Software Energy Profiling: Comparing Releases of a Software Product

Erik A. Jagroep, Jan Martijn van der Werf, Sjaak Brinkkemper
Utrecht University
Dept. of Information and Computing Sciences
Utrecht, The Netherlands
{e.a.jagroep, j.m.e.m.vanderwerf, s.brinkkemper}@uu.nl

Giuseppe Procaccianti, Patricia Lago
Vrije Universiteit Amsterdam
Dept. of Computer Science
Amsterdam, The Netherlands
{g.procaccianti, p.lago}@vu.nl

Leen Blom, Rob van Vliet
Centric
Gouda, The Netherlands
{leen.blom, rob.van.vliet}@centric.eu

ABSTRACT

In the quest for energy efficiency of Information and Communication Technology, so far research has mostly focused on the role of hardware. However, as hardware technology becomes more sophisticated, the role of software becomes crucial. Recently, the impact of software on energy consumption has been acknowledged as significant by researchers in software engineering. In spite of that, measuring the energy consumption of software has proven to be a challenge, due to the large number of variables that need to be controlled to obtain reliable measurements. Due to cost and time constraints, many software product organizations are unable to effectively measure the energy consumption of software. This prevents them to be in control over the energy efficiency of their products.

In this paper, we propose a software energy profiling method to reliably compare the energy consumed by a software product across different releases, from the perspective of a software organization. Our method allows to attribute differences in energy consumption to changes in the software. We validate our profiling method through an empirical experiment on two consecutive releases of a commercial software product. We demonstrate how the method can be applied by organizations and provide an analysis of the software related changes in energy consumption. Our results show that, despite a lack of precise measurements, energy consumption differences between releases of a software product can be quantified down to the level of individual processes. Additionally, the results provide insights on how specific software changes might affect energy consumption.

ICSE '16 Companion, May 14-22, 2016, Austin, TX, USA

 ISBN 978-1-4503-4205-6/16/05…$15.00
DOI: http://dx.doi.org/10.1145/2889160.2889216

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Metrics

Keywords

Energy Efficiency, Profiling, Software Architecture, Software Product

1. INTRODUCTION

In the search for energy efficient solutions for the Information and Communication Technology (ICT) industry, research has mostly focused on hardware aspects in order to reduce the environmental impact of the sector. 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 [18]. Although hardware ultimately consumes energy, software provides the instructions that guide the hardware behavior [33].

For example, the impact of software is clearly visible in the mobile phone domain. Although the EC of mobile applications is typically closely monitored due to battery constraints [3, 21, 25], we have reached the point of requiring quad-core processors to ensure smooth operation. Nowadays, 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. However, in contrast to the mobile domain, EC measurements with regard to 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 [29]. Recently, several of such approaches have been proposed, both hardware [7] and software based [24],

which were able to identify opportunities for considerable savings in EC.

However, these approaches have not been adopted in industrial contexts so far. Although Software Product Organizations (SPOs), i.e. independent software vendors and open-source foundations, have software development as their core activity [13], having accurate software EC measurements still requires significant investments in terms of resources and specialized knowledge. As a consequence, SPOs are unable to directly address software energy efficiency, even when required to do so (e.g. 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; contradicting goals could require a trade-off to be made [12].

A deeper understanding of the matter is required to properly address the EC of the software itself. In this research we investigate a method that can be applied by SPOs to gain control over the EC of their software products. Given the dynamics of the software industry, such as multiple releases in relatively rapid succession, the method should explicitly address these dynamics by enabling an SPO to report any improvement or deterioration in EC with a new release. Our proposed profiling method is applied in an empirical experiment on a commercial software product.

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

1.1 Contributions

The main contributions of this paper are:

Empirical study: To the best of our knowledge this is one of the few papers that compares the EC of software products across releases on a commercial software product. The study includes a demonstration of tooling and metrics to find the relevant energy hotspots [29] for software products and enables an SPO to quantify EC differences brought about by changes in the software.

Profiling method: We show in detail how to set up an environment to perform EC measurements across releases.

