional behavior of a resource).

4. Experiment Planning

This section describes our experiment planning, in terms of dependent and independent variables, hypotheses formulation, and instrumentation [7].

4.1. Variable Selection

The main **object** of our study is the energy impact of two best practices for software energy efficiency, or *Green Software practices*. We elicited those practices inspired by academic literature and industry [9, 10, 11, 12, 13, 5, 14] and collected them in a wiki[4] to share them with academics and practitioners.

For the purpose of this evaluation, we selected two practices from our wiki: *Use efficient queries* and *Put application to sleep*. Those practices were selected for two main reasons: the high relevance for practitioners, as they can be applied in a wide context of software applications, and their well–defined scope of implementation, which allows a better traceability of their impact. The practices are described in Tables 4 and 5, respectively, using the template described in [14]. More details and the rationale behind our implementation choices will be presented in Section 5.

Due to the nature of the practices and their formalization, it was not possible to automate and randomize their application to the subjects, i.e. software applications. Hence, our empirical study qualifies as a **quasi-experiment** [7], as the treatments were manually applied to two selected software applications. We chose two commonly-used, open-source products: the Apache Web Server and the MySQL Database Server. The same criteria used for object selection guided this choice: the wide usage of these products ensures relevance for practitioners, and their open-source nature allowed us to easily access their source code for instrumentation purposes.

The **dependent variables** we monitored and analyzed for answering our RQs are: the energy consumption at system-level to assess the energy impact of the practices; the energy consumption values at resource-level and its usage ratio to identify the most affected resources; and software execution measures (response time, number of request/query served) to determine their relationship with energy consumption and how the application of the practice affects them.

The main **independent variable** for our experimentation is the application of the practices, which we selected as our main factor. This factor identifies two treatments: whether the Green Software practice is applied to a subject software application, or not (absence of treatment). We identified and controlled other independent variables, to avoid potential confounding

[4] `https://wiki.cs.vu.nl/green_software/index.php/Best_practices_for_energy_efficient_software`, last visited on December 5th, 2015.

Table 4: Description of Practice 1: Use efficient queries.

Logical name	Use of efficient queries
Category	Databases
Description	Most Web applications of any size involve the use of a database. Typically, a Web application allows the addition or creation of new records (for example, when a new user registers on the site), and the reading and searching operations of many records in a database. Consequently, the traditional performance bottleneck of Web applications comes from the database. It is often caused by reading operations of a large number of records, or reading operations whose complexity requires an expensive data processing time by the database.
Rationale	Often, database queries perform complex operations, such as ordering or indexing. Those operations are done to increase the application performance at the expense of energy efficiency. Hence, limiting the utilization of indexation mechanisms or unnecessary ordering operations (use of ORDER BY keywords) can mitigate the energy consumption of our queries.
Source	Green Software Wiki
Keywords	database, coding, query

factors. For example, we have a fixed workload for our application, defined by the parameters we used for benchmarking (e.g. total number of requests, database size). We also fixed the software and hardware configuration of the test machine, as described in the remainder of this section.

4.2. Hypotheses Formulation

In the following we formulate the hypotheses that guide our experimentation, starting from our research questions.

- *RQ 1: What is the impact of each practice in terms of energy consumption?*

Table 5: Description of Practice 2: Put application to sleep.

Logical name	Put application to sleep
Category	Energy-efficient/Coding
Description	This practice makes use of a *sleep* function (or equivalent) that puts a process or thread in sleep mode for a specific period of time, i.e. in a Not Runnable state. The *sleep* function suspends the execution of the thread/process, releasing CPU resources, while other threads and processes keep running. Once the thread/process exits the Not Runnable state, it can use CPU resources again.
Rationale	A proper use of the Sleep function (e.g. when the application is no longer active, waiting for I/O or other signals) allows the application to reduce CPU utilization, and consequently improves its energy efficiency.
Source	Green Software Wiki, Wikipedia
Keywords	sleep, coding, thread

$H1_0$: $\Delta E \approx 0$
$H1_a$: $|\Delta E| \gg 0$

The null hypothesis implies a negligible impact of the practice over energy consumption. The alternative hypothesis represents instead an evident and significant impact of the practice.

- *RQ 2: Is the relationship between resources and power consumption affected by the application of each practice?*

$H2_0$: $cor(u_r, P) \approx cor(u_r, P') \; \forall \, r$,
$cor(u_r, P_r) \approx cor(u'_r, P'_r) \; \forall \, r$
$H2_a$: $\exists \, r \mid cor(u_r, P) \neq cor(u'_r, P')$,
$\exists \, r \mid cor(u_r, P_r) \neq cor(u'_r, P'_r)$

Table 6: Summary of experiment planning phase.

Object of Study	Energy impact of Green Software Practices
Subjects	Open-source Software Applications
Factor	Implementation of a practice
Dependent variables	• Energy Consumption values (at system- and resource-level) • Resource usage measures • Software execution measures
Independent variables	• Application workload • Hardware configuration • Software configuration

The null hypothesis implies that the practice does not modify the relationships between the various resources and power consumption in a significant way. The alternative hypothesis instead implies that the practice plays a role in altering this pattern.

4.3. Instrumentation and Testbed

In the following we describe the instrumentation we used in our experimentation, in terms of hardware and software tools. The hardware tools were provided by the Software Energy Footprint Lab (SEFLab) [15], which served as our laboratory environment.

Hardware

The test machine is a Dell PowerEdge SC1425 server, with the following specifications:

- 2x Intel Xeon CPUs, 3.2GHz
- 4x Infineon 1GB DDR2-333 SDRAM

- Intel E7520 chipset
- 1x Maxtor 7L250S0 250GB SATA150 HDD
- Dell power Supply Unit 450W

The instrumentation on this server consists of two Texas Instruments Data Acquisition Boards (DAQs) connected to the power supply channels of the individual resources (e.g. CPUs, memory banks), in order to record the power consumption data of the different components of the server. In addition, this server is also equipped with a Wattsup PRO[5] meter to record system-level power consumption.

