Requirements Eng (2017) 22:27–40
DOI 10.1007/s00766-015-0234-2

ORIGINAL ARTICLE

Green software requirements and measurement: random decision forests-based software energy consumption profiling

Mohamed Amine Beghoura[1] · **Abdelhak Boubetra**[2] · **Abdallah Boukerram**[1]

Received: 13 December 2014 / Accepted: 14 July 2015 / Published online: 26 July 2015

Abstract This paper proposes an explicit definition of green software requirements and a tool to support their evaluation. The proposed evaluation tool describes the green efficiency by considering the energy consumption as the main aspect to be studied during the development stage. This approach consists of building a multiple regression model, by using a supervised learning algorithm, in order to reproduce the energy consumption pattern of devices at different workload circumstances. The energy consumption model is then deployed to estimate the impact of software applications based on their resource usage. Our work has been validated on desktop and mobile devices. The experiments show the effectiveness of the proposed energy profiling tool that provided relevant information on the energy consumption of software applications.

Keywords Software engineering · Energy consumption model · Green software · Green requirements · Software development · Green software engineering · Random decision forests · Multiple regression model

✉ Mohamed Amine Beghoura
amine.beg@gmail.com

Abdelhak Boubetra
boubetraabd@yahoo.fr

Abdallah Boukerram
boukerram@hotmail.com

[1] Department of Computer Science, Faculty of Exact Sciences, University of Bejaia, 06000 Bejaia, Algeria

[2] Department of Computer Science, University of Bordj Bou Arréridj, 34000 Bordj Bou Arréridj, Algeria

1 Introduction

In the past decade, green IT practices emphasized the application of environmental constraints on hardware design to mitigate their direct negative effect [6]. However, in more recent years, the energy optimization techniques have gone beyond the hardware level to reach the software level. These optimization techniques aim at developing software which behaviour follows that of the devices' energy efficiency techniques. Software could be helpful to enhance the green efficiency of the information technology (IT) infrastructures [28], although it carries a direct rebound effect on the energy consumption of the involved IT infrastructures. Software behaviour plays a significant role in determining the energy consumption of the hardware [29] where an intensive usage of computation, communication, and data storage resources often lead to a greater negative impact [24], especially on mobile devices since they have limited battery life [15].

In this context, a green software can help to directly improve the energy efficiency of the IT hardware, where software energy efficiency is defined as the amount of energy required to complete a specific task per unit of time [29]. Moreover, Zhang and Hindle [29] argue that the energy efficiency benchmark should be user-centric and based on real applications in order to put pressure on developers to improve the energy efficiency of their applications. At this point, there are many state of the art programming techniques [9] and design guidelines to improve the energy efficiency. However, the software engineering process must be more concerned with the green issues related to the software.

The existing software development life cycle models, such as the waterfall and the spiral models, emphasize the development, the maintenance, and the business aspects of

the software. It is clear that these models do not explicitly treat the identification of the green constraints to express the impact of software on the environment [7]. To meet the importance of developing green software systems, developers need design tools with generalized characteristics and unified aspects to embody the green constraints explicitly in the development life cycle. These software development life cycle models need to be adapted to help with (1) designing and modelling, (2) analysing scenarios of long-term change, (3) and testing the software green efficiency. Therefore, the integration of green requirements into the software engineering processes is important [1, 2]. Furthermore, the analysis of these green constraints is a crucial task to evaluate the design and the implementation.

In this paper, we focus on the requirements specification and testing activities. We introduce a set of green requirements that concern the direct negative impact of software and that are specified as software quality. To analyse the proposed characteristic, we present a profiling tool to measure that quality and to evaluate the green efficiency of software. The measurement is done using a self-adaptive energy profiling tool that can be deployed on a variety of devices. It is built using a statistical approach that consists of establishing multiple regression models to fit the energy consumption with the device's resource usage under different workload variations. For each device, the energy consumption pattern can be saved and used later to investigate the energy consumption of software applications based on their workload data over the time. Moreover, to build the regression model, we rely on the random decision forests algorithm as a regression process method that makes effective the process of establishing the energy consumption model.

The remainder of this paper is organized as follows. In Sect. 2, we present a green macrospiral life cycle. In Sect. 3, we discuss the integration of the green constraints in the requirements specification. In Sect. 4, we present our methodology to build the profiling tool. The experiments are presented in Sect. 5. The related work is presented in Sect. 6. Finally, we conclude and discuss future directions in Sect. 7.

2 Green awareness

The software development life cycle involves several sequentially related activities to be followed by software engineering practitioners in order to develop the desired software, where each activity uses the results of the previous one. Reducing the negative ecological impact that can be caused by software is an important concept that needs to be integrated into the different life cycle activities. The green constraints fit into the category of non-functional requirements [20] that must be explicitly defined in the requirements specification activity and supported by the design and implementation of software. In addition, having a profiling tool is important to increase the developers' awareness about the behaviour of their software. This tool enables the measurement of the green requirements of the testing activity. Thus, a particular emphasis must be stressed on the requirements specification and consequently to the software green quality evaluation.

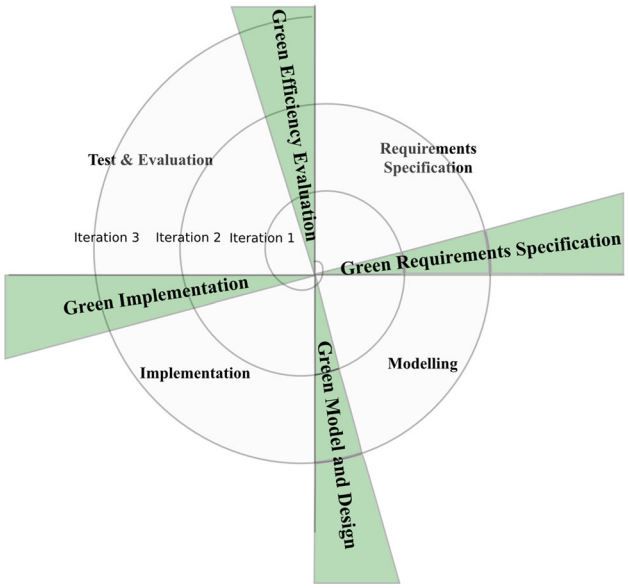

Fig. 1 Green macrospiral software development life cycle

In Fig. 1, a simple model is presented for the green software life cycle that includes the involvement of green constraints in the different activities.

The described process is a macrospiral development life cycle in which green constraints are applied as additional conditions that target the enhancement of the green aspects of the software in each phase of the macrospiral. The software prototype is evaluated to judge its efficiency as well as to acquire information on the system's components that can be greener. The assessment would suggest the plan for the next iteration. The proposed model represents an underlying software development process that consists of iteration series leading to an increase in the software green efficiency.

3 Green software requirements and metrics

One of the most useful software product quality models for requirements engineering is the ISO/IEC 25010:2011 standard.[1] The standard quality model is a revision of the

[1] ISO/IEC JTC 1/SC 7, ISO/IEC 25010:2011, Systems and software engineering—Systems and software quality requirements and evaluation (square)—System and software quality models, 2011.