Regression model: In addition to the method, we provide insight in how to create a regression model to predict EC more accurately based on performance and EC data. With this regression model we add to the 'green mining' research area [10].

2. RESEARCH QUESTIONS

Based on the problem explained in Section 1 we formulate our **main research question** as follows:

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

In the RQ, 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 our research.

A prerequisite for comparing the EC of a software product is being able to measure the software EC. Therefore our first research sub-question is:

SQ1: *How can we reliably measure the EC of a software product?*

Software-only approaches can be roughly categorized in two sets: (source code) instrumentation [24] and energy profilers [11]. Hardware-based approaches (e.g. [6]) rely instead on physical power meters to be connected to hardware devices. Given our focus on large scale software products, we see two potential issues with source code instrumentation: the *complexity* overhead and the required *investment* to apply them in terms of software development skills. Hence, we do not see them as usable in an industrial setting. Software energy profilers do not require a high effort to be adopted, but are shown to be inaccurate in their measurements [11]. On the other hand, hardware-based 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. In Section 3 we design our experiment to leverage both measurements: we use software profilers to obtain fine-grained, software-level estimations and validate them with hardware measurements obtained via power meters.

Our proposed profiling method is applied in an empirical experiment on a commercial software product, described in Section 3 and Section 4. For SPOs to actually be able to influence the EC, changes in EC should be related to the individual software elements. To this end we formulated two more sub-research questions (Section 5):

SQ2: *How can we attribute energy consumption to individual software elements?*

SQ3: *How can we relate differences in energy consumption to changes in a software element?*

The second sub-question (SQ2) is set to investigate how the EC is divided over the combination of software elements that comprise the software product. For this, we make use of a software tool, Joulemeter, as software measurements can provide these details. After relating EC to software elements, we answer SQ3 by analyzing the impact of software changes on EC.

3. EXPERIMENT DESIGN

To answer the research questions presented in the previous section, we performed an experiment to compare the EC of a commercial software product, Document Generator (DG), across different releases and to explain differences at software architecture level. Our experiment follows the guidelines provided in [14, 17, 31, 36] and the "green mining" method [9] 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. Note that compiling and analyzing the results is described in Section 4 and Section 5 respectively.

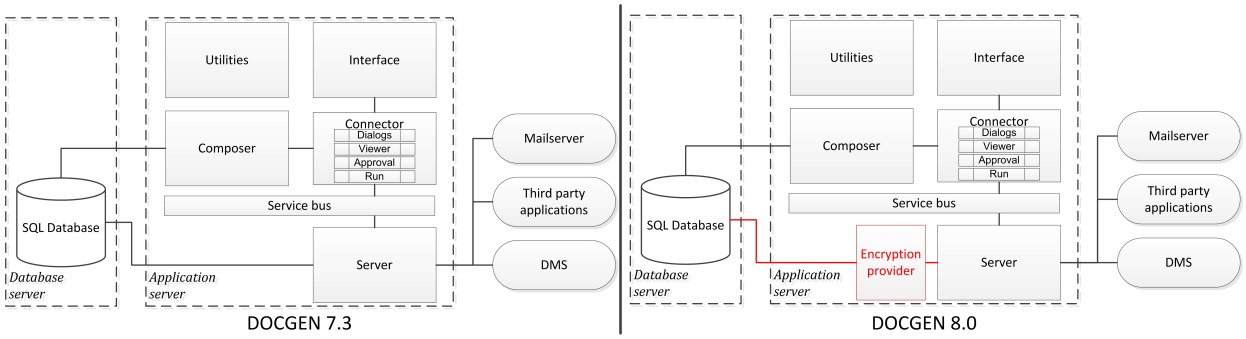

Figure 1: The functional architectures for Document Generator (DG) releases 7.3 (left) and 8.0 (right) portrayed on a commercial deployment. The changes are in red.

3.1 Product Under Study: DG

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 [37].

In Figure 1 the Software Architecture (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*.

3.1.1 Differences between Releases

Looking at the SA, the major difference 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. Cross-checking our findings with the DG architect ensured completeness of our list of relevant changes for the experiment.