Software

The Software instrumentation consists of tools able to collect power consumption data and software-related measures, along with resource usage information. All this data is timestamped, synchronized and stored in comma-separated value (CSV) files.

To collect software measures, we used the Intel Energy Checker (IEC) SDK [6]. This SDK allows developers to insert counters in the application code, to record significant events and/or operational metrics (i.e. the number of queries executed by a DBMS, the time spent in a particular function, etc.). These counters can be exported through the same API, in order to be accessible from other applications at runtime. For power consumption data, we used the Intel Energy Server tool (ESRV). ESRV is part of the IEC SDK and works under the same principle. Basically, ESRV is a simple application able to interface itself with several power meters and DAQs and export the values read by those devices through a software counter, defined in the IEC API. The use of this tool allows us to record both software events/measures and power consumption information (both per-resource, through DAQs, and system-level, through WattsUp PRO) using the same software construct. This reduces noise due to format conversions and synchronization issues.

To collect resource usage data, we used Dstat[7] for Linux/Unix. Dstat allows us to combine the output of various resource monitoring tools commonly

[5] `https://www.wattsupmeters.com/secure/index.php`

[6] Intel Energy Checker SDK, `https://www.youtube.com/watch?v=HbE98AlXSWc`, last visited on December 20th, 2015

[7] http://linux.die.net/man/1/dstat

used in Unix environments (*vmstat, iostat, netstat, ifstat*). In particular, we gathered the following resources:

- CPU statistics (user time, system time, idle time and more)
- Disk statistics (read/write)
- I/O statistics (read/write)
- Memory statistics (paging, used memory, buffered/cached memory, available memory, swap)
- System load (1m, 5m, 15m)
- Network usage (packets sent/received)

The output was collected as CSV files with a sampling interval of 1 second.

5. Execution

5.1. Preparation

The context of our experiment is a **single-object study**: we apply a single object (i.e. a software practice) to a single subject (i.e. a software application). This choice has been made due to the intrinsic complexity of the practice application: as of now, energy-efficient software practices are described in literature as high-level guidelines, hence there are no formal specifications of how to apply a practice to an application. Hence, applying the same practice to multiple applications is a challenging task in terms of generalization of the results. In addition, for our study, we decided to modify the original applications in a non–invasive way, by means of simple refactoring or query modification operations. As a matter of fact, non–invasiveness is one of the criteria we used to select our practices. This limits possible confounding factors and does not jeopardize the original architecture of our application subjects. In this section, we describe how we implemented the practices and the assumptions we made.

For each practice, we developed two different scenarios:

- The first scenario features the test application *with* code instrumentation, *before* applying the software practice under test.
- The second scenario features the test application *with* code instrumentation, *after* applying the software practice under test.

5.1.1. Practice 1: Use efficient queries

We implemented this practice using the MySQL Database Server software. As dataset, we used a full copy of the English Wikipedia articles as of 2008, which has a size of approximately 30GB. The English Wikipedia dataset has been obtained from the WikiMedia Foundation. We designed a simple query that iterates over all Wikipedia pages searching for text fragments. We disabled the MySQL internal cache, which could potentially be a confounding factor, by using the *SQL_NO_CACHE* keyword in the SQL statement. The application of the practice is simulated by issuing two different types of queries: one uses the *ORDER BY* keyword to order the results, the other one doesn't. We developed a benchmark that executes the SQL query 3 times. We decided not to control the duration of the benchmark for this practice: it varies according to the execution time of the queries. This allows us to assess the impact of the practice upon performance. Listings 1 and 2 show the SQL statements we used.

```
SELECT SQL_NO_CACHE a.old_id
FROM text a, revision b
WHERE a.old_id = b.rev_text_id
ORDER BY a.old_id;
```

Listing 1: Query before applying the practice

```
SELECT SQL_NO_CACHE a.old_id
FROM text a, revision b
WHERE a.old_id = b.rev_text_id
```

Listing 2: Query after applying the practice

5.1.2. Practice 2: Put application to sleep

We implemented this practice using a local installation of the Apache WebServer software v.2.2.25. Actually, the application already makes use of the *Put application to sleep* practice when waiting for an HTTP request. We modified the WebServer source code removing every call to the *sleep()* function in the body of the request handling procedure. This modified version represents the subject without the application of the practice. For benchmarking, we used the *ab* utility (Apache Benchmark) included in the WebServer package, with the following parameters:

ab -kc 50 -t 300 -n 5000000 http://localhost/

This configuration issues up to 5000000 requests, with a maximum of 50 concurrent requests and a time limit of 5 minutes (300 seconds). This allows us to control the length of the experiment and extract performance statistics: as opposed to the previous practice, where the number of queries was fixed and the execution time was not, in this case the number of requests varies (the WebServer is not able to process 5 million requests in 5 minutes) but the execution time is fixed.

5.2. Data Collection and Analysis

For each scenario (*before, after*) we performed 10 different executions. During each execution, we collected resource usage data through the CSV output of the *dstat* tool, energy usage through the ESRV tool (at resource level) and the WattsUp PRO meter (at system level) and software execution measures through the IEC API. We carefully checked the timestamps between our different logs to ensure synchronization. As shown in Figure 1, all the logs were collected on a separate *monitor* machine, to minimize the measurement overhead on the test machine. For this reason, the energy meters were electrically connected to the test machine, but the data channels of the meters (i.e. USB cables of the DAQs and the WattsUp PRO) were connected to the monitor machine collecting the data samples. As regards the software measurements and the resource usage data, the *dstat* tool and the IEC API calls were performed on the test machine, but the output CSV logs were remotely written on an NFS shared folder located on the monitor machine.

