EET: a device to support the measurement of software consumption

Javier Mancebo
Institute of Technology and Information Systems, University of Castilla-La Mancha
Ciudad Real, Spain
Javier.Mancebo@uclm.es

Hector O. Arriaga
Institute of Technology and Information Systems, University of Castilla-La Mancha
Ciudad Real, Spain
hectoromar.arriaga@uclm.es

Félix García
Institute of Technology and Information Systems, University of Castilla-La Mancha
Ciudad Real, Spain
Felix.Garcia@uclm.es

Mª Ángeles Moraga
Institute of Technology and Information Systems, University of Castilla-La Mancha
Ciudad Real, Spain
MariaAngeles.Moraga@uclm.es

Ignacio García-Rodríguez de Guzmán
Institute of Technology and Information Systems, University of Castilla-La Mancha
Ciudad Real, Spain
Ignacio.GRodriguez@uclm.es

Coral Calero
Institute of Technology and Information Systems, University of Castilla-La Mancha
Ciudad Real, Spain
Coral.Calero@uclm.es

ABSTRACT

Society is becoming more and more aware of the importance of preserving the environment, and is therefore increasingly concerned about issues related to sustainability. Software development should not remain indifferent to the need to construct software products that contribute towards sustainability, both during their creation and throughout their use. However, one of the main gaps is the difficulty of measuring the energy consumption when software is executed in order to detect, for example, parts of the software with excessive energy consumption. For that reason, a *Framework for Energy Efficiency Testing to Improve eNviromental Goals of the Software* (FEETINGS) has been proposed. In this paper, we focus on the core of the framework, that is, the *Energy Efficiency Tester* (EET), which is the measurement hardware device devoted to collecting the specific consumption data of the software under evaluation. The architecture and the main functions of EET, along with the reliability validation done with it are presented here, as well as an analysis of the data extracted from a case study conducted. The outcome of such analysis shows that EET is able to carry out stable energy consumption measurements of executed software.

KEYWORDS

Software consumption measurement, Green software, Software sustainability, measurement device.

1 INTRODUCTION

The energy consumption of Information Technology (IT) has become an important concern, due to how fast IT is growing. This has led to the appearance of an increasing number of green IT solutions [1]. Traditionally, most of the solutions and research proposals have focused on the reduction of the hardware's environmental impact (known as Green hardware). However, in recent years, software has also been identified as a source of negative impact on the environment, and several groups around the world have started to work on what is known as green software [2]. For example, in [3, 4] the authors analyzed desktop computers from different technological generations, in a range of software usage scenarios; they discovered that, depending on the software applications used, power consumption can increase up by to 20%.

However, software developers do not yet know how to develop energy-efficient software applications, and one of the main problems is the lack of knowledge about how to measure, profile, and optimize energy efficiency in software development and maintenance [5].

In the quest to address the problem outlined above, a fundamental step is to support reliable measurement of the energy consumption of software. Two main approaches for supporting the measurement of the energy consumption of software can be found:

- Software-based approaches, which do not require a great effort to be adopted; these are, however, shown to be inaccurate in their measurements, because they perform estimations of the energy consumption of a system at run time [6].
- Hardware-based approaches, using physical power meters connected to hardware devices. This approach is more promising than the software-based approaches,

because it allows accurate measurements of the energy consumed by a computer. The main difficulty of this approach is to trace the energy consumption of the software while running. An example of this approach is shown in [7], and is described in the next section.

Bearing all this in mind, we have designed FEETINGS (*Framework for Energy Efficiency Testing to Improve eNviromental Goals of the Software*). FEETINGS is a framework aimed at supporting the automatic measurement of the energy consumed by the software, when running in a PC. The framework also supports the automatic collection of the consumption data from the most relevant PC components, along with suitable processing and visualization of the collected data.

The final goal is to use the measurements carried out in order to:

- analyze the consumption of the software.
- know the behavior of the software and its different versions, to find out if these versions worsen the energy consumption of the software or not.
- identify consumption patterns which can guide the improvement of the energy efficiency of the software applications.
- recommend software changes in an effort to improve energy efficiency.

The framework must include a set of guidelines, empirically validated, which can help developers to understand the energy consumption effects of their software engineering decisions, and therefore promote energy-efficient software development. An overview of the framework is shown in Figure 1. As recognized in [5], traditionally the energy consumption strategy has been successfully addressed at low-lever layers, but better results could be obtained with software developer involvement in the process; FEETINGS has been designed with this purpose in mind.