3.1.2 Test Case

For the experiment we chose to stress DG with its core functionality, namely the generation of documents. 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 are to be monitored on their respective servers. As the 'Microsoft Enhanced Cryptographic Provider' is not an executable but a dynamic library, it cannot be monitored in isolation.

3.2 Experimental 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 2. The specifications of the application and database servers are provided in Table 1. To ensure consistency with regard to external factors (e.g. room temperature), the servers were installed in an operational 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.

3.2.1 Baseline Measurements

To obtain a clean measurement of the EC related to solely DG, we need to determine the idle EC for the hardware that is used. This figure 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 suggest to stop any service and process known not to be

Table 1: 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 2008 R2 Standard (64-bit), Service Pack 1	Windows 2008 R2 Standard (64-bit), Service Pack 1
Software	DOCGEN 7.3 and 8.0	Oracle 11.0.2.0.4.0

related to or required for the software product under test to minimize their effects. For example, in our case the automatic Windows update service was disabled. In our setup, we additionally use a separate logging server to minimize the overhead caused by the data collection process.

Another software related aspect 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 most likely will 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 should be 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.

3.2.2 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 rely on power metering devices. As these meters are installed between a device and its power source, a meter is required for each power supply unit of the devices under test. Although these meters are capable of achieving high levels of accuracy, their specifications should be taken into account in the analysis as even small measurement errors might prove significant at software level.

Each of the servers in our setup is instrumented with a single WattsUp? Pro (WUP) device[1] (see Figure 2). WUP devices record the total energy consumption of the hardware once per second.

Regarding software profilers, we require such tools to not only estimate the total energy consumption of the system, but also to profile individual software processes. Using their respective power models, software profilers estimate the EC of a system at run time, based on the computational resources used by applications and by monitoring the hardware resource usage. Unfortunately, although a more fine-grained interval is desired [10], these tools record measurements with a one second interval. While the usability and accuracy of energy profilers still have margins for improvement [11,22], 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 releases can still provide useful insights.

[1]http://www.wattsupmeters.com/secure/products.php?pn=0, last visited on February 10th, 2016

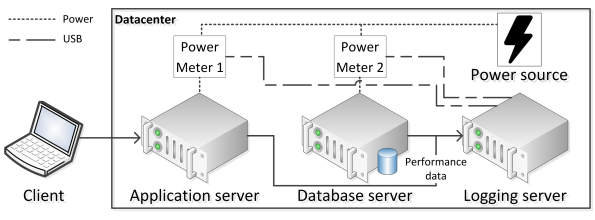

Figure 2: Experiment environment.

In our experiment, we make use of 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 [11] 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). Although relatively coarse, measurements on process level (i.e. the concurrency views on the system [30]) can be translated to more fine-grained aspects using an architectural perspective [12].

3.3 EC and Performance Metrics

Comparing literature (cf. [9,12,15]) 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 [4], especially in the case of a pioneering topic like the EC of software [7]. For the experiment we want to measure the SEC and UEC metrics as defined in [12] at the level of the concurrent processes of the product, facilitating discussion with software architect of DG.

In addition to the EC, the hardware performance needs to be recorded as performance data could fill the gap when it comes to accurately relating EC to individual software elements [2,12,15]. 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.

The performance of the application and database servers are measured using the standard performance monitor (perfmon) provided with Microsoft Windows. As perfmon does not include network performance counters at process level, we exclude these from the experiment. Following the definition of the 'Unit Energy Consumption' [12], in our experiment we set up performance counters for the most frequently monitored 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)

Performance data is remotely collected using the logging server, thereby minimizing the overhead of measurement on the actual hardware.

3.3.1 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 is not in sync, there is a risk of missing the data related to this activity. A simple solution to this problem is to synchronize the clocks for all measurement instances using the Network Time Protocol (NTP).

3.4 Protocol

While the "green mining" method [9] 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 [9]:

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 [36]. To increase the consistency across measurements, a script is used to generate 5014 documents using DG.

