Information and Software Technology 95 (2018) 165–176

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

The hunt for the guzzler: Architecture-based energy profiling using stubs

Erik Jagroep[*,a,b], Arjan van der Ent[b], Jan Martijn E.M. van der Werf[a], Jurriaan Hage[a], Leen Blom[b], Rob van Vliet[b], Sjaak Brinkkemper[a]

[a] Department of Information and Computing Science, Utrecht University, P.O. Box 80.089, Utrecht, TB 3508, The Netherlands
[b] Centric Netherlands B.V., P.O. Box 338, Gouda, AH 2800, The Netherlands

ARTICLE INFO

Keywords:
Software architecture
Energy consumption
Sustainability
Software stubs
Energy profiling

ABSTRACT

Context: Software producing organizations have the ability to address the energy impact of their software products through their source code and software architecture. In spite of that, the focus often remains on hardware aspects, which limits the contribution of software towards energy efficient ICT solutions.
Objective: No methods exist to provide software architects information about the energy consumption of the different components in their software product. The objective of this paper is to bring software producing organizations in control of this qualitative aspect of their software.
Method: To achieve the objective, we developed the StEP Method to systematically investigate the effects of software units through the use of software stubs in relation to energy concerns. To evaluate the proposed method, an experiment involving three different versions of a commercial software product has been conducted. In the experiment, two versions of a software product were stubbed according to stakeholder concerns and stressed according to a test case, whilst energy consumption measurements were performed. The method provided guidance for the experiment and all activities were documented for future purposes.
Results: Comparing energy consumption differences across versions unraveled the energy consumption related to the products' core functionality. Using the energy profile, stakeholders could identify the major energy consuming elements and prioritize software engineering efforts to maximize impact.
Conclusions: We introduce the StEP Method and demonstrate its applicability in an industrial setting. The method identified energy hotspots and thereby improved the control stakeholders have over the sustainability of a software product. Despite promising results, several concerns are identified that require further attention to improve the method. For instance, we recommend the investigation of software operation data to determine, and possibly automatically create, stubs.

1. Introduction

Software is increasingly being acknowledged as the driver behind the energy consumption of ICT solutions [1]. A striking example is the impact a single app can have on a smart-phone. Poorly written software can drain the battery, despite the increased energy efficiency of hardware components in the device. This effect on energy consumption also holds for larger applications, e.g., software products [2], where software can eliminate any energy efficient features built into the hardware [3]. As the software can account for 66% of the power consumed by, e.g., servers [4], this has a significant impact on the energy costs of the hardware. Unfortunately, where mobile apps are commonly engineered with energy consumption in mind, with larger applications hardware investments remain the preferred option to reduce energy consumption. This limits the contribution of the software towards the energy efficiency of a system.

On the other hand we find software qualities to increasingly become a decisive factor with respect to software products [5]. If a software product is not energy efficient, data center operating costs will increase the total cost of ownership and thereby diminish the economic viability of the product. Equally, if a software product does not perform as required, a functionally equivalent and better performing product is often quickly found. In light of these examples, implying trade-offs have to be made between qualities, we consider sustainability as a property of software quality with a social, environmental, technical and economic dimension [6]. Energy related concerns for the software, as discussed in this paper, can be attributed to the environmental dimension of sustainability.

Recent studies, e.g., [7–9], show the significant impact green software can have on energy consumption. For a Software Producing

* Corresponding author.
E-mail address: e.a.jagroep@uu.nl (E. Jagroep).

https://doi.org/10.1016/j.infsof.2017.12.003
Received 1 May 2017; Received in revised form 3 December 2017; Accepted 4 December 2017
Available online 06 December 2017
0950-5849/

Organization (SPO) [10], e.g., independent software vendors or open source consortia, applying green software changes requires them to measure the energy consumption of their software products: a relatively unfamiliar terrain. In practice, performance is often used as a proxy despite the fact that this aspect does not always have a direct relation with energy consumption [11]. As a result, SPOs are unable to control the contribution of software to the energy consumption of their ICT solutions. One way to obtain insights in energy consumption would be by performing measurements on the software in production. However, applying adjustments to the software in this stage of the lifecycle is often most expensive [12]. Consequently, energy efficiency of software should be studied in the earlier stages of software development.

In this paper, we study energy consumption of software in more detail. Based on the Energy Consumption Perspective (ECP) [13], we propose the Stubbed Energy Profiling Method (StEP Method) that enables SPOs to create an in-depth energy profile of their software products. The method builds upon substituting functional elements by stubs, which can be done during development, to study the change in energy consumption. The delta between the initial and stubbed version explains the energy consumption of the stubbed functional element. In this way, an SPO can investigate the software product in total, as well as of individual important constituents like functional modules and individual services.

To validate our method, we applied the method in an experiment using a commercial software product. Together with the software architect and product owner, we determined the important architectural elements in relation to the core functionality, and created an energy profile that identified the most prominent energy consumers in the software. Stakeholders of the product now know what software should be targeted to maximize the impact of sustainability efforts. Additionally, the energy profile of the software supports the architect to make better informed trade-off analysis with respect to sustainability and other qualitative aspects of the software.

The remainder of this paper is organized as follows. Section 2 provides background on the ECP and introduces our energy profiling method. In Section 3 we describe the design of our experiment where we apply the proposed method, followed by the experiment results in Section 4. In Section 5 we discuss the results and reflect on the method itself. Threats to validity and related work are presented in Sections 6 and 7. Section 8 concludes the paper.

2. Stubbed energy profiling method

In this section we present the StEP Method which stakeholders can use to create an energy profile of their software. We first explain the application of software stubs, followed by the activities comprising the proposed method.

2.1. Relating software stubs to energy concerns

An essential step in enabling the ICT industry to reduce the energy consumption of their software, is to address this qualitative aspect in the Software Architecture (SA). For this purpose, we introduced a quality perspective on energy consumption, i.e., the Energy Consumption Perspective (ECP) [13]. The ECP provides a collection of activities, tactics and guidelines stakeholders can use to ensure that a system exhibits a particular set of related quality properties [14]. However, although the introduced perspective includes a method to measure and relate energy consumption to the SA, we found that creating a detailed energy profile of a software product remains challenging.

An energy profile portrays the energy consumption of software elements related to an energy requirement; a condition or capability needed to solve a problem or achieve an objective [5]. Stakeholders of a software product, e.g., software architects and developers, use the energy profile to determine what changes can be applied to meet the requirement and where they can be applied. Hence, an energy profile is an essential instrument in addressing the energy consumption of software.

In relation to software products, we found that energy consumption is often investigated through differences between releases [7,15] or at runtime [16–18]. Such approaches are used to determine energy consumption changes in hindsight, e.g., after a development sprint has finished and the software has been deployed. However, at this stage there is often little room for addressing any remaining concerns compared to, e.g., the design or development stage. From a practical point of view, addressing concerns at a later stage in the development cycle negatively affects the economic dimension of sustainability [6].

To address qualitative concerns during software design requires a more predictive approach (e.g., [19]) and validated design decisions (e.g., [20]). However, as the actual energy consumption differs per configuration of both hardware and software [21], the end result of sustainable design efforts still requires verification. With the StEP Method we focus on analyzing a single version of a software product during development, to pinpoint the energy consuming elements within a single release. In this way, the proposed method can be seen as part of the ‘create energy profile’ activity in the ECP [13]. The StEP Method builds upon creating stubs for individual functional elements, for which guidance is provided in, e.g., [22]. The stubs are configured such that they return a standard response instead of executing the intended code, while ensuring the runtime capabilities of the release.

The main idea of the StEP Method is depicted in Fig. 1: for each architecturally important element, in this example *A, B*, and *C*, we create stubs *A′*, *B′* and *C′*, and perform tests on the resulting versions *S*1, *S*2, and *S*3. Comparing these versions with the Benchmark (i.e., the non-stubbed original) provides insights in the (the effects on) energy consumption caused by the stubbed element. For example, the difference between the Benchmark and *S*1 provides insight in the energy consumption effects of element *A*. Depending on the scope of the requirement and the test to be performed, more detailed stubs can be introduced, e.g., by individually stubbing sub-elements B1 and B2 or even stubbing individual lines of code.

2.2. Step method activities

As applying the StEP Method requires additional investments in terms of development and performing energy consumption measurements, careful planning is required to ensure the costs do not outweigh expected benefits. For example, depending on the elements under study, stubs could become complex and costly to apply. To minimize investments, the StEP Method approach could be incorporated in planned software tests (e.g., module testing [22]), thereby checking functional soundness and energy consumption in a combined effort. The same holds for energy consumption measurements, where different measurement approaches can be applied depending on the equipment and resources available, e.g., [16,17,23]. Also, using a predictive model, like GreenOracle [19], could potentially reduce the time required for energy consumption measurements and the associated costs.

