Information and Software Technology 134 (2021) 106560

Contents lists available at ScienceDirect

Information and Software Technology

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

A process for analysing the energy efficiency of software

Javier Mancebo *, Félix García, Coral Calero

Alarcos Research Group, Institute of Technology and Information Systems, University of Castilla-La Mancha, Ciudad Real, Spain

ARTICLE INFO

Keywords:
Software sustainability
Green software
Software consumption measurement
Energy consumption measurement process
Energy efficiency

ABSTRACT

Context: It is essential to be aware of the energy efficiency of software when it is running, so that it can be improved; to that end, energy consumption measurements need to be carried out. To ensure that these measurements are as reliable as possible, it is recommended that a well-defined process be followed.
Objective: To identify how the process for analysing the energy efficiency of software should be carried out (including the definition of the software to be evaluated, the selection of measuring instruments, the analysis and the presentation of results, etc.), in an endeavour to improve the reliability and consistency of the information obtained regarding energy efficiency.
Method: An analysis of related work was carried out, to extract some good practices in measuring energy consumption; based on our experience, a process to analyse the energy efficiency of the software has been defined.
Results: We have defined a process to analyse the energy efficiency of the software. We describe this process through a set of phases that covers all the steps needed to carry out a correct analysis of the energy consumption of the software executed. Moreover, this process was validated with two different studies using different measurement instruments (one with a hardware-based approach and one with a software-based approach) to ensure its applicability to all types of studies with software energy consumption measurement.
Conclusion: The steps to be followed to analyse the energy efficiency of the software need to be established. A new process has hence been defined to improve the reliability and consistency of the measurements. Furthermore, this process facilitates the replicability and comparison of the studies carried out.

1. Introduction

The expansion of the information and communication technology (ICT) sector in recent years has led to a remarkable growth in the environmental impact brought about by technology. According to a report published by Huawei Technologies [1], ICT energy consumption in 2018 already represented about 9% of total global energy consumption. Moreover, estimates indicate that global ICT energy use could exceed 20% of total energy and could emit up to 5.5% of the world's carbon emissions by 2025. This would have a large negative impact on the environment [2,3].

For that reason, an increasing number of green ICT solutions have emerged. The proposed solutions have focused mainly on improvements in hardware, with the aim of reducing the environmental impact that it generates [4,5]. In recent years, however, software has also been identified as having a negative impact on the environment and recent research has focused on the appropriate use of software resources, which has led to the development of more sustainable and environmentally friendly software [6,7]. Dick et al. [8], define software as "sustainable" where the direct and indirect negative impact resulting from its development, deployment and usage is minimal and/or has a positive effect on sustainable development as regards the economy, society, humans and the environment.

In the summary of Calero et al.'s proposal on Green and Sustainable Software [6], the authors assert that Green and Sustainable Software is a major research topic that has been very active in recent years. They also highlight that Green Software promotes the improvement of the energy efficiency of software, minimising its environmental impact and potentially having a positive impact with respect to the economy and humans [7].

In order to develop more sustainable software and limit any negative impacts that it may bring about, there is a need for methods that will measure and/or estimate the energy consumption that is induced by the software when it is running [9,10]. The measuring instruments for the analysis of software energy consumption can be classified in the following manner [10]:

* Corresponding author.
E-mail addresses: Javier.Mancebo@uclm.es (J. Mancebo), Felix.Garcia@uclm.es (F. García), Coral.Calero@uclm.es (C. Calero).

https://doi.org/10.1016/j.infsof.2021.106560
Received 26 March 2020; Received in revised form 19 January 2021; Accepted 18 February 2021
Available online 20 February 2021

- Software-based approach: software tools estimate the power consumed by the hardware when the software is executed. The adoption of this approach does not require much effort and hence is cheaper. Measurements are, however, subject to some error rates because they provide estimates. Some examples of software tools that adopt this approach are PowerAPI [11] or Joulemeter [12].
- Hardware-based approach: devices that use physical energy meters connected to PC hardware devices. This approach is more promising than the software-based one, because it allows for accurate measurements of the energy consumed by a computer. It should be said, however, that these devices are relatively more expensive and as such not available to everyone. EET [13] or Watts Up? [14] are examples of measuring instruments that follow the hardware-based approach.

Several empirical experiments on software energy consumption using either of the two approaches described to perform the measurements are presented in the available literature. Capra et al. analyse energy efficiency on a set of 63 open-source applications. Sahin et al. [15], using a hardware-based approach, investigate the energy impact of using a set of fifteen software design patterns. Hindle [16] investigates the impact of software change on energy consumption, along with the relationship to software metrics. For this purpose, three applications were chosen (Firefox, Vuze and rTorrent) and, for each application, a set of different releases was selected. In [17], the authors used a hardware device to measure the energy consumption for different software products such as word processors, web browsers, or database systems. They thereby analysed different software architectures. In a software-based approach, using the PowerAPI library, Noureddine et al. [18] analyse the impact of the energy of various programming languages with different algorithms (recursive vs. iterative). In other work, such as [14], the authors combine both approaches to measuring energy consumption and seek to compare the energy consumption of a software product in different versions and to explain the variations in energy consumption brought about by differences at the level of software architecture.

Analysing the above-mentioned studies, it can be observed that each author applies their own methodology - or does not present any method - to carry out the measurements of energy consumption. Consequently, it is difficult to compare the results obtained in different pieces of work. Another problem of not following a defined process is the difficulty of replicating energy consumption measurements.

To solve this problem, it is necessary to establish a method that will serve as a guide for carrying out software energy consumption measurements. Such method should start with the design of the study and continue right through to the analysis and reporting of the results obtained, as well as taking account of the particularities of the field of software energy consumption measurement. As a result of such an approach, it is to be expected that a greater control of the measurements made to ensure the reliability, consistency and coherence of the measurements would be achieved, which in turn would support the replication of the studies carried out [19,20].

Considering the importance of a defined method that could help researchers to analyse the energy efficiency of software and the real need that is apparent from an analysis and use of existing measurement proposals, we have defined a specific process that integrates all the activities necessary to measure and analyse the energy consumption of the software evaluated.

This process is mainly related to the technical sustainability dimension, since the main objective of the process is to define the steps to be followed to evaluate and improve the energy efficiency of the software. In addition, the process affects indirectly to the rest of the dimensions of sustainability by means of the relationship with the technical dimension.

Our proposal is composed of seven different phases which cover the main steps to be performed, from the measurement of energy consumption right through to the analysis of the results, including actions such as defining the scope of the study, details on how to conduct valid and reliable measurements and how the results obtained should be reported. To define our process, we have followed the method engineering approach [21], using the SPEM specification [22] and the EPF Composer tool to model the defined process.

We organize the content of the paper as follows: firstly, we present the software measurement frameworks and standards and compare them to the process that we propose. Subsequently, the process proposed to help researchers carry out energy consumption measurements of software is detailed (Section 3). Thereafter, in Section 4, two application examples in which the process described has been applied are presented. Finally, Section 5 sets out the conclusions of this work and presents our proposals for complementary research activities.

2. Related work

As described in the introduction, measuring the energy consumed by software is becoming increasingly important in the effort to improve software sustainability and as such it is necessary to define a process that will ensure the rigour and consistency of studies using software energy consumption measurements.

Different approaches that could be useful for this purpose can be found in the existing literature. We describe the proposals that already exist, divided into two subsections. In the first of these, we present the software measurement frameworks and standards which aim to provide guidelines for carrying out the measurement process effectively and systematically, based on the objectives defined. In the second, we give an overview of the approaches put forward for carrying out a measurement of the power consumed by the software. Finally, we include a comparison between the methodologies and standards mentioned and our proposed process.

2.1. Measurement methodologies and standards

The Goal/Question/Metric (GQM) method proposed by Basili and Weiss [23] establishes guidelines to define a measurement program: the context, the objectives and the measurement process plan. Guidelines on data collection, analysis, interpretation of results and identification of potential improvements are also provided. GQM is a method that allows the objectives of measurement to be refined into a set of quantifiable questions that are used to identify the data to be collected to support the decision-making process. Measured data allow us to answer the questions and then to analyse whether the goals have been attained [24,25]. The GQM method is composed of four phases [26]:

1 The Planning phase: in this phase, the necessary information is gathered and a project for the application of measurements is defined, characterised and planned, resulting in a project plan that documents the procedures, schedules and objectives of a measurement program.
2 The Definition phase: during which the measurement program is defined (goal, questions, metrics and hypotheses are defined) and documented.
3 The Data Collection phase: during this phase, the data collected from the measurements are defined, filled in and stored.
4 The Interpretation phase: during which the data collected is processed and the measurements then used to answer the questions, aiming to respond to the goal established.

Furthermore, there are many proposals in the literature for extensions to the GQM method. These include Goal-Driven Software Measurement (GDSM) [27], which provides an extension to the Planning phase of the GQM method, improving the way measurements are derived from business objectives and providing useful templates that help define objectives, indicators, measurements, etc. This extension of GQM is called Goal Question Indicator Metric (GQ(I)M) and provides explicit support for indicators, ensuring that a consistent collection of

metrics is available to construct an indicator. GQ(I)M also provides additional elements to ensure consistent interpretation of the indicator [28].

Another methodology for software measurement is PSM (Practical Software Measurement) [29], which aims to provide a set of good practices and guidelines for software measurement. It is based on proven measurement principles derived from actual experience in government and industry projects. PSM proposes a model of the measurement process, which is divided into four main activities: Plan measurement, Perform measurement, Evaluate measurement, and Establish and Sustain commitment. In summary, the PSM framework provides a systematic approach to planning and implementing the software measurement and analysis process.

