JOURNAL OF SOFTWARE: EVOLUTION AND PROCESS
J. Softw. Evol. and Proc. 2016; **00**:1–34
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/smr

Energy efficiency on the product roadmap: an empirical study across releases of a software product

E. A. Jagroep[1,3*], G. Procaccianti[2], J. M. E. M. van der Werf[1], S. Brinkkemper[1], L. Blom[3], R. van Vliet[3]

[1] *Utrecht University, Department of Information and Computing Sciences, Princetonplein 5, 3584 CC Utrecht, The Netherlands*
[2] *Vrije Universiteit Amsterdam, Department of Computer Science, De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands*
[3] *Centric Netherlands B.V., P.O. Box 338, 2800 AH Gouda, The Netherlands*

SUMMARY

In the quest for energy efficient ICT, research has mostly focused on the role of hardware. However, the impact of software on energy consumption has been acknowledged as significant by researchers in software engineering. In spite of that, due to cost and time constraints, many software producing organizations are unable to effectively measure software energy consumption preventing them to include energy efficiency in the product roadmap.
In this paper, we apply a software energy profiling method to reliably compare the energy consumed by a commercial software product across two consecutive releases. We demonstrate how the method can be applied and provide an in-depth analysis of energy consumption of software components. Additionally, we investigate the added value of these measurement for multiple stakeholders in a software producing organization, by means of semi-structured interviews.
Our results show how the introduction of an encryption module caused a noticeable increase in the energy consumption of the product. Such results were deemed valuable by the stakeholders and provided insights on how specific software changes might affect energy consumption. In addition, our interviews show that such a quantification of software energy consumption helps to create awareness and eventually consider energy efficiency aspects when planning software releases.

Received . . .

KEY WORDS: Energy Efficiency; Profiling; Product Roadmap; Software Product

*Correspondence to: Utrecht University, Department of Information and Computing Sciences, Princetonplein 5, 3584 CC Utrecht, The Netherlands. E-mail: e.a.jagroep@uu.nl

1. INTRODUCTION

In order to make the Information and Communication Technology (ICT) sector more environmentally sustainable, research has mostly focused on hardware improvements. Indeed, every new generation of hardware improves its Energy Efficiency (EE) by either increased performance (i.e. more performance per Watt) or decreased Energy Consumption (EC) in absolute terms. Considering the growing number of hardware devices, the impact of these improvements can be significant. However, a crucial aspect that has been long overlooked is the role of software [1]. Although hardware ultimately consumes energy, software provides the instructions that guide the hardware behavior [2].

The sustainability of software is still in its infancy as a research topic. Previous work [3,4] defines sustainability as a multi–dimensional concept that identifies requirements for multiple software Quality Attributes (QAs). In particular, *environmental* sustainability identifies EC requirements for energy efficiency. Sustainability requirements also impact other QAs: for example, in the mobile domain, the EC of mobile applications directly impacts usability, as it shortens battery life [5–7].

Despite this, we do not witness a significant increase in the energy efficiency of mobile applications over time. On the contrary, software updates require the user to buy a new mobile phone every few years, sometimes even without a clear benefit in terms of performance. Additionally, new phones are often equipped with higher capacity batteries, to prevent deterioration of the operation time. Looking at larger software products, e.g. business applications, a similar pattern can be observed. Depending on the deployment, increasingly more powerful hardware is required to run new releases of applications. In contrast to the mobile domain, though, EC measurements on business software products are more complicated to perform. The diversity of deployments and levels of abstraction (e.g. virtualization and cloud computing) require more sophisticated measurement approaches to properly analyze software EC [8]. Recently, several of such approaches have been proposed, both hardware [9] and software based [10], which were able to identify opportunities for considerable savings in EC.

However, these approaches have not been adopted in industrial contexts so far. While Software Product Organizations (SPOs), e.g. independent software vendors and open-source foundations, have software development as their core activity [11], having accurate software EC measurements still requires significant investments in terms of resources and specialized knowledge [12]. As a consequence SPOs do not plan the evolution of their product, i.e. with a product roadmap [13], on its energy efficiency aspect, potentially leading to not meeting market requirements [14]. For example, in the Netherlands the government specifies EC related requirements in their tenders.

In practice, performance is often used as a proxy for energy efficiency. Software performance optimization is a more mature field of study, hence more people with such skills are available on the market. However, although much can also be derived from performance measurements, EC and performance are not always positively correlated [15–19]; contradicting goals could require a trade-off to be made [3]. Hence, a deeper understanding of the matter is required to properly address the EC of the software itself.

For this purpose, we formulate the following **main research questions**:

RQ1: *How can we reliably compare the energy consumption of large-scale software products across different releases?*

In RQ1, we explicitly refer to large-scale software products as multi-tenant, multi-user distributed software applications, as opposed to e.g. single-user mobile applications which are out of scope for this study. RQ1 is further divided in 2 sub-research question (Section 5):

- **SQ1**: *How can we reliably measure the EC of a software product?* A prerequisite for comparing the EC of a software product is being able to measure the software EC.
- **SQ2:** *How can we attribute energy consumption to individual software elements?* For SPOs to actually optimize the EC of their products, it is necessary to identify how individual software elements affect EC. For a more precise definition of what we mean as a software element, see Section 3.

In Section 3 we describe the design of an experiment where we used software profilers to obtain fine-grained, software-level estimations and validated them with hardware measurements obtained via power meters. The results of this experiment allow us to answer RQ1.

Additionally we investigate the benefits of measuring the EC of a software product for stakeholders in SPOs responsible for a product. Comparing EC across releases of a software product will, most likely, only be done when there is added value from this effort. To put EC on the product roadmap [20] we formulated a second main research question:

RQ2: *What is the added value for a software producing organization to perform EC measurements on software products?*

In Section 3, we describe the design of a secondary empirical study encompassing interviews with the stakeholders from an SPO. The results of this study allow us to answer RQ2, from the perspectives of the different roles involved in software product development.

This paper extends our previously published work [21] in several ways. First of all, we performed a more in-depth analysis of the data, i.e. including software metrics in the analysis, propose a technique to visualize the results in the form of radar graphs and discuss the impact of energy consumption in software design. Moreover, this study poses an additional Research Question (RQ2), answered by means of a series of interviews with practitioners from the SPO which provided the product for our experimentation. During the interviews, we discussed our experimental results and their implications for their product related activities.

The remainder of this article is organized as follows: in Section 2 we review the related work. In Section 3, Section 4 and Section 5 we describe the design, execution and results of our empirical studies (experiment and interviews). We discuss the results in Section 6 and threats to validity in Section 7. Concluding remarks and an outline for future work are provided in Section 8.

2. RELATED WORK

2.1. Product Roadmap

To identify the added value of EC measurements for product development, a basic understanding of the product dynamics is required. Changes in the product market have significantly shifted the focus of software development towards the goal of achieving competitive advantage [22]. Since EC could be considered as a non-functional, strategic aspect of software [3, 4], this topic fits the software product management competence model [14] in the area of product planning, or more specifically product roadmapping. The product, or software, roadmap translates strategy into short- and long-term plans and could be considered as planning the evolution of a product [13].

An important aspect for creating a roadmap is being aware of the lifecycle phase a product is in; beginning of life, middle of life or end of life [23]. Depending on the phase different drivers, economical and technical, direct investments for the product, taking into account the current position of the product in the market. SPOs are, for example, not eager to invest in technology that has become obsolete in a specific market segment. Depending on the lifecycle phase the SPO could consider different investment strategies to minimize losses.

Parallel to the three phases, a different lifecycle representation is presented by Ebert and Brinkkemper [20] ranging from strategic management, product strategy, product planning, development, marketing, sales and distribution to service and support. The beginning of life phase is characterized by creating a product strategy and planning, which leads to the initial development of the product. Development continues in the middle of life phase where the marketing, sales and distribution, service and support activities are key to deliver a ‘mature’ product to the market and keep the product financially viable. During the end of life phase marketing, sales and distribution, service and support activities are key to minimize costs and stretch the financial viability of the product. If required, a substitute product is sought when a current product is considered end of life. Typically major investments are done in the first two stages of the lifecycle.

From an EC point of view the first two stages are where the product team forms and executes short- and long-term plans for a product and where measuring the EC could prove helpful to increase the product success. Sales, an internal stakeholder for a software product [14], could benefit from having low EC as a unique selling point for the software product. When nearing the end of life phase a product, its EC characteristics potentially contribute to extending the lifecycle by ,e.g., lowering the total cost of ownership.

Apart from creating the roadmap the product manager, the one responsible for the future of a product [20], also has to ensure development activities are in line with the roadmap. Among others, developers should obtain requirements based on the roadmap and the team has to plan their releases (or sprints) to fulfill these requirements. Not meeting the requirements, or not meeting them in time, could potentially negatively affect the success of the software product.

2.2. Software Energy Consumption Measurements

The available techniques for measuring software energy consumption are rapidly advancing, however a distinction must be made based upon the software execution environment. EC measurements on *mobile* devices are commonly performed to prevent the software from having a deteriorating effect on the battery life of the device., e.g. by software tools performing measurements on the device itself (Joulemeter [24], eprof [5]), or by emulation tools that allow developers to estimate the EC of their application on their development environments [6]. Since battery drain can be monitored relatively easily and mobile devices have similar hardware architectures, some approaches were able to relate EC to source code lines [25] with reasonable accuracy (within 10% of the ground truth), although only for Android applications. Additionally, as performance profilers are quite mature in mobile computing, EC profilers can build upon such tools [26].

In the area of *large-scale* software products, the execution environment is more complex and approaches for energy profiling are more elaborate. Hardware-based approaches (e.g. [27]) rely on physical power meters to be connected to hardware devices. Such approaches do not provide fine-grained measurements at software level, i.e. they are not able to trace the energy consumption of single software elements such as processes or architectural components.

Software-based approaches can be roughly categorized in two sets: source code instrumentation [10] and energy profilers [28]. Source code instrumentation consists in injecting profiling code into the applications code (or byte code), to capture all the necessary events related to energy consumption. For example, JalenUnit [29] is a bytecode instrumentation method that can be used to detect energy bugs and understand energy distribution. JalenUnit infers the energy consumption model of software libraries from execution traces. However, source code instrumentation always results in a noticeable overhead in performance.

Energy profilers rely on fine-grained power models [30] to deliver more accurate measurements at software level. Typically, profilers use performance measurements to explain and characterize software and its EC characteristics [31, 32]. The power model is typically generated via linear regression from performance measurements or resource usage data. This technique could be potentially applied on multiple software products using public repositories and benchmarks, an approach known as *green mining* [33]. Unfortunately, due to lack of publicly available performance data, green mining is still an immature area. Despite the differences, these approaches all focus on identifying energy hotspots [34] i.e. elements or properties, at any level of abstraction of the system architecture, that have a measurable and significant impact on energy consumption.

We see two potential issues with applying source code instrumentation on large scale, e.g. 30000 lines of code, software products: the performance overhead and the required investment (in time and money) to instrument the code [35]. Hence, we do not see them as viable in an industrial setting. On the other hand, energy profilers do not require a high effort to be adopted, but are shown to be inaccurate in their measurements [28]. Hence, for the purpose of our study (see Section 3), we use software profilers to obtain fine-grained, software-level estimations and validate them with hardware measurements obtained via power meters.

2.3. *Software Architectural Aspects of Energy Consumption*

The EC can be significantly influenced by the way software is designed and architected. For example, recent study shows data locality plays an important role in the EC of multi-threaded programs [36]. An information viewpoint [37] could be used to structurally consider this aspect. Characterizing software using performance measurements on the other hand is more related to the deployment and functional viewpoint. Combining multiple viewpoints of a software product, i.e.

creating a perspective [37], enables stakeholder to structurally address concerns on different aspects of the system design.