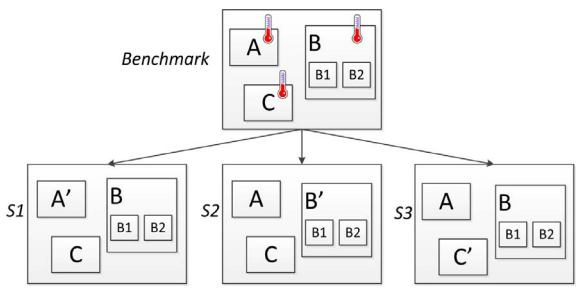

Fig. 1. By stubbing elements A, B and C the energy effect of each individual element can be determined.

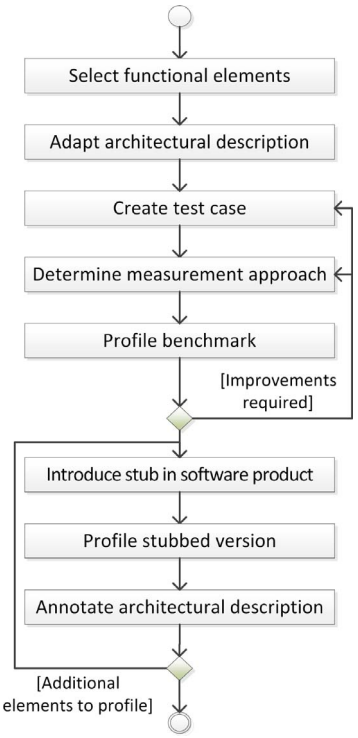

Fig. 2. The Stubbed Energy Profiling Method extending the 'Create energy profile' activity in the energy consumption perspective [13].

However, to accurately verify energy consumption changes, each product still requires product-specific measurements to be performed.

To enable stakeholders to successfully create an energy profile of their software product, we explicitly address these issues in the StEP method (Fig. 2). In the remainder of this section, we present the activities comprising the method.

Select functional elements. The first activity is to identify the target functionality and functional software elements, in relation to an energy requirement. Software operation data can be used to gain insight in frequently used functions and prioritize the elements that should be tested in relation to the requirement [24]. As the functional viewpoint is the most commonly created architectural description, this activity should be relatively easy to perform for many software products [14]. While the level of modularization of the software determines the control stakeholders can exert over the software, this aspect does not affect the ability to apply the method.

Adapt architectural description. After identifying the functional elements, the architectural description needs to be adapted to provide a sufficient level of detail on the functions comprising the target functionality. These details are used to identify where stubs are required and determine what stubs can be created for the different functions. We acknowledge that not all software elements can be stubbed individually, e.g., due to the inability to isolate the component or module under test. In such a case, the adapted architectural description should still identify the meaningful software units in relation to the requirement. In specific cases, like with software product lines, the software architect can potentially also identify what measurement results can be reused for each derived product.

Create test case. Based on the identified functional elements, a test case should be designed that executes these functional elements. While a stakeholder is free to design the test, realistic usage scenarios are found to best fit practitioners information needs when discussing energy usage [25]. To minimize the effort for performing this activity, readily available test cases (e.g., a regression test) can be expanded to include energy consumption measurements or automatic test case generation can be applied [26,27]. Stakeholders should ensure that the correct functional elements are stressed and the test can be consistently applied to profile each element.

Additionally, stakeholders should ensure the software is stressed such that differences can be observed across varying usage scenarios. The usage scenarios show the variation of resource utilization across different levels of stress and could provide valuable insights for, e.g., resource provisioning settings. To determine the usage scenarios, stakeholder can utilize software operation data to determine representative usage scenarios for the product under study. For example, tools like 'Application Insights' provide a range of monitoring and analysis options that could serve this purpose. However, while creating the test case, keep in mind that the load induced by the test case should allow stakeholders to reliably exclude incidental findings.

Determine measurement approach. Based on the test case, the appropriate metrics should be identified to quantify sustainability in relation to the target functionality and allow for evaluating whether requirements are met. In turn, the metrics, of which a solid list is available in literature [28], determine the measurement approach that a stakeholder should apply. Ideally, since different hardware could show different energy consuming behavior [21], different configurations are used to validate the energy profile. However, as this increases the costs, the minimal requirement is to have one environment that is representative for the production environment, where representative means using hardware and a software stack that closely resemble the target environment for the software. In Section 3 we provide a detailed description on how energy measurements were performed in the experiment comprising our research.

Profile benchmark. The next activity is creating the benchmark to evaluate the stubbed versions of the software against. Hence, the test case is performed on the non-stubbed version of the software. Since this is the first time that the test case is performed, the results from this activity could require that improvements are made to the test case and measurement approach.

Introduce stub in software product. If the benchmark is profiled and the test case and measurement approach are found suitable, the first functional element can be stubbed. Inserting the stub in the original software results in a new software version, S'. Keep in mind that the stubs themselves also consume energy, which depends on their build quality. However, as stubs provide simplified standard responses, an increase in energy consumption implies that either a bug is found or the quality of the stub should be improved. Performance measurements can be used to further verify the quality of the stubs and assure there is minimal impact on other parts of the product.

Profile stubbed version. Once version S' with the stub is obtained, the test case can be performed on it, resulting in a new set of measurements.

Annotate architectural description The final activity is to annotate the architectural description with the energy consumption effects caused by the stub. By comparing the newly obtained measurements with the benchmark, the difference in energy consumption can be assigned to the stubbed element. Apart from summarizing the experiment results, the resulting annotated description can be used to facilitate sustainability discussions among stakeholders of a software product. Keep in mind that the experiment results should be available to clarify the annotated architectural description when required.

The final three activities of the StEP Method are repeated until all required functional elements, as identified in the adapted architectural description, are profiled.

3. Research design

To demonstrate the StEP Method, we performed an experiment using a commercial software product. According to Wohlin et al. [29], experiments are launched when control over a situation is desired and behavior is subject to manipulation. In our study, control over the measurement approach is required to validly determine the energy

effect of stubs, whereas the StEP Method guides systematic manipulation of software behavior using stubs. To create the energy profile we aim to determine 'how' stubs affect energy consumption, which potentially helps understand 'why' specific code consumes a particular amount of energy [30].

In this section we present the design of the experiment, which follows the guidelines presented in [29–31] and is structured according to the activities comprising the StEP Method. Overall the experiment follows the design science research cycle [32], where the StEP Method is the central artifact that is field tested and adds to the existing knowledge base.

3.1. Product under study: City explorer

City Explorer (CITEX)[1] is a commercial software product that enables visitors of a city to explore Points Of Interest (POIs), such as historical buildings, through predefined routes. The routes are created by route managers, which are the actual CITEX customers, and made available to visitors via a smartphone app. Additional information is provided through the app with each Point Of Interest (POI), as well as an augmented reality experience using, e.g., historical pictures. The product is used by seven route managers, mostly municipalities and tourist information offices, and serves over 8000 visitors on an annual basis with peaks of 500 daily visitors during special events. CITEX is developed and maintained by a large Dutch SPO, with over 5000 employees and a portfolio containing approximately 80 commercial software products.

The SA of CITEX is depicted in Fig. 3. The application consists of three layers: Data Access, Business Logic and User Interface. The User Interface Layer contains two major elements: the WebAPI and Portal, that provide an interface for the app and a management portal for route managers, respectively. A SQL Server instance is used to store route information. The use of third party components is limited to (i) a Routeplanner, that provides the routes to follow based on GPS locations, and (ii) multiple information hosts (e.g., websites) providing the information concerning the POIs. All third party components are accessed directly by the app running on a visitors' smartphone.

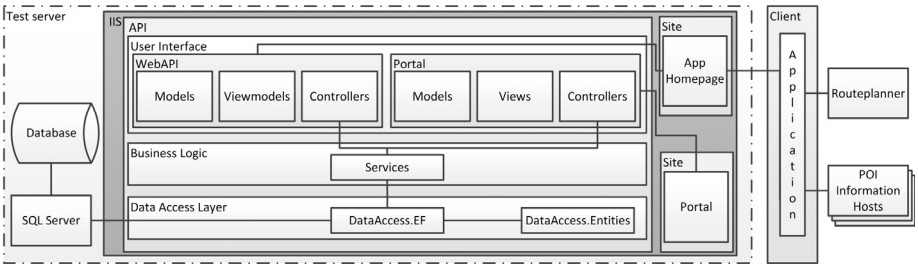

Fig. 3. The software architecture of City Explorer (CITEX).

3.2. Select functional elements and adapt architectural description

Together with the software architect of CITEX, we formulated the requirement for CITEX to increase the energy efficiency while executing its core functionality: providing routes to visitors. For this requirement, the focus was put on the back-end of the app (i.e., the 'Test server' in Fig. 3) as this is main energy consuming hardware under control of the SPO. Following the StEP Method, a second architectural description was created with the architect to obtain detailed insight into the core functionality (Fig. 4). This description allowed us to break down the core functionality into three separate activities:

A obtaining a route including route information,
B obtaining the POIs,
C obtaining the information related to POIs.