Table 1 Green internal/external quality model

Characteristic	Subcharacteristics
Functional suitability	Functional completeness, functional correctness, functional appropriateness
Reliability	Maturity, availability, fault tolerance, recoverability
Usability	Appropriateness recognizability, learnability, operability, user error protection, user interface aesthetics, accessibility
Performance efficiency	Time behaviour, resource utilization, capacity
Maintainability	Modularity, reusability, analysability, modifiability, testability
Portability	Adaptability, installability, replaceability
Security	Confidentiality, integrity, non-repudiation, accountability, authenticity
Compatibility	Coexistence, interoperability
Green efficiency	Green computation efficiency, green data management, green data communication, energy consumption awareness

ISO/IEC 9126[2] (2001), which specifies the essential software quality attributes that are used to judge how software is intended to be. The ISO/IEC 25010:2011 quality model classifies the software constraints into a set of characteristics and associates their corresponding subcharacteristics to describe the behaviour of the software in development (internal/external) and usage context (in-use) stages. Furthermore, the standard affiliates the corresponding metrics to test and evaluate each of those attributes.

The standard quality model implicitly involves the green aspects of the software in the quality in-use characteristics division as freedom from risk characteristic. It defines the risk as the occurrence probability function of a given threat and the potential adverse consequences of the threat event. The freedom from risk is specified to mitigate the risk caused by the software to the economy, health, and environment. However, no further recommendations are provided to elicit the requirements of developing a green software nor to discriminate the green aspects of the software at the internal/external view.

The primary factors that determine the green aspects of the software as a product are related to the energy consumption [13], since it is the principal resource used by the hardware to run the software. To represent reducing the energy consumption as a requirement that will be considered at the development stage, we incorporate the green efficiency (Table 1) as an additional characteristic of the software quality model. The defined characteristic expresses the degree to which the software behaviour mitigates the energy consumption of the infrastructure. In other words, the specification of the green efficiency characteristic in the requirements specification activity aims to enhance the software behaviour towards the energy usage. Furthermore, analysing and evaluating the green efficiency at the internal/external views leads to a better understanding of green software behaviour.

In general, an intensive usage of resources such as the processor, network interfaces, and the storage components requires more energy than in an optimized usage. Moreover, some hardware components have the tendency to spend energy more than others. Therefore, developers must specify the green efficiency with more details to express which category of hardware components usage must be optimized using adequate energy optimization techniques (depending on the nature of their software). Sabharwal et al. [24] group the main green techniques to optimize the energy consumption of the implemented software into three sets: computation efficiency, data efficiency, and context awareness. Based on that categorization of the optimization techniques, we derive the corresponding subcharacteristics of the green efficiency characteristic. We propose four subcharacteristics: green computation efficiency, green data management, green data communication, and energy consumption awareness. The first three subcharacteristics overcome the green efficiency issues at the computation, communication, and storage levels of the software. The fourth subcharacteristic provides a descriptive behavioural energy consumption awareness chart. We define the corresponding subcharacteristics as:

- Green computation efficiency: expresses the ability of the software to process the demanded workload efficiently by consuming an optimal amount of energy.
- Green data management: expresses the effectiveness of the implemented data management strategies to perform I/O operations at low energy consumption.
- Green data communication: expresses the efficiency of energy management policies when the software sends and receives data over the communications network.
- Energy consumption awareness: describes the whole energy consumption of the software in order to determine the different consumption levels and to

[2] ISO/IEC JTC1/SC7, Software engineering product quality model, 2001.

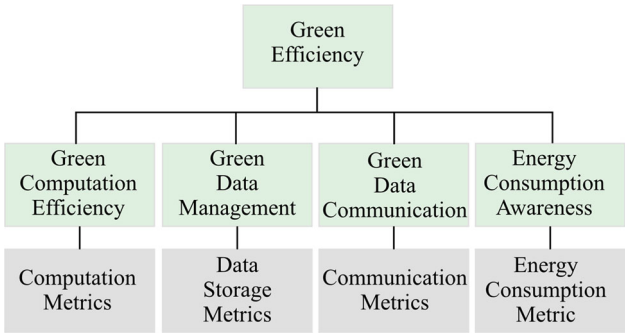

Fig. 2 Green requirements associated metrics

define the expected behaviour of the software at the peak, the average, and the low energy usage.

The effectiveness of the defined requirements varies depending on the nature of the software, the implementation techniques, the software design, and the configuration of the hardware used to run the software. In order to assess their effectiveness, a set of measures should be linked to the proposed requirements in the model of software quality. The goal is to quantify the extent to which these requirements are fulfilled in the implementation activity. We connect the green attributes with the corresponding metrics related to the hardware components responsible for the energy consumption when being used by the software. Figure 2 shows the associated metrics to the green subfeatures of the software.

Computation metrics: measurement counters related to the set $CP = \{cp_1, cp_2, \ldots, cp_z\}$ of processing units that are dedicated to perform the computation operations. Computation metrics are associated with the green computation efficiency subcharacteristic, which corresponds to the quantification of the energy consumed by the software when performing operations on the computing resources (e.g. CPU usage).

Data storage metrics: measurement counter related to the set $ST = \{st_1, st_2, \ldots, st_w\}$ of storage components used by the software to perform data storage operations on the local storage components. This category of metrics is associated with the green data management efficiency subcharacteristic in order to measure the amount of energy consumed when software is performing the operations of reading and writing on the storage devices (read/written bytes).

Communication metrics: measurement counters related to the set $CO = \{co_1, co_2, \ldots, co_m\}$ of communication components to count the number of operations or the amount of data that have been sent or received over the network interfaces.

Energy consumption metric: this metric answers the question "How much energy does a software consume?" The measurement indicates the energy consumption due to a software by relying on the hardware resources usage.

In addition to the computation, storage, and communication components, many external factors play a role in determining the pattern of the energy consumption in IT devices (e.g. the hardware configuration, the product design, the battery ageing factors). An adaptive energy consumption profiling tool has to be built to measure the energy consumption of software on different platforms at different conditions. It aims at providing an answer to the question: "How much energy does a software consume?" It must estimate the impact of a particular software application on the energy consumption of the device rather than determining the energy usage of the whole device (including the background process and the operating system). Furthermore, we believe that the energy model should also be extensible to include the diversity of the hardware components that may be available on the devices since the energy consumption depends on those factors.

4 Random decision forests for profiling the energy usage

The design of an energy profiling tool must satisfy a set of requirements. The first concern is estimating the energy usage of the software based on its resource utilization data. The estimation must be specific to the software by separating its impact from the background consumption caused by the other running software in a multitasking operating system (infrastructure energy overhead [26]). Secondly, it has to track the behaviour of the software components to evaluate their impact at different granularity levels (e.g. process, thread, routines). The third objective concerns the tool itself, in that it should be adaptive to fit the variety of the hardware configurations where the workload data can be rendered. It must also be usable on different platforms (PCs, smartphones, tablets) and reusable when the power measurements given by the an external instrumentation are no longer needed during the estimation process.

The methodology used to establish the estimation tool consists of four successive stages. These steps are performed to build an energy consumption model for each device in which the software will be tested and evaluated:

1. Classifying the components of the device into their corresponding category (computation, storage, communication) and defining the metrics to measure their usage.
2. Collecting the data related to the usage of components at the device level along with its corresponding energy consumption.