The analysis of the data was performed using the R software for statistical computing[8]. We applied the following analysis techniques on the data (most of them are described in [36], references provided otherwise):

- Descriptive statistics (e.g. mean, median)
- Shapiro-Wilk test of normality [37]
- Correlation analysis using Spearman's rank coefficient
- Wilcoxon signed-rank test for assessing the impact of the practice
- Effect size computation using Cohen's d, Hedges' g and Vargha-Delaney A measure [38]

[8] `http://www.r-project.org/`

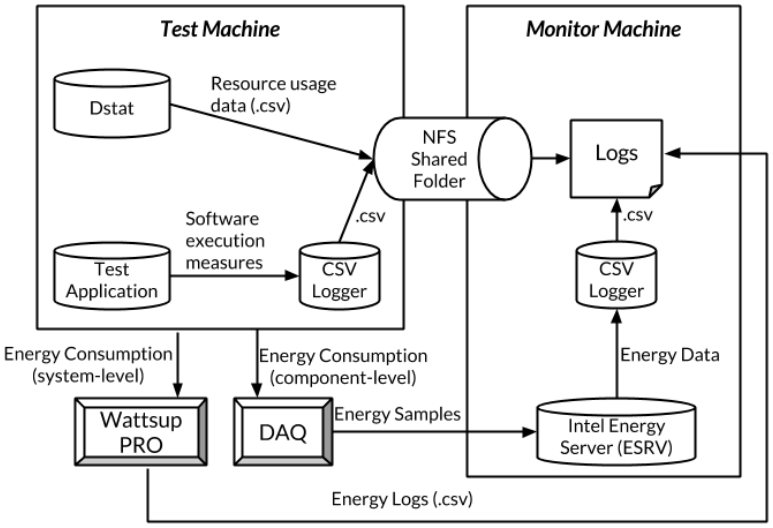


Figure 1: Experiment execution.

We choose a significance level $\alpha = 0.05$ for all of our tests, i.e. we accept a 5% chance of type I error (rejecting the null hypothesis when it is actually true). When evaluating correlations, due to the high number of comparisons we perform (21 comparisons for system-level analysis, 168 for resource-level analysis) we applied the Bonferroni correction to our significance level. Hence, at system level we have

$$\alpha_s = 0.05/21 \approx 0.002 \tag{1}$$

while at resource-level we have

$$\alpha_c = 0.05/168 \approx 0.0003 \tag{2}$$

All the raw data, the reports and the R scripts reproducing the reports are publicly available online[9].

[9] `http://www.s2group.cs.vu.nl/wp-content/uploads/2014/07/green-practices-online-package.zip`

6. Results

In this section, we present the results of our experiment. For each practice, we present the results of hypothesis testing and other relevant findings. We also discuss the results for each practice in detail.

6.1. Practice 1: use efficient queries

6.1.1. Hypothesis testing

RQ1. In Table 7 we summarize the results for hypothesis testing. The Wilcoxon signed-rank test shows that the application of the practice induces a significant decrease in energy consumption (*Z=-3.915, p-value=0.00009*). Thus we can safely reject the null hypothesis.

Table 7: Practice 1: Results of hypothesis testing for RQ1 - Energy consumption

	Before	After
Median	47.85	35.82
Mean	43.83	35.82
% Diff.	-25.1 %	
Wilcoxon's Z	-3.915	
Cohen's d	-140.206 (large)	
Hedges' g	-134.282 (large)	
Vargha & Delaney's A	0 (large)	

RQ2. As regards RQ2, before and after the application of the practice, Spearman's ρ correlation coefficient calculation returned different results in 15 over 21 pairs of resource–energy variables at system level (all p-values $< \alpha_s$, see Equation 1 in Subsection 5.2) and in 88 over 168 pairs of resource–energy variables at resource-level (all p-values $< \alpha_c$, see Equation 2 in Subsection 5.2). Thus, our null hypothesis is rejected. In Table 8 we report all the significant correlation coefficients at system level. In Table 9, for the sake of brevity, we report only the resource-level coefficients that had a significant variation ($\Delta\rho > 0.4$) before and after applying the practice.

Table 8: Practice 1: Results of hypothesis testing for RQ2 - system-level

u	P	$cor(u_r, P)$	$cor(u'_r, P')$
CPUusr	Watts	0.673	0.715
CPUsys	Watts	0.643	-0.163
CPUidl	Watts	-0.189	-0.319
CPUwai	Watts	-0.691	-0.632
CPUsiq	Watts	0.108	0.264
DSKread	Watts	-0.489	0.04
IOread	Watts	-0.696	-0.688
LOAD1m	Watts	-0.087	-0.177
LOAD5m	Watts	-0.079	-0.143
LOAD15m	Watts	-0.06	-0.092
MEMbuff	Watts	0.045	0.127

6.1.2. Discussion on Practice 1

Our hypothesis testing confirms that our first practice, *Use efficient queries*, is successful in increasing energy efficiency. However, some additional considerations need to be done. As emerges from Table 10, the decrease in power consumption is significantly lower than energy, in percentage terms. This indicates that after the application of the practice, there is also an improvement in performance, which is reasonable due to the missing *ORDER BY* clause. Indeed, we report a significant difference in execution time: before the practice, we measured an average of 257 seconds per query, while after the practice the average time per query was 200 seconds.

The decrease in power consumption also indicates a different usage of the resources, as emerges from the results of RQ2: after applying the practice, we can observe a direct correlation rising between CPU activity and motherboard/disk consumption, that indicates a more energy-proportional behavior [34]. This behavior becomes evident when analyzing the relationship between the CPU activity and the system-level power consumption, as shown in Figure 2. Another interesting insight regards the relationship between the I/O operation and power consumption: before the practice, there was no significant correlation, while after we observe negative correlation coefficients. This

Table 9: Practice 1: Most significant results of hypothesis testing for RQ2 - resource-level

u	P_r	$cor(u_r, P_r)$	$cor(u'_r, P'_r)$
CPUusr	MB.5V..Watts.	0.043	0.583
CPUsys	HDD1.5V..Watts.	-0.553	0.156
IOread	HDD1.5V..Watts.	0.618	0.177
IOread	MEM.12v..Watts.	-0.076	-0.628
IOread	MB.5V..Watts.	-0.07	-0.659