The international standard ISO/IEC/IEEE 15939 [30] is based on PSM. This standard identifies the activities and tasks which are necessary to successfully identify, define, select, apply and improve software measurement within a generic project or within the measurement organisation structure. According to this standard, the main objective of the measurement process is to collect, analyse and provide relevant data regarding the implemented products and processes, in the quest to manage the processes and to demonstrate objectively the quality of products, services and processes. The software measurement process defined by ISO/IEC/IEEE 159339 consists of four activities that are similar to those in the framework.

Although these software measurement frameworks provide guidelines for defining and implementing software measurement programs, none of the above-mentioned methodological frameworks focuses on measuring the energy efficiency of software, so these frameworks cannot be fully adopted for our purposes.

2.2. Energy measurement methodologies

With regard to works that present guidelines for energy measurement in software, to the best of our knowledge the only one of relevance is “Green Mining Methodology” [16]. This methodology describes how to conduct experiments using energy consumption measurements and is composed of seven activities: *(1)* choose the software product and the context in which it should be checked, *(2)* decide the types of data that will be registered, *(3)* choose a set of versions of the software, *(4)* develop the test cases that are to be run, *(5)* configure the testing ground, *(6)* carry out the measurements for each version and gather the data registered, and *(7)* analyse the results. The main defect in this methodology is that it does not provide any protocol or good practices regarding how to carry out the measurement in a way that is valid and reliable. For this reason, Jagroep, et al. [14] present a measurement protocol, in which an extension of activity (6) of "Green Mining" is performed, detailing the specific tasks to be carried out. These tasks include: (a) run the test within the testbed and record the instrumented data, (b) compile and store the recorded data, and (c) clean up the test and testbed.

The above works present some guidelines for carrying out measurements of the energy consumption of software. They do not, however, provide details on how to carry out a full process for analysing the energy efficiency of software.

2.3. Comparison of measurement methodologies

As mentioned in the Introduction, before developing our specific process for the measurement of the energy consumption of the software, an analysis of existing proposals was conducted and each one of them was tested to see whether they could be adapted to this type of measurements.

Firstly, generic methodologies for the measurement of software energy consumption were reviewed, such as the PSM [29] or the GQM method [26], among others. This type of generic methodologies provided guidelines for conducting the measurement of the energy consumption of software, and informed our process for the same. Nevertheless, these methods were not wholly suitable due to the particularities that exist in the measurement of the energy efficiency of software, such as the tools required to make these measurements or the definition of the scenarios in which certain factors have to be taken into account, including the consumption of the operating system or the running of other applications in the background, which can affect the accuracy of the measurement.

Once the use of generic methodologies had been discarded, our studies turned to the specific methodologies for the measuring of the power consumed by the software. With this focus, we identified the Green Mining methodology [16] and its extension proposed by Jagroep et al. [14], which did present at a high level a range of activities and standards for making measurements of the energy consumption of software. However, these authors did not provide details on how to achieve the full process, in other words, how to not only conduct the measurement but also how the results obtained should be analysed and reported.

From an analysis and use of the existing work, we can conclude that there is a lack of guidelines to help analyse the energy consumption of software. Bearing this in mind, we propose a new process for measuring and reporting energy consumption of software products, to improve the results obtained from the measurements. Table 1 summarises the main aspects of the approaches mentioned above, comparing them with our proposed process.

As explained above and shown in Table 1, none of the methodologies or standards mentioned are completely adapted to carry out the entire process of measuring the energy consumption of software, from its planning through to its analysis and reporting. It is for this reason that we consider it necessary to define a process for use by researchers when analysing the energy consumption of software when it is running. Our proposal includes guidelines that describe the entire process of performing measurements, including the preparation and configuration of the environment and software to be measured, the steps to perform the measurement and analysis of the data obtained, and how to report and prepare the laboratory packages so that they can be reproduced.

3. Process for analysing the energy efficiency of the software

In this section, we will describe the proposed process for analysing the energy consumed by the software when it is running. This process consists of seven phases (see Fig. 1), which are divided into different activities with input and output devices.

The phases of the process are based on the grouping of the different activities defined in the Green Mining Methodology [16]. In addition, to define some aspects or artefacts of the process, we have based ourselves on well-known approaches to software measurement and good practices related to green software proposed by other authors.

We use the SPEM 2.0 [22] and the EPF composer version 1.5.2. to describe the process employed to analyse the energy consumption.

The description of the process includes the participant roles, phases and activities (with inputs, outputs and guidelines). A more detailed and comprehensive version of the process and its elements is available at https://alarcos.esi.uclm.es/FEETINGS/.

3.1. Roles

In this subsection, we present the roles that are involved in the different phases of the process. We have identified four different roles: Client, Measurement Analyst, Measurement Performer and Data Analyst, all of which are described in Table 2. A participating person in the process can play one or several of these roles.

Moreover, and in line with the SPEM guide, roles can operate in two different ways, depending on the relationship between an activity and the role: Primary Performs (PP), which refers to the roles that participate in the realisation of the activity; and Additionally Performs (AP), which

Table 1
Comparison of the proposals.

	GQM	GQ(I)M GDSM	PSM	ISO/IEC/IEEE 15939	Green Mining Metodology	Jagroep et al. proposal	Our proposal
Guidelines for the software measurement process	X	X	X	X	X	X	X
Guidelines for the process of analysing software energy efficiency					X	X	X
Specific guidelines for carrying out energy measurement of software						X	X
Specific guidelines for analysing the data obtained from measurement		X	X				X
Specific guidelines for reporting measurement results			X	X			X

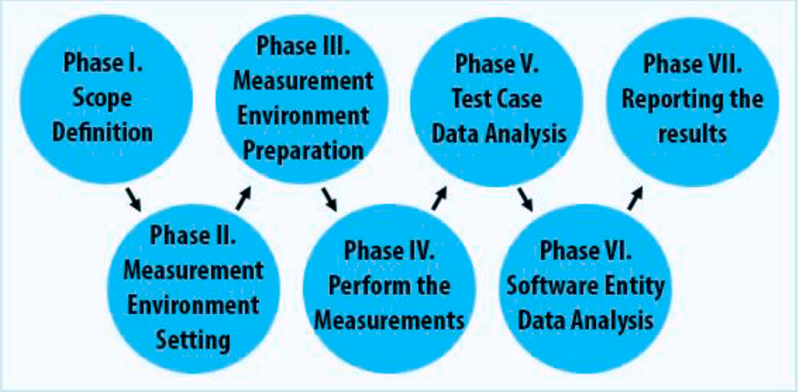


Fig. 1. Process for evaluating the energy efficiency of the software.

Table 2
Roles participating in the process.

Role name	Description
Client (C)	They are interested in the results obtained by measuring the energy consumption of the selected software. They are responsible for providing information about the software to be evaluated and the requirements needed to carry out the energy consumption measurement.
Measurement Analyst (MA)	The person responsible for defining in detail the scope of measurements and the configuration of the measurement environment. They are also in charge of reporting and documenting the results obtained.
Measurement Performer (MP)	Prepares the measurement environment and sets up the testbed. Furthermore, this role is responsible for carrying out the energy consumption measurements in the selected environment.
Data Analyst (DA)	The person responsible for processing and analysing the data extracted from the measuring device and converting it into software energy consumption information.

are the roles that must be informed or which are in some way interested in the realisation of the activity.

3.2. Phases

Our process is intended to be performed iteratively, so the phases are interrelated to each other. The initial phase focuses primarily on the definition of the requirements and the software system to be evaluated. The next two phases focus on the configuration and preparation of the measurement environment. In phase four, energy consumption measurement activities are carried out. Finally, the last phases are the analysis and reporting of the data obtained. In the following subsections, we will describe in detail each of the phases, which will be represented with the diagrams obtained from the EPF Composer tool.

3.2.1. Phase I: scope definition

The main goal of this phase is to obtain a complete specification of the requirements for the evaluation of energy efficiency. Moreover, the software to be analysed must be defined. To achieve this, four different activities are undertaken in this phase, with the inputs and outputs shown in Fig. 2. The figures of the process phases have been obtained with the EPF Composer tool.

The first activity of this phase is to elicit the requirements (Activity A1.1) for the analysis of software energy consumption. To do this, the *Client* provides the *Measurement Analyst* with information about the software to be evaluated. In addition, all the requirements to carry out the energy consumption measurement must be detailed. This information must be documented in the Requirements Specification.

Once the *Client* has provided all the necessary information, the *Measurement Analyst* performs the definition of the objective (Activity A1.2) and chooses the collection of all the entities that satisfy the determined purpose, known as Software Entity Class [31]. We suggest using the recommendations of Wohlin et al. [19], based on the application of the Goal/Question/Metric (GQM) method to correctly define the Goal and the Software Entity Class.

After choosing the Software Entity Class, it is necessary to select the Software Entity [31], which is the software that is to be characterised by measuring its attributes. This corresponds to the third activity in this phase (Activity A1.3). It is essential to check that all the selected Software Entity are available and can be installed and/or run on the Device Under Test (DUT). In the effort to facilitate the selection of the Software Entity, a template is included and can be downloaded from the website indicated.