3. Establishing an energy model using a regression process to simulate the energy consumption pattern of the device.
4. Deploying the established regression model to estimate the energy consumption of a software based on its workload.

The first step consists of setting a list of hardware components with which the software will be involved (Table 2). With each component, we associate the corresponding metric that will be used to measure the workload data. In the second step, we estimate the processed workload and the energy consumption of the device at a fixed sampling rate. During the fourth step, the collected data are then used to build the regression model that emulates the energy consumption pattern of the device. Finally, the model is deployed to estimate the energy consumption of software based on their workload.

From a hardware perspective, the energy consumption of a device at the time t has the tendency to vary depending on the affected workload of the different components. Knowing the characteristics of the affected workload and the energy usage model of the device allows the prediction of the consumption related to a software. It has been shown in [8] that an energy model of a device can be represented by a function g that links the predictors $x_{t,1}, x_{t,2}, \ldots, x_{t,k}$, which correspond to resource usage data to the response of the model y_t (energy usage). The formulation of the function g is presented in Eq. 1.

$$y_t = g(x_{t,1}, x_{t,2}, \ldots, x_{t,k}) \quad (1)$$

In other words, having a set C_k of computation, storage, and communication components, where $C_k = CP \cup ST \cup CO$, the problem can be stated as finding a mapping function $g(v_t)$ that correlates the measured workload data (input data) $v_t = \{x_{t,1}, x_{t,2}, \ldots, x_{t,k}\}$ with the energy consumption (output data) y_t at a given time t.

Figure 3 illustrates the relationship between the input and the output data in addition to the role of the mapping function g. $C_j/(j = 1, 2, \ldots, k)$ is the hardware component j and $x_{i,j}$ is the sampled usage measurement of workload

Table 2 A descriptive table of energy consumption regression model's data (input/output)

	Components	Types	Metrics	Unites
Input	C1	Computation	Usage	Per cent
			Frequency	GHz
	C2	Storage	Read	Bytes
			Write	Bytes
	C3	Communication	Sending	Bytes
			Reception	Bytes
Output	C4	Energy	Usage	Joule or uA

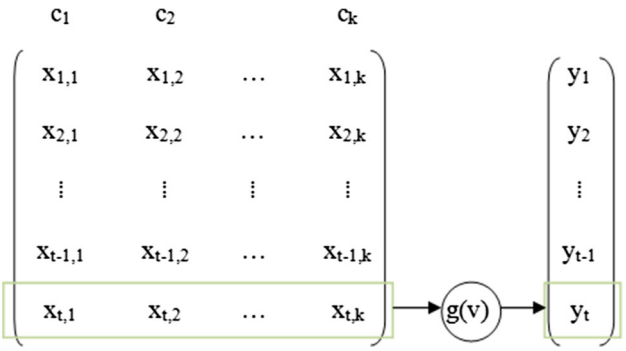

Fig. 3 Workload and energy consumption data

performed by C_j at time i. y_i is the measured value of energy consumption at time i related to component usage data of the components set C. In this context, y_t is designated as a dependent variable that has continuous values measured in joules (or watts) and that quantifies the energy consumption. The variables $x_{t,1}, x_{t,2}, \ldots, x_{t,k}$ are considered as a set of independent predictors that quantify the workload at components $c_1, c_2, \ldots, c_k$. Thus, an adequate way to find the mapping function $g(v_t)$ is to define the problem as a multiple regression problem where the resource usage are used to predict the energy consumption. Despite the relationship between the input and the output data (linear or nonlinear), our goal is to establish an easily adaptable and deployable tool that can be used in a wide variety of cases and devices. We use the machine learning techniques to solve the presented multiple regression problem.

In this work, we deploy the nonparametric random decision forests (RDFs) [3] technique as a general data mining algorithm to build the energy consumption regression model. The RDF method fits within the machine learning algorithms category. It is shown to be useful in several types of data mining tasks such as regression, classification, semiautomatic learning, and density estimation. The RDF method consists of a set of decision trees [22]. It was first introduced for the recognition of handwritten digit recognition and then was successfully deployed to solve many other machine learning problems. The main strength of this method is its accuracy when processing previously unseen data. The diversification generated by the randomization process improves the results obtained from the different models (Fig. 4). Randomness in the decision forests technique is presented in several forms, such as the random selection of the features [11] and the random selection of the training samples [3]. The difference between the trees offers more precision and efficiency in processing new data.

A decision tree is a predictive model of a set of nodes that form a graph without loops and uses a set of binary rules to calculate a target value. The nodes in the decision tree graph designate split test functions that are applied to

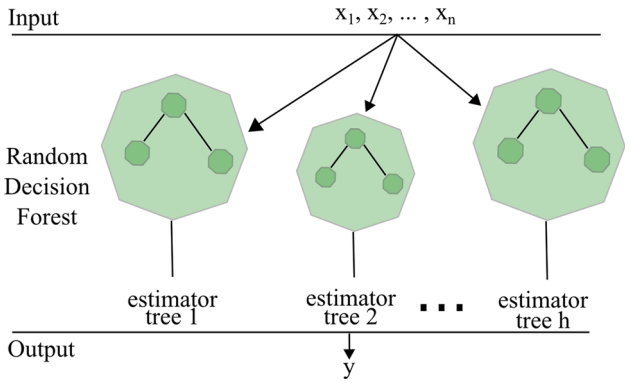

Fig. 4 Random decision forests process

the data. The leaves of the tree represent the decision or the class of the data processed. The processing of the decision trees is established in two stages:

1. The first stage is the learning stage. Parameters optimization of the tree nodes is performed using a set of training data that is constituted of labelled data. The learning algorithm determines the feature to be processed in each node, and the best split value of the node that optimizes a criterion (e.g. entropy). In our case, the criterion to be optimized is the mean-squared error function that is given in Eq. 2.
2. The second stage is the processing stage. In this phase, series of tests are applied to the input vector v, at each node, starting from the root of the tree. The tests are repeated until reaching a leaf node that determines the decision.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - y_i')^2 \quad (2)$$

After defining the input and output metrics and measurement units, the next step is to collect data that will be divided into two data sets. The training data set is used to build the regression model. The test data set is used to determine the optimal RDF algorithm parameters values that minimize the regression error and consequently increase the efficiency of the energy consumption estimator. In our case, the learning process of the RDF algorithm is performed to simulate the energy consumption pattern of the device in which the software will be tested. The learning process uses a data set that contains the workload measurements and the energy usage data. Table 2 shows an example of a general chart of data and their metrics that will compose the data set.