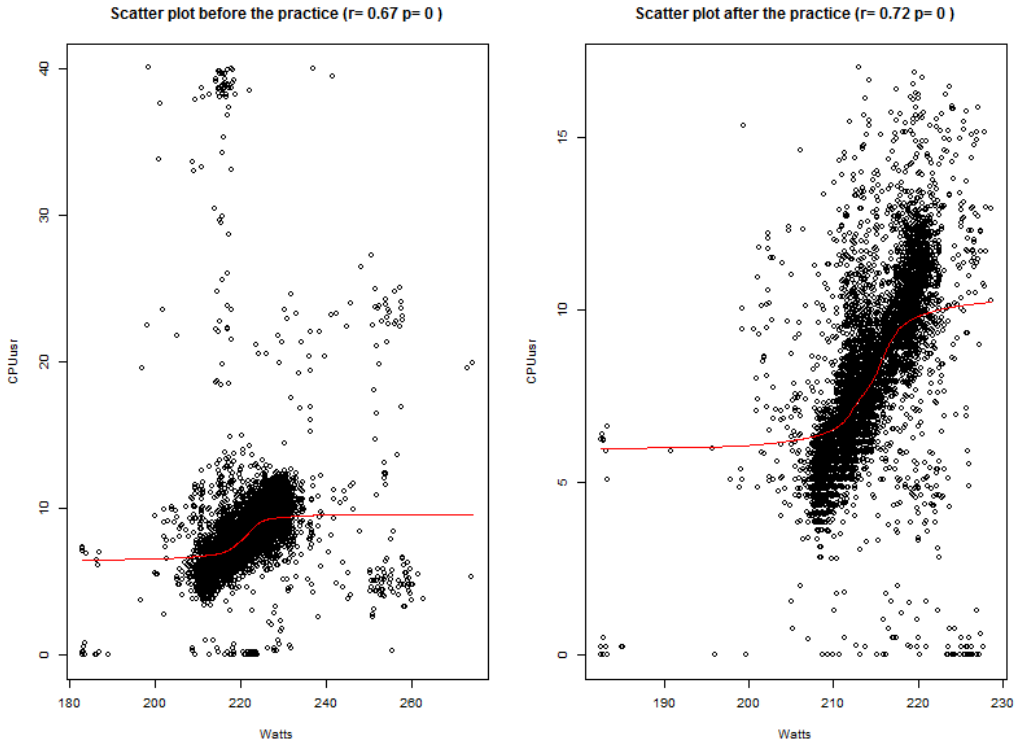


Figure 2: Scatter plots of the variables CPUusr and Watts before and after practice 1 was applied.

might be due to the fact that I/O activity is typically less power–intensive, as the most power consuming resource, the CPU, is inactive. Applying the

Table 10: Practice 1: Effect size analysis for RQ1 - Power Consumption

	Before	After
Median	221.22	214.06
Mean	219.64	213.47
% Diff.	-2.81 %	
Wilcoxon's Z	-3.468	
Cohen's d	-1.775 (large)	
Hedges' g	-1.700 (large)	
Vargha & Delaney's A	0.09 (large)	

practice reduces I/O activity by removing the *ORDER BY* clause which translates in less I/O read and writes, hence the negative impact of I/O over the overall power consumption is more evident, also due to the reduced execution time. This is also supported by the enhanced proportionality between energy and memory usage in Table 8.

6.2. Practice 2: put application to sleep

6.2.1. Hypothesis testing

RQ1. In Table 11 we summarize the results for hypothesis testing. The Wilcoxon signed-rank test shows that the application of the practice induces a significant decrease in energy consumption (*Z=-3.929, p-value=0.00008*). Thus we can safely reject the null hypothesis.

RQ2. As regards RQ2, before and after the application of the practice, Spearman's ρ correlation coefficient calculation returned different results in 7 over 21 pairs of resource–energy variables at system level (all p-values $< \alpha_s$, see Equation 1 in Subsection 5.2) and in 55 over 168 pairs of resource–energy variables at resource level (all p-values $< \alpha_c$, see Equation 2 in Subsection 5.2).Thus, our null hypothesis is rejected. In Table 12 we report all the significant correlation coefficients at system level. In Table 13, for the sake of brevity, we report only the resource-level coefficients that had a significant variation ($\Delta\rho > 0.4$) before and after applying the practice.

Table 11: Practice 2: Results of hypothesis testing for RQ1 - Energy consumption

	Before	After
Median	24.11	22.06
Mean	24.10	22.06
% Diff.	-8.48%	
Wilcoxon's Z	-3.929	
Cohen's d	-42.069 (large)	
Hedges' g	-40.292 (large)	
Vargha & Delaney's A	0 (large)	

Table 12: Practice 2: Results of hypothesis testing for RQ2 - System-level

u	P	$cor(u_r, P)$	$cor(u'_r, P')$
CPUusr	Watts	0.691	0.886
CPUsys	Watts	0.159	0.761
CPUidl	Watts	-0.736	-0.905
CPUsiq	Watts	0.527	0.423
MEMused	Watts	-0.727	-0.849
MEMcach	Watts	-0.43	-0.392
MEMfree	Watts	0.596	0.593

6.2.2. Discussion on Practice 2

The results of hypothesis testing, as for Practice 1, confirm the usefulness of Practice 2 in improving energy efficiency. That being said, the two practices affect energy consumption and resource usage in different ways.

First of all, for Practice 2 there is almost no difference in the impact between energy and power consumption, as shown in Table 14. This indicates a less evident impact on performance: indeed, benchmarks before the practice indicated an average time per request of 0.210 milliseconds, as opposed to 0.196 milliseconds after the practice, hence a mere 6% improvement. However, the average energy consumed per request is 16.89 pWh before the