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 the performance measures for both the application and the database server;
- the start and end timestamp for each measurement;

After each measurement, both servers have been reverted to the initial state. To mitigate the risk of mismatched time data, all devices are continuously synchronized using the Network Time Protocol (NTP).

4. EXPERIMENT RESULTS

In this section we extensively report our experimental results. 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.

4.1 Baseline Measurements

The results of the idle and JM overhead measurements are presented in Table 2 along with the measurement time to determine the averages. 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. If we use the JM measurements to determine the average power consumption (right of Table 2), a larger difference is perceived which is in line with the findings presented in [11].

4.2 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 3.4. Table 3 summarizes the results in terms of the averages for the application and database server. 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 zero by JM, despite them being active. The same holds for the 'Interface' process on the application server, that runs the GUI of DG: it was not active during the experiment as the DG execution was scripted.

Comparing the measurements between releases, two differences are clearly visible. First is the difference in average run length of 12 seconds, which is surprising considering the fact that the scripts used to stress both releases were identical. A second difference is the overall increase in energy consumption of DG 8.0 as compared to 7.3 with 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. If the 'idle with JM' EC is subtracted from these differences for 12 extra seconds, to remove the effect of having longer

Table 2: Comparison of server power consumption in "idle" and "idle with JM" scenarios.

Server	Idle		Idle with JM		Idle with JM according to JM	
	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

measurements, we still find a difference of 2.05 Wh and 0.32 Wh that is on the account of DG.

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. These EC figures are obtained by calculating the area under the power consumption curve. 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.** Although small, all differences found could add up significantly with each installation and generated document. Consider [9], where a savings of 0.25 W is shown to potentially equal the power use of an American household for a month.

4.3 Joulemeter Estimations

The SEC can also be calculated using the estimations provided by JM. 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. Straightaway we notice the 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 of a multitude of processes that become active in the background alongside the DG processes.

5. DISCUSSION

In this section we discuss the results presented in the previous Section and answer our research sub-questions. The complete dataset of the experiment is openly available[2].

5.1 SQ1: Measuring the EC

The profiling method that was applied in the experiment encompasses activities to ensure 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. We also did not come across any peculiarities while collecting and processing the data. This allows us to conclude that **the measurement method we adopted is able to reliably measure the EC of a software product**.

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

5.2 SQ2: Relating EC to Software Elements

The percentages of EC that JM leaves unexplained 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.

One possible explanation is a lack of accuracy of JM. The profiling tool is based upon a linear model that takes into account only a limited amount of hardware resources [16]. Hence, it is reasonable to conclude that this energy estimation gap is due to unaccounted resources in the linear model. For this reason, we tried to build a special–purpose linear model, trained by using performance data and the energy consumption measured by the WUP. The model was built by means of penalized linear regression [34], 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. This assumption builds upon the rationale that a software process will use a positive and finite share of the system resources.

Our special-purpose model outperforms JM at machine-level prediction i.e. trying to predict the total system EC, see Figure 3. The model has a MAPE of 0.004 when compared to WUP measurements, whereas JM has 0.005. However, the process-level prediction is quite overestimated, probably due to an incorrect determination of the intercept term. This is a strong indication that other factors are playing a role. Examples might be networking devices, or OS-level processes and system calls that the profiler is unable to detect as separate processes. Hence, further work must be

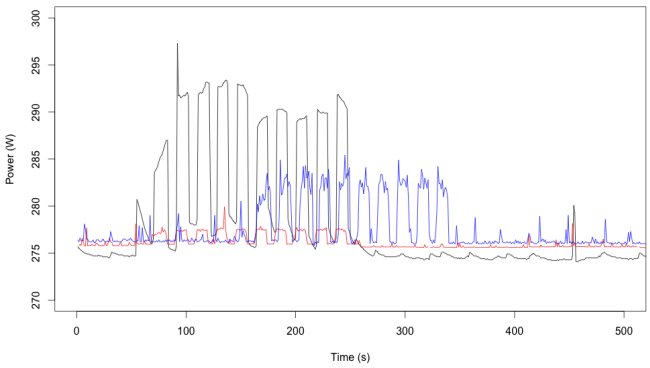

Figure 3: Performance of our special-purpose regression model (in red) vs. Joulemeter (in blue). Measured values by WUP are in black.