Software Architecture (SA) also allows a stakeholder to explore design trade-offs for the software [3]. Increased performance, a quality attribute for the software, does not always have a direct relation with EC [38]. A different design trade-off is to exchange modules or services for more energy efficient sustainable variants, e.g. cloud federation [39]. SA helps to identify adjustments on different levels in complex environments [40].

2.4. *Energy Consumption Comparison Between Releases*

Comparing aspects across releases is often discussed in terms of software evolution [41]. However, only few papers were found that investigate the EC of software and include a comparison between different releases. In [42] a comparison is made between three releases of rTorrent by 'mining'EC and performance data. A direct relation is described between the granularity of the measurements and the ability to determine the cause of changes in EC. Another approach is to characterize software using Petri nets [43]. Assuming that a complex software product can be fitted into a Petri net, analysis could show the path of lowest EC to perform a specific task. If the changes in a new release can be included in the Petri net, the difference(s) between releases can be quantified.

2.5. *Awareness*

A different approach is to increase developer awareness in software energy efficiency. The 'Eco' programming model [44], for example, introduces energy and temperature awareness in relation to the software and challenges developers to find energy friendly solutions. Awareness of the software community about the impact of software on EC is increasing [45]. However, Pinto et al. [12] point out that this is still far too little to make a difference. In spite of recent progress, the state-of-the-Art in software energy efficiency did not reach sufficient quality yet to deliver reliable, detailed measurements. Comparing the EC between releases can be used to create awareness at the right place for a SPO, and hence exert control over the EC of their software.

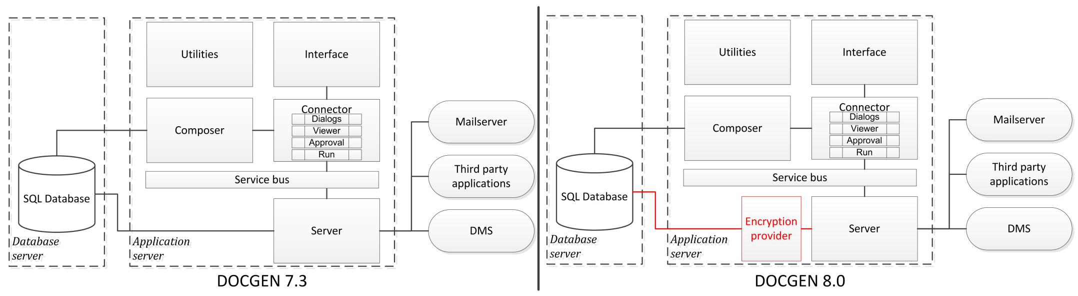


Figure 1. The functional architectures for Document Generator (DG) releases 7.3 (left) and 8.0 (right) portrayed on a commercial deployment. The changes are in red.

Table I. Specifications of the hardware and software used for the experiment.

	Application server	Database server
Hardware	HP Proliant G5, 2 x Intel Xeon E5335 (8 cores @ 2GHz), 8 GB DDR2 memory, 300 GB hard disk @ 15.000 RPM	HP Proliant G5, 1 x Intel Xeon E5335 (4 cores @ 2GHz), 8 GB DDR2 memory, 300 GB hard disk @ 15.000 RPM
Operating system	Windows Server 2008 R2 Standard (64-bit), Service Pack 1	Windows Server 2008 R2 Standard (64-bit), Service Pack 1
Software	DOCGEN 7.3 and 8.0	Oracle 11.0.2.0.4.0

3. STUDY DESIGN

To answer the research questions presented in the Section 1, we performed two empirical studies: an experiment to compare the EC of a commercial software product (Document Generator (DG)) across different releases and an interview with stakeholders from an SPO.

3.1. *Experiment design*

Our experiment follows the guidelines provided in [46–49] and the "green mining" method [42] consisting of seven prescribed activities; (1) choose a product and context, (2) decide on measurement and instrumentation, (3) choose a set of versions, (4) develop a test case, (5) configure the testbed, (6) run the test for per each version and configuration, and (7) compile and analyze the results. In this Section we describe our experimental design, in terms of Product Under Study, setup, metrics and protocol used for the experimentation. We report on compiling and analyzing the results in Section 4 and Section 5 respectively.

3.1.1. Product Under Study: Document Generator (DG) is a commercial software product that is used to generate a variety of documents ranging from simple mailings to complex documents concerning financial decisions. The product is used by over 300 organizations in the Netherlands, counting more than 900 end-users, and annually generates more than 30 million documents. This experiment focuses on two releases of DG, 7.3 and 8.0, allowing us to compare the effects of a major release change [50].

In Figure 1 the SA is shown for the DG releases included in the experiment. Starting with the *Connector* element, we have a central hub in the SA responsible for receiving user input through the *Interface*, collecting data from the *Composer* and handling communication with the *Service bus*. Together with the *Composer* element, responsible for merging document templates and definitions with database data, the *Connector* element handles all activities before documents are generated. *Utilities* and *Interface* respectively provide configuration options and an interface for DG. The final element on the application server is the *Server* element responsible for the actual generation of the documents and delivering the documents to where they are required. The database server hosts an Oracle SQL Database. Specifications of the hardware used in our experiment, i.e the application and database server, are provided in Table I.

3.1.2. Differences between Releases: Looking at the SA, the major difference between the two selected releases is the encryption provider introduced on the application server in release 8.0. Data encryption was introduced in release 8.0 in order for DG to comply with the upcoming General Data Protection Regulation (GDPR) set up for the European Union. In the case of DG 'Microsoft Enhanced Cryptographic Provider' is used: a module that software developers can dynamically link when cryptographic support is required. Encryption is applied in relation to the 'Server' element to remain independent from the database that is used, i.e. encrypted data is sent to the database

Another difference, which is not visible in the SA, can be found in the data model for the database. As release 8.0 is compliant with a new document management system, the datastructure is more complicated compared to release 7.3. We cross-checked our findings with the DG architect, to ensure completeness of our list of relevant changes between releases.

3.1.3. Test Case: For the experiment we selected the core functionality of DG, the generation of documents, as test case. DG was instructed to erase existing documents of a certain type and consecutively regenerate these documents. The selected document type contains both textual information and financial calculations and a total number of 5014 documents was generated per each execution of the test case. During each execution, the 8 processes 'Interface', 'Run', 'Connector', 'Server', 'Oracle', 'TNSLSNR', 'omtsreco' and 'oravssw' processes were monitored on their respective servers. As the 'Microsoft Enhanced Cryptographic Provider' is not an executable but a dynamic library, it could not be monitored in isolation.

3.1.4. Metrics: Comparing literature (cf. [31, 42, 51]) we find similarities in the measurement method that is applied, but a clear difference in the reported metrics. Although all report EC, the metrics target different stakeholders while still providing the details required to be in control of the software EC. During the design of an experiment, a choice should be made on what metrics are to be reported, as they should facilitate discussion between stakeholders, e.g. product managers and (potential) customers [20], especially in the case of a pioneering topic like the EC of software [9]. In Figure 2 we show how the research questions driving our empirical experiment (RQ1, further divided into SQ1 and SQ2) are answered in terms of quantitative metrics. In the following, we further motivate our metric selection and rationale.

As regards the energy consumption of software, we measured the *Software Energy Consumption (SEC) and Unit Energy Consumption (UEC)* metrics [51]. The Software Energy Consumption (SEC) is the total energy consumed by the software, whereas the UEC is the measure for the energy consumed by a specific unit of the software. In our experiment the units, i.e. *software elements* in our RQ, are the individual processes that comprise the product. This is not to be intended as a formal

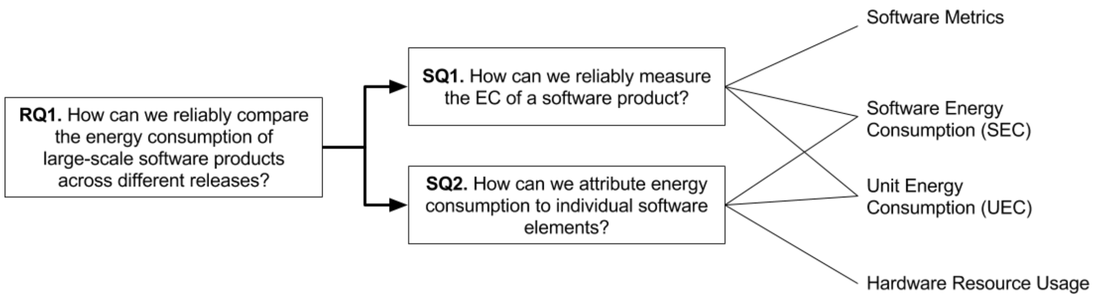

RQ1. How can we reliably compare the energy consumption of large-scale software products across different releases?
SQ1. How can we reliably measure the EC of a software product?
SQ2. How can we attribute energy consumption to individual software elements?
Software Metrics
Software Energy Consumption (SEC)
Unit Energy Consumption (UEC)
Hardware Resource Usage

Figure 2. Overview of how the RQ and SQs of the experiment are linked to the reported metrics.

definition of what a software element is, but it is rather a choice determined by a practical aspect: our profiling method and tools are only able to attribute energy consumption at process level. Any finer granularity, although desirable, is not possible with current techniques.

In addition to the EC, we recorded *hardware resource usage*, as it can be used to accurately relate EC to individual software elements [31,32,51]. Profiling the performance requires the user to have a basic understanding of the hardware components that have to be monitored (e.g. hardware-specific details) and the context in which they are installed.

Following the definition of the 'Unit Energy Consumption' [51], in our experiment we monitored the following hardware resources:

- Hard disk: disk bytes/sec, disk read bytes/sec, disk write bytes/sec
- Processor: % processor usage
- Memory: private bytes, working set, private working set
- Network: bytes total/sec, bytes sent/sec, bytes received/sec
- IO: IO data (bytes/sec), IO read (bytes/sec), IO write (bytes/sec)

We also collected *software metrics* for both DG releases using CppDepend 6.0.0[†]. The tool provides several software size and complexity measures, such as 'cyclomatic complexity' and 'nesting depth' which allows us to more extensively identify differences between DG releases. These metrics are related to SQ1 as ideally they could provide an early indication of software EC at design time: by analyzing whether there are any correlations between specific software metrics and the EC of the different releases, we could provide such an indication. Reporting software metrics is also useful to identify potential trade-offs between energy efficiency and other aspects of software quality (e.g. maintainability).

3.2. *Interview Design*

To follow up our experiment we investigated how the results were picked up by the DG team through interviews. More specifically, we looked into their views on the information provided by the EC measurements and the effects of having this information on their tasks. Additionally we explored the opinions and views on how EC measurements in relation to software can be promoted within their organization. To provide the most complete answer on RQ2, we aim to include different roles within the DG team in the study and provide insight on the operational aspects in relation to DG, e.g. its development, and strategic aspects like the product roadmap. For the interviews we followed the guidelines presented in [46,52].

As the interviews took place after the case study, we could build on a common understanding of SEC between the interviewer and interviewees. However, given the relatively little experience of the team with SEC, we still decided to conduct semi-structured interviews. Structured interviews would have limited the interviewees to only think of those aspects that have a direct relation with the specific questions, instead of actively considering SEC in relation to their tasks and responsibilities. On the other hand, an unstructured interview could result in interviewees focusing on only those aspects they are more experienced in and might not be directly related to SEC.