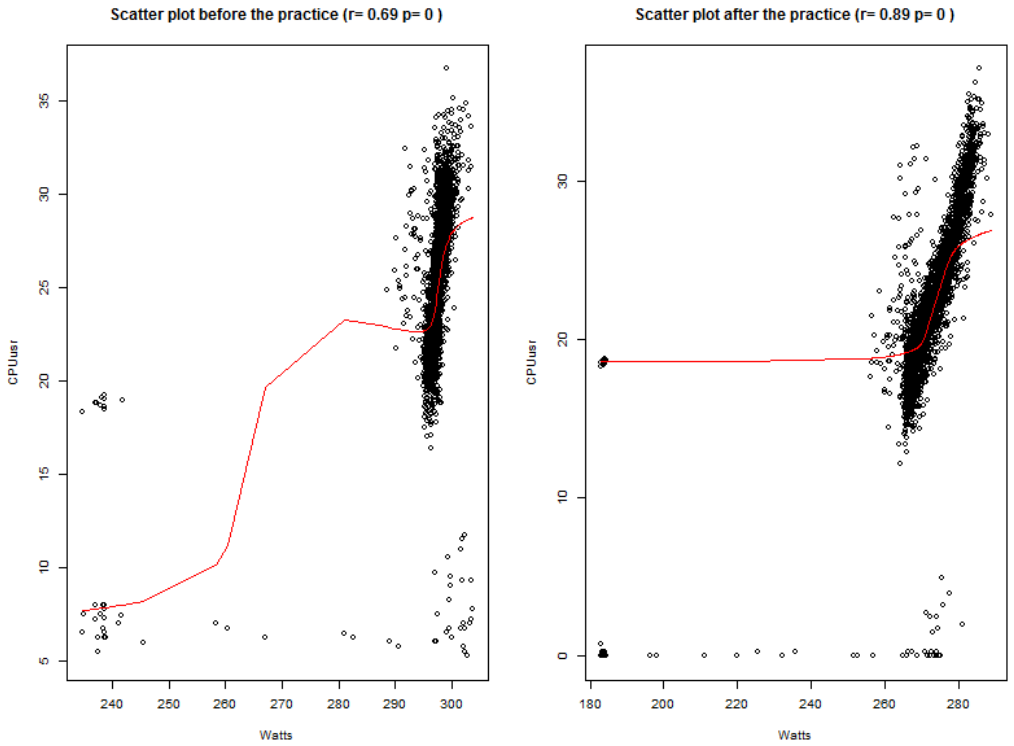
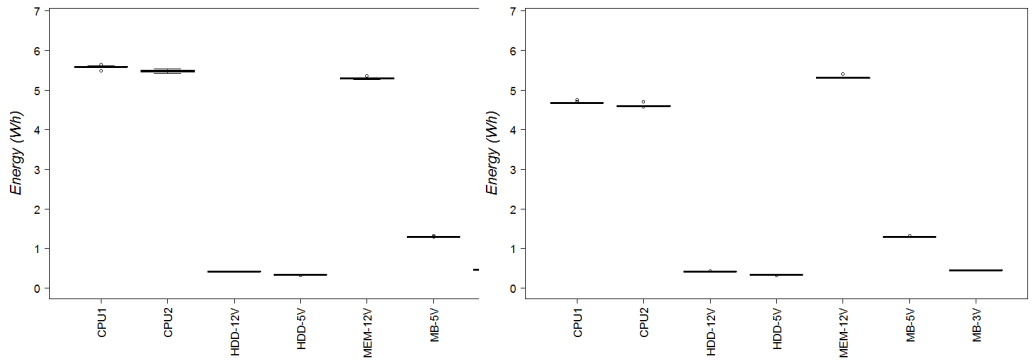


Figure 3: Scatter plots of the variables CPUusr and Watts before and after practice 2 was applied.

practice and 14.41 pWh after, with a reduction of 14%. Thus, energy savings are not only due to a shorter execution time.

The correlation analysis shows us clearly that, as for Practice 1, applying this practice leads to a more energy–proportional behavior, as all CPU–power coefficients increase ($CPUidl$ represents the time spent by the CPU in idle time, which is negatively correlated with power, as expected). This phenomenon becomes evident by looking at the scatter plot in Figure 3.

However, as shown in the box-plot of Figure 4, memory (MEM-12V) also plays an important role: the average energy consumed by the memory amounts to 5.297 Wh per run, as opposed to the 5.47 and 5.58 Wh consumed by the two CPUs (CPU1 and CPU2, respectively).



(a) (b)

Figure 4: Energy consumed per resource before (a) and after (b) applying Practice 2.

Table 13: Practice 2: Results of hypothesis testing for RQ2 - Resource-level

u	P_r	$cor(u_r, P_r)$	$cor(u'_r, P'_r)$
CPUusr	CPU1.12V..Watts.	0.144	0.773
CPUusr	CPU2.12V..Watts.	0.078	0.8
CPUusr	MB.5V..Watts.	0.086	0.631
CPUsys	MEM.12v..Watts.	0.117	0.655
CPUidl	CPU1.12V..Watts.	-0.158	-0.793
CPUidl	CPU2.12V..Watts.	-0.075	-0.815
CPUidl	MB.5V..Watts.	-0.095	-0.655
MEMused	CPU1.12V..Watts.	-0.111	-0.771
MEMused	CPU2.12V..Watts.	-0.103	-0.788
MEMfree	CPU1.12V..Watts.	0.072	0.553
MEMfree	CPU2.12V..Watts.	0.111	0.55

7. Reflection

In this section, we provide the reader with some reflection points that emerge from our results.

The first point regards the main outcome of this study: the impact of

Table 14: Practice 2: Effect size analysis for RQ1 - Power Consumption

	Before	After
Median	297.06	272.63
Mean	296.62	272.06
% Diff.	-8.28%	
Wilcoxon's Z	-4.548	
Cohen's d	-16.890 (large)	
Hedges' g	-16.176 (large)	
Vargha & Delaney's A	0 (large)	

the Green Software practices. We reported very significant energy consumption reductions, up to 25%, that show the relevance of Green Software research. This relevance even increases when considering emerging contexts such as Cloud Computing and Big Data, where software applications run in countless instances, hence their energy efficiency becomes crucial. Another relevant context is High Performance Computing: as we approach the so-called Exascale Computing era [39], when the power consumption of HPC systems is predicted to be in the order of tens of MWs, hardware engineers and researchers are already developing extremely energy efficient technologies. Software engineering, in turn, does not provide yet a consistent body of knowledge on energy efficiency.