Finally, the fourth activity is the development of test cases to execute and measure energy consumption (Activity A1.4). Based on the Software Entities defined in the previous activity, a representative test case must be built that will exercise the necessary functionality of the software product whose energy consumption is to be measured. The test case is expected to be independent and should not affect the next test case [16]. A test case could simulate user input, focus on specific software tasks, or

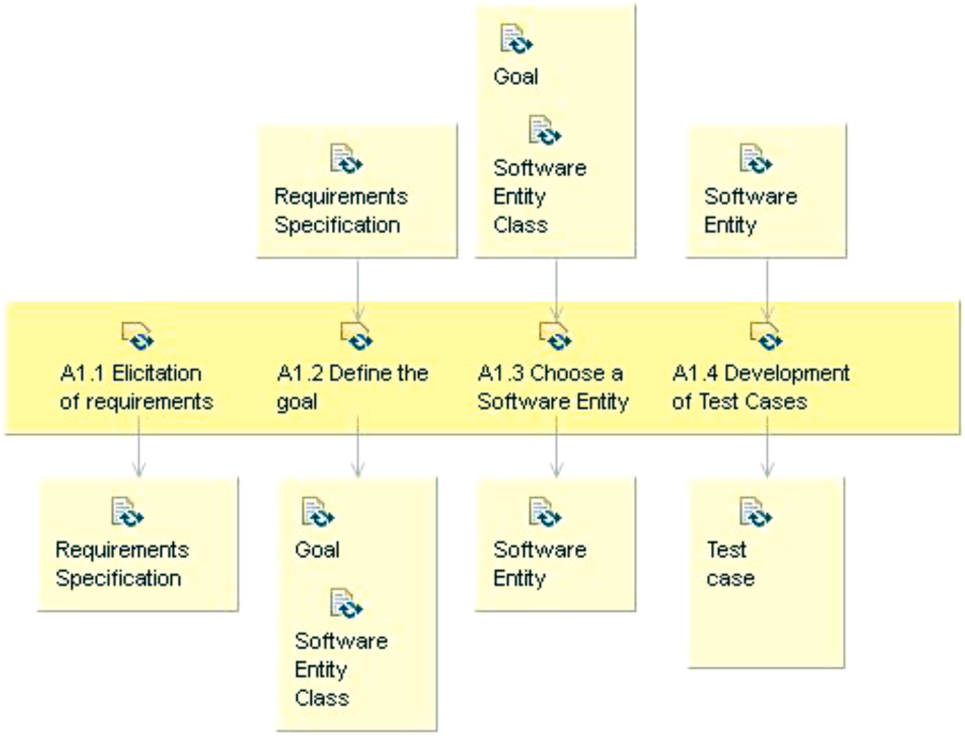


Fig. 2. Phase I: Scope definition.

on the execution of an algorithm. Moreover, if several Software Entities have been chosen, the defined test cases should be able to be tested in all Software Entities. This activity is very important, because if the test cases are not well-defined, it can cause problems in the analysis of energy consumption of the software product.

The main outputs that we obtain at the end of this phase are the specification of the context in which the measurements will be carried out, the Software Entity, and the test cases that will measure the energy consumption.

Table 3 shows the involvement of each role in each activity of this phase.

3.2.2. *Phase II: measurement environment setting*

The second phase has four activities, which are shown in Fig. 3. The purpose of this phase is the definition of the measurement environment that will be used to satisfy the goal defined in the first phase.

The first activity carried out in this phase is the selection of the measuring instrument (Activity A2.1). The measuring instrument is used to perform the power consumption measurements of the software analysed. This measuring instrument may be either a hardware device or a software tool [32]. Depending on whether we want to obtain very precise measurements, and on the availability of the measuring instrument, we will follow one of two approaches. On the one hand, the software-based approaches estimate the power consumption of a system at run time, which indicates that measurements are subject to some error rates. In addition, software measurement tools have the ability to obtain the consumed energy at different levels of granularity, i.e., this type of tool allows us to know the energy consumed by an application, a process or a method [9,12]. On the other hand, we have the hardware-based approaches, which use physical power meters. This approach is much more accurate than software tools in measuring energy consumption. Hardware devices provide power readings at low frequencies, thus increasing the reliability of the measurements, but also increasing post-processing time and effort [9,13].

Table 3
Roles and their responsibilities in Phase I.

Roles Activities	Client (C)	Measurement Analyst (MA)	Measurement Performer (MP)	Data Analyst (DA)
A1.1 Elicitation of requirements	PP	AP		
A1.2 Define the goal		PP		
A1.3 Choose a Software Entity		PP		
A1.4 Development of Test Cases		PP		

The second activity in this phase consists of defining the specifications that the Device Under Test (DUT) must have (Activity A2.2). The test cases defined in Activity A1.4 will be executed in the selected DUT in order to carry out the energy consumption measurements. To choose the right DUT, we should first consider the features of the Software Entity, as it must be possible to install and run it on the DUT. Moreover, depending on the results we want to obtain, the DUT will necessarily have different specifications. For example, if we want the results obtained to be more general, we must choose a DUT without special processing or storage capabilities and with a conventional configuration. However, if we want to know the energy consumption in a specific environment where the software will usually be executed, we must simulate this environment by configuring the DUT to be as close as possible to it.

The next step is to decide on the set of measures to be used for the analysis (Activity A2.3). The main measure of interest is obviously the energy consumption (EC), which is obtained by the measuring instrument. Sometimes it is necessary to recover other measures, however, such as the performance of certain hardware components or different kinds of measurements that are necessary for further analysis; e.g. information about the executed source code (Total Lines of Code or Complexity).

The fourth activity is to check that no other software is running in the background, as well as to interrupt all services and processes that may

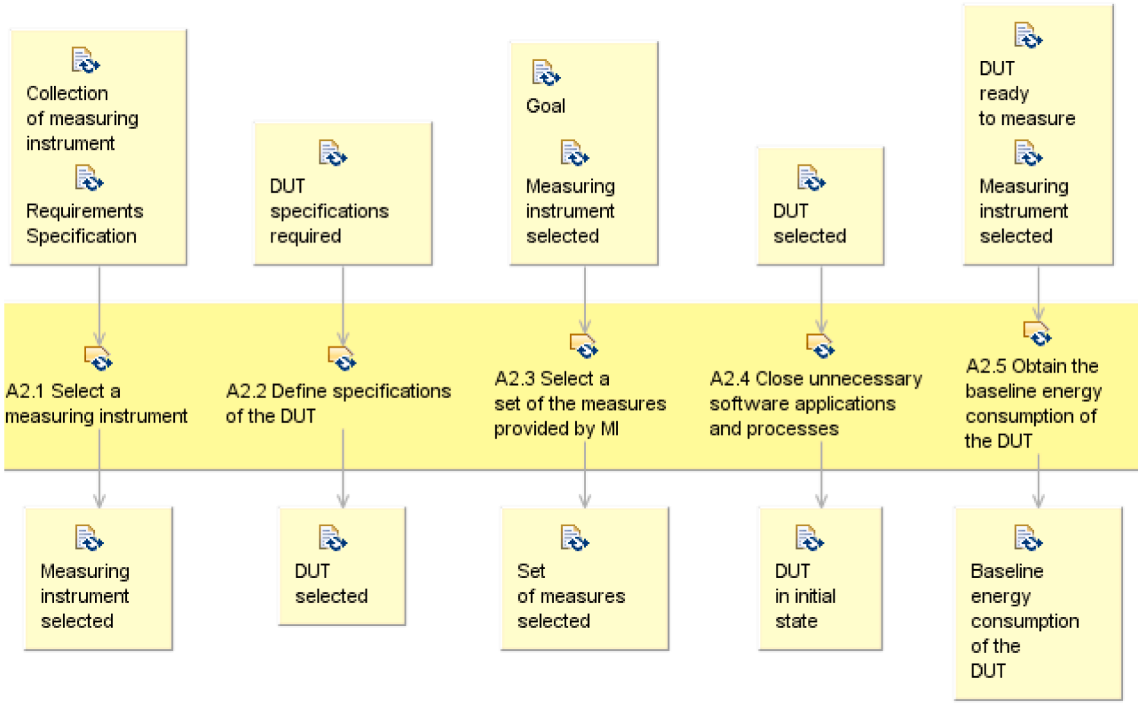


Fig. 3. Phase II: Measurement environment setting.

affect the baseline measurement of consumption (Activity A2.4).

Finally, the fifth activity is to obtain baseline energy consumption (Activity A2.5). The baseline measurement determines the idle energy consumption for the DUT that is used. As the idle energy consumption depends mainly on the hardware used, this value must be determined separately for each DUT used, by carrying out measurements while the DUT is running without any active software [33]. The baseline energy consumption allows us to calculate the energy consumption induced by the execution of the selected test cases, under the assumption that the increase in the energy consumed by the DUT depends exclusively on running the Software Entity under test.

Table 4 shows the roles that have participated in this phase, following the levels of relationships between activities and roles that were explained in the previous section.

Table 4
Roles and their responsibilities in Phase II.

Roles Activities	Client (C)	Measurement Analyst (MA)	Measurement Performer (MP)	Data Analyst (DA)
A2.1 Select a measuring instrument		PP		
A2.2 Define specifications of the DUT		PP		
A2.3 Select a set of the measures provided by measuring instruments		PP		
A2.4. Close unnecessary software applications and processes		PA	PP	
A2.5 Obtain the baseline energy consumption of the DUT		PA	PP	

3.2.3. *Phase III: measurement environment preparation*

Phase III focuses on the preparation of the energy consumption measurements to be performed and on the configuration of the measurement environment that was defined in the second phase. This phase is composed of three activities, which are summarized in Fig. 4.

The first step (Activity A3.1) before starting the energy consumption measurements is to check that no other software is running in the background. After that, we must interrupt any services and processes that are not required by the software under test, seeking to minimise the effect they may have on the power consumption of the DUT (for example, the automatic update service or virus scans).

The next activity (Activity A3.2) is to determine the number of times each measurement should be repeated. We consider a measurement to be a set of energy consumption samples from a single test case run. There is no exact and correct number of repetitions to be measured. The choice of this value depends on the objective we have defined, as well as on the resources available. Some authors [17] recommend that, for measurements of software energy consumption in a controlled environment, 30 measurements are usually a sufficient sample size for an analysis of each of the test cases devised, as the sampling distribution will tend to be normal.

The last activity (Activity A3.3) in the preparation of the measurement environment is the configuration of the testbed. The Software Entity and the services required in the DUT need to be installed. Once the measurements for one of the Software Entities are completed, the DUT is restored, such that it returns to its initial state. This procedure is repeated for the different Software Entities that are going to be assessed. In this activity, the selected Software Entity must also be prepared, so that it can execute the test cases defined.