[†]http://www.cppdepend.com/

Table II. The questions comprising the interview including the goal for each question.

Question	Goal
What do you think of measuring the energy consumption of software?	Elicit position of interviewee.
Does it seem useful to measure this aspect of the software?	Elicit position of interviewee.
What do you think of the changes that are measured across releases?	Determine opinion on measurements and differences.
Are you able to relate the measurement to your tasks as $< role >$?	Gain insight in $< role >$-perspective.
How would you apply the information that is provided?	Gain insight in value of measurements for $< role >$.
Looking at the data, did you miss aspects that would have been useful to include in the measurements?	Identify gaps in measurement information.
What do you think of software energy consumption in relation to quality attributes of the software?	Identify relations with SEC and determine opportunities for trade-off analysis.
What do you think of software energy consumption in relation to software metrics (e.g. lines of code, number of types, complexity measures)?	Identify relations with SEC and determine opportunities for further analysis.
In your opinion, what is required to put SEC on the agenda within the organization?	Identify strategic opportunities from SPO perspective.
What is required to have you consider this aspect as part of the job?	Identify opportunities from $< role >$-perspective.

The questions comprising the semi-structured interview were formulated during multiple brainstorming sessions between the authors, and tailored to help answer RQ2 in light of the experiment results. For each question (Table II) a goal was formulated in relation to determining the added value for an SPO. Note that the order of presentation corresponds with the order in which the questions will be posed to the interviewees. Given the novelty of the topic (i.e SEC) and the focus on the added value from the perspective of a product team and SPO, we were not able to validate our questions through a pilot interview with a person independent from the research.

For the interview itself, each interview was conducted following a protocol where the interviewees are first presented with a summary of the data, i.e. our previous work [21], followed by the interview questions. During the interviews notes were made on the important aspects mentioned by interviewees and, with consent of the interviewee, the interviews were recorded. Processing the interviews was done directly after the interview, to prevent inaccuracies due to poor recall [52], and encompassed the identification of themes across interviews: aspects that are mentioned by multiple interviewees or are stressed from the perspective of a specific role. The notes made during interviews served as a guide to identify themes and were completed, e.g. by adding missed aspects, using the recordings. As such the notes served as a qualitative summary of the individual interviews and the main source to extract the final set of themes.

4. STUDY EXECUTION

4.1. Experiment Execution

4.1.1. Setup: in line with the deployment portrayed in Figure 1, two servers have been used: one for the application and one for the database. The setup is depicted in Figure 3. The specifications of the application and database servers are provided in Table I. To ensure consistency with regard to external factors (e.g. room temperature), the servers were installed in the same data center.

Both releases of DG were installed on the application server and Oracle was installed on the database server. The setup of the experiment, including the servers, is comparable with a commercial setting of the product. In the experiment, both releases use the same data set of an actual customer. To increase the consistency across measurements, a script is used to generate 5014 documents using DG.

4.1.2. Baseline Measurements: to obtain a clean measurement of the EC related to solely DG, we determined the idle EC for the hardware that is used. This represents our *baseline*, and as such is subtracted from the total EC during a measurement, under the assumption that the increase in EC solely depends on running the software under test. As the idle EC heavily depends on the used hardware, this number should be determined separately for each hardware device in the experiment by performing measurements while the hardware is running without any active software.

However, using this method, the EC is not only related to DG, but also includes the effects of measurement software and Operating System (OS)-specific activities (e.g. background daemons), which we are not (yet) able to consider separately and thus considered to be part of the idle measurement. As we cannot completely control these aspects, we stopped any service and process known not to be required by the software product under test to minimize their effects (e.g. the automatic Windows update service). Additionally, we used a separate logging server to minimize the overhead caused by the data collection process.

Another aspect that we had to consider is the *cooldown time* a server needs after rebooting: after a reboot, several services related to the OS are active without direct instructions from a user. As these services require computational resources, they will most likely pollute measurements if the experiment starts while these services are running. Hence, measurements have to be taken in a "steady state" i.e. when the extra services become inactive.

As with the idle baseline, the cooldown time was determined for every hardware device included in the experiment. EC and performance measurements give an indication of when the steady state is reached. The cooldown time for our servers was determined to be 15 minutes.

4.1.3. Hardware– and Software–Based Measurements: a measurement method concerning software EC should include both hardware and software approaches to obtain the right level of detail in the measurements. In terms of hardware measurements, we relied on power metering devices. As these meters are installed between a device and its power source, a meter was needed for each power

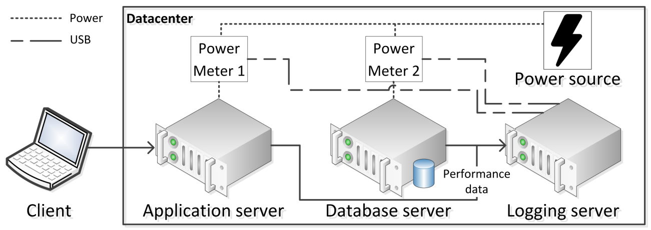


Figure 3. Experiment environment.

supply unit of the devices under test. Although these meters are capable of achieving high levels of accuracy, their specifications was taken into account in the data analysis as even measurement errors of a fraction of a percentage point might prove significant at software level. Each of the servers in our setup is instrumented with a single WattsUp? Pro (WUP) device[‡] (see Figure 3). WUP devices record the total energy consumption of the hardware once per second.

In order to profile individual software processes, we used software energy profilers (see Section 2). These tools estimate the EC of both the whole system and individual processes at run time, using power models based on computational and hardware resource usage. Unfortunately, most energy profilers record measurements with a 1-second interval, although a higher frequency is desirable [33]. While the usability and accuracy of energy profilers still have margins for improvement [28, 30], the reported measurements could still be used to detect differences in EC. In other words, although measurements in absolute terms may not be fully accurate, the relative differences between EC of the two releases we analyzed still provided useful insights.

In our experiment, we used the tool Joulemeter (JM) of Microsoft, that allows to estimate the power consumption of a system down to the process level. JM estimates EC on a model that first needs to be calibrated for the hardware it runs on. Previous experience with JM [28] shows that although JM provides a general idea of EC, it differs significantly from the actual EC. Since only one process can be measured per instance of JM, a separate instance for each of the concurrent DG processes is instantiated (see Section 3.1.1). Although relatively coarse, measurements on process level (i.e. the concurrency views on the system [37]) can be translated to more fine-grained aspects using an architectural perspective [51].

The hardware resource usage of the application and database servers were measured using the standard performance monitor (*perfmon*) provided with Microsoft Windows. Performance data is remotely collected using the logging server, thereby minimizing the overhead of measurement on the actual hardware.

Summarizing the data collected for each individual measurement we have:

- WUP measurements of the energy consumption at the level of the hardware;
- JM estimates for each of the processes together with an estimate of the total energy consumption;
- one *perfmon* file containing resource usage data for both the application and the database server;
- the start and end timestamp for each measurement;

After each measurement, both servers were been reverted to the initial state, restarted and were left untouched for the determined cooldown times.

4.1.4. Data Synchronization: an important requirement for data analysis is to have synchronized measurements. As measurements are obtained from different sources, their timestamps have to be synchronized to avoid irregularities in the data. For example, if a specific activity is performed and the timestamps across sources are not in sync, there is a risk of missing the data related to this

[‡]http://www.wattsupmeters.com/secure/products.php?pn=0, last visited on Monday 19th December, 2016

activity. To address this issue, in our experiment we continuously synchronized the clocks for all measurement sources using the Network Time Protocol (NTP).

4.1.5. Measurement Protocol: while the "green mining" method [42] provides a solid basis for designing an experiment, no details are provided on how to actually perform reliable measurements within an experiment. To this end, we propose the following protocol applying the information presented in this section, which is an extension to the activities presented by [42]:

i Restart environment;

ii Check time synchronization;

iii Close unnecessary applications;

iv Start performance measurements;

v Remain idle for a sufficient amount of time;

vi Start EC measurements;

vii Run measurement and wait for run to finish;

viii Collect and check data;

ix Revert environment to initial state;

The protocol ensures consistency across measurements and improves the reliability of each measurement [46].

4.2. Interview Execution

The interviews were conducted with the architect, the product manager [20], a developer and a tester of the DG team, the latter also being the 'Scrum master', and took place between four to seven months after the results of the SEC measurements (i.e. [21]) became available. Given the nationality of the team the interviews were conducted in Dutch, which meant we had to translate the interview questions to Dutch and the interview results from Dutch to English. Also, as not the entire team was situated in the same office building, we had to conduct one interview remotely. On average an interview lasted approximately one and a half hour.

For the analysis, the notes made during the interviews appeared sufficient to identify all relevant themes and in practice the recordings were only played back once to confirm the themes. Unfortunately, even though all interviewees gave their consent for recording the interview, only three out of the four interviews were successfully recorded. In the case where we lacked the recording, we cross-checked the processed results with the interviewee for inaccuracies: none were identified.

The results of the interviews are reported in the results sections (Section 5), sorted by the themes that we identified. Combined with the other information at hand, these results are used to provide an answer to RQ2 (Section 6).

5. RESULTS

5.1. Experiment Results

In this section we extensively report our experimental results. The complete dataset is openly available[§].

Both the WUP as well as the JM measurements report the EC as an average of the instantaneous power over the sampling interval. To calculate the total EC, we either multiply the average power with the time the system was running, or sum up the recorded energy measurements. We report our findings in Watt (W) or Watthour (Wh) where applicable.

5.2. Baseline Measurements

The results of the idle and JM overhead measurements are presented in Table III along with the measurement time to determine the averages. The measurements were collected over 5 runs per scenario, spanning more than 50 hours of measurement time. Starting with the idle EC we found an average power consumption of 274.54 W and 252.59 W for respectively the application and database server. Considering that the servers are almost identical, we can only allocate this difference of 21.95 Watt (W) to the extra processor available in the application server.

An interesting finding is the fact that there is minimal to no overhead on the account of JM. Further investigation showed a base memory usage by JM, which increased when JM was actually logging measurement data. While logging, performance measurements show increases in the memory usage of the JM instances which are periodically 'reset' to a base memory usage. Our guess is that the pattern in memory usage corresponds to incrementally adding measurements to the CSV file. Despite this variability in memory usage we could not detect any change in EC.

As part of the baseline measurements, we also determined the power consumption interval of the servers. Based on 36 hours of running the servers at full capacity, we determined a maximum power consumption of 367.3 W for the application server and 291.2 W for the database server providing a range of 92.02 W and 38.41 W respectively. Again we can only explain the difference due to the impact of the additional processor, showing that, all other things equal, the power consumption range increases with a factor of 2.4. Using the range we are able to normalize the measured power consumption and better investigate the impact of the software on the hardware EC.

5.3. DG measurements

We performed 20 executions for each DG release (7.3 and 8.0). During each execution, we collected the data described in Section 4.1.5. Tables IV and V summarize the results in terms of mean (μ) and

[§]https://www.dropbox.com/sh/kk9kastzo2cypur/AABA3ZuWbSi-F4k8o8Af6KJJa?dl=0

Table III. Comparison of server power consumption in different "idle" scenarios including measured time.

Server	**Idle**		**Idle (JM running)**		**Idle (JM measuring)**	
	Total time	Avg. Power (W)	Total time	Avg. Power (W)	Total time	Avg. Power (W)
Application	57:11:30	274.54	54:06:21	275.28	54:06:21	276.18
Database	57:11:30	252.59	54:06:21	252.79	54:06:21	253.39

standard deviation (σ) for the application and database server, respectively. Notice that the process-level results for the database server only include the JM results for the 'Oracle' process. The other processes were excluded from the table as their EC was reported as 0 by JM, despite them being active. The 'Interface' process on the application server, that runs the GUI of DG, was not active during the experiment as the DG execution was scripted.