Indeed, the related work we summarized in Section 2 reports a number of findings in heterogeneous contexts, sometimes presented as suggestions/guidelines for developers. In this work, we present two Green Software practices, documented with a structured template, and we assess their impact. The template gives an added value in terms of reusability of the practices, while the characterization of their impact helps developers in planning for reaching a specific level of energy efficiency, provided the compatibility of the practice with the application domain and other possible constraints.

The two Green Software practices we investigated can be applied on a variety of contexts and applications. As we discuss in the next Section, we cannot generalize our findings to such a wide population. However, from the answer to our RQ2 it emerges that the application of the practices significantly impacts the usage of system resources like CPU, memory and I/O –

which are quite consistent, at least from a logical standpoint, throughout all IT platforms. Hence, the enhanced energy proportionality stemming from the application of the practices will most likely be observed in other similar contexts.

The impact of the practices is also very different in nature. The energy impact of Practice 1 can be largely explained by a performance increase, due to less I/O operations being performed. The impact of Practice 2, on the other hand, is more complex to explain: definitely, the usage of sleep instructions modifies the execution scheme of the different threads of the Apache Webserver. Using the sleep instruction forces a thread to release the CPU, hence allowing a more efficient interleaving between the threads of the webserver pool. This results in comparable performance, but more efficient CPU usage, hence less energy consumption. This complex interaction shows how contextualized empirical evidence is crucial for professionals in need to make informed decisions about the energy efficiency of their applications.

If we look at the impact of Green Software on the software engineering process, it introduces energy efficiency as a new concern that potentially affects all phases of software development. From an architectural perspective, we already provided elements of reusable knowledge on software energy efficiency, in the form of architectural tactics [40]. Accordingly, reusable Green Software practices guide software architects and developers in making *green* design decisions and implementation choices, hence ensuring software to be energy efficient in the first place. However, it has to be mentioned that energy efficiency is possibly in trade–off with other software quality attributes. For example, applying Practice 2 ("put application to sleep") might result in a performance loss, although this was not the case in our experiment. Other practices that require more invasive refactorings of the application (e.g. "reduce abstraction layers") might have a negative impact on maintanability. However, more research is required to evaluate and assess such trade–offs, most likely at the level of software architectures. This is part of our research agenda [40].

Finally, our experimental design and setting represents an important part of our scientific contribution. A dedicated laboratory environment for assessing software energy efficiency, such as the SEFLab, represents the starting point of a sound methodology for Green Software research, as well as a future testbench, when software testing for energy efficiency will be common practice, as it is now for other software qualities. The tools and analysis methods we adopted fit into a more general, reusable framework for Green Software

Engineering [41, 42], that we will further develop in our next research and education [43] efforts.

8. Threats to Validity

In this section we present the threats to validity and their mitigation. Our aim is to illustrate the premises and the assumptions behind our experimentation. The classification of the threats follows the one by Cook and Campbell [44].

8.1. Conclusion validity

Threats to conclusion validity affect the statistical significance of the findings. In our experimentation, we identify the following conclusion validity threats:

- *Reliability of measures.* When performing energy consumption analysis, the precision and accuracy of the measurement equipment is of utmost importance. For this reason, we performed our measurement in the SEFLab, a state-of-the-Art laboratory purposely built to perform energy consumption analysis. We also collaborated with the staff and technicians who set up the lab in order to ensure the highest measurement quality (see Acknowledgements).

- *Reliability of treatment implementation.* As mentioned in Section 5, the application of the Green Software Practices to our application subjects is a complex process that cannot be standardized or automated (this is also a threat to construct validity, see below). Hence, we cannot guarantee that a different implementation would give similar results. To mitigate this threat, the implementation of the practice was performed by two different researchers and its meaningfulness was cross-checked with experts in the field.

8.2. Internal validity

Threats to internal validity affect the interpretation of our findings as regards the causality link between treatment and outcome.

- *Treatment assignment.* As we stated in Section 4, our empirical study is a *quasi-experiment*, for feasibility reasons: the assignment of a practice to a single software application is an operation that cannot be

automated nor randomized at the present time. We are aware that this prevents us from fully establish causation. Part of our future efforts (see Section 9) will be devoted to generalizing the practices in order to make them automatically applicable on multiple products, hence enabling randomized assignment.

- *Code instrumentation.* Due to the usage of the IEC API (see Section 4), we had to insert several additional calls in our application subjects. That might result in a confounding factor. To mitigate this threat, we also performed a benchmark of each application before performing instrumentation (a so-called *vanilla* scenario). This allowed us to estimate the impact of the instrumentation and take it into account.

8.3. Construct validity

Threats to construct validity affect the relationship between theory and observation. Our main threat to construct validity regards the *operational explication of constructs*, meaning that the practices are not formalized in a standard and objective way, hence their translation into operational constructs is subject to interpretation. To mitigate this threat, we documented the practices to the best of our knowledge and we provided references of their sources. We seek, however, for researchers to challenge our interpretation and provide further examples to improve our knowledge base on Green Software practices.

8.4. External validity

Threats to external validity affect the generalization of our findings. This aspect is also discussed in Section 2 where we describe our research vision. We identified the following external validity threats:

- *Subject selection.* For feasibility reasons, also due to the complexity of the application of a practice, we designed a single-object study, so we selected only two software applications for our study. Accordingly, we cannot guarantee that our subjects are representative of the whole population. To mitigate this risk, we chose our application subjects to be as representative as possible, being widely-used open source software applications.

- *Experimental setting.* We conducted our experiments in a controlled environment. Hence, we cannot guarantee that our results would be

the same in a different setting, e.g. the production environment of a company. However, the test machine and the application versions were as up-to-date as possible and they are widely used in industrial settings as well.

9. Conclusions

In this contribution, we presented empirical validation of two best practices for energy-efficient software development, i.e. Green Software practices, selected from a collection of practices elicited from both academic and industrial sources. Although energy efficiency is gaining importance as a crucial property for software systems, the related literature did not provide, up to now, a consistent body of knowledge to guide practitioners to design and develop energy-efficient software. Our contribution is a first step in that direction.