Here, the objective is to minimize the total error of prediction that is the squared difference between the real value y_i and the predicted value y_i' for n observations. Given a training set $d = (v_i, y_i)/i = 1, 2, \ldots, n$ of input vectors v_i representing the hardware components usage

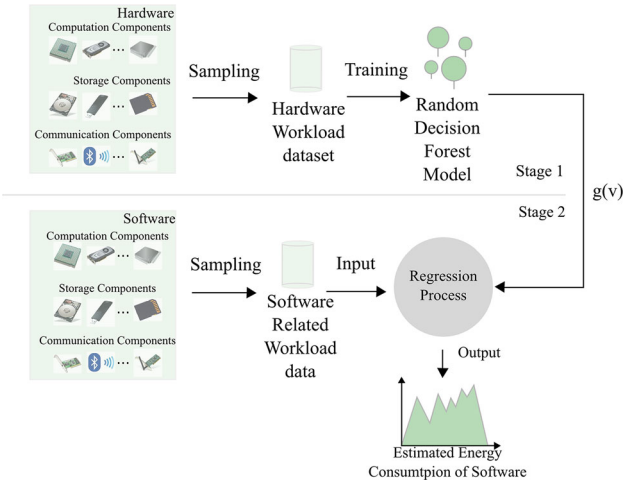

Fig. 5 A conceptual schema of the proposed energy profiling tool

data associated with their corresponding energy consumption y_i. The goal is to fit a function $g(v)$, which approximates the relation inherited between v_i and y_i. The established model that is represented by g will be used during the fourth step to infer the output y of a new input data related the tested software. In Fig. 5, we present the conceptual framework of the energy profiling tool.

5 Experimental study

5.1 Evaluation of mobile software applications

In this section, we present a case study to measure the green efficiency of mobile applications. The description of the mobile device's configuration is provided in Table 3.

The classification of the device components: first, we designate the hardware components that will be included in the testing and the metrics to measure their usage. In this case, we activate only the components defined in Table 4, and we disabled the rest of the components such as the 2G/3G network interfaces for the whole experiment.

Data collection: during the second step, we collected the workload measurement and the energy usage of the test device. The fulfilment of this task is done by a data collection programme that is executed on a desktop computer. The measured data are obtained using the Android Debug Bridge (ADB) tool and recorded by a Python script (Fig. 6). The resulting data set is saved in a comma-separated values file (*.CSV). The columns in the saved file correspond to the performed workload data resulting from each component listed in Table 4 at a fixed sampling time.

In this experiment, the data set is collected during 2 h of recording at a sampling rate of 1 s (7200 samples). The

Table 3 Experiment's mobile device characteristics

Characteristic	Type
Device name	Samsung Galaxy Tab 3 10.1 P5200
OS	Android OS v4.2.2 (Jelly Bean)
CPU	Dual-core 1.6 GHz
Memory	Internal 16/32 GB, 1 GB RAM
2G network	GSM 850/900/1800/1900
3G network	HSDPA 850/900/1900/2100
WLAN network	Wi-fi 802.11 a/b/g/n dual-band
Battery	Li-Po 6800 mAh battery

Table 4 Mobile platform regression model (input/output data)

	Components	Types	Metrics	Unites
Input	CPU	Computation	Usage	Per cent
			Frequency	GHz
	Memory	Storage	Read	Bytes
			Write	Bytes
	WLAN	Communication	Sending	Bytes
			Reception	Bytes
Output	Battery	Energy	Usage	uA

```
p = sub.Popen([
      "adb", "shell", "cat",
 "/proc/stat",
 "/sys/devices/system/cpu/cpu0/cpufreq/scaling_cur_freq",
 "/proc/meminfo",
       "/proc/net/dev",
        "/sys/class/power_supply/battery/batt_current_ua_now"
], stdout=sub.PIPE, stderr=sub.PIPE)

   output, errors = p.communicate()
```

Fig. 6 Data sampling script

collected data correspond to a diverse workload. Figure 7 shows a portion of the training data set.

Building the energy consumption model: Following the acquisition of the training data set, we calibrated the RDF algorithm to achieve a minimal value of the mean-squared error criterion, by using a test data set composed of 600 samples. Then, we study the influence of the number of estimators (decision trees) on the *MSE* given by the trained model. In Fig. 8, we present the *MSE* values obtained by different forests sizes.

The minimal *MSE* value is obtained by using 152 trees in the decision forests algorithm. However, the *MSE* value can be minimized further by studying the effect of combining the previous workload samples at time $t-p$ to estimate the consumption at time t where $p = \{0, 1, 2, \ldots, p\}$. Thus, the mapping function will take the following form presented in Eq. 3. The corresponding results are presented in Table 5.

$$g(v_t, v_{t-1}, v_{t-2}, \ldots, v_{t-p}) = y_t \tag{3}$$

The analysis of using the previous workload samples to estimate the actual energy usage at the time t shows an enhancement in the minimal *MSE* value. The best configuration is given by using three previous workload sample data. Therefore, the energy consumption at a time t on our device is relative to the current and three previous workloads. Figure 9 shows the difference between the estimated energy consumption and the real consumption as it was observed during the recording of the test data set.

Evaluating the energy consumption of software: first, we estimate the energy consumption of YouTube's (video-sharing website) mobile application that decodes the videos using the H.264 video encoder.[3] Figure 10 shows the resource usage and the estimated energy consumption data of the software while being used to play the same video in two different display resolutions (standard-definition 360p and high-definition 720p).

We notice from the estimation that the high-quality video consumes more energy than a standard-definition video. In other words, the operation of decoding a standard-definition resolution video is more green efficient (computation and communication) than decoding a high-definition resolution. Therefore, users should select the standard-definition quality on mobile devices to save energy (battery life).

In the second test, we measure the energy consumption of three Web browsers (Mozilla Firefox, Google Chrome, Opera's Opera Mini Web Browser). The browsers are tested during different tasks:

(a) Idle state.
(b) Connecting to the website *Y*.[4]
(c) Scrolling over the index page of *Y*.
(d) Connecting to *F*[5] account.
(e) Scrolling over the accounts home page *F*.
(f) Connecting to *G*.[6]
(g) Switching from page *Y* to page *F* then page *G*.
(h) Closing all tabs.

The total consumption per browser is presented in Table 6, which shows that the Mini Opera Browser consumes less

[3] http://www.itu.int/rec/T-REC-H.264.2/.
[4] www.youtube.com.
[5] www.facebook.com.
[6] www.google.com.