Table 5 shows the roles that have participated in the Measurement Environment Preparation phase.

3.2.4. *Phase IV: perform the measurements*

During this phase, energy consumption measurements will be carried out. The fourth phase consists of only two activities, as shown in Fig. 5. Both activities are an iteration, as these nested activities can be repeated more than once; indeed, they will be repeated as many times as test cases

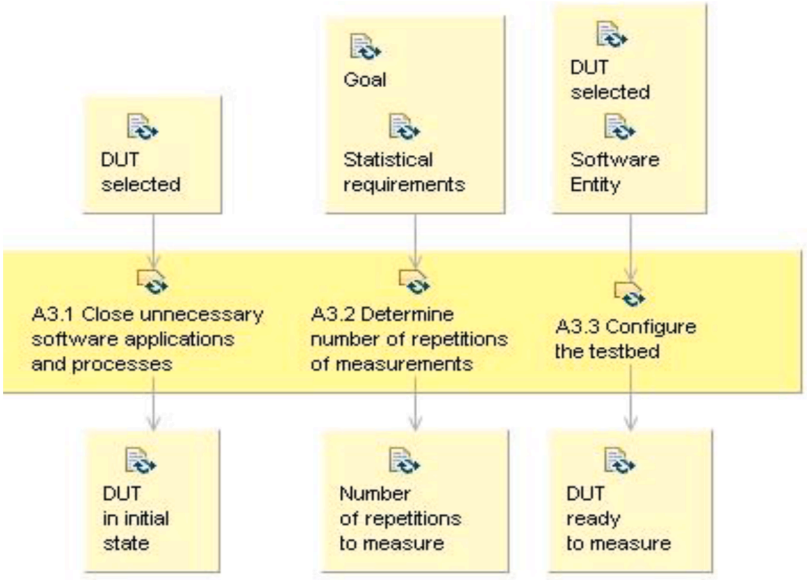


Fig. 4. Phase III: Measurement environment preparation.

Table 5
Roles and their responsibilities in Phase III.

Roles Activities	Client (C)	Measurement Analyst (MA)	Measurement Performer (MP)	Data Analyst (DA)
A3.1 Close unnecessary software applications and processes			PP	
A3.2 Determine number of repetitions of measurements		AP	PP	
A3.3 Configure the testbed			PP	

were defined in the first phase. Measuring the energy consumed by the selected Software Entities is the first activity of this phase (Activity A4.1). Once the measurement is completed, the test bed should be cleaned, to avoid affecting the power consumption when another test case is run.

After that, it is time to collect the raw energy consumption data taken from the measuring instrument (Activity A4.2). Later, the data obtained will be processed to make its analysis easier. When storing the results of each test, the relevant information such as the details of the DUT should be recorded, as should the definition of the test cases, the current configuration, the start and end time, or the power monitor trace itself.

In Phase IV, only the role of Measurement Performer participates, performing both activities as primary performer. Table 6 summarizes this information.

3.2.5. *Phase V: test case data analysis*

From this phase onwards begins the analysis of the energy consumption data obtained by the measuring instrument. The main goal of Phase V is the processing and analysis of the energy consumption data of each of the test cases that were defined in the first phase. This phase is composed of two different activities, which are summarised in Fig. 6.

The first activity focuses on the preparation of the raw data obtained by the measuring instrument (Activity A5.1). The steps to be performed in this activity depend on the source of the data, but it is crucial to achieve a transformation of the raw data into useful information for

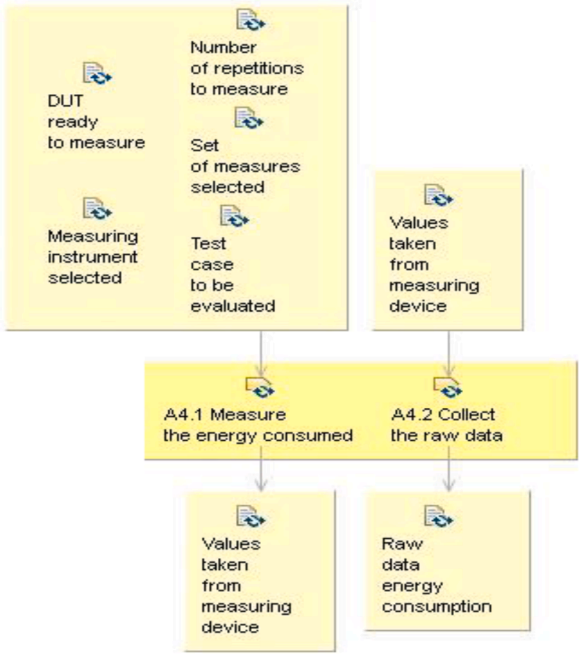


Fig. 5. Phase IV: Perform the measurements.

Table 6
Roles and their responsibilities in Phase IV.

Roles Activities	Client (C)	Measurement Analyst (MA)	Measurement Performer (MP)	Data Analyst (DA)
A4.1 Measure the energy consumed			PP	
A4.2 Collect the raw data			PP	

performing an analysis. This process of data transformation is known as Data Wrangling. The most outstanding tasks to be performed in Data Wrangling, according to Kandel, S. et al., [34] are:

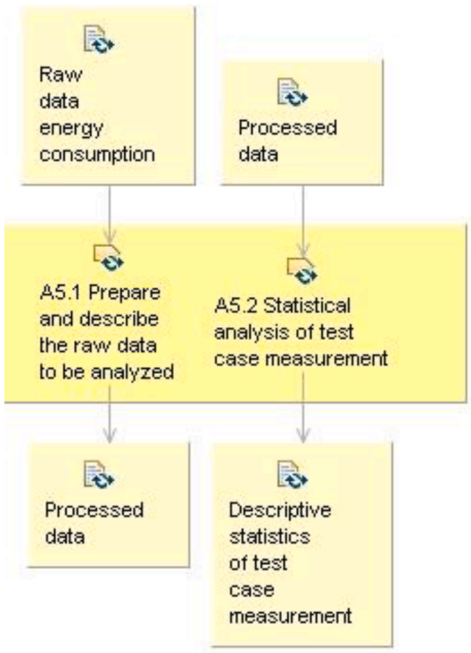


Fig. 6. Phase V: Test case data analysis.

- *Data formatting*: reformatting and integrating data from different sources so that they can be analysed correctly.
- *Correcting erroneous values:* Once the data has been formatted, data preparation begins. Data preparation includes the detection of outliers, the imputation of missing values and the resolution of duplicate records. For the identification of possible outliers that may be present in the samples of the measurements, we recommend the use of robust parametric methods such as the median of the absolute deviations from the median (MADN) [35,36].
- *Validating the measurements:* check that each of the measurements performed is correct. To find unusual measurements, you can use the interquartile range method (IQR). With this method, all values that fall below Q1 - 1.5*IQR or above Q3 + 1.5*IQR, where Qi is the quartile, are considered extraneous or incorrect. Another method of identifying incorrect measurements is to use a confidence interval. A problem inherent in defining a confidence interval, however, is that it is necessary to have made a large number of measurements beforehand.

The next step to be performed, once the data have been processed, is the statistical analysis of the values obtained from the measurements of the defined test cases (Activity A5.2). To carry out the analysis, the descriptive statistics for each test case analysed need to be calculated. To obtain the most complete information available on energy consumption, we suggest the calculation of the following descriptive statistics: on the one hand, standard descriptive statistics (maximum and minimum value, range, mean, standard deviation, variance or interquartile range) and, on the other hand, the robust descriptive statistics such as median, trimmed mean, winsorised mean or median absolute deviation. It is not compulsory to calculate all of the descriptive statistics mentioned. We need choose only those that adapt to the statistical analysis that we are going to carry out.

Table 7 shows the roles that have participated in this phase, following the levels of relationship between the activities and the roles identified in SPEM.

3.2.6. Phase VI: software entity data analysis

Once we have analysed the energy consumption data of the test cases, we will be able to determine how much energy was consumed when the Software Entity was executed in the DUT. As a result of this phase, we will carry out an analysis of the information on energy consumption, based on the goal defined at the beginning of the process of measuring the energy efficiency of a software. To that end, there are two activities in this phase, which are summarised in Fig. 7.

Table 7
Roles and their responsibilities in Phase V.

Roles Activities	Client (C)	Measurement Analyst (MA)	Measurement Performer (MP)	Data Analyst (DA)
A5.1 Prepare and describe the raw data to be analysed				PP
A5.2 Statistical analysis of test case measurement				PP

The first activity in this phase consists of calculating the energy consumed by the execution of the Software Entity (Activity A6.1). As mentioned above, the software energy consumption depends mainly on the DUT used. Hence, to calculate the energy required for the running of the software, it is necessary to subtract the baseline energy consumption of the DUT (Activity A2.4) from the average energy of the Software Entity measurements. Before we can subtract the baseline energy consumption from the DUT, we must adjust it to the software measurement performed. The adjusted baseline energy consumption is calculated by dividing the average energy of the baseline by the average duration of the baseline, and multiplying it by the average duration of the measurement:

$$\text{Adjusted Baseline Energy Consumption} = \frac{\overline{EC}\ \text{Baseline}}{\overline{T}\ \text{Baseline}} * \overline{T}\ \text{Measurement}$$

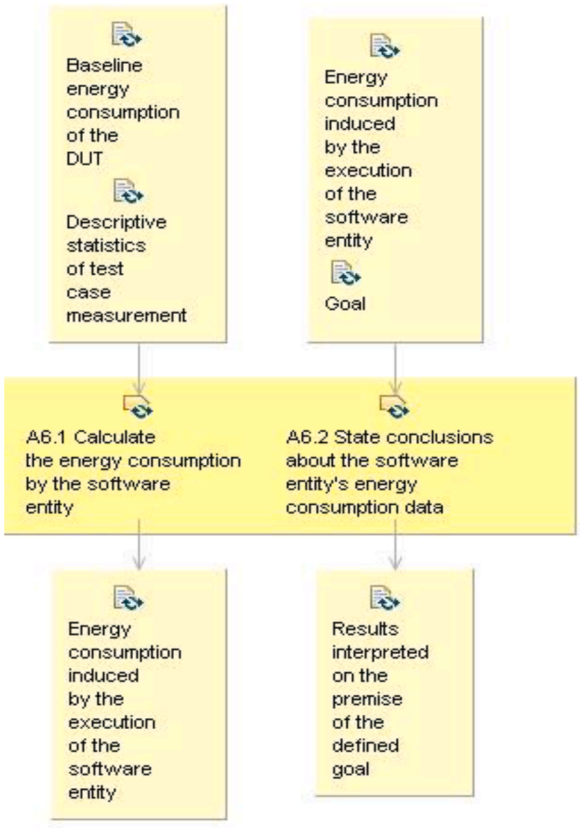


Fig. 7. Phase VI: Software entity data analysis.