Comparing the measurements between releases, two differences are clearly visible. First, the run length increases by 12 seconds on average in the 8.0 release. A second difference is the overall increase in energy consumption of DG 8.0 as compared to 7.3 of 4.14 Wh according to the WUP measurements: 2.97 Wh for the application server and 1.17 Wh for the database server. Such increase, to a lower extent, is also reflected in the JM data. This difference cannot be explained only by the increase in execution time: if we subtract the average amount of energy consumed by DG in 12 seconds from the 8.0 average, we still find a difference of 2.05 Wh and 0.32 Wh.

The SEC for both DG releases is calculated by subtracting the 'idle with JM' EC from the total EC as reported by the WUP for the length of the run. For release 7.3 we find a SEC of **2.57 Wh for the application server and 8.03 Wh for the database server.** Measurements for release 8.0 provide a SEC of **4.61 Wh and 8.34 Wh for the application and database server.**

Placing the SEC in the perspective of the ranges calculated for each server, we find that only a relative low portion of the available resources is actually used by DG. Even when considering the total power consumed by the servers, the average power consumption figures for release 7.3 are 276 W and 255.66 W for the application and database server. For release 8.0, the averages are 277 W and 256.00 W respectively. Considering this in relation with the power interval we reported in our baseline measurements, at most 1.87% and 8.36% of the application and database server capacity is used, respectively. These figures in our opinion underline why virtualization, or resource sharing in general, could still be an important aspect to reduce the EC related to software.

Table IV. Summary of the experimental results on the Application server for both DG releases.

		Application server				
		7.3		**8.0**		**Diff**
		μ	σ	μ	σ	Δ
Run length	(hh:mm:ss)	2:48:16	4 s	2:48:28	7 s	+12 s
Processed Documents		5014		5014		
WUP (Wh)		774.59	1.18	777.56	0.84	+2.97
Run	Total (Wh)	765.20	0.32	766.21	0.63	+1.01
	Process (Wh)	0.0002	0.00009	0.0003	0.0001	+0.0001
Server	Total (Wh)	765.18	0.33	766.21	0.63	+1.03
	Process (Wh)	0.744	0.00002	0.758	0.007	+0.014
Connector	Total (Wh)	765.19	0.34	766.22	0.63	+.03
	Process (Wh)	0.144	0.004	0.22	0.004	+0.76

Table V. Summary of the experimental results on the Database server for both DG releases.

		Database server				
		7.3		**8.0**		**Diff**
		μ	σ	μ	σ	Δ
Run length	(hh:mm:ss)	2:48:16	4 s	2:48:28	7 s	+12 s
Processed Documents		5014		5014		
WUP (Wh)		716.99	0.45	718.16	0.61	+1.17
Oracle	Total (Wh)	706.37	0.29	707.27	0.51	+0.9
	Process (Wh)	5.63	0.02	5.62	0.02	-0.01

5.4. *Joulemeter Estimations*

The SEC can also be calculated using the estimations provided by JM (Table VII). Using this data we find a SEC of **1.45 Wh and 5.69 Wh** for the application and database server with release 7.3, and **1.57 Wh and 5.72 Wh** with release 8.0. There are evident differences between these SEC figures and the ones obtained using WUP. In our data we observe that the WUP on average provides a higher SEC of 1.12 Wh and 2.34 Wh for the application and database servers. This difference is probably due to an underestimation given by the JM power model.

Apart from the total EC, the JM data allows us to calculate the SEC according to measurements on process level, i.e. the 'Run', 'Server' and 'Connector' processes on the application server and the 'Oracle' process on the database server. The measurements for release 7.3 provide a SEC of **0.89 Wh and 5.69 Wh** for the application and database server. With release 8.0 we find a SEC of **0.97 Wh and 5.62 Wh** respectively. The large differences in the SEC figures could be an indication that, despite our efforts, several processes are still active in the background alongside the DG processes.

5.5. *Software Metrics*

The results of the analysis on software metrics are shown in Table VI. Our results show a size increase of DG 8.0 in terms of lines of code (LOC) (6.3%) and number of types (19.64%), projects (33.19%), namespaces (31.46%), methods (13.88%), fields (33.23%) and source files (27.80%). Since our case study was performed after the releases were commercially available, we were not able to determine all churn measures presented in [42]. Specifically, the added and removed lines and the file churn require a fine-grained tracking during development.

If we consider EC in relation to the metrics, we find that the EC per line of code is 0.047 Wh for release 7.3 and 0.044 Wh for release 8.0. suggesting an increased efficiency per line. This increased efficiency also holds for the other size-related metrics. However, any usage of LOCs for quantitative analysis of software products is under the strong assumption that every LOC is equivalent in terms of efficiency. Inefficiently written code (e.g. resulting in more LOC) could result in a lower and incorrect EC per line of code.

5.6. *Interview results*

The interview results on the stakeholders' views on EC measurements are presented below, arranged by the common topics that emerged across the interviews. For each topic we combined the results gained from each interviewee.

Sustainability: in general, sustainability, including EC, is perceived as an important topic in the Dutch software industry and this importance has increased with the recent climate deals[¶]. Dutch municipalities, which comprise a large part of the DG customer base, are compelled to consider sustainability in their processes and are becoming aware of the role IT can play. However, given the novelty of this area there are no hard requirements from the customers (yet).

[¶]http://ec.europa.eu/clima/policies/international/negotiations/paris/index_en.htm

Size metrics	DG 7.3	DG 8.0	
Lines of code	31770	33770	
Types	1389	1663	
Projects	74	103	
Namespaces	89	117	
Methods	9658	10999	
Fields	10757	14726	
Source files	1698	2170	
Complexity metrics			Units
Max cyclomatic complexity for methods	152	165	Paths
Max cyclomatic complexity for types	2723	3145	Paths
Average cyclomatic complexity for methods	2.48	2.53	Paths
Average cyclomatic complexity for types	37.35	40.78	Paths
Max nesting depth for methods	30	32	Scopes
Average nesting depth for methods	0.89	0.82	Scopes
Max # methods for types	535	614	Methods
Average # methods for types	7.63	7.2	Scopes

Table VI. Software metrics obtained using CppDepend 6.0 for the DG releases.

The tester, from his product owner perspective, and product manager were enthusiastic about measuring the EC of DG as a way to gain competitive advantage. Striving for a reduced EC and thereby environmental impact is seen as desirable for the product and the company as a whole.

However, the team was convinced that the impact of renewing hardware on the EC is higher than changing software. According to the developer ‘hardware changes are easily made and are still the low-hanging fruits when it comes to EC.’ Although it is important to monitor the resource utilization, e.g. CPU utilization, to control and improve software, renewing hardware is expected to grant higher economic savings.

Experimentation: overall, the interviewees were satisfied with the results of the experiment and found no reasons for concern. The functionality, i.e. encryption, was added to comply with regulation and the differences were not significant. On a strategic level, the product manager was pleased by the fact that this aspect of the software could be measured and made tangible. Until now the aspect of EC was relatively abstract, especially in relation to software.

The results did raise the interest of the architect and developer: specifically, they were surprised by the elements and processes that were shown to be affected. However, further investigation into the matter would (among others) require isolating the encryption provider and testing this aspect separately (e.g. encrypt and decrypt a number of rows) to determine its impact. Given that, analysis of the code would still be required to find the actual cause(s) of the unforeseen change.

Consequently, a business case was made to further investigate the impact of the encryption provider on the software including the costs for investigating and potentially redesigning. Weighing

Table VII. The SEC according to Joulemeter calculated using the total power consumption and the power consumption per process.

	Application server		Database server	
Release	7.3	8.0	7.3	8.0
Total EC (Wh)	1.45	1.57	5.69	5.72
Process level EC (Wh)	0.89	0.97	5.69	5.62

the costs against the projected benefits (i.e. lower energy consumption and potential increased performance) it did not appear beneficial to invest in this matter at this point in time. Still, this aspect will be monitored as it could become more important in the future.

Software energy efficiency might become more important when the scale of the transactions increases. In the experiment DG was instructed to generate 5014 documents, which is only a small fraction of the 30 million annually generated documents. If the software is made more efficient, this is bound to have a significant effect on the resources, and as a consequence this will also affect the forthcoming EC.

In this sense, performing experimentation on a larger scale would be useful. For example, the tester and the developer suggested increasing the duration of the experiments. Simulating DG usage for an entire working day could help the tester detect EC patterns over time and possibly provide input for a smarter scheduling schema. For the developer, a longer experiment duration could help detecting errors and bugs that only show after a longer period in time. The effect of small loops or try-catch recursions, for example, can keep piling up over time until their presence is noticed. On the long run they could significantly affect the resource usage by the software. Even though errors like these are often not critical and can be resolved by rebooting the system, as a developer they are valuable to ensure system stability over time. Also testing in different environments, e.g. SaaS, with multiple servers, layers of virtualization and different hardware resources in general was considered an interesting increase in scale.

The team also felt strong towards the idea of presenting results in relation to a benchmark instead of ‘raw’ measurement data. Comparison between releases directly identifies differences and can be used to pinpoint those aspects that have evolved dis-proportionally. Presenting ‘raw’ measurement data, e.g. CPU utilization, would require more effort to understand the measurements, correctly interpret the results and translate results to concrete actions.

With respect to EC the interviewees did not see a clear relation with software metrics. Software metrics are mostly used as an indication for the maintainability attribute of the software and as a means to monitor the evolution of DG in general. Higher technical debt, for example, could be an indication of poor maintainability of the software. Especially the comparison with other products is important, which is an internal indicator for the quality of the development activities.

Functional vs. non-functional aspects: in general the software development practice of the team can be characterized as functionality-driven [53]. The architect stated: ‘writing code concerns functional aspects, not performance or energy. Developers do not consider non-functional aspects while writing code’. It was made clear that the only way a developer would work on, for example, performance is to make the requirement very concrete and functional; e.g. a window should open in one second.

With respect to EC and the quality attributes of the software product, the product manager admitted that this aspect was, due to the functionality driven development, not high on the priority list. As there are currently no customer requirements on this aspect, the product manager could not justify a trade-off in favor of sustainability against, e.g., performance. It would be valuable to consider EC on this level, but from a strategic perspective that would require more awareness on the customer side to justify why certain decisions are made. For DG the risk was considered too high to make these trade-offs themselves.

On the other side the developer pointed out the necessity of certain design decisions that have been made. Although not related to DG, the developer mentioned the usage of the HTTPS protocol which according to him requires twice as much resources compared to simple HTTP. On large scale systems the decision to apply HTTPS is bound to have a significant impact on the EC sometimes without having a clear benefit in certain cases. Any design decision should include trade-off considerations between the relevant quality attributes.

The team agreed that if EC is labeled as a key factor by the organization, then decisions on adding or changing functionality should be made in the design phase and EC should be a prominent factor in the decision-making process. In a sense EC should be considered the same as the other quality attributes for the software and trade-offs should be made depending on the organizational policies. The tester admitted that testing on non-functional aspects, which EC is considered to be, is in general not done extensively. Given the current stage of the product life cycle where the product is transformed to a SaaS solution, there is no high priority to do so.

Relating measurements to roles: with respect to the measurements in relation to his tasks, the developer argued that the measurements are foremost an indication of whether he has done his job right. If large, unexpected discrepancies are observed, it could be an indication that a mistake has been made in the code itself. As such the measurements are used as a check. The same holds for software metrics, which essentially are used as a means to check whether any changes are in line with the adjustments that have been made.