Each activity was mapped on the corresponding functional element in Fig. 4, which provided guidance for applying the stubs.

Ideally, individual stubs would have been applied for the 'Route', 'POI' and 'INFO' elements to determine their energy impact. However, investigating the code learned that there is a dependency between the elements that would not allow such an approach. Within CITEX it appeared that the POIs, and corresponding information, are hard linked to a route in the database. Hence, stubbing the 'Route' element also prevents CITEX from validly obtaining POIs. Similarly, the dependency between POIs and POI information prevented us from stubbing the 'POI' element individually without breaking the associated functionality of the 'INFO' element. Consequently, we were only able to apply two stubs to CITEX in our experiment: (1) stub the POI information (NoInfo) and (2) stub both POIs and POI information (NoPOI_NoInfo).

As the 'RouteController', 'GetRoutes', 'Entities' and 'Context' elements do not contribute to our energy requirement, these were not stubbed. The remaining elements, i.e., 'POI Information Hosts', 'Routeplanner' and 'Portal', were considered out of scope. A summary of the CITEX versions included in the experiment is provided in Table 1. In the remainder of this paper we will refer to each version through their respective labels, which summarizes the adjustment made to that specific version. To get an impression of the database impact of the stubs, the SQL Server database will be monitored separately.

3.3. Creating a test case

For the experiment, in line with the requirement, we chose to stress CITEX with its core task, i.e., providing routes to visitors. As no test case was readily available, a test case was designed where all CITEX versions were stressed by simulating three different usage scenarios: 1000, 3000 and 10,000 visitors that each follow three different routes for three times. The varying number of visitors per scenario allowed us to investigate the software behavior with different levels of stress intensity, whereas the number of routes per user corresponds to the average number of routes walked by a visitor based on historical data.

Initial testing showed that our usage scenarios had a significant impact on the software. This allowed us to allocate changes in resource utilization to our test case and exclude incidental findings. Additionally, our initial tests showed a limitation with respect to the number of simultaneous requests that could be processed. To prevent any errors from occurring, the test case was adjusted to steadily increased the number of simultaneously active users at a rate of three additional users per second.

The actual test case was designed using Postman[2] and Apache JMeter[3] to respectively obtain and simulate the calls made by a client. Using batch scripts we were able to automate JMeter tests, thereby

[1] A sample of the CITEX content is available via http://www.xplregouda.nl after selecting a language preference in the menu. Last accessed on 25-08-2017.

[2] https://www.getpostman.com/.

[3] http://jmeter.apache.org/.

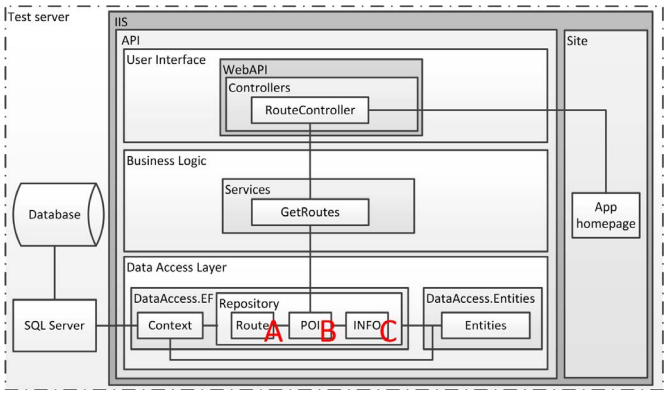

Fig. 4. The adapted architectural description of CITEX for applying the StEP Method.

Table 1
The CITEX releases included in the experiment.

Version	Version label	Description
1	Benchmark	Production version of CITEX without any stubbed elements.
2	NoInfo	Version based on the Benchmark where the 'Info' element is stubbed. The code to obtain POI information is not executed.
3	NoPOI_NoInfo	Version based on the Benchmark where the 'POI' and 'Info' elements are stubbed. The code to obtain POIs and associated information is not executed.

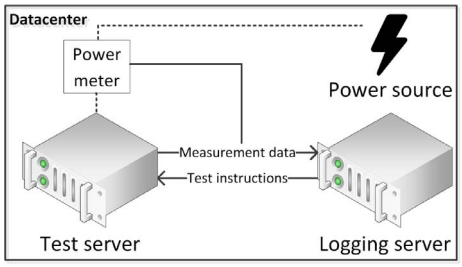

Fig. 5. Experiment environment.

ensuring no human interference would pollute our measurements and consistency across executions. To obtain a reliable data set, we set out to obtain 30 valid test case executions per CITEX version. To ascertain validity, all CITEX instructions were logged which allowed us to monitor for errors.

3.4. Determining the measurement approach

For the experiment a setup was created resembling a commercial setting (Fig. 5), which could be characterized as an off-line situation [29]. The software was deployed on a test server [4] that included the 'API', 'App Homepage', 'SQL server' and 'Database' elements as portrayed in Fig. 3. A second server, the logging server, was included to remotely collect measurement data (e.g., performance data) and minimize the impact of logging activities on our measurements. In addition, the logging server was used to perform the test case on the test server, thereby functioning as the Client device. To ensure consistency with regard to external factors, e.g., room temperature, the server was installed in an operational data center. All CITEX versions used the same data set for the experiment, which was a copy of the production database. The role of the power meter is explained in Section 3.4.1.

3.4.1. Power consumption and performance measurements

Following the methods described in [13] and [15], we applied a software- and hardware-based approach to measure the energy consumed by the software. For the software-based approach Microsoft Joulemeter (JM) was used: a software tool that estimates the total power (in Watt) consumed by a system and individual processes based on the required computational resources [16]. After calibration using WattsUp? Pro-(WUP), JM allows us to monitor energy consumption changes on both system and process level. Process level measurements were performed on the SQL Server, Homepage and API processes respectively. The first represents the 'SQL Server' and 'Database', whereas the latter two are related to Internet Information Services (IIS) and provide insight in the 'App Homepage' and 'API' (Fig. 3).

The hardware-based approach was applied using a WUP device[5], i.e., the power meter in Fig. 5, to more accurately determine the power consumption on system level [33]. By determining the average %-difference between JM and WUP, we are able to correct the JM estimations [33], which positively contributes to the quality of our data set. As WUP does not provide measurements at process level, the estimations at process level are reported as provided by JM. Both JM and WUP operate with a one second interval between measurements.

The performance of the test server was measured using Windows Performance monitor, a standard tool that is part of the Windows operating system. Following [21], the relevant performance metrics were determined with the product owner:

- CPU: % processor time
- Memory: private working set
- Hard disk: % disk time, % disk idle time
- Network: bytes sent/sec, bytes received/sec

The performance measurements were collected remotely on the logging server to minimize the overhead of the data collection process. Similar to the power measurement tools, the performance measurements are logged with measurement interval of one second. To ensure accuracy across measurement tools, i.e., be able to link events across sources on specific times, we synchronized the clocks across systems and devices using the Network Time Protocol (NTP).

3.4.2. Software energy consumption

To determine the energy consumed by the product under study, i.e., the Software Energy Consumption (SEC) [13], we subtract the idle power consumption from the power consumption as measured during an execution. The difference is considered to be the power consumed by CITEX, provided that no other applications were active during an execution. The idle power consumption, i.e., the baseline, is determined by performing power consumption measurements while the test server is idle, i.e., running without any active software. To obtain the SEC, power measurements need to be converted into energy consumption. Hence, we multiply the average power between two consecutive measurements with the time between measurements, i.e., one second, and sum up the results for the duration of each separate execution. We report our findings in Watt (W) or Joule (J) where applicable.

The SEC also forms the basis to determine the energy consumed by the individual architectural elements using the stubs. We are able to determine the SEC for each element as follows:

- 'Route': $SEC_{Route} = SEC_{NoPOI,NoInfo}$

By stubbing the 'POI' element, and thereby disabling the 'Info'

[4] HP Proliant DL380 G5, Intel Xeon E5335 CPU (4 cores @ 2 GHz), 8GB PC2-5300, 300GB hard disk (15.000 rpm), 64-bit MS Windows Server 2008R2, Service Pack 1, .Net Framework 4.5, IIS7, SQL Server 2016.

[5] https://www.wattsupmeters.com/ last visited 20th April 2017.

element, the total SEC for the NoPOI_NoInfo version effectively measures the SEC of the element that is not stubbed, being 'Route'.

- 'INFO': $SEC_{INFO} = SEC_{Benchmark} - SEC_{NoInfo}$

Deducting the SEC of the NoInfo version from the Benchmark results in the SEC for the 'INFO' element.