Table 3: Summary of the experimental results for both DG releases.

		Application server					Database server				
		7.3		8.0		Diff	7.3		8.0		Diff
		μ	σ	μ	σ	Δ	μ	σ	μ	σ	Δ
Run length	(hh:mm:ss)	2:48:16	4 s	2:48:28	7 s	+12 s	2:48:16	4 s	2:48:28	7 s	+12 s
Processed Documents		5014		5014			5014		5014		
WUP (Wh)		774.59	1.18	777.56	0.84	+2.97	716.99	0.45	718.16	0.61	+1.17
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					
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

done to reliably attribute EC to specific software elements.

That being said, **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' process in the database server is by far the most energy–consuming. This indicates that the database is a potential *hotspot* [29] and, as such, a candidate for optimization.

5.3 SQ3: Relate EC Differences to Software Changes

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 dll, we were not able to perform measurements specifically on this element and thus could not be identified separately from the SEC figures. 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. A difference that 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 that JM reports that energy is consumed increases with release 8.0 for the 'Server', 'Connector' and 'Oracle' processes. Combining this finding with the linear model applied by JM, more seconds of measurement, i.e. more activity, should mean 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 with 4.14 Wh for the generation of 5014 documents. 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 accept the increase in EC. To increase efficiency, however, the software architect will still look into the 'Connector' element.

The results in this section show that, **by using the results of the profiling method, we are able to think of grounded explanations for differences found in the EC across releases of a software product**. Although our profiling method requires more detailed measurements to draw hard conclusions, we are able to provide guidance when EC aspects are discussed and point out possible unexpected differences.

6. THREATS TO VALIDITY

This section presents the threats to internal, external and construct validity as required by [14, 31, 36].

6.1 Internal Validity

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

JM 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, reveals a much higher impact of single processes on total energy consumption than estimated by Joulemeter. Hence, additional work is needed to have a clear and reliable attribution of the energy impact of single processes.

Measurement 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 applied in the state of the art [9].

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 during a measurement. Ideally, the OS would be considered as a separate layer with its own, distinguishable EC. Also, there is the possibility of OS processes and services that might become active during a measurement without a direct, controllable trigger. Deep analysis of the performance measurements could show whether such an activity has occurred during a measurement, provided that the activity can be measured to begin with.

EC Overhead. 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 do not include any effect of having multiple software applications running simultaneously. Further research is required to fully understand and control this effect.

6.2 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 boast improved performance and EE. For this reason we explicitly added, among others, idle EC in the measurement method, to create a matching EC profile for the hardware. 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 tooling. Both hardware and software measurement approaches are applied to obtain the experiment 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.

6.3 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. In the experiment method we included a section on choosing the appropriate metrics for the experiment and the stakeholders. In our experiment design, the measurements reflect the data required to calculate the metrics. With regard to the metrics themselves, a solid list is already available in the literature [2].

Definition of change. Our goal 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.

7. RELATED WORK

Measurement method: 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 [8], eprof [25]), or by emulation tools that allow developers to estimate the EC of their application on their development stations [21]. Since battery drain can be monitored relatively easily and mobile devices have similar hardware architectures, approaches surface where EC is related to source lines [19]. Additionally, as performance profilers are quite mature in mobile computing, EC profilers can build upon such tools [20].

In the area of large scale software products, multiple approaches can be identified for energy profiling in complex environments. A commonly accepted approach is to use performance measurements to explain and characterize software and its EC characteristics [2, 15]. Others focus on finding more fine-grained power models [22] used by multiple tools to deliver more accurate measurements at software level. Finding a regression model using EC and performance data could be considered part of green mining [10]. Unfortunately, due to lack of publicly available data, green mining is still an immature area.