The task of subtracting the baseline energy consumption of the DUT from the average energy of the Software Entity's measurements may not be performed if we provide relative information on energy consumption. That is, if we classify or sort according to the energy consumptions of each scenario that has been measured in the same DUT, all the results will have been equally affected by the baseline energy, and the classification will not vary.

The last activity of this phase deals with interpreting the data of the energy consumed by the Software Entity analysed and with establishing some conclusions (Activity A6.7). As a result of this activity, information is obtained on energy efficiency in response to the objective defined. It is essential to have fulfilled all the requirements proposed by the *Client* at the beginning of the process if the objective is to be completely satisfied.

The involvement of a *Data Analyst* and the participation of the *Measurement Analyst* are required in performing the tasks of analysing the energy consumption of the selected Software Entity. In addition, and as with the previous phases, in Table 8 we show the implication of each role in each activity, using the SPEM relationship levels.

3.2.7. Phase VII: reporting the results

Finally, the last phase is about documenting the study performed, describing the entire process followed, along with the results on the energy consumption of the software that had been extracted. Fig. 8 contains all the activities, inputs and outputs of this phase.

The first activity of this phase focuses on the development of a laboratory package (LP) intending to achieve repeatability of the experiment performed (Activity A7.1). The main objective of LPs is to be an instrument for supporting knowledge transfer, as well as for conducting replications; they should support all activities in the experimental process, and not only the implementation. Laboratory packages should contain all the information and materials required to replicate an experiment or case study [37,38]. The content of an LP should not be static; it needs to be adapted to the needs of the researcher and the limitations of the experiment. In order to develop a correct LP, we suggest that the proposal put forward by Jedlitschka and D. Pfahl [37,38] be followed, in which the content and structure of the laboratory packages for software engineering experiments are indicated. Considering the indications of these authors, the LP should include the following information:

- *Planning:* description of each of the activities to be carried out, and the order in which they are to be performed. It is also recommended that the estimated workload for the replicant experimenter be indicated.
- *Study conception*: description of the high-level attributes that are studied by the experiment, together with its goals. In addition, the variables used in the experiment should be shown.
- *Experimental design*: information about the design of the experiment. It should include details on what the subject of the evaluation will be and in what cases.

Table 8
Roles and their responsibilities in Phase VI.

Roles Activities	Client (C)	Measurement Analyst (MA)	Measurement Performer (MP)	Data Analyst (DA)
A6.1 Calculate the energy consumption by the Software Entity				PP
A6.2 State conclusions about the Software Entity's energy consumption data		AP		PP

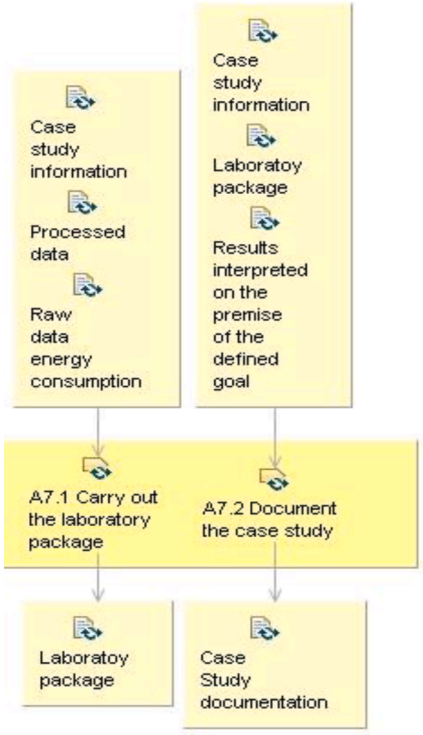


Fig. 8. Phase VII. Reporting the results.

- *Operation:* information for the creation of the laboratory environment to be used. This includes specific software engineering objects (such as programs, specifications, or test cases) and instruments used for measurement and analysis of the data.
- *Analysis:* specification of the data wrangling process followed, as well as the analysis methods applied. A report of the experiment should be included, and the analysis should conclude with a high-level interpretation of the results. In addition, the raw data should be included in standard format, to allow other researchers to repeat all the analysis activities of the results.

The last activity of the process for the measurement of the energy efficiency of the software is to make detailed documentation, in which the whole process is explained, along with the results obtained in the study (Activity A7.2). The main difference with the laboratory package is that while that is oriented to other researchers who want to replicate the experiment, the documentation here, containing the information that has been obtained, is directed at the *Client* and other stakeholders. The LP can be considered as a piece of this documentation. To report a study where we evaluate the energy consumption of the software, we can use the guidelines proposed by Jedlitschka and Pfahl [39].

Table 9 shows the roles that have participated in this phase, along with the involvement of each role in each activity using the SPEM relationship levels.

Table 9
Roles and their responsibilities in Phase VII.

Roles Activities	Client (C)	Measurement Analyst (MA)	Measurement Performer (MP)	Data Analyst (DA)
A7.1 Carry out the laboratory package		PP	AP	AP
A7.2 Document the case study		PP		

3.3. Considerations for the validity of energy consumption measurements of software

Although the process described above provides a solid basis for carrying out energy consumption measurements, the assumptions that may occur, and which could jeopardise the validity of the measurements, must be identified. Table 10 shows the assumptions that can threaten the validity of energy consumption measurements of software.

4. Application of the software energy consumption measurement process

This section presents the application of the process for evaluating the energy efficiency of the software, which was defined in the previous section. To demonstrate that this process can be adapted to any study in which energy consumption is evaluated, two case studies are presented. In the first (Case Study A), a hardware device was used to make accurate and real consumption measurements. In the second (Case Study B), so as to illustrate a software-based approach, we adapted to our process the case study performed by Chandra et al. [41].

Table 10
Considerations for the validity of energy consumption measurements.

ID	Name	Description
C.1	Sampling interval	The frequency with which samples of the power consumed are provided must be taken into consideration. If the frequency is too low, this might lead to an underestimation of the energy consumed, due to the high frequency of the hardware components [33]
C.2	OS effects and interaction with other software	The energy used by the operating system (OS) is usually included in the energy consumption measurements. In addition, other applications or services of the operating system may be activated during the measurement. We mitigate this threat by performing a large number of measurements and by obtaining the baseline of DUT consumption.
C.3	Laboratory temperature	Not having direct control over the temperature in the laboratory where measurements are performed can be harmful to measuring accurate energy consumption. This risk can be mitigated by repeating the measurements several times [40].
C.4	Experiment settings	The choice of the Software Entity to be analysed, together with the creation of the test cases to be run to measure energy consumption, can be considered a limitation of the experiment. Hence, we cannot generalise the results obtained for other Software Entities, although they may be useful for future experiments.
C.5	Measuring instrument	There is an inevitable dependence on the measuring instrument in terms of accuracy and detail of measurements, as these may vary when a different measuring instrument is used. However, whenever possible it is useful to provide comparisons about different instruments by clearly stating their settings.
C.6	DUT Specificity	One of the main factors that can influence energy consumption measurements is the configuration of the DUT in which the software being evaluated is running, since the energy consumption obtained is specific to the DUT used. It is therefore possible to use the results as absolute values, if the DUT used is too similar to the one where the software will normally be run. Otherwise, the values obtained must be considered relative, and should serve to determine in which test cases there is a greater or lesser consumption of energy

4.1. Case study A: using a hardware device to measure energy consumption

In this case study, a hardware device is used, which allows us to obtain accurate and real measurements. The activities that have been carried out following the process are detailed below:

- Phase I. Scope definition:

The objective of this case study is to find out how software changes can alter the energy consumption behaviour of the software. With this feedback, when we then develop a new version of the software we will be able to avoid carrying out the changes that have had the most negative impact on the software's energy consumption (Activity A1.1).

The first step is to select the Software Entity class to be analysed in the case study (Activity A1.2). In this case, we chose Apache Hadoop, which is a framework that enables [evenly] distributed storage and the processing of large data sets.

The next step is to choose the individual Software Entities to be measured (Activity A1.3). These software products must be available for installation and execution, and the source code must be accessible if the changes are to be analysed. In addition, the selected Software Entities must all include at least the same functionality. Given these criteria, we will analyse the energy consumption of the three different versions of Apache Hadoop shown in Table 11.

The last activity to be carried out in this phase is the creation of the test cases to be executed (Activity A1.4). In this case study, one of the test cases uses the Hadoop Distributed File System (HDFS) and the other does not. These two test cases defined are detailed below:

- *Estimation of the Pi number:* this test case runs a REDUCE program for maps that estimates *Pi* using a quasi-Monte Carlo method. The program takes two inputs: the number of maps and the number of samples. We run with 50 maps and 5000 samples per map.
- *Count the words of the novel "Don Quixote of La Mancha":* this algorithm counts the number of times each different word appears in the famous novel, using Hadoop's HDFS.