As a software producing organization the product manager saw added value in having a unique selling point and also saw potential to strengthen the organizations' image with respect to sustainability. Compared to competing products, simply providing insight in the EC of the software could help in winning over customers. Performing EC measurements not only potentially helps the customer become more sustainable, but also visibly lets the organization take responsibility for their contribution.

The tester noted that a focus on non-functional aspects requires different tests performed in different environments. The added value for him as tester specifically was marginal in the form of the knowledge gained by performing these tests. Finally, the architect noted the strategic advantage of performing these measurements and added the potential increase in software quality. An aspect like EC requires trade-offs to be made and stimulates to rethink design decisions.

Putting EC on the agenda: To put EC of software products on the agenda would require a change in mindset within the organization. Progress with the software itself, i.e. functional improvements, is most important, but there are other developments that require a broader perspective on the software. For example, the scale of software products is changing, e.g. on-premise to cloud, which also affects the resources used by the software. In the end, to structurally consider EC, all interviewees agreed that the costs and financial gains should be made visible.

Also making EC tangible, like in this study, is essential. Presentations on being sustainable have been given in the past and often left the team with more questions than answers. From the perspective of the product manager this topic can not be forced upon products due to other factors weighing in, but making EC concrete helps to include this aspect in the decisions that need to be made. The tester however disagreed and noted that a top-down approach would help to have an organization-wide focus on this aspect of software and will hopefully stimulate attention from the bottom up.

The future of DG: Currently release 9.0 for DG is being developed where the system is redesigned to a software as a service (SaaS) product. The bulk of the work for the architect is to redesign the system in terms of (functional) aspects that were originally not designed to be, for example, multi-tenant. Again the architect stressed that a new development viewpoint would only guide development activities (e.g. by providing insight in changed dependencies) while still a lot of work has to be done on code level.

Apart from the development activities, there is awareness on the 'different dynamics' with a SaaS product; shared resources, multi-tenancy, a changing pricing model, continuous delivery and the total cost of ownership in general. Deploying DG as a SaaS product will most likely emphasize non-functional requirements for the system, thus requiring a better comprehension on these aspects. In line with the insights provided earlier, the team expected EC to be more relevant in SaaS deployments where a lower energy consumption can directly affect the total cost of ownership and thereby the strategy for a product.

6. DISCUSSION

In this section we discuss the results presented in the previous Section, answer the research sub-questions for RQ1 and provide an answer to RQ2.

6.1. SQ1: Measuring the EC

The protocol that was applied in the experiment ensures that the relevant variables (that can be influenced) are under the control of the researcher. It also provides guidelines for the data collection and processing. By following the measurement protocol we obtained consistent and comparable data across measurements, confirmed by the small standard deviations found with each item, and were able to compare different releases of DG from an EC perspective (Figure 4). This allows us to conclude that **the measurement method we adopted can be used to reliably measure the EC of a software product**.

In terms of software metrics, due to our limited dataset we could not perform a statistical correlation analysis. Although the size metrics show an increased efficiency per line, we cannot claim a causation link between such increase and the increase in energy consumption. However, we argue more valuable insights can be gained from the complexity metrics. The cyclomatic complexity for types and methods is expressed in the number of independent paths through program source code where an independent path is a path that has at least one edge that has not been traversed before in any other paths. A high cyclomatic complexity over time increases the probability of errors while maintaining the software (i.e. decreases maintainability). The nesting depth represents the depth of a nested scope in a method body where a lower nesting depth is better for the readability and testablity of the software.

As per the size metrics, we cannot claim a direct causation link between the increase in complexity and the EC. However, given their relation to QAs (see the ISO 25010 standard), the complexity metrics could indicate a potential impact on the design of a system architecture in terms of allowing or precluding other QAs.

This allows trade-offs between different sustainability requirements, thereby enabling decision-making with respect to the EC of a software product. For example, one could consider EC in relation to its maintainability and specifically its technical debt (i.e. results of past decisions that negatively affect its future [54]). Maintainability is a QA that is normally associated with the technical sustainability dimension.

Looking at the reported complexity metrics we find an increase in the average cyclomatic complexity values, whereas the average nesting depth for methods decreases with release 8.0. Most notable is the increase perceived in the average cyclomatic complexity for types from 37.35 to 40.78 paths. A lower cyclomatic complexity in general indicates an improved maintainability and testability of the software, and an improved readability of the code itself. While this might seem a deterioration of the software quality, this finding might indicate a (deliberate) trade-off has been made between the maintainability and security QAs.

6.2. *SQ2: Relating EC to Software Elements*

In order to answer SQ2, we used Joulemeter to estimate the energy consumption of individual software processes composing our application. We also validated the accuracy of Joulemeter, building a separate model from our performance dataset using linear regression. The model outperforms JM at machine-level prediction i.e. trying to predict the total system EC, see Figure 5. More details on this model are provided in the Appendix.

However, if we aggregate the estimation obtained using our model for all the processes running in the application, we obtain very similar percentages to those computed via JM. Given this validation, we must conclude that JM provides a fairly accurate estimation of the energy consumption of specific processes.

Although both Joulemeter and our regression model are unable to attribute the total SEC to specific processes, **our profiling method allows us to observe relevant changes between the different processes composing our software product** which allows us to make informed hypotheses about the impact of each elements on our software product. For example, the 'Oracle'

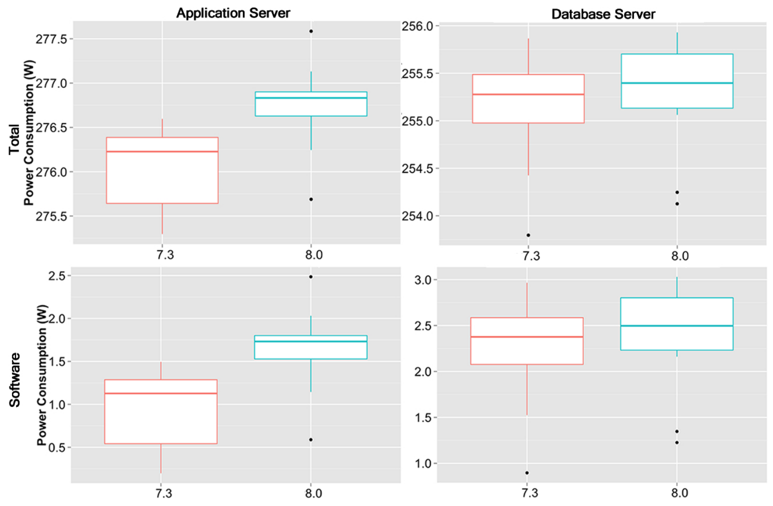


Figure 4. Boxplots summarizing the total and software power consumption measurements for the application and database servers.

process in the database server is by far the most energy–consuming. This indicates that the database is a potential *hotspot* [8] and, as such, a candidate for optimization.

The most apparent difference between the DG releases is the introduction of the encryption provider element on the application server. Unfortunately, as this element is a library, we were not able to perform measurements specifically on it and isolate its energy impact. We are, however, able to analyze the effects that are caused by the addition of this elements and infer possible explanations for EC differences.

According to the architect, the introduction of the encryption provider was accompanied by minor changes in the 'Server' element. Interestingly though, while an increase in EC is found in the 'Server' element, the main EC difference was found in the 'Connector' element going from 0.144 Wh to 0.215 Wh. This difference could not be explained based on the adjustments applied in release 8.0. This *unforeseen change* in EC was reason for the architect to further investigate the matter in the near future.

With regard to the difference in run length an explanation is sought in the encryption that is applied, possibly extending the time required to set up a connection and communicate data. Apart from increased duration of the run, we also found that the net number of seconds reported by JM increases with release 8.0 for the 'Server', 'Connector' and 'Oracle' processes. Considering that JM uses a linear model to estimate EC, a higher execution time should result in a higher EC for these processes. However, this only holds for the processes running on the application server.

Overall we can conclude that the changes applied in release 8.0 increased the SEC by 4.14 Wh for the generation of 5014 documents. Although small, these differences could add up significantly with each installation and generated document: in literature [42], a savings of 0.25 W is shown to potentially equal the power use of an American household for a month.

With these results stakeholders of DG are now able to quantify and justify changes in EC. Considering the cause of this increase, i.e. being compliant with a new document management system and ready for the General Data Protection Regulation, the stakeholders deemed the increase

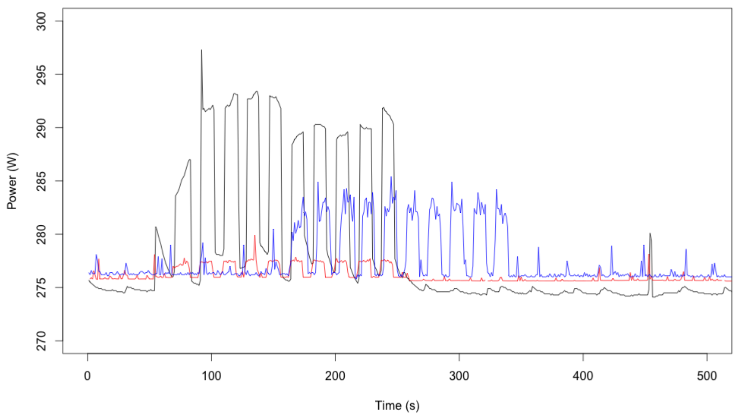


Figure 5. Performance of the penalized regression model (in red) vs. Joulemeter (in blue). Measured values by WUP are in black.

in EC as acceptable. To increase efficiency, however, the software architect will still look into the 'Connector' element.

6.3. Visualizing Software Energy Consumption

Besides measuring on process level, the resource usage data described in Section 3.1.4 was measured at server level. This data allows us to characterize the hardware aspects that affect the EC range (Section 5.2) and *visualize* the impact of release DG 8.0 on the servers. Specifically we use the disk bytes/sec, % processor usage and working set to respectively calculate the hard disk, CPU and memory dimensions. Note that the network dimension is not included since these metrics were also excluded for measurement on server level.

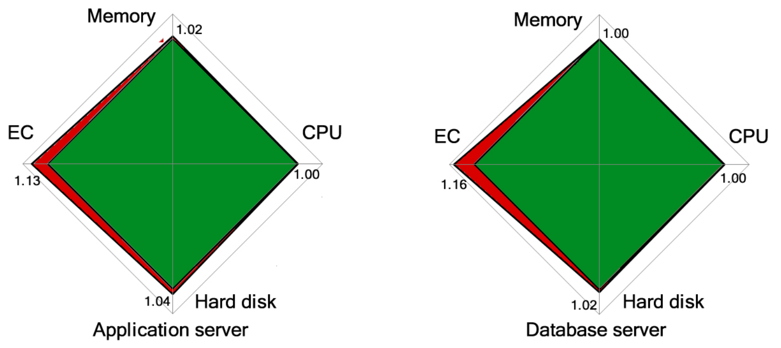


Figure 6. Radar graph showing the impact of DG release 8.0 on EC and the relevant hardware aspects.

In order to create the visualization, we follow the approach described in [55] to create a radar graph. For each dimension, we calculate the normalized delta using the averages across measurements. Since the values are normalized, a delta larger than '1' shows a deterioration compared to release 7.3 and vice versa. Given the focus on EC, we included this dimension in our calculation.