- 'POI': $SEC_{POI} = SEC_{Benchmark} - SEC_{Route} - SEC_{INFO}$

Deducting the energy consumption associated with the Route and INFO elements (i.e., SEC_{Route} and SEC_{INFO}) provides the SEC of the POI element.

In our approach, we assume that any increase in power consumption is caused by the activities performed by the software. To minimize overhead, we stopped services and processes not related to or required for CITEX, such as the Windows update service. Additionally we determined the cooldown time [15] for our test server; the time after which uncontrolled services and processes become inactive after a reboot. In our case we could start an execution eight minutes after rebooting.

3.4.3. Measurement protocol

To actually perform the measurements and ensure the validity of the Energy Consumption (EC) measurements, a protocol was followed to perform the test case:

1. Restart the test server.
2. Close unnecessary applications, services and processes.
3. Remain idle for the duration of the cooldown time.
4. Start WUP, performance and JM measurements.
5. Start test and wait for test to finish.
6. Collect data.

Each execution resulted in a data set containing a performance log, three JM logs for the SQL Server, Homepage and API processes (including measurements of the entire system), a WUP log, and CITEX log files. To validate the protocol, multiple test executions were performed which involved a mid-execution check for unexpected issues. In these specific execution no issues occurred, leading us to conclude that the test case was being performed as designed.

3.5. Introducing stubs in CITEX

The activity of introducing the required stubs in the software was performed by a software developer in collaboration with the CITEX team. The quality of every stub was verified before performing the test case, which included deployment of the stubbed versions on the test server.

4. Results

In this section we report on the execution and results of our experiment for all three versions. This corresponds to the remaining activities of the StEP Method, i.e., 'Profile benchmark', 'Profile stubbed version' and 'Annotate architectural description', resulting in the energy profile for CITEX.

4.1. Profiling software

By following the protocol as described in the previous section, we managed to obtain the required number of valid executions for each CITEX version. Despite consistency in performing the test case, twelve executions across different versions were found to contain errors and thus excluded from further processing. Further investigation into these executions was performed to assure that the impact of the errors was limited to the twelve single executions. We found the number of errors to be small, e.g., 20 errors on a total of 90,000 activities, which were caused by incidental time-outs of modules in the application or third party components. Hence, we can assume our results to provide valid insights into the energy consumption differences between CITEX versions.

The results are summarized in Tables 2, 3 and 4 according to each usage scenario. The reported SEC is based on the total energy consumption measurements provided by the WUP device, whereas the measurements for the SQL Server and API processes are provided by JM. On average, an execution lasted for 5 min. 45 s (SD = 1 s) with 1000 visitors, 17 min and 16 s (SD = 1 s) with 3000 visitors and 57 min and 40 s (SD = 11 s) with 10,000 visitors.

4.1.1. Process level measurements

At process level, we noticed an unexpected lack of energy consumption by the Homepage process. Investigating the issue more closely, learned that this specific process only shows activity when the process itself is starting up. Additionally, it is very likely that the webserver created separate processes for calls, which could not be measured by JM. Hence, the Homepage process was excluded from further processing. The remaining energy consumption figures show clear differences between CITEX versions.

In addition to energy consumption estimates per process, JM also provides estimates for the total energy consumed by a system. In line with previous experience, i.e., [33], we found that JM overestimates the idle energy consumption compared to WUP and underestimates the energy consumed while processing a varying load, i.e., during an execution. As a result, we found that JM estimations for total energy consumption cannot cope with subtracting the baseline energy consumption. This activity in some cases even resulted in a negative SEC.

In contrast to process level estimations, the JM total energy consumption estimations can be corrected using WUP measurements. For example, the baseline for the test server was determined to be 136.3 W using WUP and 136.95 W using JM. After correction with the average difference, i.e., -0.48%, a corrected JM baseline of 136.3 W was found. Similarly, a correction of 2.67% was applied to the total energy consumption figures during an execution.

With these corrections, the corrected SEC based on JM estimations was obtained as follows:

$$SEC = EC_{Total} - EC_{corrected,baseline} + (d \cdot EC_{Total})$$

First, the correction is applied to the compensate for underestimating the energy consumed while processing a varying load [33]. This correction is found by multiplying d, i.e., the load correction, with the total energy consumption for an execution. Second, the baseline energy consumption is subtracted from the estimated energy consumed during an execution. Finally, the correction is added resulting in the energy consumption on behalf of the software being active. The results are presented in Fig. 6, showing the differences between the SEC calculated according to WUP (SEC WUP), the corrected JM estimations (SEC corrected) and the uncorrected JM estimations (SEC JM).

Based on the large differences found between measurements sources, in the remainder of this paper we will use WUP measurements to report total energy consumption and JM estimations to report process level energy consumption. Keep in mind that we were not able to triangulate, and thus correct, process level estimations with the available equipment and tools.

4.1.2. Performance

The results of the performance measurements provide interesting findings per usage scenario. In line with our expectations, we found a decreasing CPU utilization with decreased software activity with all scenarios. However, the opposite was found for hard disk utilization with the 1000 and 3000 visitors scenarios. Even more noticeable is that the average hard disk utilization percentage with these two scenarios, is

Table 2
The results for each CITEX version with 1000 simulated visitors.

Version	SEC (J)	SQL Server (J)	API (J)	CPU (%)	Memory (MB)	HDD (%)	Network received (MB)	Network sent (MB)
Benchmark	1938.86	237.94	455.48	28.39	1066.93	3.78	0.05	0.55
NoInfo	887.32	51.81	191.75	11.26	1183.79	3.53	0.04	0.27
NoPOI_NoInfo	696.65	27.07	121.88	7.67	1156.04	5.36	0.04	0.25

Table 3
The results for each CITEX version with 3000 simulated visitors.

Version	SEC (J)	SQL Server (J)	API (J)	CPU (%)	Memory (MB)	HDD (%)	Network received (MB)	Network sent (MB)
Benchmark	5188.96	405.63	1323.38	24.14	1158.93	4.37	0.05	0.55
NoInfo	2978.89	255.43	576.51	12.67	1153.73	3.52	0.04	0.27
NoPOI_NoInfo	2016.30	96.19	378.67	8.21	1181.67	6.48	0.04	0.25

Table 4
The results for each CITEX version with 10,000 simulated visitors.

Version	SEC (J)	SQL Server (J)	API (J)	CPU (%)	Memory (MB)	HDD (%)	Network received (MB)	Network sent (MB)
Benchmark	19786.74	2428.28	4596.39	28.43	1090.55	3.85	0.05	0.55
NoInfo	8710.28	653.25	1979.10	11.71	1114.08	3.34	0.04	0.27
NoPOI_NoInfo	5983.41	297.12	1348.18	7.85	1096.14	2.85	0.04	0.24

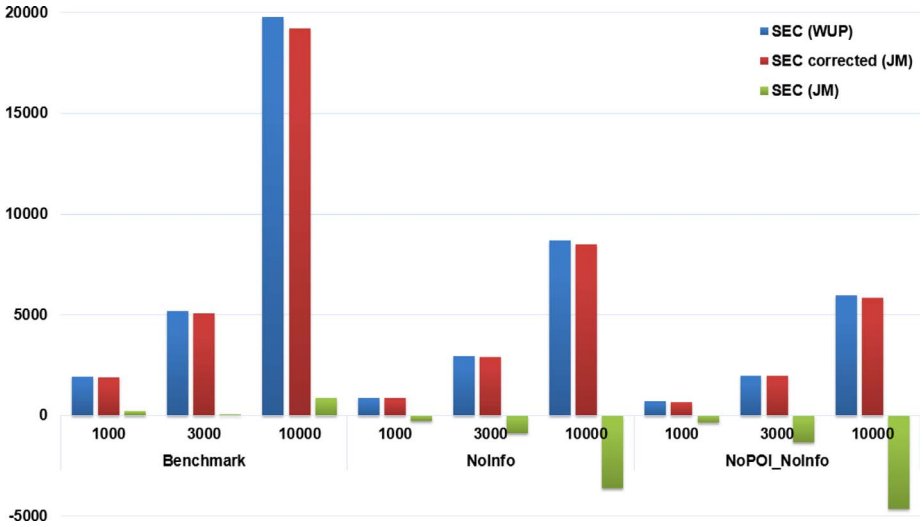

Fig. 6. The SEC (in Joule) for each usage scenario and version according to different measurement sources.

greater compared to the 10,000 visitors scenario. We did not find any discrepancies in our data that could explain this finding, although this finding might be an indication of cashing performed by the database.

Notice that we report one metric in relation to hard disk utilization, whereas two metrics were monitored. While the "% disk time" metric seems a logical metric to indicate hard disk activity, this metric was found to exaggerate the disk utilization[6]. Instead, we subtracted the "% idle time" from the maximum 100% load, and proceeded to calculate the average hard disk utilization.