The outcome of this phase is that we have now defined the test cases that are to be executed in each of the selected Hadoop versions, in order to analyse if the changes in each of the versions have affected the power consumption.

- ***Phase II. Measurement Environment Setting:***

The measurement environment we used to evaluate the energy efficiency of the software is FEETINGS (Framework for Energy Efficiency Testing to Improve eNviromental Goals of the Software) [13], a framework for measuring and analysing the energy consumption of a software application. FEETINGS consists of two main elements: (i) EET, which is a device that allows the energy consumption of a set of hardware components to be measured when a Software Entity is executed in the DUT (Device Under Test); and (ii) ELLIOT, which is the software application that processes and analyses the data collected by EET (Activity A2.1).

We also used the SonarCloud platform, to be able to establish what changes the software may have undergone between its different versions, as selected in Phase I; this platform is a cloud service for the continuous inspection of the quality of the code, providing detailed

Table 11
Selected software entities in case study A.

Versions	Last Modified
Apache Hadoop 2.2	Nov. – 14
Apache Hadoop 2.6.5	Oct. – 16
Apache Hadoop 3.0.3	Jun. – 18

information on software maintainability measurements.

The chosen Software Entities will be executed in a DUT without any special processing or storage capabilities, so that the results obtained are of more general application (Activity A2.2). The specifications of the DUT are provided in Table 12.

Concerning the set of measures to be used in this case study, we can identify not only the energy consumption obtained by the measuring instrument, but also the measurements obtained from the use of the SonarCloud tool, such as the Total Lines of Code (TLOC), the Cyclomatic Complexity (CC), the Percentage of Comments in the Code (PCC), and the Percentage of Duplicate Code lines (PDC) (Activity A2.3).

In this case study it was not necessary to obtain the basal power consumption of the DUT, since our goal was to perform a classification of the different versions of Apache Hadoop based on consumption. Moreover, all scenarios were to be executed in the same DUT, so it was not necessary to isolate the power consumption of the operating system, as this consumption would be the same in all executions, hence, there would be no distortion of the classification obtained - as it would affect all results equally.

- ***Phase III. Measurement Environment Preparation:***

Before starting this phase, we checked that no other software was running in the background. Any process or software not related to the Software Entity to be analysed must be closed (Activity A3.1).

Another aspect to be defined in this phase is the number of repetitions to be performed for each measurement of a test case (Activity A3.2). We consider that each test case should be measured 20 times, since the fact of being in a controlled environment is enough to mitigate the effect of any other processes that could be executed at the same time.

Once it has been checked, the DUT is configured and the chosen Software Entity is installed (Activity A3.3). Apache Hadoop can be set on a single machine, called a Single Node or Pseudo-Distributed Cluster, (since it simulates a complete cluster environment for testing Hadoop applications, using its HDFS module, which is a distributed file system that provides high-performance access to application data). Alternatively, Apache Hadoop can be run on different machines in a distributed manner, which is known as a Multi-Node or Fully Distributed Cluster. In our case study, to carry out real measurements of energy consumption, we have chosen to configure it on a single machine (Pseudo-Distributed Cluster).

- ***Phase IV. Perform the Measurements:***

In this phase, power consumption measurements will be made for each of the Apache Hadoop test cases defined (Activity A4.1). The measuring instrument used is the EET device. This measuring instrument provides information on the power consumption of the following hardware components of the DUT: processor, hard disk, graphics card and the DUT as a whole.

After the execution of each test case, the results of the energy measurement are recorded in a log file, see Fig. 9 (Activity A4.2).

- ***Phase V. Test Case Data Analysis:***

During this phase, the analysis of the energy consumption data for each of the test cases is carried out, using the ELLIOT analysis tool. The first activity is the preparation of the raw data, which has been obtained from the EET device in the previous phase. Here, the average values of each of the measurements of the Apache Hadoop test cases are calculated, the outliers are identified and eliminated, and the obtained values are checked for validity (Activity A5.1).

Once the data have been processed and prepared, the descriptive statistics of the values obtained are calculated (Activity A5.2), as can be seen in Fig. 10.

- ***Phase VI. Software Entity Data Analysis:***

In this phase, the results obtained for each of the Software Entities analysed (versions of Apache Hadoop) are compared (Activity A6.1). As we can see in Fig. 11, in both the test cases, the power consumption increases in the most recent versions.

With the results of energy consumption obtained, the next step is to analyse the differences that exist between each of the versions, seeking to determine why the consumption increases in the newest versions of the software (Activity A6.2).

- ***Phase VII. Reporting the results:***

Finally, all the results obtained, along with the process followed to achieve them, are documented (Activity A7.1). In addition, the laboratory package[1] of the experiment is created, so that it can be analysed and replicated by other researchers (Activity A7.2). The LP includes all the raw data obtained from EET and the information processed by the ELLIOT tool. Also attached are the templates, filled in with the information from the study that has been performed.

4.2. Case study B: using a software tool to estimate energy consumption

In this case study, the energy efficiency evaluation process will be adapted to the experiment presented by Chandra et al. [41], whose objective is to evaluate the energy efficiency of programming languages, using a software tool to estimate energy consumption.

Our purpose is to demonstrate that the process is valid for energy consumption analysis experiments, regardless of the measurement approach followed. Below we detail the activities carried out in each phase, using information provided by the authors of the paper [41].

- ***Phase I. Scope definition:***

The main goal of this experiment is to find out which programming language consumes the least amount of energy (Activity A1.1 and A1.2). For this purpose, the authors of this experiment focused on analysing the power consumption of three standard programming languages: Visual Basic, Java and C#.Net. These programming languages are the Software Entities that they selected for evaluation (Activity A1.3).

They then defined the test cases that would be executed to evaluate the Software Entities (Activity A1.4). As test cases, they implemented four sorting algorithms (Bubble sort, Insertion sort, Selection sort and Quicksort) in the different programming languages. Fig. 12 shows an excerpt of the template used for the definition of the test cases to be evaluated.

- ***Phase II. Measurement Environment Setting:***

The measuring instrument used is the Joulemeter tool (Activity A2.1) [42]. This software tool allows the power consumption (in Watts/s) of a system to be estimated when it is running a software application. The sorting algorithms implemented in the three programming languages were executed in the DUT with these specifications: Intel Core i5 and 4th

Table 12
Specifications of the DUT used in case study A.

Hardware	Motherboard: Asus M2N-SLI Delux Processor: AMD Athom x2 6000+ HDD: Seagate BarraCuda 7200 rpm 500 GB RAM: 4×1 GB 666 MHz Kingston Graphics Cards: Nvidia GForce 8600 GTS
Operating system	Xubuntu 16.04.2 LTS

[1] https://zenodo.org/record/3669902#.XkpanS1DlhE

```
sample, run time, sensor 5A      samples, time, HDD1, HDD2,  Graf1, Graf2, Proc1, Proc2,
1,  519,  57.439,  259.405       1 ,200 ,1.82304687 ,14.43476581 ,0.39062500 ,1.56250000 ,1.22070312 ,2.00195312
2,  1021,  48.935,  264.200      2 ,209 ,1.77421879 ,14.43476581 ,0.48828125 ,1.51367187 ,1.22070312 ,2.05078125
3,  1524,  50.299,  404.395      3 ,220 ,1.77421879 ,14.48359394 ,0.48828125 ,1.46484375 ,1.36718750 ,2.00195312
4,  2026,  55.146,  441.039      4 ,229 ,1.82304687 ,14.38593769 ,0.43945312 ,1.51367187 ,1.26953125 ,2.05078125
5,  2528,  50.282,  421.041      5 ,238 ,1.82304687 ,14.53242206 ,0.39062500 ,1.51367187 ,1.31835937 ,2.05078125
6,  3030,  52.200,  445.628      6 ,255 ,1.82304687 ,14.43476581 ,0.34179687 ,1.46484375 ,1.22070312 ,2.09960937
7,  3532,  53.867,  420.934      7 ,264 ,1.77421879 ,14.43476581 ,0.43945312 ,1.51367187 ,1.22070312 ,1.90429687
8,  4033,  57.730,  429.409      8 ,274 ,1.92070312 ,14.43476581 ,0.43945312 ,1.46484375 ,1.17187500 ,2.05078125
9,  4536,  52.737,  407.655      9 ,283 ,1.82304687 ,14.43476581 ,0.43945312 ,1.46484375 ,1.26953125 ,2.09960937
10,  5038,  51.353,  372.158     10 ,292 ,1.82304687 ,14.48359394 ,0.34179687 ,1.46484375 ,1.36718750 ,2.00195312
11,  5539,  52.696,  255.360     11 ,303 ,1.82304687 ,14.43476581 ,0.43945312 ,1.51367187 ,1.22070312 ,2.05078125
12,  6041,  52.786,  357.395     12 ,317 ,1.87187500 ,14.53242206 ,0.43945312 ,1.51367187 ,1.22070312 ,2.05078125
13,  6544,  51.616,  240.784     13 ,326 ,1.82304687 ,14.43476581 ,0.43945312 ,1.41601562 ,1.36718750 ,2.09960937
14,  7047,  53.151,  284.539     14 ,335 ,1.87187500 ,14.48359394 ,0.34179687 ,1.46484375 ,1.31835937 ,2.05078125
15,  7548,  55.436,  368.523     15 ,346 ,1.77421879 ,14.43476581 ,0.48828125 ,1.51367187 ,1.31835937 ,2.14843750
16,  8050,  60.755,  467.808     16 ,355 ,1.82304687 ,14.38593769 ,0.39062500 ,1.56250000 ,1.26953125 ,2.05078125
17,  8558,  65.908,  468.197     17 ,364 ,1.92070312 ,14.53242206 ,0.34179687 ,1.51367187 ,1.26953125 ,2.09960937
18,  9059,  55.772,  439.902     18 ,380 ,1.87187500 ,14.43476581 ,0.43945312 ,1.51367187 ,1.26953125 ,2.09960937
19,  9563,  50.879,  428.235     19 ,390 ,1.72539067 ,14.48359394 ,0.39062500 ,1.51367187 ,1.26953125 ,2.09960937
20,  10065,  50.809,  373.40     20 ,399 ,1.82304687 ,14.48359394 ,0.39062500 ,1.51367187 ,1.31835937 ,2.09960937
```

Fig. 9. Excerpt of a log file of Apache Hadoop's energy measurement generated by the EET device.