For our study we selected two Green Software practices, *Put application to sleep* and *Use efficient queries*, and we applied them on well-known and widely adopted open source products. The results of our empirical experimentation show a significant and consistent impact of both practices in increasing the energy efficiency of the selected subjects, namely the Apache WebServer and the MySQL Database Server. A more detailed analysis also showed that the practices significantly alter the resource usage pattern of the applications, inducing a more energy-proportional behavior after their application.

We have set up a wiki to distribute our practices, along with a structured template for their documentation where we will include the results of our empirical validation, so that practitioners can learn how adopting a practice can improve the energy efficiency of their product. In the next future, we plan to set up a regular experimentation activity on all the practices collected so far, formalizing our current experimental design and incrementally building a general framework for empirical research on Green Software. Other future efforts will be aimed at generalizing the Green Software practices, by abstracting the general concepts behind them and eventually providing automated refactoring procedures to apply them on existing products. This will speed up our experimentation and hence increase the amount (and maturity) of evidence we can gather through our methodology. In a previous publication [42] we also propose an experimentation approach to increase generalization: we observe software behavior on a large scale (e.g. at datacenter level)

and elicit potential candidates for energy optimization ("energy hotspots") by analyzing a large amount of data from multiple levels of abstraction and different hardware contexts. Then, in order to increase internal validity, we perform an in-depth experimentation, reproducing such candidate hotspots in a controlled environment.

Our ultimate goal is to increase the validity of empirical results on Green Software and build a solid, empirically sound body of knowledge for Green Software Engineering.

Acknowledgments

This work has been partially sponsored by the European Fund for Regional Development under project MRA Cluster Green Software. The authors would like to thank Bo Merkus and Eric Hoekstra for their support in setting up and operating the equipment in the SEFLab.

[1] E. Capra, C. Francalanci, S. A. Slaughter, Measuring application software energy efficiency, IT Professional 14 (2) (2012) 54–61. doi:http://doi.ieeecomputersociety.org/10.1109/MITP.2012.39.

[2] G. Procaccianti, L. Ardito, A. Vetro, M. Morisio, Energy Efficiency in the ICT - Profiling Power Consumption in Desktop Computer Systems, in: Energy Efficiency - The Innovative Ways for Smart Energy, the Future Towards Modern Utilities / InTech, Prof. Moustafa Eissa, Helwan, 2012, pp. 353–372. doi:10.5772/48237.
URL `http://porto.polito.it/2502197/`

[3] A. Sinha, A. P. Chandrakasan, Energy aware software, in: Thirteenth International Conference on VLSI Design, IEEE, 2000, pp. 50–55.

[4] A. Kansal, F. Zhao, Fine-grained energy profiling for power-aware application design, SIGMETRICS Perform. Eval. Rev. 36 (2) (2008) 26–31. doi:10.1145/1453175.1453180.
URL `http://doi.acm.org/10.1145/1453175.1453180`

[5] P. Larsson, Energy-efficient software guidelines, Tech. rep., Intel Software Solutions Group (2011).
URL `https://goo.gl/YuGKR8`

[6] TERNA, Dati Statistici sull'energia elettrica in Italia, Tech. rep., SISTAN (2013).
URL http://download.terna.it/terna/0000/0607/94.ZIP

[7] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, A. Wesslén, Experimentation in software engineering, Springer, 2012.

[8] V. R. Basili, R. W. Selby, D. H. Hutchens, Experimentation in software engineering, IEEE Transactions on Software Engineering SE-12 (7) (1986) 733–743.

[9] T. Simunic, L. Benini, G. De Micheli, Energy-efficient design of battery-powered embedded systems, IEEE Trans. Very Large Scale Integr. VLSI Syst. 9 (1) (2001) 15–28.

[10] E. Saxe, Power-efficient software, Commun. ACM 53 (2) (2010) 44–48. doi:10.1145/1646353.1646370.
URL http://doi.acm.org/10.1145/1646353.1646370

[11] N. Jones, Eight software approaches can enable Energy-Efficient computing, Tech. Rep. G00151775, Gartner (16 Oct. 2007).

[12] B. Steigerwald, R. Chabukswar, K. Krishnan, J. Vega, Creating Energy-Efficient software, Tech. rep., Intel Corp. (2007).

[13] W. Baek, T. M. Chilimbi, Green: A framework for supporting energy-conscious programming using controlled approximation, in: Proceedings of the 2010 ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '10, ACM, New York, NY, USA, 2010, pp. 198–209.

[14] S. Gude, Energy efficient software, Master's thesis, VU University Amsterdam (Sep 2010).

[15] M. Ferreira, E. Hoekstra, B. Merkus, B. Visser, J. Visser, Seflab: A lab for measuring software energy footprints, in: Green and Sustainable Software (GREENS), 2013 2nd International Workshop on, 2013, pp. 30–37.

[16] P. Lago, S. A. Koçak, I. Crnkovic, B. Penzenstadler, Framing sustainability as a property of software quality, Commun. ACM 58 (10) (2015)

70–78. doi:10.1145/2714560.
URL http://doi.acm.org/10.1145/2714560

[17] V. R. Basili, The experimental paradigm in software engineering, in: Experimental Software Engineering Issues: Critical Assessment and Future Directions, Lecture Notes in Computer Science, Springer Berlin Heidelberg, 1993, pp. 1–12.

[18] E. Capra, C. Francalanci, S. A. Slaughter, Is software "green"? application development environments and energy efficiency in open source applications, Information and Software Technology 54 (1) (2012) 60–71. doi:10.1016/j.infsof.2011.07.005.
URL http://dx.doi.org/10.1016/j.infsof.2011.07.005

[19] C. Sahin, F. Cayci, I. L. M. Gutierrez, J. Clause, F. Kiamilev, L. Pollock, K. Winbladh, Initial explorations on design pattern energy usage, in: Green and Sustainable Software (GREENS), 2012 First International Workshop on, 2012, pp. 55–61.

[20] A. Noureddine, A. Bourdon, R. Rouvoy, L. Seinturier, A preliminary study of the impact of software engineering on Green IT, in: First International Workshop on Green and Sustainable Software (GREENS), IEEE, 2012, pp. 21–27.