With respect to the memory utilization and network throughput, fairly consistent figures were obtained across versions and usage scenarios. Memory utilization averaged 1135 MB, 1164 MB and 1096 MB, whereas the variation with network throughput is found due to rounding the measurements. More specifically, we obtained 0.0246 MB, 0.0247 MB and 0.0243 MB for respectively the 1000, 3000 and 10,000 visitors scenarios. As we did expect larger network throughput differences between usage scenarios, our presumption is that the bulk of data sent and received was in relation to the test case itself and not on behalf of CITEX.

[6] https://technet.microsoft.com/en-us/library/cc938959.aspx.

4.2. Annotate architectural description

The final activity of the StEP Method is to annotate the adapted architectural description with the SEC for the elements under study. We performed this activity separately for each usage scenario and excerpts of the annotated architectural descriptions are depicted in Figs. 7, 8 and 9.

To calculate the SEC per element, we followed the procedure explained in Section 3.4.2. Taking the 10,000 visitors scenario as an example (i.e., Table 4 and Fig. 9), the SEC for the "Route" element equals the SEC found with the NoPOI_noInfo version of CITEX, i.e., 5983.41 J. For the SEC of the "INFO" element we subtract the SEC of the NoInfo version from the Benchmark, resulting in an energy consumption of 11076.46 J. Finally, to obtain the SEC for the "POI" element we subtract the SEC of the "Route" and "Info" elements from the benchmark, providing a SEC of 2726.87 J.

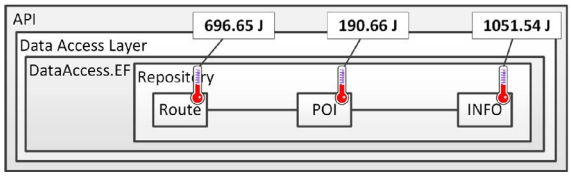

Fig. 7. The energy consumption per software element in the 1000 visitors usage scenario.

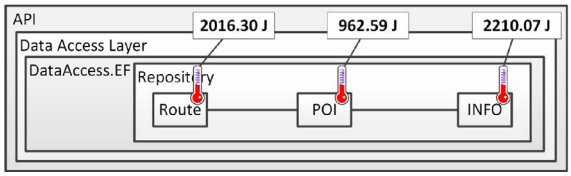

Fig. 8. The energy consumption per software element in the 3000 visitors usage scenario.

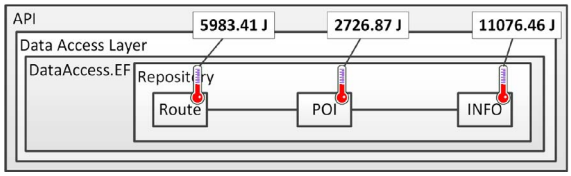

Fig. 9. The energy consumption per software element in the 10,000 visitors usage scenario.

5. Discussion

In this section, we discuss the main results and findings of our experiment in which we applied the proposed StEP Method, followed by a reflection on the method itself.

5.1. Energy consumption of CITEX

Recall that our test case was designed to simulate 1000, 3000 and 10,000 visitors that obtain three routes, i.e., the core functionality, for three times. The energy profile for this test case shows that obtaining the POI information is the most energy consuming activity related to this functionality, accounting for at least 42.59% of the SEC (left side of Table 5). This result suggest that if an SPO wants to reduce the SEC of CITEX, the most prominent code to address is related to obtaining POI information, i.e., the source code comprising the "INFO" element. The performance measurements presented in the previous section support this finding through the significant decrease of CPU utilization with the NoInfo version.

A different approach is to consider the energy consumption related to the actual activities that are performed, i.e., the Task Energy Consumption (TEC) [13]. The TEC provides an energy consumption for each individual time a task has been performed. For example, in the 10,000 visitors scenarios CITEX obtained 540,000 POI information items, 570,000 POIs and 90,000 routes per execution. Dividing the SEC of each element by these numbers, learns that CITEX consumes 0.021 J per POI information item, 0.005 J per POI and 0.066 J per route. A finding that is stable across usage scenarios. Hence, analyzing the TEC suggests that the "Route" element should be the first object for optimization. In the calculation of the TEC, note that the discrepancy between POI information items and POIs is caused by a so-called waypoint POI: a POI that does not contain information and is only used to ensure a visitor follows the prescribed route.

Different analysis methods provide CITEX stakeholders with multiple approaches to address the formulated energy requirement. Product stakeholders are free to decide what analysis best supports their vision related to the energy concern. The activities of determining what adjustments should be made and validating their effect, are beyond the StEP Method and are included in the ECP.

5.1.1. Process level energy consumption

Continuing the analysis on process level ('SQL Server' and 'API' in Table 5), we find results that are for a large part consistent with the findings based on the total SEC. Except for the 3000 visitors usage scenario, both the SQL Server and API processes indicate that obtaining POI information is the most energy consuming activity. As no differences were found in terms of performance or energy consumption, the exception with the SQL Server process was unexpected. Potentially a turning point was reached with the number of POIs where the database had trouble finding its most efficient processing speed. Still, the analysis on process level does not provide any reason to reconsider the analysis methods presented above.

5.1.2. Energy consumption scaling

A test case with multiple usage scenarios also provides the opportunity to investigate the scaling of resource utilization, including energy consumption, with different load intensities. With the transition from 1000 to 3000 visitors the SEC of the Benchmark version increases with a factor of 2.68, and again with a factor of 3.81 from 3000 to 10,000 visitors. These factors indicate that the available hardware is capable of more efficiently handling the first load increase, i.e., 3 times as much visitors at the cost of 2.68 times more resources, whereas the second increase appears inefficient. This translates to 1.94 J, 1.73 J and 1.98 J SEC per visitor for respectively the 1000, 3000 and 10,000 visitors scenarios. While these findings are not likely to affect any software-based efforts towards fulfilling the energy requirements, the results contain a potential lesson for load balancing the existing hardware. With respect to the latter, a more detailed analysis on scaling could be performed to determine the energy consumption 'near sweet-spot' [34].

5.2. Stubbing software for SEC measurements

Applying a stub and thereby providing a standard response for a specific software element, is already commonly used to enable SPOs to test specific functionality early on in the development cycle [22]. The stub allows the software to process requests as intended, without requiring the (incomplete) code to be fully executed. Hence, while stubbing software is not new, its application in relation to energy consumption had not yet been made explicit.

In relating software stubs to energy consumption we were inspired by genome research, where disabling a gene to investigate its effects is a commonly applied method. With the StEP Method, we translated this approach to the SA domain and focus on the effects caused by the disabled software elements. Without requiring deep knowledge on the

Table 5
The SEC percentages for each software element based on the total and process level energy consumption measurements.

	SEC			SQL Server			API		
Element	1000	3000	10000	1000	3000	10000	1000	3000	10000
INFO	54.24%	42.59%	55.98%	78.23%	37.03%	73.10%	57.90%	56.44%	56.94%
POI	9.83%	18.55%	13.78%	10.40%	39.26%	14.67%	15.34%	14.95%	13.73%
Route	35.93%	38.86%	30.24%	11.38%	23.71%	12.24%	26.76%	28.61%	29.33%

code and internal structure comprising CITEX, the StEP Method allowed us to quantify qualitative aspects of the product and provide input for future design decisions and trade-offs.

Based on our experiment, we identify two aspects that should be taken into consideration by an SPO in applying the StEP Method: the stakeholders in relation to a requirement and the resources (e.g., time) available to apply the StEP Method. These aspects determine the context in which the StEP Method is to be applied and thereby the preparations required to successfully apply the method. For example, a software architect is likely to investigate the target functionality on the level of major architectural elements. As such, an energy profile would be used to compare between releases, on an environment that resembles the production environment. Additionally, the test case can be set on a large scale to simulate longer periods of operational time of the software. In this context, the same measurement approach can be applied as described in this paper. The resulting energy profile is not likely to include measurements on the level of individual software elements.

On the other hand, a developer is more likely to investigate the target functionality on code level and wants to know the exact functions that are performed in relation to the target functionality. With developers, the energy profile is created during development using the local system. As using the system during a load test will pollute measurements, the test case should be quick to perform while still producing observable discrepancies between the stubbed versions and the benchmark. A more predictive approach (e.g., [19]) could prove more valuable in such contexts, although the actual effects of any adjustments should still be verified by profiling the software. Concerning the measurement approach, several tools exist [35–37] that are able to replicate our profiling approach on a smaller scale. However, keep the accuracy of these tools in mind during selection.

5.3. Reflections on applying the step method

Although the experiment provided valuable insights for CITEX, there are limitations to our proposed profiling method.