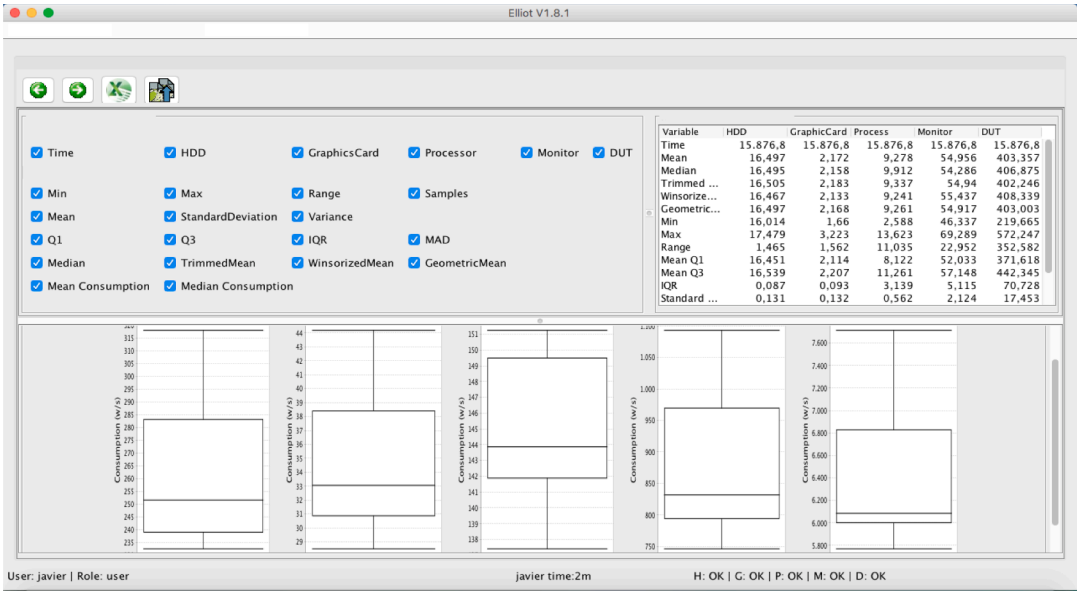


Fig. 10. Data analysis of a test case with ELLIOT tool.

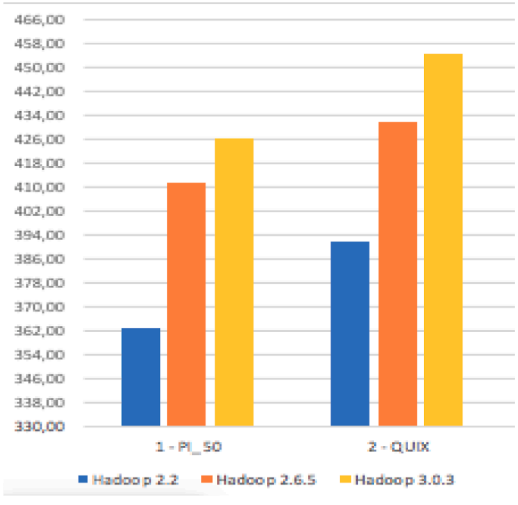
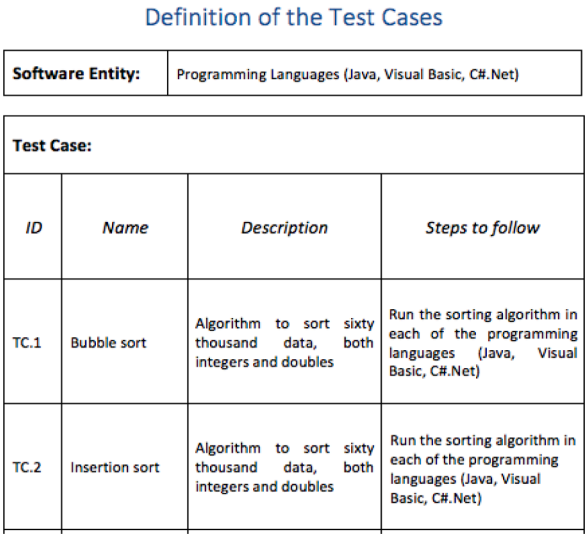


Fig. 11. Total energy consumed by each version of Apache Hadoop.

generation CPU with Windows 8.1 (Activity A2.2). The baseline energy consumption of the DUT was not considered in this experiment. Fig. 13 presents the definition of the measurement environment, including the



Definition of the Test Cases

Software Entity:	Programming Languages (Java, Visual Basic, C#.Net)

Test Case:

ID	*Name*	*Description*	*Steps to follow*
TC.1	Bubble sort	Algorithm to sort sixty thousand data, both integers and doubles	Run the sorting algorithm in each of the programming languages (Java, Visual Basic, C#.Net)
TC.2	Insertion sort	Algorithm to sort sixty thousand data, both integers and doubles	Run the sorting algorithm in each of the programming languages (Java, Visual Basic, C#.Net)

Fig. 12. Excerpt from the template for defining the test cases.

Definition of the Measurement Environment

Measuring Instrument:	
Name	*Description*
Joulemeter tool	This software tool allows estimating the power consumption (in Watts/s) of a system when running a software application. *Joulemeter. https://www.microsoft.com/en-us/research/project/joulemeter-computational-energy-measurement-and-optimization/*

Device Under Test (DUT)		
Name	*Specifications Hardware*	*Specifications Software*
Workstation 1	Intel Core i5 and 4th generation CPU	Windows 8.1 (Version 6.3.9600)

Set of measures selected		
ID	*Measure*	*Description*
M1	Energy Consumption (EC)	The energy consumption obtained by the measurement instrument

Fig. 13. Template for definition of the measurement environment.

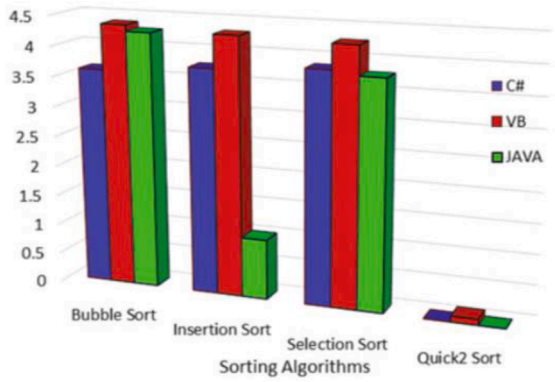


Fig. 14. Power consumed measurement.

information about the measuring instrument, the DUT and the recorded measurement (Activity A2.3).

- ***Phase III. Measurement Environment Preparation:***

The researchers decided that the consumption measurements of each algorithm in each of the languages should be executed four times on the same data set (Activity A3.2).

They implemented the four sorting algorithms in the three programming languages. They then prepared a set of elements with more than sixty thousand data, both integers and doubles. The algorithms had to sort this data set (Activity A3.3).

- ***Phase IV. Perform the Measurements:***

They performed four tests with the same data set and took their average; the aim was to find the average power consumption per second in each programming language for all the ranking algorithms defined above (Activity A4.1 and A4.2).

- ***Phase V & VI. Data Analysis:***

Once each of the algorithms had been measured, the values were calculated based on the power consumption represented in watts per second (Activity A5.2 and A6.1). It was found that the most efficient sorting algorithm, in terms of energy, is Quicksort, followed by Bubble sort. In contrast, the most energy-demanding algorithm is the Insertion sort. To implement these sorting algorithms, the most energy-inefficient programming language is Visual Basic. However, with Java and C#.Net implementations the results are more similar. Moreover, the classification of double-type data elements consumes more power than the integer type data set (Activity A6.2).

- ***Phase VII. Reporting the results:***

The researchers reported the results found in their paper [41] (Activity A7.2) (Fig. 14). However, the authors did not indicate whether they had a laboratory package that would enable the experiment to be replicated.

5. Conclusions and future work

The development of environmentally friendly software is no trivial project. And, in order to determine how efficient software is from an energy point of view, it is essential to be able to evaluate the energy consumed when it is running. However, simply having measuring instruments that allow us to fully analyse consumption may not in itself be enough. To ensure that the results obtained are correct and appropriate it is imperative to follow the correct steps. This has inspired us to present, in this paper, our proposal for a process to evaluate the energy efficiency of software. By the process we propose it is possible to improve reliability and consistency when measuring energy consumption. To support the systematic development, management and growth of our proposed process by using a standardized representation, we have chosen to use SPEM 2.0; this has also allowed us to generate documentation in a standard format that is available to anyone who wants to consult it on the web. This process covers all the necessary phases in carrying out this type of studies, from the definition of the scope and configuration of the environment, through the performance of the measurements, to the subsequent analysis of the data obtained and the reporting of the results. Furthermore, this process was designed to be valid with any measuring instrument used, regardless of whether it follows the hardware-based or software-based approach. The phases of the process are based on the grouping of the different activities defined in the Green Mining Methodology. Moreover, to define some aspects or artefacts of the process, we have based ourselves on well-known approaches to software measurement and good practices related to green software proposed by other authors.

To illustrate this process, we have provided two examples of how to use the defined process to achieve more reliable software energy consumption results, one of them using a measuring instrument following the hardware-based approach, and another using an estimation software tool. The process is therefore shown to be applicable to both cases.