Although different information is used, the 'JalenUnit' [23] can be used to, a.o., detect energy bugs and understand energy distribution. The 'JalenUnit' infers the energy consumption model of software libraries from execution traces. Despite the differences in approach and accuracy, measurement methods all focus on identifying energy hotspots [24].

Software architecture: Although diverse, the results in this field of research share a common principle able to relate them to software and software design; SA. For example, recent study shows data locality plays an important role in the EC of multi-threaded programs [27]. An information viewpoint [30] 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 [30], enables stakeholder to structurally address concerns on different aspects of the system design.

SA also allows a stakeholder to explore design trade-offs for the software. Increased performance, a quality attribute for the software, does not always have a direct relation with EC [35]. A different design trade-off is to exchange modules or services for more energy efficient sustainable variants, e.g. cloud federation [28]. SA helps to identify adjustments on different levels in complex environments [5].

EC comparison between releases: Comparing aspects across releases is often discussed in terms of software evolution [32]. However, only few papers were found that investigate the EC of software and include a comparison between different releases. In [9] 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 [38]. 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.

Awareness: A different approach is to increase developer awareness in software energy efficiency. The 'Eco' programming model [39], 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 [1]. However, Pinto et al. [26] 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.

8. CONCLUSIONS

In this paper we present the results of an experiment performed on the EC of a commercial software product. We consider the perspective of a SPO aiming to exert control over the EC of its software products, and posed the following **main research question**: 'How can we reliably compare the energy consumption of large scale software products across different releases?'. We provide an answer to this question by investigating three sub-research questions.

To reliably measure the EC of a software product (SQ1), we followed a prescribed methodology to perform an experiment extended with hands-on instructions to create an energy profile of the hardware, obtain EC data that is consistent across measurements and ensure that the data is ready for analysis. We managed to successfully perform an experiment in an industrial setting, using a commercial software product. The reported EC metrics proved useful to communicate and discuss the results with industrial stakeholders.

The second and third sub-questions aim at providing an SPO with the required information to actually control the EC of a software product. Starting with relating EC to individual software elements (SQ2), our experiment includes the estimation of the EC at process level by means of energy profilers. 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, but were (yet) unable to create an accurate model at the process level. However, our method successfully identified changes in EC at process level.

The final sub-question (SQ3) addresses how changes in EC can be related to changes in the software elements. This requires a clear overview of the changes that are made and an energy profile of the application. 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 using the data available. 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. Our experiment showed that the total EC of DG increased with release 8.0 w.r.t. 7.3. While this increase was expected, actual EC data (provided by the WUP) now verifies it quantitatively. Relating the increased EC to the changes in the software, stakeholders deemed this increase as justifiable, and the SPO experts could use the quantification to establish a better causation link.

An SPO that follows the experiment design can compare the EC of their products across different releases and gather insights in where adjustments should be made to decrease EC. In future work, we plan to apply our method during the development phase, aiming to provide software developers with direct EC feedback while developing.

Software energy efficiency is a pioneering field that requires a large amount of empirical evidence to be produced. We strongly encourage other researchers to contribute to this field of research, and we make our data available (see Section Section 4)for reproduction and replication of our results, or to allow the discovery of new and interesting findings.

9. ACKNOWLEDGMENTS

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 to improve the experiment, and Fabiano Dalpiaz, Garm Lucassen and Leo Pruijt for their valuable discussions and feedback.

10. REFERENCES

[1] C. Becker, R. Chitchyan, L. Duboc, S. Easterbrook, B. Penzenstadler, N. Seyff, and C. Venters. Sustainability Design and Software: The Karlskrona Manifesto. In *Software Engineering (ICSE), 2015 IEEE/ACM 37th IEEE International Conference on*, volume 2, pages 467–476. IEEE, May 2015.

[2] P. Bozzelli, Q. Gu, and P. Lago. A systematic literature review on green software metrics. Technical report, Technical Report: VU University Amsterdam, 2013.

[3] H. Chen, B. Luo, and W. Shi. Anole: A case for energy-aware mobile application design. In *Parallel Processing Workshops (ICPPW), 2012 41st International Conference on*, pages 232–238, 2012.