The results for each server are shown in Figure 6. The line surrounding the green area is the benchmark for the normalized delta and represents release 7.3. The line surrounding the red area, representing release 8.0, indicates that the impact of the encryption provider on the available resources is mainly through the memory and hard disk usage, i.e. delta $>$ 1. Note that the radar graph is zoomed in, i.e. the maximum value of the graph is set to 1.2, to better portray the results. This finding is contrast with the expectations of the DG development team: the encryption was regarded as increasing the load on the CPU, however the normalized delta for the CPU usage is 1.00 and shows no difference across releases. Most prominent is the deterioration of the EC, with deltas of 1.13 and 1.16 respectively, which clearly skews into the red area of the graph.

6.4. RQ2: the added value of EC measurements for the SPO

Through interviews we obtained insights into how EC information for a software product affects different roles in the DG product team. With respect to the differences found between the DG releases, there was no reason for concern as the EC increase was considered marginal. However, a clear lack of reference material also prevented the interviewees to put the increase in perspective with other products. As such, a first aspect of added value for each role was to have EC measurements in the first place.

The lack of reference material implies that EC measurements could potentially provide a strategic advantage with respect to competitors. Even though all roles acknowledged the potential, only the product manager and architect roles are actually able to extract value from this advantage. The product manager can promote improvements on the energy consuming behavior of the product and stress the (temporary) uniqueness of these efforts, potentially leading to increased sales and an improved market position. On top of that, the product manager is able to include EC requirements on the roadmap and steer development towards strengthening this aspect of the software. The architect on the other hand, can plan technical adjustments that help to reduce the total cost of ownership for a product.

One added value that holds for all interviewees is the **increased awareness on the topic of sustainability and the relation with software products**. Before the experiment was performed, sustainability was considered as a topic that should be addressed with other aspects in the organization, i.e. renewing hardware. The relation with software would not have come to mind, which would be a missed opportunity according to the interviewees.

A final added value is related to non-functional aspect in general and not specifically to the EC measurements. By positioning EC in relation to quality attributes of the software, the measurements helped to re-introduce non-functional aspects on the agenda. The focus on functionality made that non-functional aspects were often only considered when issues were experienced by the customer. Keeping non-functional aspects in mind allows to have more control over the software and the quality thereof.

6.5. Energy Consumption on the Product Roadmap

Although the provided delta analysis provides practitioners valuable insights in the EC of their software product, the creation of energy efficient software starts with the design of the software [45], i.e. with its architecture. Changes on this level often require thorough preparation and development, and should be planned ahead on the roadmap. Especially when a balance has to be found with planning and realizing customer requirements [22].

As the performed case study shows, the development team tends to mainly focus on the functional aspects, and postpone trade-off analysis of the relevant QAs. With the presented analysis, EC becomes tangible for the developers, and as such, allows them to start reasoning about the consequences of implementation choices. In this way, EC can be introduced as one of the QAs to be taken into account and the measurements themselves serve to **create awareness** on the topic. Ideally increased awareness would result in the inclusion of EC related requirements on the roadmap.

With EC related requirements on the planning EC becomes an aspect of the system design. Different studies have evaluated EC with respect to system design. For example, a recent study shows data locality plays an important role in the EC of multi-threaded programs [36]. Similarly, Trefethen and Thiyagalingam [38] have shown that increased performance does not always have a direct relation with EC. These studies show the need for a separate EC perspective on software architecture. We envision a EC perspective [3] for software architects to analyze and evaluate the EC of a software product. Applying a separate EC perspective enables practitioners to structurally consider aspects that concern the design of a system. On top of that, the knowledge gained from applying the perspective helps in making informed decisions on trade-offs with other design decisions.

For example, relating the energy consumption to the functional views of a software architecture, the architect can analyze the EC per functionality, and decide, based on performance indicators, such as execution time or frequency, whether the functionality requires separate attention. With the presented delta-analysis, the consequences of the proposed changes can be analyzed. By applying the EC perspective different trade-offs are possible. For example, to exchange modules or services for more energy efficient sustainable variants, such as cloud federation [39].

Since the roadmap encompasses the future direction of a product, the roadmap could be considered a 'to be' system design. As such, a product roadmap allows for the **controlled evolution of the product**.

From an economic perspective, (re-)designing software should be done with the lifecycle in mind. For each investment in the product a **business case** should be created to ensure scarce resources are directed to where they add most to the product, and organizational, strategy. The lifecycle stage for DG, for example, is two-fold as there is a current product and a planned new product. Release 8.0 is considered a mature product, i.e. middle of life, but is going towards end of life on the technological aspect. However, strategic management has decided that DG is to be renewed and the product is redesigned to a Software as a Service solution. While the new version will replace the current one, the decision was also made to label this new version as release 9.0 to maintain the marketing position for the product. Release 9.0 is currently in its beginning of life phase and the decision of the architect to look into the 'Connector' element is should be considered with release 9.0 in mind. Investigating this element for release 8.0 would not be considered as economically viable.

7. THREATS TO VALIDITY

This section discusses the threats to internal, external and construct validity [46, 48, 49] of our research.

7.1. Internal Validity

The internal validity is concerned with the uncontrolled factors that might affect the results of the experiment.

Energy measurement reliability. Although we were able to clearly identify differences between the estimated energy consumption of the selected processes, the estimations only accounted for percentages of the variation in EC. A brief cross–validation conducted by means of a self-obtained regression model based on resource consumption information (see Appendix) was still unable to fully explain the total EC. Hence, additional work is needed to have a clear and reliable attribution of the energy impact of single processes.

Sampling Interval. Both hardware and software measurement approaches have a sampling interval of one second. Given the nature of electrical power, this low sampling frequency might result in an underestimation of EC due to high-frequency energy components. However, this interval is also commonly used in the state of the art [42].

OS Effects. In the experiment the EC of the OS was included in the reported SEC for DG as we could not measure the OS separately. Ideally, the OS would be considered as a separate layer with its own, distinguishable EC. Also, we cannot fully exclude the possibility of OS processes and services becoming active in the background during a measurement. A deeper analysis of the performance measurements could detect such background activities, however this was deemed out of scope for our study. Instead, we mitigated this threat by performing a large number of trials (20 per each release) that should average out these effects as much as possible.

Interaction among multiple applications. The EC of software not related to DG was measured and taken into account (as overhead) while calculating the SEC. These measurements were performed separately to obtain clean overhead figures. However, by doing so we assume a negligible impact of the interaction among DG and other applications running in the system.

7.2. External validity

The external validity addresses the extent to which the results can be generalized beyond the experiment.

Experiment Setting. Our experiment is limited to a single application and tested on a single testbed. Hence, we cannot generalize the effect size of changes in the EC on our target population of commercial software products. Nevertheless, we argue that our work can be useful to generate awareness in software developers and architects about the knowledge gap in software energy efficiency.

Hardware Specificity. One of the main factors that could influence the EC measurements is the specific hardware; new generations of hardware often yield improved performance and EE. We mitigated this thread by performing extensive baseline measurements to determine the idle EC. We argue that differences might be found when comparing the absolute numbers, but that the relative proportions should be consistent across different hardware setups.

Measurement equipment. We applied both hardware and software measurement approaches to obtain our experimental data. Given the diversity of power meters and software tools available, each with their own advantages and limits, there is an unavoidable dependency on the equipment when it comes to the accuracy and detail of the measurements.

Team and organization dynamics. The added value identified for the different roles included in the interview could be specific for the team and the organization in which the team operates. An organization that does not have policies on sustainability or does not operate in a market that requires to do so, will probably not experience the added value as described. The same holds for a team that does not have any affinity with the topic of sustainability.

Interview results. We acknowledge that the number of interviewees, four out of a six-person team, is too small to generalize the results. However, given the specific focus of the interview on the experiment results in the context of a software product team and SPO, we could not extend our population beyond our specific case. Still, we managed to include all roles represented in the DG team and our results should be considered as a first insight into SEC from the relevant perspectives for a software product.

7.3. Construct validity

Construct validity addresses the degree to which the measures capture the concepts of interest in the experiment.

Metrics vs. outcome. A central aspect in performing EC measurements is to have a clear view on the metrics that should be reported. To mitigate this threat, in our experiment design we extensively report on our metrics selection and rationale for the experiment and the stakeholders. With regard to the metrics themselves, a consolidated list is already available in the literature [32].

Definition of change. The goal of our study is to relate software changes with their effects on the EC. Although we can empirically assess the difference between the energy consumption of the two application releases, we do not aim to provide a general definition of what a 'change' represents in software. For that purpose, we simply use two different releases of the DG product. Then, we provide insight as to which specific changes could affect the observed difference in EC. Further work is needed to pinpoint (and predict) the exact energy consumption impact of a generic software change.

Interview questions. The interview questions allowed us to identify potential added value of EC measurements for an SPO, and the semi-structured nature provided room for the interviewees to express themselves beyond our predefined set of questions. However, as the questions themselves were not validated, we acknowledge a different approach, i.e. direct questions on the added value of EC measurements, could yield different results. While formulating the interview questions however, we carefully considered the trade-off with the generalizability of our results as such an approach could yield results that are too specific with respect to the software product, the target market and the SPO.

8. CONCLUSIONS

Software sustainability, and in particular software energy efficiency, is hardly addressed in industrial contexts. Previous work [12] shows that due to lack of tools and knowledge, energy efficiency is not on the software roadmap of most SPOs. To investigate this aspect, we posed two **main research questions**: 'How can we reliably compare the energy consumption of large-scale software products across different releases?' (**RQ1**), further divided in 2 sub-research questions: 'How can we reliably measure the EC of a software product?' (*SQ1*) and 'How can we attribute energy consumption to individual software elements?' (*SQ2*), and 'What is the added value for a software producing organization to perform EC measurements on software products?' (**RQ2**). We presented the design and results of two empirical studies: an experiment performed on the EC of a commercial software product and an interview with the stakeholders from the SPO of the product. In the previous section we provided an answer to both RQs.

To reliably measure the EC of a software product (SQ1), we followed a rigorous methodology and we extensively documented our energy profiling method. Our experimental results show that the total EC of DG increased with release 8.0 w.r.t. 7.3. While this increase was expected, actual EC data now verifies it quantitatively. The stakeholders deemed this increase as justifiable, and the SPO experts could use the results to establish a better causation link.

The second sub-question (SQ2) addresses how to attribute EC to individual software elements. By means of energy profilers our experiment includes the estimation of the EC at process level. Our analysis showed that energy profilers can only explain percentages of the total EC for the application server and an even lower percentage for the database server. We tried to find a regression model to fill this gap in the data (see Appendix), but were unable to create an accurate model at the process level. However, our method successfully identified changes in EC at process level. Any differences found in the measurements between releases are considered to be caused by at least one of these changes and as such should be further investigated in further experimentation. Ideally, aspects of the energy profile can be related to the individual software elements in order to find quantifiable possible explanations for any changes in the EC.

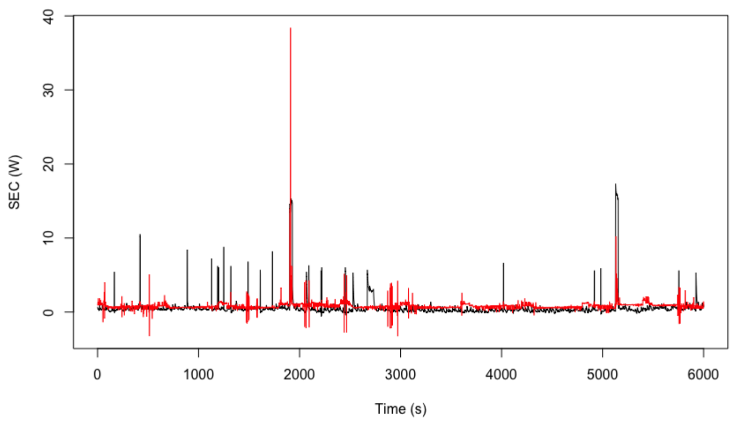
With respect to the second main research question (RQ2) we identified that the added value of EC measurements is on both operational and strategic level. On operational level, such measurements provide a new technique to identify inefficiencies in software. On a more strategic level, the SPO is able to increase the success of a software product by planning its evolution also in terms of energy efficiency. In general, putting energy efficiency on the software roadmap is expected to produce an improvement in the overall quality of the product, also in relation to other quality attributes. However, in order to exploit the added value of software energy efficiency to its fullest, the team must be aware of its potential.