The process defined allows us to analyse the energy efficiency of software, and so enables researchers to obtain greater control over the measurements made, guaranteeing the reliability and consistency of the same. It also means that the studies we carried out can be easily replicated and the results obtained can be compared with those of other studies.

This contribution, we believe, helps software professionals to be aware that there are processes and tools to evaluate the energy efficiency of the software applications they develop. They can thus develop software that is environmentally friendly.

If we analyse the impact that the defined process can have on the environment in the software life cycle with regards to the three order effects defined in [43], we can conclude that the proposed process is mainly related to second-order or indirect effects, which are the effects arising from the usage of software. This is because the defined process

allows to evaluate and measure the energy consumed by a software when it is running. The process also has some influence on the direct or first-order effects of software, since it can be applied to measure the energy consumed by the tools involved in the design and production of software. Finally, our process may involve third-order effects to improve the software as a result of a maintenance task.

As future work, our proposed process will be validated through more studies conducted in real environments, thereby allowing us to improve the process and the activities to be carried out. In addition, the process will be improved by a more detailed definition of all activities and artefacts. In addition, the process will be completed with new guidelines that will make it easier for researchers to apply the process in their energy consumption measurements.

CRediT authorship contribution statement

Javier Mancebo: Conceptualization, Visualization, Validation, Investigation, Writing - original draft. **Félix García:** Conceptualization, Writing - review & editing, Supervision. **Coral Calero:** Conceptualization, Methodology, Writing - review & editing, Supervision.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This work was part of the BIZDEVOPS-Global (RTI2018-098309-B-C31), supported by the Spanish Ministry of Economy, Industry and Competitiveness and European FEDER funds, and was also part of the SOS project (No. SBPLY/17/180501/000364), funded by the Department of Education, Culture and Sports of the Directorate General of Universities, Research and Innovation of the JCCM (Regional Government of the Autonomous Region of Castilla-La Mancha).

References

[1] A. Andrae, Total consumer power consumption forecast, Nordic Digital Bus. Summit 10 (2017).
[2] A. Andrae, Prediction studies of electricity use of global computing in 2030, Int. J. Sci. Eng. Invest. 8 (2019) 27–33.
[3] J. Vidal, Tsunami of data'could consume one fifth of global electricity by 2025, Climate Home News 11 (2017).
[4] A.C. Moises, A. Malucelli, S. Reinehr, Practices of energy consumption for sustainable software engineering, in: 2018 Ninth International Green and Sustainable Computing Conference (IGSC), IEEE, 2018, pp. 1–6.
[5] G. Procaccianti, P. Lago, S. Bevini, A systematic literature review on energy efficiency in cloud software architectures, Sustain. Comput.: Inform. Syst. 7 (2015) 2–10.
[6] C. Calero, et al., 5Ws of green and sustainable software, Tsinghua Sci. Technol. 25 (3) (2019) 401–414.
[7] C. Calero, M. Piattini, Puzzling out software sustainability, Sustain. Comput.: Inform. Syst. 16 (2017) 117–124.
[8] M. Dick, S. Naumann, N. Kuhn, A model and selected instances of green and sustainable software. What Kind of Information Society? Governance, Virtuality, Surveillance, Sustainability, Resilience, Springer, 2010, pp. 248–259.
[9] T.A. Ghaleb, Software energy measurement at different levels of granularity, in: 2019 International Conference on Computer and Information Sciences (ICCIS), IEEE, 2019, pp. 1–6.
[10] G. Pinto, F. Castor, Energy efficiency: a new concern for application software developers, Commun. ACM 60 (12) (2017) 68–75.
[11] A. Bourdon, A. Noureddine, R. Rouvoy, L. Seinturier, Powerapi: a software library to monitor the energy consumed at the process-level, ERCIM News 2013 (92) (2013).
[12] E. Jagroep, J.M.E. van der Werf, S. Jansen, M. Ferreira, J. Visser, Profiling energy profilers, in: Proceedings of the 30th Annual ACM Symposium on Applied Computing, 2015, pp. 2198–2203.
[13] J. Mancebo, et al., EET: a device to support the measurement of software consumption, in: Proceedings of the 6th International Workshop on Green and Sustainable Software, 2018, pp. 16–22.
[14] E.A. Jagroep, et al., Software energy profiling: comparing releases of a software product, in: Proceedings of the 38th International Conference on Software Engineering Companion, ACM, 2016, pp. 523–532.
[15] C. Sahin, et al., Initial explorations on design pattern energy usage, in: 2012 First International Workshop on Green and Sustainable Software (GREENS), IEEE, 2012, pp. 55–61.
[16] A. Hindle, Green mining: a methodology of relating software change and configuration to power consumption, Empir. Software Eng. 20 (2) (2015) 374–409.
[17] E. Kern, et al., Sustainable software products—towards assessment criteria for resource and energy efficiency, Future Gen. Comput. Syst. 86 (2018) 199–210.
[18] A. Noureddine, A. Bourdon, R. Rouvoy, L. Seinturier, A preliminary study of the impact of software engineering on greenit, in: 2012 First International Workshop on Green and Sustainable Software (GREENS), IEEE, 2012, pp. 21–27.
[19] C. Wohlin, et al., Experimentation in Software Engineering, Springer Science & Business Media, 2012.
[20] N. Fenton, J. Bieman, Software Metrics: a Rigorous and Practical Approach, CRC press, 2014.
[21] B. Henderson-Sellers, Method engineering for OO systems development, Commun. ACM 46 (10) (2003) 73–78.
[22] Software & Systems Process Engineering Metamodel specification (SPEM) Version 2.0, 2008.
[23] V.R. Basili, D.M. Weiss, A methodology for collecting valid software engineering data, IEEE Trans. Softw. Eng. (6) (1984) 728–738.
[24] V.R. Basili, H.D. Rombach, The TAME project: Towards improvement-oriented software environments, IEEE Trans. Softw. Eng. 14 (6) (1988) 758–773.
[25] G. Caldiera, V.R. Basili, H.D. Rombach, Goal question metric paradigm, Encyclopedia Software Eng. 1 (1994) 528–532.
[26] D.R. van Solingen, E.W. Berghout, The Goal/Question/Metric Method: a Practical Guide for Quality Improvement of Software Development, McGraw-Hill, 1999.
[27] R.E. Park, W.B. Goethert, W.A. Florac, Goal-Driven Software Measurement. a Guidebook, Carnegie-Mellon Univ Pittsburgh, 1996.
[28] W. Goethert and J. Siviy, "Applications of the Indicator Template for Measurement and Analysis," Carnegie-Mellon Univ Pittsburgh PA Software Engineering INST2004.
[29] PSM: Practical Software and Systems Measurement, A Foundation for Objective Project Management, 2000.
[30] ISO/IEC/IEEE International Standard - Systems and Software Engineering–Measurement Process, 2017, ISO/IEC/IEEE 15939, 2017, pp. 1–49.
[31] F. García, et al., Effective use of ontologies in software measurement, Knowl. Eng. Rev. 24 (1) (2009) 23–40.
[32] J. 200:2012, The International Vocabulary of Metrology—Basic and General Concepts and Associated Terms (VIM), 2012. Available, http://www.bipm.org/vim.
[33] E. Jagroep, et al., Energy efficiency on the product roadmap: an empirical study across releases of a software product, J. Software: Evol. Process 29 (2) (2017) e1852.
[34] S. Kandel, et al., Research directions in data wrangling: visualizations and transformations for usable and credible data, Inf. Visualization 10 (4) (2011) 271–288.
[35] B. Kitchenham, et al., Robust statistical methods for empirical software engineering, Empir. Software Eng. 22 (2) (2017) 579–630.
[36] R.R. Wilcox, Introduction to Robust Estimation and Hypothesis Testing, Academic press, 2011.
[37] V.R. Basili, R.W. Selby, D.H. Hutchens, Experimentation in software engineering, IEEE Trans. Softw. Eng. (7) (1986) 733–743.
[38] A. Brooks, J. Daly, J. Miller, M. Roper, M. Wood, Replication of experimental results in software engineering, in: International Software Engineering Research Network (ISERN), 2, University of Strathclyde, 1996. Technical Report ISERN-96-10.
[39] A. Jedlitschka, D. Pfahl, Reporting guidelines for controlled experiments in software engineering, in: 2005 International Symposium on Empirical Software Engineering 2005, IEEE, 2005, 10 pp.
[40] S. Chowdhury, S. Borle, S. Romansky, A. Hindle, Greenscaler: training software energy models with automatic test generation, Empir. Software Eng. 24 (4) (2019) 1649–1692.
[41] T.B. Chandra, P. Verma, A.K. Dwivedi, Impact of programming languages on energy consumption for sorting algorithms. Software Engineering, Springer, 2019, pp. 93–101.
[42] N. Kothari, A. Bhattacharya, Joulemeter: virtual machine power measurement and management, MSR Tech. Rep. (2009).
[43] C. Becker, et al., The Karlskrona Manifesto for Sustainability Design, 2014 arXiv preprint arXiv:1410.6968.

Javier Mancebo is a Ph.D. in Computer Science at the University of Castilla-La Mancha. His research interests are software sustainability and business process management. He is a member of the Alarcos Research Group. He holds the following professional certifications: PMP, CISA, ITIL Foundation and Scrum Manager.

Félix Garcia is Professor in the Department of Information Technologies and Systems at the UCLM. He is a member of the Alarcos Research Group and his research interests include business process management, software processes and software measurement. He holds the following professional certifications: PMP, CISA and Scrum Manager.

Coral Calero is Professor in the Department of Information Technologies and Systems at the University of Castilla-La Mancha (Spain). She holds PMP certification. Her research interests include software quality, software quality models, software measurement and software sustainability. She is a member of the Alarcos Research Group.