[4] C. Ebert and S. Brinkkemper. Software product management - an industry evaluation. *Journal of Systems and Software*, 95(0):10 – 18, 2014.

[5] A. M. Ferreira and B. Pernici. Managing the complex data center environment: an integrated energy-aware framework. *Computing*, pages 1–41, 2014.

[6] M. A. Ferreira, E. Hoekstra, B. Merkus, B. Visser, and J. Visser. Seflab: A lab for measuring software energy footprints. In *GREENS*, pages 30–37. IEEE, May 2013.

[7] K. Grosskop and J. Visser. Identification of application-level energy optimizations. *Proceeding of ICT for Sustainability (ICT4S)*, pages 101–107, 2013.

[8] A. Gupta, T. Zimmermann, C. Bird, N. Nagappan, T. Bhat, and S. Emran. Detecting energy patterns in software development. *Microsoft Research Microsoft Corporation One Microsoft Way Redmond, WA*, 98052, 2011.

[9] A. Hindle. Green mining: a methodology of relating software change and configuration to power consumption. *Empirical Software Engineering*, pages 1–36, 2013.

[10] A. Hindle, A. Wilson, K. Rasmussen, E. J. Barlow, J. C. Campbell, and S. Romansky. Greenminer: A hardware based mining software repositories software energy consumption framework. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 12–21, New York, NY, USA, 2014. ACM.

[11] E. Jagroep, J. M. E. M. van der Werf, S. Jansen, M. Ferreira, and J. Visser. Profiling energy profilers. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, pages 2198–2203. ACM, 2015.

[12] E. A. Jagroep, J. M. E. M. van der Werf, R. Spauwen, L. Blom, R. van Vliet, and S. Brinkkemper. An energy consumption perspective on software architecture. In *9th European Conference on Software Architecture*, number 9278 in LNCS, pages 239–247. Springer, 2015.

[13] S. Jansen, S. Brinkkemper, J. Souer, and L. Luinenburg. Shades of gray: Opening up a software

producing organization with the open software enterprise model. *Journal of Systems and Software*, 85(7):1495–1510, 2012.

[14] N. Juristo and A. M. Moreno. *Basics of Software Engineering Experimentation*. Springer Publishing Company, Incorporated, 1st edition, 2010.

[15] G. Kalaitzoglou, M. Bruntink, and J. Visser. A practical model for evaluating the energy efficiency of software applications. In *ICT for Sust. 2014 (ICT4S-14)*. Atlantis Press, 2014.

[16] A. Kansal, F. Zhao, J. Liu, N. Kothari, and A. A. Bhattacharya. Virtual machine power metering and provisioning. In *Proceedings of the 1st ACM Symposium on Cloud Computing*, SoCC '10, pages 39–50, New York, NY, USA, 2010. ACM.

[17] B. A. Kitchenham, S. L. Pfleeger, L. M. Pickard, P. W. Jones, D. C. Hoaglin, K. E. Emam, and J. Rosenberg. Preliminary guidelines for empirical research in software engineering. *IEEE Transactions on Software Engineering*, 28(8):721–734, 2002.

[18] P. Lago, R. Kazman, N. Meyer, M. Morisio, H. A. Müller, F. Paulisch, G. Scanniello, B. Penzenstadler, and O. Zimmermann. Exploring initial challenges for green software engineering: summary of the first GREENS workshop, at ICSE 2012. *ACM SIGSOFT Software Engineering Notes*, 38(1):31–33, 2013.

[19] D. Li, S. Hao, W. G. J. Halfond, and R. Govindan. Calculating source line level energy information for android applications. In *Proceedings of the 2013 International Symposium on Software Testing and Analysis*, ISSTA 2013, pages 78–89, New York, NY, USA, 2013. ACM.

[20] Y. Liu, C. Xu, and S.-C. Cheung. Characterizing and detecting performance bugs for smartphone applications. In *Proceedings of the 36th International Conference on Software Engineering*, pages 1013–1024. ACM, 2014.