In future work, we will further focus on the development phase i.e. to provide software developers with direct EC feedback during development. The insights we gained from the follow-up interview allow us to understand what form of feedback is more suitable and helpful for the team. A related direction for future research is to further investigate the metrics that characterize a software product in terms of energy efficiency. Finally, an accurate attribution of EC to specific software elements requires further research. Current tools and techniques such as regression models, while promising, still suffer from many limitations. Ideally, such models should be extended to also include (e.g.) OS-level processes, or better, to accurately separate the EC of these processes from the SEC. More complex data analysis and machine learning techniques have to be investigated.

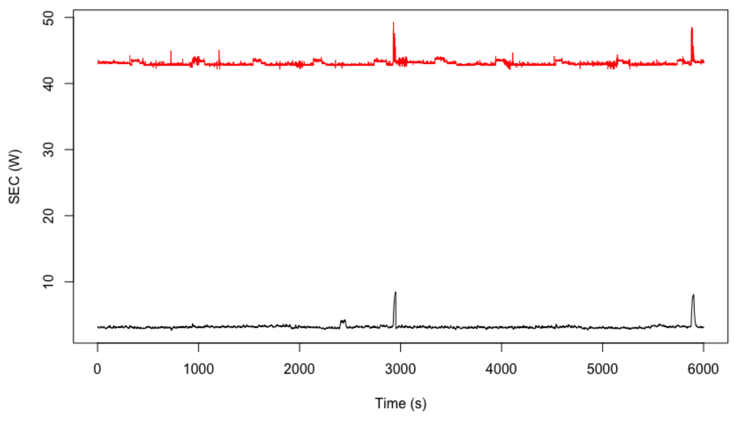
Our research contributes towards levering research on the EC of software products to a new level, instead of maintaining focused on the 'low-hanging fruits' as found with the interviews. The results show that software energy efficiency is a pioneering field, that still requires a large amount of empirical evidence before providing solid foundations and principles. For this reason, we strongly encourage other researchers to contribute to this field of research, and we make our data available (see Section 5) for reproduction and replication of our results, to stimulate the community towards new and interesting findings.

ACKNOWLEDGEMENTS

We would like to thank Edwig Huisman, Yuri Idris and Ronald Roos for their help in setting up the experiment and actively proposing and discussing possibilities for improvement, and Fabiano Dalpiaz, Garm Lucassen and Leo Pruijt for their valuable discussions and feedback. Also we would like to thank Jordy Broekman for his contribution concerning the energy consumption visualization techniques. Furthermore, as this work is an extension of a previously published paper [21], we would like to thank the reviewers and the editors of this special issue for their valuable comments.



(a) Application server predict set.



(b) Database server predict set.

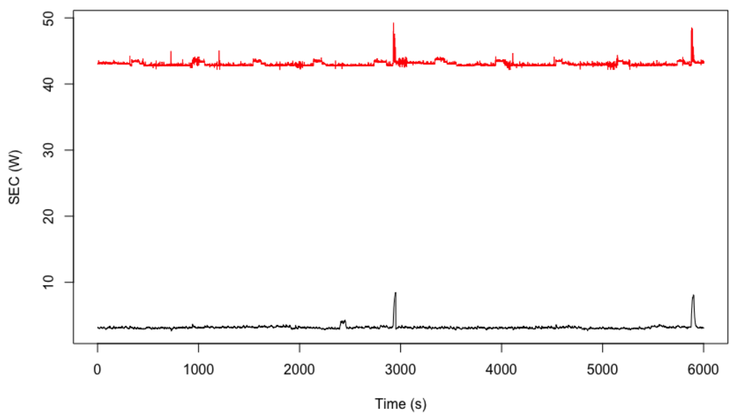
Figure 7. Overfitting of level 2 linear regression model trained on Application server data.

APPENDIX

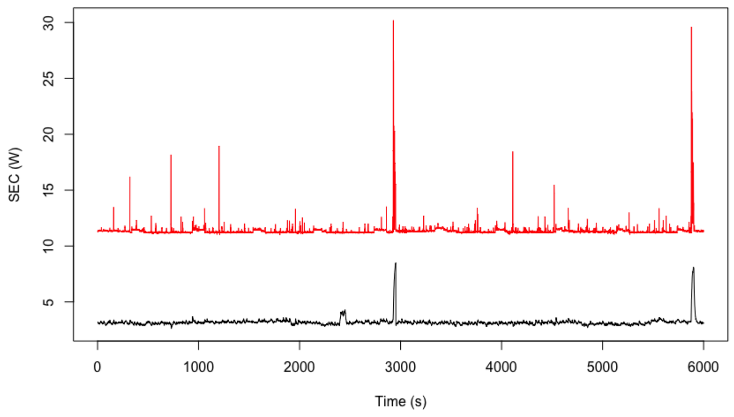
Regression Model to Predict the EC of Software Elements

The percentages of EC that JM reports on process level (on average 61.9% for the application server and 69.3% for the database server) indicate that we are still unable to explain a relatively large amount of the energy overhead of software execution. We initially attributed this to a lack of accuracy of JM: the tool is based upon a linear model that takes into account only a limited amount of hardware resources [56]. Hence, we hypothesized that this energy estimation gap could be due to unaccounted resources in the linear model. For this reason, we built a special–purpose linear model, trained by using performance data and the energy consumption measured by the WUP. In this Section we briefly explain the techniques we adopted and our results. In order to train the models, we used the values from a single experiment execution (see Section 4) as training set. We then use the remaining executions as test sets to evaluate the performance of our models.

Our first version of the regression model was obtained by means of multiple generalized linear models selection [57]. We selected the SEC as a response and through a genetic algorithm we generated multiple instances of linear models, using resource usage data as predictors (specifically, the used predictors were



(a) Non-robust model.



(b) Robust model.

Figure 8. Comparison of non-robust and robust regression models, trained on Application server data and predicting Database server data.

CPU time, IO bytes/sec, memory private bytes and working set: the other predictors (Section 3.1.4) were excluded due to collinearity). We fitted both level-1 and level-2 models, by analyzing interactions between the predictors. The models generated with this method were characterized by strong overfitting to the specific machine. Figure 7 shows why the model performs poorly: if fitted to the application server data, it performs well when predicting data from the same machine (Figure 7a), but is unable to predict the database server data with reasonable error (Figure 7b).

By performing some diagnostics on the model, we found out that there were a number of observations with high leverage (i.e. significantly influencing the regression coefficients). In addition, the data was also characterized by a high number of outliers. Hence, we opted for robust regression [58], a form of regression analysis that gives more reliable results in such conditions. Indeed, such method performed significantly better: in Figure 8 you can see a comparison of the performance of the two models. A large systematic error is still present, but in terms of Mean Absolute Percentage Error (MAPE) we were able to improve from of 12.6 to 2.6.

However, the fitted regression models exhibit negative coefficients. If we assume that a software process will use a positive and finite share of the system resources, this is probably not realistic. Hence, we adopted

Total SEC (Wh)	**Energy impact: Oracle - Joulemeter (Wh)**	**Energy impact: Oracle - Model (Wh)**
8.239	5.618 (68.18%)	5.724 (69.47%)

Table VIII. Example comparison of the energy estimation for Joulemeter and our special-purpose linear model for the Oracle process.

penalized linear regression [59], a regression technique that enables to specify constraints for the model features. This was done in order to enforce a positive value for the predictors.

The model obtained through penalized regression outperforms JM at machine-level prediction i.e. trying to predict the total system EC, see Figure 5. Our model has a MAPE of 0.005 when compared to WUP measurements, as opposed to the 0.08 of JM. Given these promising results, we used the same model to predict the impact at process-level. The prediction values were obtained by using the resource usage data of the single processes (as measured by Perfmon, see Section 3.1.4) as an input to the model. The intercept coefficient of the model was subtracted from the prediction, to remove the machine-dependant idle power estimation. Through this technique, we are able to obtain a realistic estimation of the energy impact for each process (in Table VIII an example for an execution of Oracle is shown).

If we aggregate the estimation obtained using our model for all the processes running in our application, however, we obtain very similar percentages to those computed via JM. Given this validation, we must conclude that JM provides a fairly accurate estimation of the energy consumption of specific processes.

Hence, a relatively high percentage of energy consumption cannot be attributed to specific processes. This is a strong indication that other factors are playing a role. Examples might be OS-level processes and system calls that the profiler is unable to detect as separate processes. Further work must be done to reliably attribute EC to specific software elements.