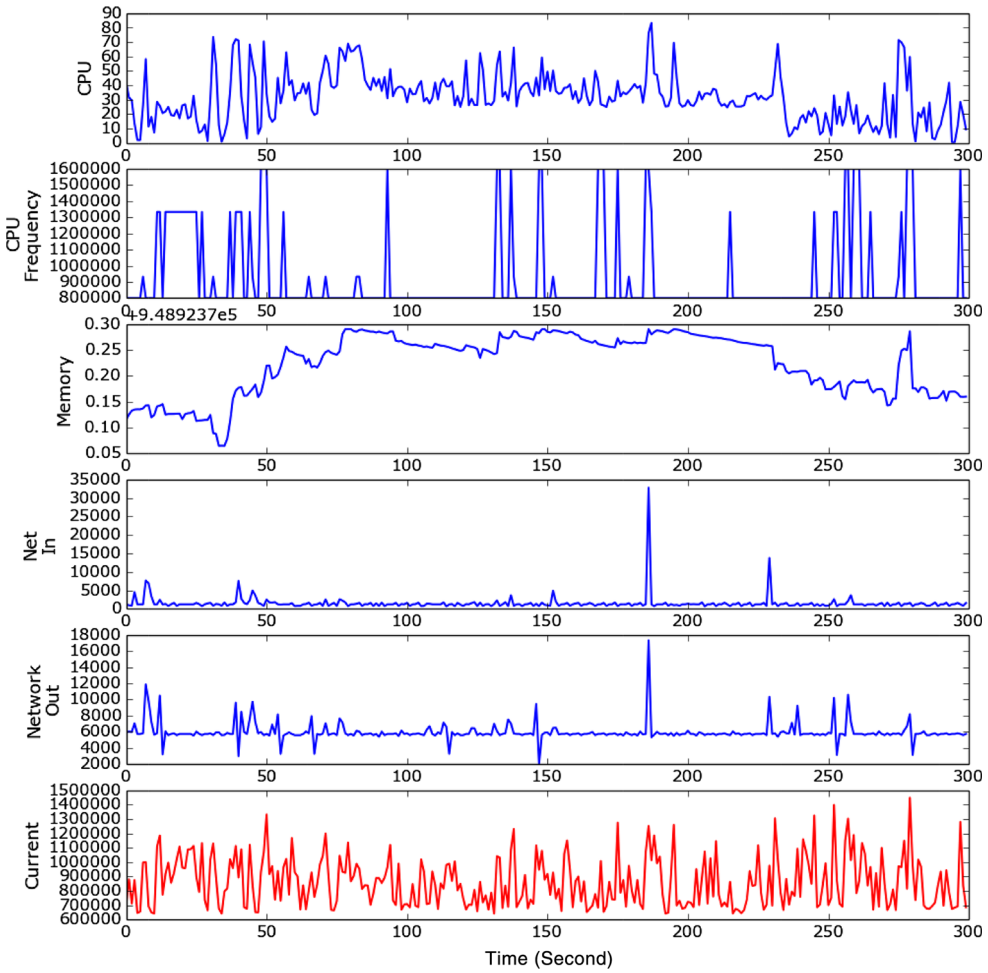

Fig. 7 Three hundred samples of the collected data set

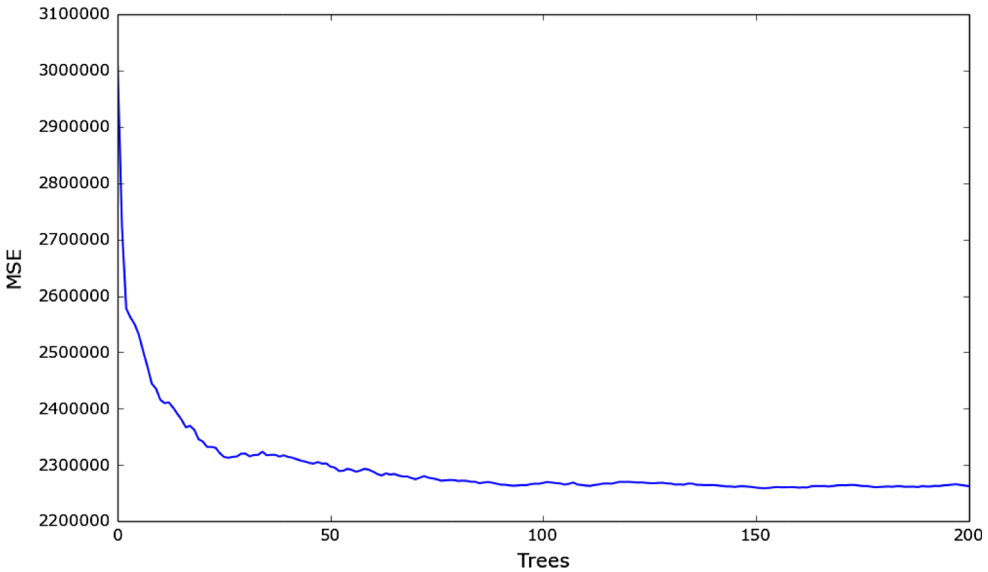

Fig. 8 Mean-squared error per estimator number

energy than the Google's Chrome and that consumes less energy than Firefox. The measurement results of the different operations are presented in Fig. 11.

Table 5 Mean-squared error per different time lags

p	0	1	2	3	4
MSE	2259e3	2196e3	2153e3	2153e3	2166e3

5.2 Evaluation of PC-based software applications

In this experiment, we adapted the energy profiling tool to measure the energy consumption of desktop computer-based software applications. The hardware configuration is composed of an Intel Core i5-3550 @ 3.30GHz (4 cores) processor, 4096 MB memory, Intel DZ77GA-70K motherboard, a hard disk, and an the Intel 82579V Gigabit Connection network interface. The device is also equipped