[21] A. Hindle, Green mining: a methodology of relating software change and configuration to power consumption, Empirical Software Engineering 20 (2) (2015) 374–409. doi:10.1007/s10664-013-9276-6.
URL http://dx.doi.org/10.1007/s10664-013-9276-6

[22] Y.-W. Kwon, E. Tilevich, The impact of distributed programming abstractions on application energy consumption, Information and Software Technology 55 (9) (2013) 1602–1613.

[23] I. Manotas, C. Sahin, J. Clause, L. Pollock, K. Winbladh, Investigating the impacts of web servers on web application energy usage, in: Green and Sustainable Software (GREENS), 2013 2nd International Workshop on, ieeexplore.ieee.org, 2013, pp. 16–23.

[24] A. R. Tonini, L. M. Fischer, J. C. B. de Mattos, L. B. de Brisolara, Analysis and evaluation of the android best practices impact on the efficiency

of mobile applications, in: 2013 III Brazilian Symposium on Computing Systems Engineering (SBESC), IEEE Computer Society, 2013, pp. 157–158.

[25] G. Pinto, F. Castor, Y. D. Liu, Understanding energy behaviors of thread management constructs, in: Proceedings of the 2014 ACM International Conference on Object Oriented Programming Systems Languages & Applications, OOPSLA '14, ACM, New York, NY, USA, 2014, pp. 345–360. doi:10.1145/2660193.2660235.
URL http://doi.acm.org/10.1145/2660193.2660235

[26] D. Li, W. G. J. Halfond, An investigation into energy-saving programming practices for android smartphone app development, in: Proceedings of the 3rd International Workshop on Green and Sustainable Software, GREENS 2014, ACM, New York, NY, USA, 2014, pp. 46–53. doi:10.1145/2593743.2593750.
URL http://doi.acm.org/10.1145/2593743.2593750

[27] M. Linares-Vásquez, G. Bavota, C. Bernal-Cárdenas, R. Oliveto, M. Di Penta, D. Poshyvanyk, Mining energy-greedy api usage patterns in android apps: An empirical study, in: Proceedings of the 11th Working Conference on Mining Software Repositories, MSR 2014, ACM, New York, NY, USA, 2014, pp. 2–11. doi:10.1145/2597073.2597085.
URL http://doi.acm.org/10.1145/2597073.2597085

[28] G. Pinto, F. Castor, Y. D. Liu, Mining questions about software energy consumption, in: Proceedings of the 11th Working Conference on Mining Software Repositories, MSR 2014, ACM, New York, NY, USA, 2014, pp. 22–31. doi:10.1145/2597073.2597110.
URL http://doi.acm.org/10.1145/2597073.2597110

[29] R. Perez-Castillo, M. Piattini, Analyzing the harmful effect of god class refactoring on power consumption, Software, IEEE 31 (3) (2014) 48–54.

[30] I. Manotas, L. Pollock, J. Clause, SEEDS: A software engineer's energy-optimization decision support framework, in: Proceedings of the 36th International Conference on Software Engineering, ICSE 2014, ACM, New York, NY, USA, 2014, pp. 503–514.

[31] C. Sahin, L. Pollock, J. Clause, How do code refactorings affect energy usage?, in: Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM '14, ACM, New York, NY, USA, 2014, pp. 36:1–36:10.

[32] J. Gui, S. Mcilroy, M. Nagappan, W. G. J. Halfond, Truth in advertising: The hidden cost of mobile ads for software developers, in: Proceedings of the 37th International Conference on Software Engineering - Volume 1, ICSE '15, IEEE Press, Piscataway, NJ, USA, 2015, pp. 100–110.

[33] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Pearson Education, 1994.

[34] L. A. Barroso, U. Holzle, The case for energy-proportional computing, Computer 40 (12) (2007) 33–37. doi:http://doi.ieeecomputersociety.org/10.1109/MC.2007.443.

[35] C. Spearman, The proof and measurement of association between two things, Am. J. Psychol. 15 (1) (1904) 72–101.

[36] D. C. Montgomery, Design and Analysis of Experiments, Student solutions manual, John Wiley & Sons, 2008.

[37] S. S. Shapiro, M. B. Wilk, An analysis of variance test for normality (complete samples), Biometrika 52 (3/4) (1965) 591–611.

[38] A. Vargha, H. D. Delaney, A critique and improvement of the "CL" common language effect size statistics of McGraw and wong, J. Educ. Behav. Stat. 25 (2) (2000) 101–132.

[39] T. H. Szymanski, Impact of future trends on exascale grid and cloud computing, in: Supercomputing, Springer International Publishing, 2014, pp. 215–231.

[40] G. Procaccianti, P. Lago, G. Lewis, Green architectural tactics for the cloud, in: Software Architecture (WICSA), 2014 IEEE/IFIP Conference on, 2014, pp. 41–44. doi:10.1109/WICSA.2014.30.

[41] L. Ardito, G. Procaccianti, M. Torchiano, A. Vetrò, Understanding green software development: A conceptual framework, IT Professional 17 (1) (2015) 44–50.

[42] G. Procaccianti, P. Lago, A. Vetro, D. Mendez Fernandez, R. Wieringa, The green lab: Experimentation in software energy efficiency, in: Software Engineering (ICSE), 2015 IEEE/ACM 37th IEEE International Conference on, Vol. 2, 2015, pp. 941–942. doi:10.1109/ICSE.2015.297.

[43] P. Lago, A master program on engineering energy-aware software, in: J. M. Gómez, M. Sonnenschein, U. Vogel, A. Winter, B. Rapp, N. Giesen (Eds.), Proceedings of the 28th Conference on Environmental Informatics - Informatics for Environmental Protection, Sustainable Development and Risk Management, BIS-Verlag, 2014, pp. 469–476, iSBN 978-3-8142-2317-9.

[44] T. D. Cook, D. T. Campbell, Quasi-experimentation: Design & analysis issues for field settings, Houghton Mifflin Company, Boston, 1979.