REFERENCES

1. Lago P, Kazman R, Meyer N, Morisio M, Müller HA, Paulisch F, Scanniello G, Penzenstadler B, Zimmermann O. Exploring initial challenges for green software engineering: summary of the first GREENS workshop, at ICSE 2012. *ACM SIGSOFT Software Engineering Notes* 2013; **38**(1):31–33.
2. Sun Y, Zhao Y, Song Y, Yang Y, Fang H, Zang H, Li Y, Gao Y. Green challenges to system software in data centers. *Frontiers of Comp. Sc. in China* 2011; **5**(3):353–368, doi:10.1007/s11704-011-0369-3. URL `http://dx.doi.org/10.1007/s11704-011-0369-3`.
3. Jagroep E, van der Werf JM, Brinkkemper S, Blom L, van Vliet R. Extending software architecture views with an energy consumption perspective. *Computing* 2016; :1–21doi:10.1007/s00607-016-0502-0. URL `http://dx.doi.org/10.1007/s00607-016-0502-0`.
4. Lago P, Koçak SA, Crnkovic I, Penzenstadler B. Framing sustainability as a property of software quality. *Commun. ACM* Sep 2015; **58**(10):70–78, doi:10.1145/2714560. URL `http://doi.acm.org/10.1145/2714560`.
5. Pathak A, Hu YC, Zhang M. Where is the energy spent inside my app?: fine grained energy accounting on smartphones with eprof. *Proceedings of the 7th ACM european conf. on Computer Systems*, EuroSys '12, ACM: New York, NY, USA, 2012; 29–42, doi:10.1145/2168836.2168841. URL `http://doi.acm.org/10.1145/2168836.2168841`.
6. Mittal R, Kansal A, Chandra R. Empowering developers to estimate app energy consumption. *Proceedings of the 18th annual international conference on Mobile computing and networking*, Mobicom '12, ACM: New York, NY, USA, 2012; 317–328, doi:10.1145/2348543.2348583. URL `http://doi.acm.org/10.1145/2348543.2348583`.
7. Chen H, Luo B, Shi W. Anole: A case for energy-aware mobile application design. *Parallel Processing Workshops (ICPPW), 2012 41st International Conference on*, 2012; 232–238, doi:10.1109/ICPPW.2012.34.
8. Procaccianti G, Lago P, Vetro A, Fernández DM, Wieringa R. The green lab: Experimentation in software energy efficiency. *Proceedings of the 37th International Conference on Software Engineering (ICSE)*, 2015.
9. Grosskop K, Visser J. Identification of application-level energy optimizations. *Proceeding of ICT for Sustainability (ICT4S)* 2013; :101–107.
10. Noureddine A, Rouvoy R, Seinturier L. Monitoring energy hotspots in software. *Automated Software Engineering* 2015; :1–42.
11. Jansen S, Brinkkemper S, Souer J, Luinenburg L. Shades of gray: Opening up a software producing organization with the open software enterprise model. *Journal of Systems and Software* 2012; **85**(7):1495–1510.
12. Pinto G, Castor F, Liu YD. Mining questions about software energy consumption. *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, ACM: New York, NY, USA, 2014; 22–31, doi:10.1145/2597073.2597110. URL `http://doi.acm.org/10.1145/2597073.2597110`.
13. Fricker SA. *Software Product Management*. Springer Berlin Heidelberg: Berlin, Heidelberg, 2012; 53–81, doi: 10.1007/978-3-642-31371-4_4.
14. Bekkers W, van de Weerd I, Spruit M, Brinkkemper S. A framework for process improvement in software product management. *Systems, Software and Services Process Improvement*, *Communications in Computer and Information Science*, vol. 99, Riel A, O'Connor R, Tichkiewitch S, Messnarz R (eds.). Springer Berlin Heidelberg, 2010; 1–12, doi:10.1007/978-3-642-15666-3_1. URL `http://dx.doi.org/10.1007/978-3-642-15666-3_1`.
15. Esmaeilzadeh H, Cao T, Yang X, Blackburn S, others. What is happening to Power, Performance, and Software? *IEEE Micro* 2012; .
16. Trefethen AE, Thiyagalingam J. Energy-aware software: Challenges, opportunities and strategies. *J. Comput. Sci.* Nov 2013; **4**(6):444–449.
17. Rangan KK, Wei GY, Brooks D. Thread Motion: Fine-grained Power Management for Multi-core Systems. *SIGARCH Comput. Archit. News* Jun 2009; **37**(3):302–313.
18. Pinto G, Castor F. On the Implications of Language Constructs for Concurrent Execution in the Energy Efficiency of Multicore Applications. *Proceedings of the 2013 Companion Publication for Conference on Systems, Programming, & Applications: Software for Humanity*, SPLASH '13, ACM: New York, NY, USA, 2013; 95–96.
19. Cao T, Blackburn SM, Gao T, McKinley KS. The Yin and Yang of Power and Performance for Asymmetric Hardware and Managed Software. *Proceedings of the 39th Annual International Symposium on Computer Architecture*, ISCA '12, IEEE Computer Society: Washington, DC, USA, 2012; 225–236.
20. Ebert C, Brinkkemper S. Software product management - an industry evaluation. *Journal of Systems and Software* 2014; **95**(0):10 – 18, doi:http://dx.doi.org/10.1016/j.jss.2013.12.042. URL `http://www.sciencedirect.com/science/article/pii/S0164121214000156`.
21. Jagroep EA, van der Werf J, Brinkkemper S, Procaccianti G, Lago P, Blom L, van Vliet R. Software energy profiling: Comparing releases of a software product. *IEEE/ACM International Conference on Software Engineering*, IEEE,

2016. To appear.

22. Fotrousi F, Fricker SA. *Software Analytics for Planning Product Evolution*. Springer International Publishing: Cham, 2016; 16–31, doi:10.1007/978-3-319-40515-5_2.
23. Li J, Tao F, Cheng Y, Zhao L. Big data in product lifecycle management. *The International Journal of Advanced Manufacturing Technology* 2015; **81**(1):667–684, doi:10.1007/s00170-015-7151-x.
24. Gupta A, Zimmermann T, Bird C, Nagappan N, Bhat T, Emran S. Detecting energy patterns in software development. *Microsoft Research Microsoft Corporation One Microsoft Way Redmond, WA* 2011; **98052**.
25. Li D, Hao S, Halfond WGJ, Govindan R. Calculating source line level energy information for android applications. *Proceedings of the 2013 International Symposium on Software Testing and Analysis*, ISSTA 2013, ACM: New York, NY, USA, 2013; 78–89, doi:10.1145/2483760.2483780. URL http://doi.acm.org/10.1145/2483760.2483780.
26. Liu Y, Xu C, Cheung SC. Characterizing and detecting performance bugs for smartphone applications. *Proceedings of the 36th International Conference on Software Engineering*, ACM, 2014; 1013–1024.
27. Ferreira MA, Hoekstra E, Merkus B, Visser B, Visser J. Seflab: A lab for measuring software energy footprints. *GREENS*, IEEE, 2013; 30–37.
28. Jagroep E, van der Werf JMEM, Jansen S, Ferreira M, Visser J. Profiling energy profilers. *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, ACM, 2015; 2198–2203.
29. Noureddine A, Rouvoy R, Seinturier L. Unit testing of energy consumption of software libraries. *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, SAC '14, ACM: New York, NY, USA, 2014; 1200–1205, doi:10.1145/2554850.2554932. URL http://doi.acm.org/10.1145/2554850.2554932.
30. Noureddine A, Rouvoy R, Seinturier L. A review of energy measurement approaches. *SIGOPS Operating Systems Review* Nov 2013; **47**(3):42–49, doi:10.1145/2553070.2553077. URL http://doi.acm.org/10.1145/2553070.2553077.
31. Kalaitzoglou G, Bruntink M, Visser J. A practical model for evaluating the energy efficiency of software applications. *ICT for Sust. 2014 (ICT4S-14)*, Atlantis Press, 2014.
32. Bozzelli P, Gu Q, Lago P. A systematic literature review on green software metrics. *Technical Report*, Technical Report: VU University Amsterdam 2013.
33. Hindle A, Wilson A, Rasmussen K, Barlow EJ, Campbell JC, Romansky S. Greenminer: A hardware based mining software repositories software energy consumption framework. *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, ACM: New York, NY, USA, 2014; 12–21, doi:10.1145/2597073.2597097. URL http://doi.acm.org/10.1145/2597073.2597097.
34. Procaccianti G, Lago P, Vetro A, Méndez Fernández D, Wieringa RJ. The Green Lab: Experimentation in Software Energy Efficiency. *International Conference on Software Engineering (ICSE)*, 2015.
35. Yang Q, Li JJ, Weiss DM. A Survey of Coverage-Based Testing Tools. *Comput. J.* 1 Aug 2009; **52**(5):589–597.
36. Pinto G, Castor F, Liu YD. Understanding energy behaviors of thread management constructs. *SIGPLAN Not.* Oct 2014; **49**(10):345–360, doi:10.1145/2714064.2660235. URL http://doi.acm.org/10.1145/2714064.2660235.
37. Rozanski N, Woods E. *Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives*. Addison-Wesley, 2011.
38. Trefethen AE, Thiyagalingam J. Energy-aware software: Challenges, opportunities and strategies. *Journal of Computational Science* 2013; **4**(6):444 – 449, doi:http://dx.doi.org/10.1016/j.jocs.2013.01.005. URL http://www.sciencedirect.com/science/article/pii/S1877750313000173, scalable Algorithms for Large-Scale Systems Workshop (ScalA2011), Supercomputing 2011.
39. Procaccianti G, Lago P, Lewis GA. A catalogue of green architectural tactics for the cloud. *Maint. and Evol. of Service-Oriented and Cloud-Based Systems (MESOCA), 2014 IEEE 8th Int'l Symp. on the*, 2014; 29–36, doi:10.1109/MESOCA.2014.12.
40. Ferreira AM, Pernici B. Managing the complex data center environment: an integrated energy-aware framework. *Computing* 2014; :1–41doi:10.1007/s00607-014-0405-x. URL http://dx.doi.org/10.1007/s00607-014-0405-x.
41. Shang W, Jiang ZM, Adams B, Hassan AE, Godfrey MW, Nasser M, Flora P. An exploratory study of the evolution of communicated information about the execution of large software systems. *Journal of Software: Evolution and Process* 2014; **26**(1):3–26.
42. Hindle A. Green mining: a methodology of relating software change and configuration to power consumption. *Empirical Software Engineering* 2013; :1–36doi:10.1007/s10664-013-9276-6. URL http://dx.doi.org/10.1007/s10664-013-9276-6.
43. Zhang G, Zhang K, Zhu X, Chen M, Xu C, Shao Y. Modeling and analyzing method for cps software architecture energy consumption. *Journal of Software* 2013; **8**(11). URL http://www.ojs.academypublisher.com/

index.php/jsw/article/view/jsw081129742981.

44. Zhu H, Lin C, Liu Y. A programming model for sustainable software. *Software Engineering (ICSE), 2015 IEEE/ACM 37th IEEE International Conference on*, vol. 1, 2015; 767–777, doi:10.1109/ICSE.2015.89.
45. Becker C, Chitchyan R, Duboc L, Easterbrook S, Penzenstadler B, Seyff N, Venters C. Sustainability Design and Software: The Karlskrona Manifesto. *Software Engineering (ICSE), 2015 IEEE/ACM 37th IEEE International Conference on*, vol. 2, IEEE, 2015; 467–476, doi:10.1109/ICSE.2015.179.
46. Wohlin C, Runeson P, Hst M, Ohlsson MC, Regnell B, Wessln A. *Experimentation in Software Engineering*. Springer Publishing Company, Incorporated, 2012.
47. Kitchenham BA, Pfleeger SL, Pickard LM, Jones PW, Hoaglin DC, Emam KE, Rosenberg J. Preliminary guidelines for empirical research in software engineering. *IEEE Transactions on Software Engineering* 2002; **28**(8):721–734, doi:http://doi.ieeecomputersociety.org/10.1109/TSE.2002.1027796.
48. Runeson P, Höst M. Guidelines for conducting and reporting case study research in software engineering. *Empirical software engineering* 2009; **14**(2):131–164.
49. Juristo N, Moreno AM. *Basics of Software Engineering Experimentation*. 1st edn., Springer Publishing Company, Incorporated, 2010.
50. Xu L, Brinkkemper S. Concepts of product software. *European Journal of Information Systems* 2007; **16**(5):531–541.
51. Jagroep EA, van der Werf JMEM, Spauwen R, Blom L, van Vliet R, Brinkkemper S. An energy consumption perspective on software architecture. *Software Architecture*, no. 9278 in LNCS, Springer, 2015; 239–247.
52. Yin R. *Case Study Research: Design and Methods*. Applied Social Research Methods, SAGE Publications, 2009. URL http://books.google.ca/books?id=FzawIAdilHkC.
53. Bass L, Clements P, Kazman R. *Software Architecture in Practice*. SEI Series in Software Engineering, Pearson Education, 2012.
54. Kruchten P, Nord RL, Ozkaya I. Technical debt: from metaphor to theory and practice. *Ieee software* 2012; **29**(6):18–21.
55. Mosley H, Mayer A. Benchmarking national labour market performance: A radar chart approach. *Technical Report*, WZB Discussion paper 1999.
56. Kansal A, Zhao F, Liu J, Kothari N, Bhattacharya AA. Virtual machine power metering and provisioning. *Proceedings of the 1st ACM Symposium on Cloud Computing*, SoCC '10, ACM: New York, NY, USA, 2010; 39–50.
57. Burnham KP, Anderson DR. *Model selection and multimodel inference: a practical information-theoretic approach*. Springer Science & Business Media, 2003.
58. Andersen R. *Modern methods for robust regression*. Sage, 2008.
59. Tibshirani R. Regression shrinkage and selection via the lasso. *J. R. Stat. Soc. Series B Stat. Methodol.* 1 Jan 1996; **58**(1):267–288.