5.3.1. Scalability of the step method

One of the main limitations of the StEP Method is the effort required to apply the method. In our experiment, the test case encompassed three usage scenarios and two stubs were applied to CITEX. Including the benchmark, a total of nine different configurations were required to create the energy profile related to the energy requirement. This is a significant effort for SPOs, which potentially prevents them from applying the method. In addition, the results obtained by applying the StEP Method are focused on a single application thus requiring the method to be repeated for each software product.

With respect to the effort of applying the StEP Method, our experiment demonstrates one of the most extensive applications by investigating relatively large architectural elements. As explained, the StEP Method can also be applied on a smaller scale (e.g., lines of code) and focused on a specific software module instead of the entire product. Similar to other qualitative requirements [5], stakeholders should consider what level of investment is required to analyze the qualitative aspects of a product. Furthermore, instead of separate usage scenarios, a test case can be designed to simulate a varying load (e.g., a varying number of visitors) for a software product. This reduces the number of configurations required for a test case.

On repeating the method for different software products, we acknowledge that general guidelines and best practices can be distilled from one product that are relevant for others. However, the effects of applying these lessons learned should be checked with each application. Different implementations potentially impact the energy consumed by the software, thus negating the possibility to generalize findings across products. In addition, different hardware configurations can significantly impact the effect of a specific implementation [21]. Hence, each product would still require its own analysis when energy concerns are addressed at code level.

5.3.2. Step method activities

The activities comprising the StEP Method proved sufficient to systematize energy consumption analysis with CITEX and guide the stakeholders through the process of analysis. A short discussion on this with the CITEX software architect, learned that the activities provided sufficient insight into the process and made clear what was expected from the CITEX team during the experiment. As a result, most activities could be performed fairly easily supported by, e.g., architectural descriptions being readily available. One activity that was a concern at first was determining the measurement approach as the team had no prior experience with software energy consumption analysis. However, this concern was dispelled after a short explanation of the topic.

5.3.3. Complex environments

The StEP Method was applied in a relatively simple test environment, representative for a production deployment of CITEX. In our view, applying the method in a more complex, e.g., virtualized, environment does not impact the method, provided that reliable SEC measurements can be performed. However, in general, as additional elements are present, more effort will be required to control these elements to obtain valid measurements. For example, a stakeholder should know, or even influence, how the load is distributed over the available resources. Hence, with increased complexity we expect an increase with respect to the costs of applying the StEP Method.

Our advice is to apply the method in a test environment that is representative for the production environment and maintain focus on optimizing the software. Once optimized and deployed, e.g., on a cloud based platform, the focus of optimization shifts to the synergy between software and hardware. At this stage, power saving features in relation to networking activity [38] and the resource allocation algorithm [39] can be implemented to even further improve the sustainability of a software product.

5.3.4. Stubs and stub effects.

A limitation in relation to stubs is the inability to measure the energy effect of a stub itself. Consequently, the energy consumed by a stub is included in the measurements of the different stubbed versions of a software product. Additionally, despite the quality of the stubs that are applied, modifying software could lead to unexpected changes in software behavior that potentially affect the measurements that are performed. Such a change could occur outside of the scope of the test case and thus go by unnoticed. Detailed performance measurements help to identify such occurrences, allowing the stakeholder to adjust the stub accordingly.

5.3.5. Energy overhead

A large part of the SEC, on average 66.77% with 10,000 visitors, could not be attributed to the CITEX processes monitored with JM. However, as we were not able to monitor the effects on OS level, we can only summarize this portion as overhead. Similar to CPU intensity and energy consumption, the overhead changes with each version and is not considered to have a linear relation with software activity. In general though, increased activity from an application also implies increased activity from the OS to coordinate tasks, as shown in Fig. 10. Further research is required to fully understand and control this effect.

5.3.6. Visualizing the energy profile

To visualize the results of the energy profiling method, we annotated the architectural description with the energy consumption figures for the respective elements (i.e., Figs. 7–9). However, especially when the energy profile is created for multiple small elements, an annotated architectural description could overwhelm stakeholders with too much information at once. Different visualization methods could be applied that better fit the needs of stakeholders, simplify the interpretation of

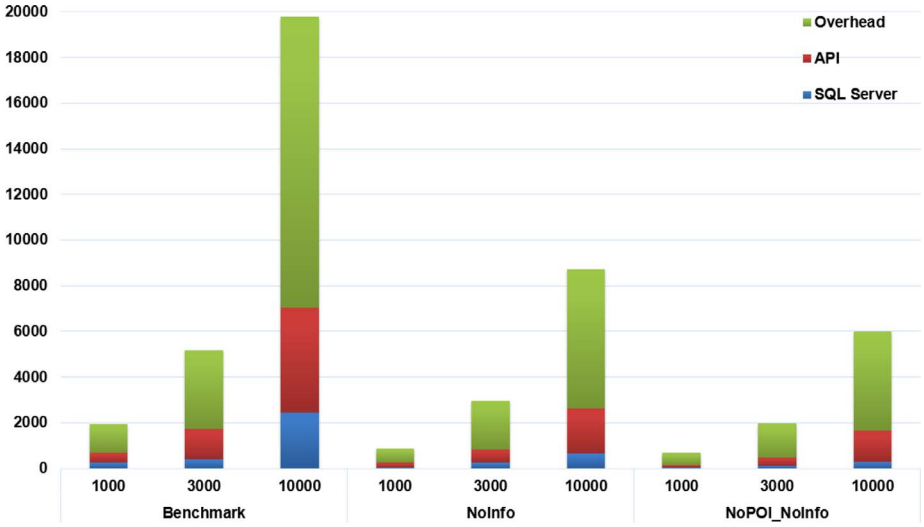

Fig. 10. The energy consumption overhead (in Joule) not explained by the SEC of the individual processes comprising CITEX.

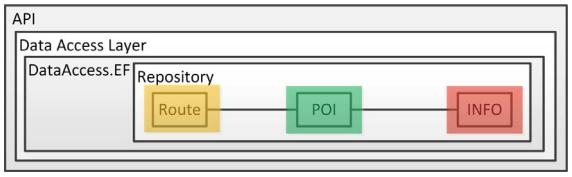

Fig. 11. A color scheme overlay to visualize the energy profile for CITEX. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

the measurements and thereby facilitate discussion on software energy consumption [25]. For example, a color coding scheme (Fig. 11) could be applied to indicate the 'hotspots' of the software. Additionally, the profile does not have to be limited to the SEC. Including, for example, performance measurements could provide a more clear message for the stakeholders.

6. Threats to validity

Following the discussion of the StEP Method itself in Section 5, this section presents the threats to internal, external and construct validity of our experiment, as required by [29,31,40].

6.1. Internal validity

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

Observer effect. While our experiment was designed to minimize the impact of performing the experiment on our results, we could not completely rule out the effects of JM and Windows Performance Monitor. We know JM uses a small amount of memory, adding to the performance measurements, which does not translate to observable energy consumption [15]. With respect to Windows Performance Monitor, we were not able to determine a potential effect on the energy consumption. However, as performance measurements were consistent across executions and these measurements represent a consistent part of the monitored network activity.

OS effects. Despite our efforts we cannot be fully certain that no OS processes became active during an execution of the load test. Configuration tools do not provide full control over, e.g., services and internal timers that might induce activity without direct input from a user. We analyzed our performance measurements in detail, i.e., for each process separately, and did not find unusual activity during the executions.

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 the SEC due to high-frequency energy components. However, this interval is also commonly used in the state of the art [7].

6.2. External validity

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

Experiment environment. Given the relation between power consumption and the hardware components, performing the experiment in a different environment could yield different measurement results. While the generalization of the study could be improved by diversifying the hardware involved in the experiment, obliging stakeholders to perform measurements on multiple hardware environments would significantly increase the required investments for applying the StEP Method. And even if such an effort is made, the generalizability of the results across hardware platforms is expected to be limited [21]. Hence, to obtain the best energy consumption measurements for CITEX, we chose to perform the experiment in an environment that closely resembles the actual production environment. Ideally, energy consumption measurements are performed on the target hardware, i.e., the hardware that the software will be executed on. However, involving the target hardware is not always be possible due to, e.g., limited control and access to such an environment. This is particularly the case when an external hosting party is involved.

Measurement equipment. As shown in Fig. 6, large differences exist with respect to the quality of the measurements between the tools we utilized. Given the diversity of power meters and software tools available to perform SEC measurements, there is an unavoidable dependency with respect to the accuracy and detail of the obtained measurements.

6.3. Construct validity

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

Sustainability and software quality. Central to performing measurements is to have a clear vision on the metrics that should be reported. In our research we quantify the environmental dimension of sustainability through the SEC, however, we acknowledge different metrics could have been applied. The measurements explained in the research design reflect the data required to calculate the SEC and were identified in collaboration with the CITEX architect.