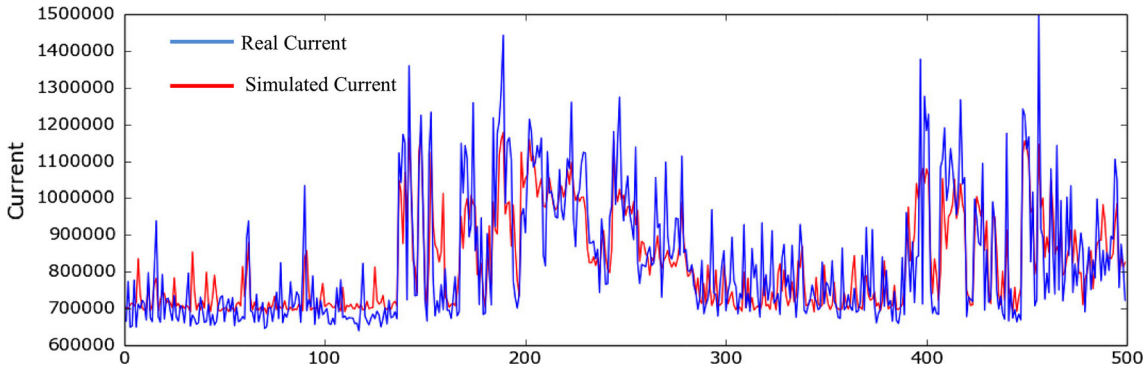

Fig. 9 Observed and estimated energy consumption

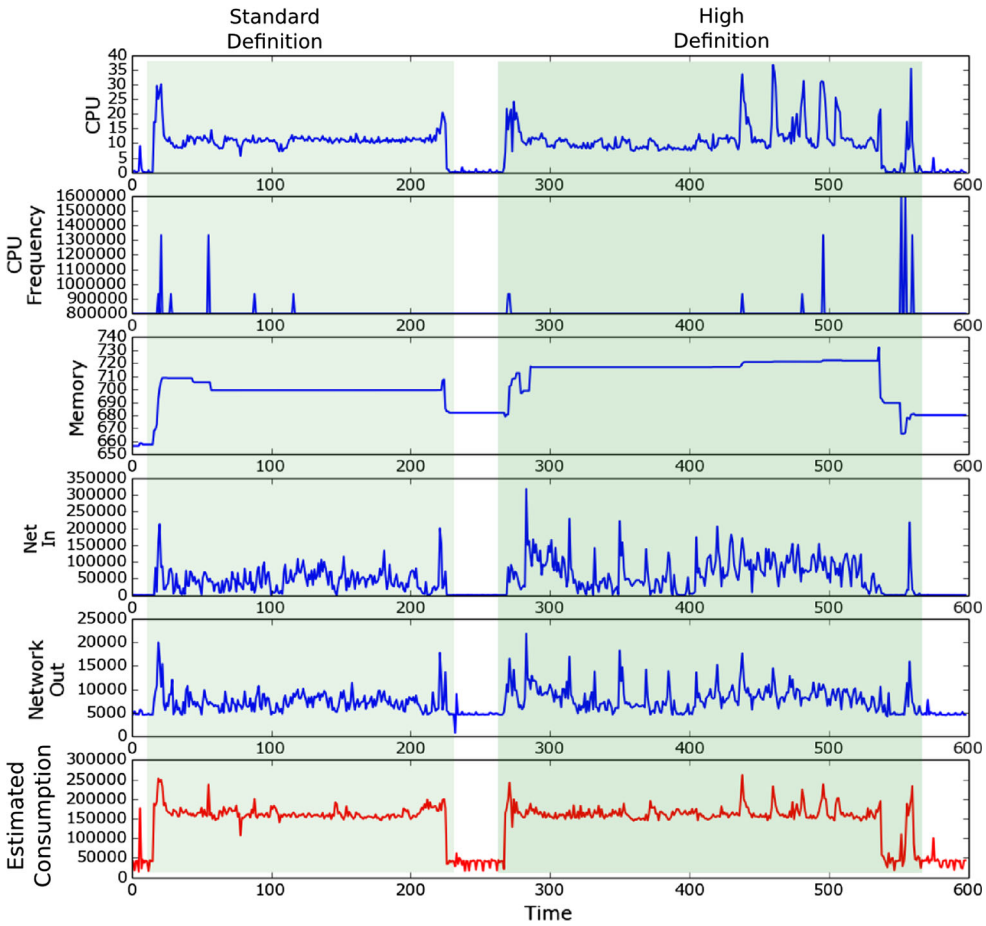

Fig. 10 Youtube's mobile application energy consumption at different video resolutions

Table 6 Browser energy consumption comparison

Browser	Firefox	Chrome	Opera
Energy consumption (A)	31.50	26.90	22.58

with an internal power supervisor to provide the energy usage.

The established energy consumption model is used to evaluate and compare the green efficiency of some most well-known sorting algorithms, which are the bubble sort, the modified bubble sort, the insertion sort, the selection sort, the merge sort, and the quick sort. Usually, the sorting algorithms are judged by their algorithmic efficiency that is based on the number of elements to sort. The best case, average case, and worst case of the algorithmic efficiency of the presented algorithms are presented in Table 7. The presented sorting algorithms are implemented using the C++ programming language, and they are used to sort an array of one hundred thousand (100,000) integers in an ascending order. The energy consumption of each algorithm is presented in Fig. 12.

Table 8 provides the total energy consumption of the different sorting algorithms. The most green efficient sorting algorithms are the merge sort and the quick sort algorithms. They both consumed 2.90 J to accomplish the sorting of the array.

In this case, we evaluated the computational efficiency of the sorting algorithms. The optimal energy consumption is due to the sorting using the merge sort and the quick sort algorithms that have a similar algorithmic efficiency of $O(n\ log(n))$. The greatest energy consumption is performed by the modified bubble sort algorithm (144.07 J). From comparison of the CPU usage and frequency related to the bubble sort and modified bubble sort algorithms, we notice that the modified bubble sort algorithm involves a high CPU frequency to sort the array as it is shown in Fig. 13.

Table 7 Comparative chart of the algorithmic efficiency of the sorting algorithms

Algorithm	Algorithmic efficiency		
	Best	Average	Worst
Bubble sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Modified bubble sort	$O(n)$	$O(n^2)$	$O(n^2)$
Insertion sort	$O(n)$	$O(n^2)$	$O(n^2)$
Selection sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Merge sort	$O(n\ log(n))$	$O(n\ log(n))$	$O(n\ log(n))$
Quick sort	$O(n\ log(n))$	$O(n\ log(n))$	$O(n^2)$

6 Related work

This section presents the related work in three areas: green software development process, the issues raised by the green requirements, and measuring software green efficiency.

Green software development process: several works have been proposed to express the green issues in the software engineering process. Dick and Naumann et al. [7] propose an extension of the software development model to cover the sustainability by introducing four subprocesses to the software engineering process: sustainability reviews and previews, process assessment, sustainability journal,