[21] R. Mittal, A. Kansal, and R. Chandra. Empowering developers to estimate app energy consumption. In *Proceedings of the 18th annual international conference on Mobile computing and networking*, Mobicom '12, pages 317–328, New York, NY, USA, 2012. ACM.

[22] A. Noureddine, R. Rouvoy, and L. Seinturier. A review of energy measurement approaches. *SIGOPS Operating Systems Review*, 47(3):42–49, Nov. 2013.

[23] A. Noureddine, R. Rouvoy, and L. Seinturier. Unit testing of energy consumption of software libraries. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, SAC '14, pages 1200–1205, New York, NY, USA, 2014. ACM.

[24] A. Noureddine, R. Rouvoy, and L. Seinturier. Monitoring energy hotspots in software. *Automated Software Engineering*, pages 1–42, 2015.

[25] A. Pathak, Y. C. Hu, and M. Zhang. Where is the energy spent inside my app?: fine grained energy accounting on smartphones with eprof. In *Proceedings of the 7th ACM european conf. on Computer Systems*, EuroSys '12, pages 29–42, New York, NY, USA, 2012. ACM.

[26] G. Pinto, F. Castor, and Y. D. Liu. Mining questions about software energy consumption. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 22–31, New York, NY, USA, 2014. ACM.

[27] G. Pinto, F. Castor, and Y. D. Liu. Understanding energy behaviors of thread management constructs. *SIGPLAN Not.*, 49(10):345–360, Oct. 2014.

[28] G. Procaccianti, P. Lago, and G. A. Lewis. A catalogue of green architectural tactics for the cloud. In *Maint. and Evol. of Service-Oriented and Cloud-Based Systems (MESOCA), 2014 IEEE 8th Int'l Symp. on the*, pages 29–36, Sept 2014.

[29] G. Procaccianti, P. Lago, A. Vetro, D. M. Fernández, and R. Wieringa. The green lab: Experimentation in software energy efficiency. In *Proceedings of the 37th International Conference on Software Engineering (ICSE)*, 2015.

[30] N. Rozanski and E. Woods. *Software systems architecture: working with stakeholders using viewpoints and perspectives*. Addison-Wesley, 2012.

[31] P. Runeson and M. Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical software engineering*, 14(2):131–164, 2009.

[32] W. Shang, Z. M. Jiang, B. Adams, A. E. Hassan, M. W. Godfrey, M. Nasser, and P. Flora. An exploratory study of the evolution of communicated information about the execution of large software systems. *Journal of Software: Evolution and Process*, 26(1):3–26, 2014.

[33] Y. Sun, Y. Zhao, Y. Song, Y. Yang, H. Fang, H. Zang, Y. Li, and Y. Gao. Green challenges to system software in data centers. *Frontiers of Comp. Sc. in China*, 5(3):353–368, 2011.

[34] R. Tibshirani. Regression shrinkage and selection via the lasso. *J. R. Stat. Soc. Series B Stat. Methodol.*, 58(1):267–288, 1 Jan. 1996.

[35] A. E. Trefethen and J. Thiyagalingam. Energy-aware software: Challenges, opportunities and strategies. *Journal of Computational Science*, 4(6):444 – 449, 2013. Scalable Algorithms for Large-Scale Systems Workshop (ScalA2011), Supercomputing 2011.

[36] C. Wohlin, P. Runeson, M. Hst, M. C. Ohlsson, B. Regnell, and A. Wessln. *Experimentation in Software Engineering*. Springer Publishing Company, Incorporated, 2012.

[37] L. Xu and S. Brinkkemper. Concepts of product software. *European Journal of Information Systems*, 16(5):531–541, 2007.

[38] G. Zhang, K. Zhang, X. Zhu, M. Chen, C. Xu, and Y. Shao. Modeling and analyzing method for cps software architecture energy consumption. *Journal of Software*, 8(11), 2013.

[39] H. Zhu, C. Lin, and Y. Liu. A programming model for sustainable software. In *Software Engineering (ICSE), 2015 IEEE/ACM 37th IEEE International Conference on*, volume 1, pages 767–777, May 2015.