Non-stubbed elements. In our experiment, we were not able to stub every element identified to the activities comprising the target functionality. Hence, our results assume the impact of the other elements to scale with the stubs applied in the NoInfo and NoPOI_NoInfo versions, and we deduced the SEC for the 'POI' element. Despite this constraint, we were able to quantify SEC differences, supported by performance measurements, and demonstrate our proposed StEP Method.

7. Related work

Measurements on the energy consumption of software products are difficult to perform accurately, and have proven to be time-consuming [41]. Each change in the software, e.g., code obfuscation [42], requires its own test to be designed, executed multiple times and the results analyzed afterwards. Although specialized equipment can be utilized, e.g., [43], such an approach is both difficult and expensive to apply in an industrial context.

The complexity of performing energy consumption measurements increases in cloud computing environments due to, e.g., virtualization techniques and load balancers. The additional layers between the software and hardware, make it difficult to allocate energy consumption of a distributed system to specific software activity [18]. Although these layers can also assist with identifying the main sources of energy consumption, e.g., [44], performance-based power models should be used with caution [45].

In comparison, developers of mobile software are able to estimate the SEC of their apps on their workstation through emulation tools [46]. However, as the energy impact of a software product scales with each installation, the efforts towards this aspect are worthwhile. For example, if four million users changed to a different, functionally equivalent web browser, the monthly energy consumption of an American household could be saved each hour [7].

Our approach of using stubs to analyze software is not new. For example, in gray-box testing [22], stubs are used to study the effects of certain components in isolation. The software architecture can be used to define stubs, as for example has been done in [47] to test a flight software product line. However, in our approach, we invert the idea of gray boxing: comparing the overall system with the system where some functional element has been substituted by a stub, provides information about the isolated functional element, rather than the effect on the complete system.

In this way, our approach may support feature slicing for hypothesis based software development [48]. This approach may assist architects in studying different design alternatives, also called design diversity [49,50]. With our proposed method, different alternatives and tactics can be compared to obtain a catalogue of green tactics [20].

Additionally, considering energy consumption as a first class citizen allows for trade-off analyses, e.g., with respect to reliability [51]. Traditional software architecture evaluation methods, such as ATAM [52] mainly rely on expert opinions. Relating test-based measurements with the software architecture, allows to create better, objective insights for software architects, supporting the longetivity of the architecture [53]. Compared to more recent methods, e.g., KLAPER [54], test-based measurements do not require the construction and maintenance of expensive performance models to evaluate alternative system designs [55].

8. Conclusions

In this paper we propose the Stubbed Energy Profiling Method to determine the energy consumption of software (i.e., SEC) on the level of architectural elements, and applied this method in an experiment performed using a commercial software product. The method is intended as a detailed elaboration of how to create an energy profile, as required for applying the Energy Consumption Perspective [13]. The StEP Method is based on creating multiple versions of a software product, each including a stubbed software element, which are then compared to a non-stubbed benchmark. **Quantifying the differences between stubbed and non-stubbed software versions provides the SEC of the stubbed elements, which creates the energy profile for the software**.

In total, three different versions of CITEX were included in the experiment, which allowed us to thoroughly investigate the energy implications related to the core functionality of the product. Next to the non-stubbed benchmark, two stubbed versions were created that enabled the SPO to create an energy profile of CITEX and identify the major energy consuming architectural elements while performing its core functionality. The level of detail also helped to identify the source code developers should target to address energy related concerns. Using different analysis methods, we were able to **identify multiple energy hotspots in the software**. With CITEX we identified the element that consumed the most energy in absolute terms and the element was the most energy consuming in relation to the number of tasks it performed.

An SPO that applies the StEP Method improves its control over the environmental sustainability aspect of their software products. Specifically, the SPO gains the ability to target the most energy consuming elements of its software products. Additionally, as the method can be applied during development, energy related concerns can be addressed when the costs of doing so are lowest.

For future work we look into refining the StEP Method and the SEC measurements. For example, to further validate the method, the method should be applied on different levels of granularity (e.g., individual lines of code) and on different hardware platforms. In addition, further investigation is required into, among others, automating software tests to minimize the required investments for applying the method. Second, while the approach relies on creating stubs, it is not always possible to create stubs for a functional element. By recording sufficient software operation data for each functional element, we envision the automatic creation of stubs that are more realistic, and thus provide better approximations in the analysis. Furthermore, in our experiment the overhead on average accounted for 66.77% of the SEC. We look to performing an in-depth analysis of the overhead as this is a significant contributor to the SEC of a software product.

Acknowledgments

The authors would like to thank Teodora Colac for her help on performing the experiment with CITEX and the anonymous IST reviewers for their valuable and constructive feedback to improve this paper.

References

[1] P. Lago, R. Kazman, N. Meyer, M. Morisio, H.A. Müller, F. Paulisch, G. Scanniello, B. Penzenstadler, O. Zimmermann, Exploring initial challenges for green software engineering: summary of the first GREENS workshop, at ICSE 2012, ACM SIGSOFT Softw. Eng. Notes 38 (1) (2013) 31–33.
[2] L. Xu, S. Brinkkemper, Concepts of product software, Eur. J. Inf. Syst. 16 (5) (2007) 531–541.
[3] B. Steigerwald, A. Agrawal, Green software, Harnessing Green It, John Wiley & Sons, Ltd, 2012, pp. 39–62, http://dx.doi.org/10.1002/9781118305393.ch3.
[4] J. Beckett, R. Bradfield, Power efficiency comparison of enterprise-class blade servers and enclosures, Technical Report, Dell, 2011. URL http://www.dell.com/downloads/global/products/pedge/en/BladePowerStudyWhitePaper_08112010_final.pdf .
[5] L. Bass, P. Clements, R. Kazman, Software Architecture in Practice, SEI Series in Software Engineering, Pearson Education, 2012.
[6] P. Lago, S.A. Koçak, I. Crnkovic, B. Penzenstadler, Framing sustainability as a property of software quality, Commun. ACM 58 (10) (2015) 70–78.
[7] A. Hindle, Green mining: a methodology of relating software change and configuration to power consumption, Empirical Softw. Eng. (2013) 1–36.
[8] G. Procaccianti, H. Fernàndez, P. Lago, Empirical evaluation of two best practices for energy-efficient software development, J. Syst. Softw. 117 (2016) 185–198.
[9] E. Jagroep, G. Procaccianti, J.M. van der Werf, S. Brinkkemper, L. Blom, R. van Vliet, Energy efficiency on the product roadmap: an empirical study across releases of a software product, J. Softw. 29 (2) (2017) e1852–n/a, http://dx.doi.org/10.