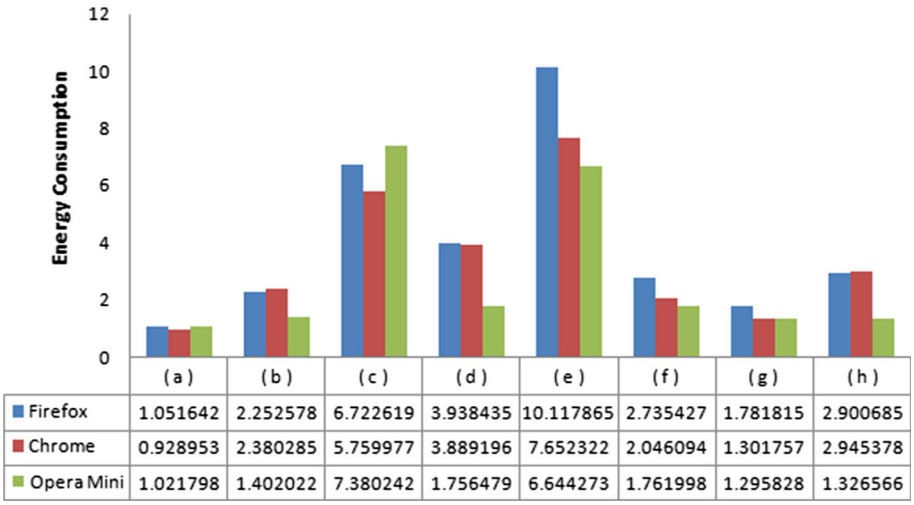

Fig. 11 Mobile browsers energy consumption comparison over different tasks

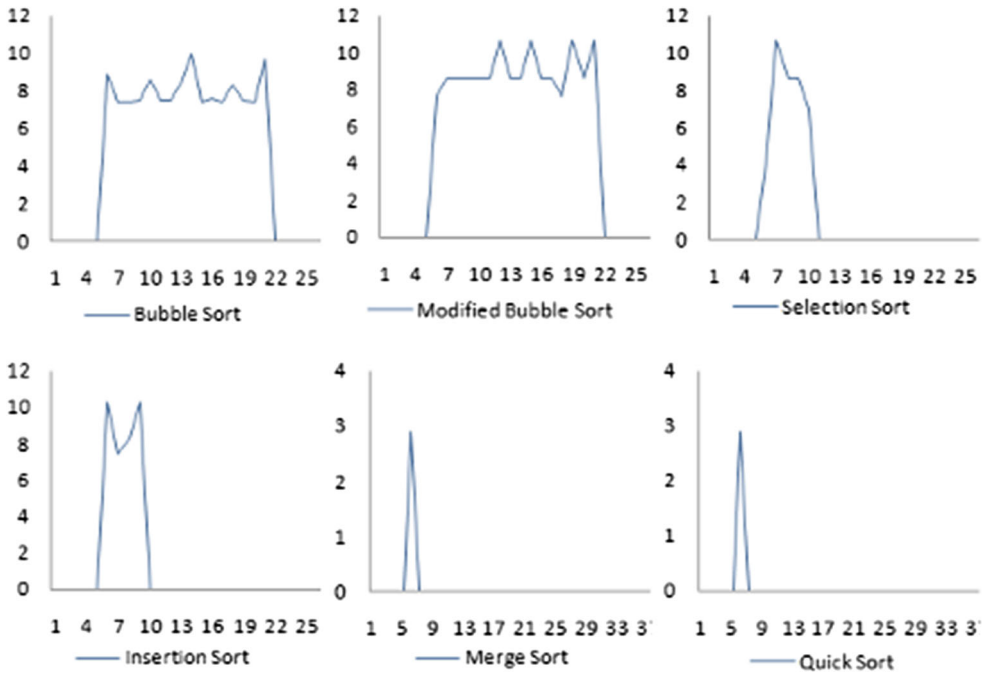

Fig. 12 Energy consumption of different sorting algorithms

Table 8 Total energy consumption of sorting algorithms

Algorithm	Bubble sort	Modified bubble sort	Selection sort
Energy consumption (joules)	129.17	144.07	38.99
Algorithm	Insertion sort	Merge sort	Quick sort
Energy consumption (joules)	36.40	2.9	2.9

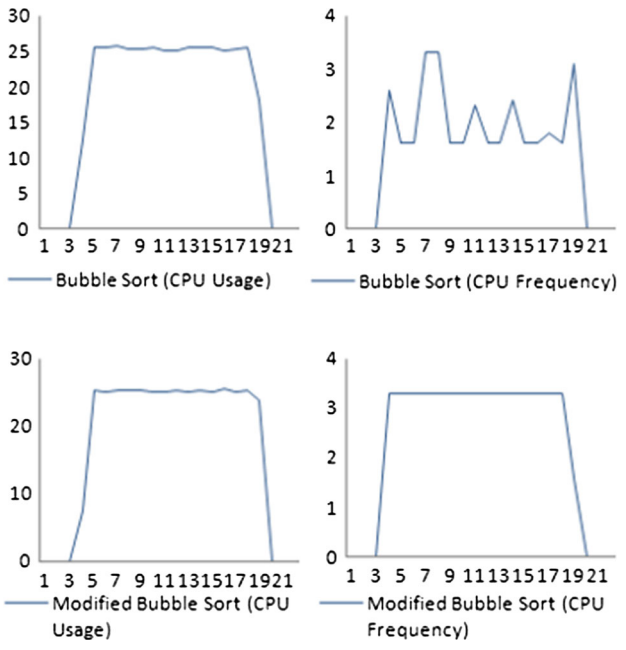

Fig. 13 CPU frequency and usage metrics measurement of the bubble and modified bubble sorting algorithms

and sustainability retrospective. Additionally, they recommend a list of exemplary tools, checklists, and guidelines to assist the software sustainable development. The presented model in [7] has been integrated into the GreenSoft model [17] that is proposed as a four-part model: (1) the first part assesses sustainability at software's life cycle (development, usage, and disposal); (2) the second part represents the green metrics and the criteria, where metrics have been classified into three categories: common quality, directly related, and indirectly related; (3) the third part consists of the procedures explained in the previous related work [7]; and (4) the fourth part of the GreenSoft model is devoted to assessing the recommendations and tools to contain guidelines and checklists for the green practices. Furthermore, a detailed model of sustainability criteria and metrics of the GreenSoft model has been provided in research [12] where Dick et al. proposed the following metrics: energy efficiency, CPU-intensity, memory usage, peripheral intensity, and idleness to measure the sustainability of the software. In another work, Lami et al. [14] focused on greening the software development process itself by identifying green drivers of the project and deriving a set of

measurable sustainability indicators. On the basis of the software development life cycle, the authors [27] proposed some practices for each phase of the development process to help in developing eco-friendly software systems.

Green software requirements: many works in the literature assessed the need of specifying the green requirements. Lami et al. [13] proposed "sustainability" as a characteristic to be integrated into the ISO/IEC 25010/2011 standard and to be seen from both the internal/external and in-use views. The level of interest in their proposition is related to Direct Effects caused by Information and Communication Technology (ICT) infrastructures and equipments (i.e. the resource consumption when producing ICT equipment, the energy consumption when using ICT, and the effects of the resulting electronic waste.). As a result, they presented a set of sustainability factors grouped into five assets: people, project infrastructure, processes, institutional context, and product. Raturi et al. [23] presented a description of a non-functional requirements (NFR) framework that characterizes the sustainability of the software based on the sustainability dimensions (human, social, economic, environmental sustainability, and technical sustainability) and the three orders of Green IT (direct effects, indirect effects, broad-scale socio-economic structural changes). The authors discuss how to use the framework to elicit and to describe sustainability related requirements by presenting an example of the Hotel Tracking Resources System. Furthermore, Calero et al. [4] propose the ISO/IEC 25010+S model to overcome the sustainability of the software as product and in-use qualities. The authors adopted the characteristics and subcharacteristics that have a direct impact on the sustainability (e.g. time behaviour, resource utilization, learnability, accessibility) and adapted the characteristics that do not express the sustainability (e.g. sustainability functional appropriateness, sustainability capacity, sustainability effectiveness).

Green metrics: Adel Noureddine et al. [18] propose the POWER API to estimate the energy consumption of running processes. The authors rely on the information about the CPU and the network card components, which are provided by the sensors associated with these components of the device, to establish the energy models. The authors presented the impact of the algorithm's implementation and programming languages on the energy consumption behaviour of the software. In [19], the authors review the main approaches and tools for measuring and estimating the energy consumption of the software. In addition, they provided the existing formulas to calculate the energy consumption of the hardware using linear regression analysis. Moreover, Povoa et al. [21] propose and evaluate a linear regression model to estimate data centre energy consumption based on information about the utilization of computational resources (e.g. processor, memory, network interface, disc) to discover the biggest energy consumption components in order to identify the most influential components using the maximal information coefficient (MIC) approach. They used an external power sensor to measure the energy consumed by the nodes in the data centre and deployed several stress tests in order to generate a workload to measure the resource usage. Similarly, Samak et al. [25] present a regression model to predict the energy consumption of two clusters in the Grid5000 Test-bed by analysing a data set that compromise a 6-month period in order to establish the coefficients of the auto-regression moving average (ARMA) model as a time series approach using the energy consumption records as a single time series. Another work [5] has presented a model for evaluating energy consumption in the process layer starting from the analysis of the characteristics of the activities composing the process and the resources in use. A way to improve energy efficiency is proposed through a list of energy-aware adaptation strategies that are based on a set of green performance indicators (GPIs) and quality of system (QoS). In the work presented in [16], the authors presented Wattson, which is an application to identify the segments of a mobile application that have high energy consumption. The presented application allows the determination of the components (display, network, or CPU) that consume the most energy during the mobile app run, which allow the researchers to study the energy consumption in different circumstances (different phone models, 2G or 3G network usage, different screen brightness, low signal strength area). Hindle [10] proposed the green mining methodology, which is composed of seven steps to measure and extract the power consumption information related to software changes. In this work, the author performed tests on two software (Firefox and Azureus) using an external device to study the impact of the software changes on the energy consumption.