1002/smr.1852.
[10] S. Jansen, S. Brinkkemper, J. Souer, L. Luinenburg, Shades of gray: opening up a software producing organization with the open software enterprise model, J. Syst. Softw. 85 (7) (2012) 1495–1510.
[11] A.E. Trefethen, J. Thiyagalingam, Energy-aware software: challenges, opportunities and strategies, J. Comput. Sci. 4 (6) (2013) 444–449.
[12] S. Naumann, M. Dick, E. Kern, T. Johann, The greensoft model: a reference model for green and sustainable software and its engineering, Sustainable Comput. 1 (4) (2011) 294–304, http://dx.doi.org/10.1016/j.suscom.2011.06.004.
[13] E. Jagroep, J.M.E.M. van der Werf, S. Brinkkemper, L. Blom, R. van Vliet, Extending software architecture views with an energy consumption perspective, Computing (2016) 1–21.
[14] N. Rozanski, E. Woods, Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives, Addison-Wesley, 2011.
[15] E.A. Jagroep, J.M.E.M. van der Werf, S. Brinkkemper, G. Procaccianti, P. Lago, L. Blom, R. van Vliet, Software energy profiling: comparing releases of a software product, ICSE Companion '16, ACM Press, 2016, pp. 523–532.
[16] A. Kansal, F. Zhao, J. Liu, N. Kothari, A.A. Bhattacharya, Virtual machine power metering and provisioning, Proceedings of the 1st ACM Symposium on Cloud Computing, SoCC '10, ACM, New York, NY, USA, 2010, pp. 39–50, http://dx.doi.org/10.1145/1807128.1807136.
[17] C. Seo, G. Edwards, S. Malek, N. Medvidovic, A framework for estimating the impact of a distributed software system's architectural style on its energy consumption, Seventh Working IEEE/IFIP Conference on Software Architecture (WICSA 2008), (2008), pp. 277–280, http://dx.doi.org/10.1109/WICSA.2008.28.
[18] C. Seo, S. Malek, N. Medvidovic, Component-level energy consumption estimation for distributed java-based software systems, in: M.R.V. Chaudron, C. Szyperski, R. Reussner (Eds.), Component-Based Software Engineering: 11th International Symposium, CBSE 2008, Karlsruhe, Germany, October 14–17, 2008. Proceedings, Springer Berlin Heidelberg, Berlin, Heidelberg, 2008, pp. 97–113, , http://dx.doi.org/10.1007/978-3-540-87891-9_7.
[19] S.A. Chowdhury, A. Hindle, Greenoracle: estimating software energy consumption with energy measurement corpora, Proceedings of the 13th International Conference on Mining Software Repositories, MSR '16, ACM, New York, NY, USA, 2016, pp. 49–60, http://dx.doi.org/10.1145/2901739.2901763.
[20] G. Procaccianti, P. Lago, G.A. Lewis, A catalogue of green architectural tactics for the cloud, MESOCA 2014, IEEE Computer Society, 2014, pp. 29–36.
[21] E. Jagroep, J.M.E.M. van der Werf, J. Broekman, S. Brinkkemper, L. Blom, R. van Vliet, A resource utilization score for software energy consumption, Proceedings of the 4th International Conference ICT for Sustainability, Advances in CSR, Atlantis Press, 2016, pp. 19–28.
[22] G.J. Myers, C. Sandler, T. Badgett, The Art of Software Testing, John Wiley & Sons, 2011.
[23] A. Noureddine, R. Rouvoy, L. Seinturier, Monitoring energy hotspots in software, Autom. Softw. Eng. (2015) 1–42.
[24] J. van der Werf, C.v. Schuppen, S. Brinkkemper, S. Jansen, P. Boon, G. van der Plas, Architectural intelligence: a framework and application to e-learning, EMMSAD, 2017.
[25] I. Manotas, C. Bird, R. Zhang, D. Shepherd, C. Jaspan, C. Sadowski, L. Pollock, J. Clause, An empirical study of practitioners' perspectives on green software engineering, ICSE '16, ACM Press, 2016, pp. 237–248.
[26] F. Palomba, A. Panichella, A. Zaidman, R. Oliveto, A. De Lucia, Automatic test case generation: what if test code quality matters? Proceedings of the 25th International Symposium on Software Testing and Analysis, ISSTA 2016, ACM, New York, NY, USA, 2016, pp. 130–141, http://dx.doi.org/10.1145/2931037.2931057.
[27] V. Tarasov, H. Tan, M. Ismail, A. Adlemo, M. Johansson, Application of Inference Rules to a Software Requirements Ontology to Generate Software Test Cases, Springer International Publishing, Cham, pp. 82–94. doi: 10.1007/978-3-319-54627-8_7.
[28] P. Bozzelli, Q. Gu, P. Lago, A systematic literature review on green software metrics, Technical Report, Technical Report: VU University Amsterdam, 2013.
[29] C. Wohlin, P. Runeson, M. Hst, M.C. Ohlsson, B. Regnell, A. Wessln, Experimentation in Software Engineering, Springer, 2012.
[30] R. Yin, Case Study Research: Design and Methods, Applied Social Research Methods, SAGE Publications, 2009.
[31] N. Juristo, A.M. Moreno, Basics of Software Engineering Experimentation, first ed., Springer, 2010.
[32] A. Hevner, S. Chatterjee, Design Science Research in Information Systems, Springer US, Boston, MA, pp. 9–22. doi: 10.1007/978-1-4419-5653-8_2.
[33] E. Jagroep, J.M.E.M. van der Werf, S. Jansen, M. Ferreira, J. Visser, Profiling energy profilers, SAC '15, ACM Press, 2015, pp. 2198–2203.
[34] G. Kalaitzoglou, M. Bruntink, J. Visser, A practical model for evaluating the energy efficiency of software applications, Proceedings of the 2nd International Conference on ICT for Sustainability, Atlantis Press, 2014, pp. 77–86.
[35] T. Do, S. Rawshdeh, W. Shi, ptop: A process-level power profiling tool, Proceedings of the 2nd Workshop on Power Aware Computing and Systems (HotPower09), (2009).
[36] N. Amsel, B. Tomlinson, Green tracker: a tool for estimating the energy consumption of software, CHI '10 Extended Abstracts on Human Factors in Computing Systems, ACM Press, 2010, pp. 3337–3342.
[37] A. Noureddine, R. Rouvoy, L. Seinturier, A review of energy measurement approaches, SIGOPS Oper. Syst. Rev. 47 (3) (2013) 42–49.
[38] X. Chen, C. Phillips, An evolutionary based dynamic energy management framework for ip-over-dwdm networks, Sustainable Comput. 4 (2) (2014) 94–105. Special Issue on Selected papers from EE-LSDS2013 Conference. doi: 10.1016/j.suscom.2014.03.001 .
[39] A. Beloglazov, J. Abawajy, R. Buyya, Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing, Future Gener. Comput. Syst. 28 (5) (2012) 755–768. Special Section: Energy efficiency in large-scale distributed systems. doi: 10.1016/j.future.2011.04.017 .
[40] P. Runeson, M. Höst, Guidelines for conducting and reporting case study research in software engineering, Empirical Softw.Eng. 14 (2) (2009) 131–164.
[41] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R.P. Dick, Z.M. Mao, L. Yang, Accurate online power estimation and automatic battery behavior based power model generation for smartphones, CODES/ISSS '10, ACM Press, 2010, pp. 105–114.
[42] C. Sahin, M. Wan, P. Tornquist, R. McKenna, Z. Pearson, W.G.J. Halfond, J. Clause, How does code obfuscation impact energy usage? J. Softw. 28 (7) (2016) 565–588. JSME-15-0021.R2. doi: 10.1002/smr.1762 .
[43] M.A. Ferreira, E. Hoekstra, B. Merkus, B. Visser, J. Visser, Seflab: a lab for measuring software energy footprints, GREENS, IEEE, 2013, pp. 30–37.
[44] A. Berl, E. Gelenbe, M. Di Girolamo, G. Giuliani, H. De Meer, M.Q. Dang, K. Pentikousis, Energy-efficient cloud computing, Comput J 53 (7) (2010) 1045, http://dx.doi.org/10.1093/comjnl/bxp080.
[45] J. Mair, D. Eyers, Z. Huang, H. Zhang, Myths in power estimation with performance monitoring counters, Sustainable Comput. 4 (2) (2014) 83–93. Special Issue on Selected papers from EE-LSDS2013 Conference. doi: 10.1016/j.suscom.2014.03.007 .
[46] R. Mittal, A. Kansal, R. Chandra, Empowering developers to estimate app energy consumption, Mobicom '12, ACM Press, 2012, pp. 317–328.
[47] D. Ganesan, M. Lindvall, D. McComas, M. Bartholomew, S. Slegel, B. Medina, Architecture-based unit testing of the flight software product line, SPLC 2010, LNCS 6287 Springer, 2010, pp. 256–270.
[48] H.H. Olsson, J. Bosch, From requirements to continuous re-prioritization of hypotheses, International Workshop on Continuous Software Evolution and Delivery, ACM, 2016, pp. 63–69.
[49] B. Littlewood, P. Popov, L. Strigini, Modeling software design diversity: a review, ACM Comput. Surv. 33 (2) (2001) 177–208.
[50] D. Sobhy, R. Bahsoon, L. Minku, R. Kazman, Diversifying software architecture for sustainability: a value-based perspective, ECSA 2016, LNCS 9839 Springer, 2016, pp. 55–63.
[51] I. Meedeniya, B. Buhnova, A. Aleti, L. Grunske, Architecture-driven reliability and energy optimization for complex embedded systems, in: G.T. Heineman, J. Kofron, F. Plasil (Eds.), Research into Practice–Reality and Gaps: 6th International Conference on the Quality of Software Architectures, QoSA 2010, Prague, Czech Republic, June 23, - 25, 2010. Proceedings, Springer Berlin Heidelberg, Berlin, Heidelberg, 2010, pp. 52–67, , http://dx.doi.org/10.1007/978-3-642-13821-8_6.
[52] R. Kazman, M. Klein, M. Barbacci, T. Longstaff, The architecture tradeoff analysis method, 4th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS) 1998, IEEE, 1998, pp. 68–78.
[53] H. Koziolek, D. Domis, T. Goldschmidt, P. Vorst, Measuring architecture sustainability, IEEE Softw. 30 (6) (2013) 54–62.
[54] V. Grassi, R. Mirandola, E. Randazzo, A. Sabetta, Klaper: an intermediate language for model-driven predictive analysis of performance and reliability, in: A. Rausch, R. Reussner, R. Mirandola, F. Plášil (Eds.), The Common Component Modeling Example: Comparing Software Component Models, Springer Berlin Heidelberg, Berlin, Heidelberg, 2008, pp. 327–356, , http://dx.doi.org/10.1007/978-3-540-85289-6_13.
[55] S. Kounev, F. Brosig, N. Huber, R. Reussner, Towards self-aware performance and resource management in modern service-oriented systems, 2010 IEEE International Conference on Services Computing, (2010), pp. 621–624, http://dx.doi.org/10.1109/SCC.2010.94.