7 Conclusion

In this paper, we introduced green efficiency as a software quality in order to target the usage of the energy optimization techniques during the development stage. The goal of proposing such a non-functional characteristic is to consolidate green practices during the implementation of software. The proposed characteristic concerns the software itself and aims to mitigate the energy consumption caused by the computation, storage, and communication workload. Furthermore, we proposed an approach to establish an energy profiling tool to locate the energy-consuming portions of code. To estimate the energy usage of software on different information technology platforms,

the energy consumption model is built to fit a variety of hardware configurations and devices. The approach is based on the random decision forests method to develop a model of energy consumption. Our approach eliminates the noise that can be produced by other running applications, which means that the provided energy usage can be reliable enough to evaluate the green aspects of the software adequately. The second advantage of the established model is its extensibility and adaptability to estimate the energy consumption at different platforms. However, this approach is highly dependent on the recorded data collection that is used to train the RDF regression algorithm. A model that does not cover the software behaviour at certain conditions may fail to perform accurate estimations. Moreover, the data sampling rate influences the granularity of the estimation, and subsequently, the energy profiling approach is less efficient in a complex environment where rendering the computations is difficult to elicit and conduct.

Green efficiency is considered as a primary aspect that affects the software itself and that is expressed as an internal/external quality in the requirements model. However, to establish an entirely sustainable software, the in-use quality model must also be studied to overcome all the sustainability pillars (environmental, economic, and social). A green in-use quality model might enhance the sustainability issues that reflect on the software during the final usage stage (e.g. the reduction in the paper and ink wastes).

References

1. Amsel N, Ibrahim Z, Malik A, Tomlinson B (2011) Toward sustainable software engineering (nier track). In: ICSE'11 Proceedings of the 33rd International Conference on Software Engineering, pp 976–979
2. Ardito L, Procaccianti G, Vetro A, Morisio M (2013) Introducing energy efficiency into sqale. In: ENERGY 2013, The Third International Conference on Smart Grids, Green Communications and IT Energy-aware Technologies, pp 28–33
3. Breiman L (2001) Random forests. Mach Learn 45:5–32
4. Calero C, Bertoa MF, Moraga MA (2013) A systematic literature review for software sustainability measures. In: Proceedings of 2nd International Workshop on Green and Sustainable Software (GREENS), pp 46–53
5. Cappiello C, Fugini M, Ferreira AM, Plebani P, Vitali M (2011) Business process co-design for energy-aware adaptation. In: IEEE International Conference on Intelligent Computer Communication and Processing (ICCP), pp 463–470
6. Capra E, Francalanci C, Slaughter SA (2012) Measuring application software energy efficiency. IT Prof 14(2):54–61. doi:10.1109/MITP.2012.39
7. Dick M, Naumann S (2010) Enhancing software engineering processes towards sustainable software product design. In: Proceedings of the 24th International Conference EnviroInfo: Integration of Environmental Information in Europe, pp 706–715
8. Dong M, Zhong L (2011) Self-constructive high-rate system energy modeling for battery-powered mobile systems. In: Proceedings of the 9th international conference on Mobile systems, applications, and services - MobiSys '11, pp 335–348. doi:10.1145/1999995.2000027
9. Gong L, Xie J, Li X, Deng B (2013) Study on energy saving strategy and evaluation method of green cloud computing system. In: 2013 8th IEEE Conference on, Industrial Electronics and Applications (ICIEA), pp 483–488
10. Hindle A (2012) Green mining: a methodology of relating software change to power consumption. In: Proceedings of the 9th IEEE Working Conference on Mining Software Repositories (MSR), pp 78–87
11. Ho TK (1998) The random subspace method for constructing decision forests. IEEE Trans Pattern Anal Mach Intell 20:832–844
12. Kern E, Dick M, Naumann S, Guldner A, Johann T (2013) Green software and green software engineering. In: Proceedings of the First International Conference on Information and Communication Technologies for Sustainability
13. Lami G, Buglion L, Fabbrini F (2013) Derivation of green metrics for software spice 2013. Commun Comput Inf Sci 349:13–24
14. Lami G, Fabbrini F, Fusani M (2013) A methodology to derive sustainability indicators for software development projects. In: Proceedings of the 2013 International Conference on Software and System Process, pp 70–77
15. Li D, Halfond WGJ (2014) An investigation into energy-saving programming practices for android smartphone app development. In: Proceedings of the 3rd International Workshop on Green and Sustainable Software—GREENS 2014, pp 46–53. doi:10.1145/2593743.2593750
16. Mittal R, Kansal A, Chandra R (2012) Empowering developers to estimate app energy consumption. In: Proceedings of the 18th annual international conference on Mobile computing and networking (Mobicom'12), pp 317–328
17. Naumann S, Dick M, Kern E, Johann T (2011) The greensoft model: a reference model for green and sustainable software and its engineering. Sustain Comput Inf Syst 1:294–304
18. Noureddine A, Bourdon A, Rouvoy R, Seinturier L (2012) A preliminary study of the impact of software engineering on greenit. In: Proceedings of First International Workshop on Green and Sustainable Software, pp 21–27
19. Noureddine A, Rouvoy R, Seinturier L (2013) A review of energy measurement approaches. ACM SIGOPS Oper Syst Rev 47:42–43
20. Penzenstadler B, Raturi A, Richardson D, Tomlinson B (2014) Safety, security, now sustainability: the nonfunctional requirement for the 21st century. IEEE Softw 31:40–47
21. Povoa LV, Junior PWB, Monteiro CE, Mueller D, Marcondes CAC, Senger H (2013) A model for estimating energy consumption based on resources utilization. In: IEEE Symposium on Computers and Communications (ISCC), pp 1–6
22. Quinlan JR (1986) Induction of decision trees. Mach Learn 1:81–106
23. Raturi A, Penzenstadler B, Tomlinson B, Richardson D (2014) Developing a sustainability non-functional requirements framework. In: Proceedings of the 3rd International Workshop on Green and Sustainable Software (GREENS 2014), pp 1–8
24. Sabharwal M, Agrawal A, Metri G (2013) Enabling green it through energy-aware software. IT Prof 15:19–27
25. Samak T, Morin C, Bailey D (2013) Energy consumption models and predictions for large-scale systems. In: Proceedings of the 2013 IEEE 27th International Symposium on Parallel and Distributed Processing Workshops and PhD Forum (IPDPSW '13), pp 899–906

26. Seo C, Malek S, Medvidovic N (2008) Estimating the energy consumption in pervasive java-based systems. In: 6th Annual IEEE International Conference on Pervasive Computing and Communications, PerCom 2008, pp 243–247. doi:10.1109/PERCOM.2008.85
27. Shenoy SS, Eeratta R (2011) Green software development model: an approach towards sustainable software development. In: 2011 Annual IEEE, India Conference (INDICON), pp 1–6
28. Sierszecki K, Mikkonen T, Steffens M, Fogdal T, Savolainen J (2014) Green software: Greening what and how much? IEEE Softw 31:64–68. doi:10.1109/MS.2014.63
29. Zhang C, Hindle A (2014) The impact of user choice on energy consumption. IEEE Softw 31(3):69–75. doi:10.1109/MS.2014.27