As can be observed in Figure 1, FEETINGS is composed of two main elements: the core is the **Energy Efficiency Tester (EET)**, which is the measurement hardware device whose task it is to collect the specific consumption data of the evaluated software; the second main element is the software part, known as SEA (Software Energy Assessment).This processes the collected data, analyses them, and supports a suitable visualization of the results (according to their nature), seeking to identify the consumption patterns and suggest proper guidelines and recommendations to help practitioners to improve software energy efficiency.

This paper focuses on the EET device, by explaining its architecture and functions. We will also present the reliability validation done with EET. We consider EET to be a key element for progress in the Green in Software area. With the obtained results of the hardware component energy measurements when a piece of software is running, we will be able to establish a knowledge base to define theories, heuristics, and prediction models, as well as other tools that are useful for building energy-efficient software.

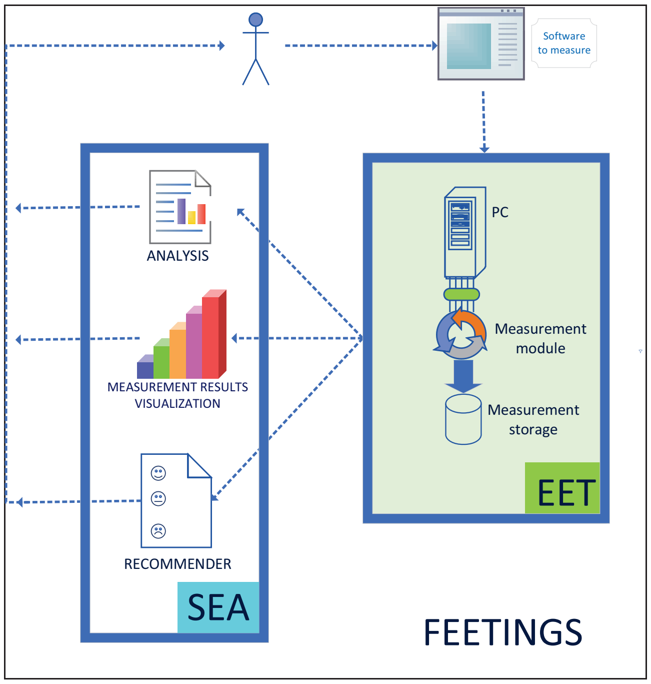


Figure 1. FEETINGS overview

The remainder of this paper is organized as follows: section 2 analyzes different proposals for measuring the energy consumption of software; in section 3 the characteristics and configuration of EET are presented; section 4 summarizes the tests carried out with the device; finally, some conclusions and future lines of work are shown in section 5.

2 RELATED WORK

It is possible to find several pieces of work that present different tools or applications for measuring or estimating energy consumption. Jalen [9] is a software-level profiling architecture, created to control the energy consumption of software applications while taking into account the code granularity. Jalen uses mathematical models to estimate the software consumption, dividing this amount by the hardware resources used.

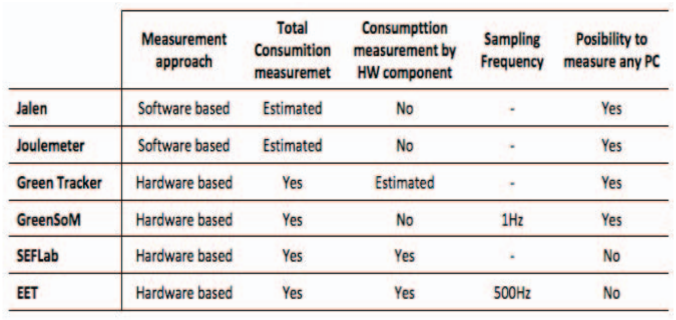
The second proposal is the Joulemeter tool (JM) from Microsoft; this is a software tool for estimating the energy consumption of the hardware resources used to execute a software application on a given PC. JM uses mathematical models to estimate the consumption on the basis of the information obtained from resources such as CPU usage, monitor brightness, etc [6]. Another related software tool is Green Tracker [10], which calculates the energy consumption of the software by estimating its CPU usage. Unlike the previous tools presented, Green Tracker requires an external device to register the energy consumption and to do the follow-up of the energy consumption of the computer. In [11], the authors propose the GreenSoM software tool. This is intended to measure software energy consumption, seeking to detect consumption peaks using a hardware data logger; this logger measures and records system consumption. That information is merged with the execution information, making it possible to generate energy graphics where the energy peaks can be observed.

This tool presents some problems, such as the sampling frequency (which is too high, around one hertz), or the fact that the consumption measurement is derived from the consumption of the whole equipment. Another paper that reports on tools for measuring consumption is [12]~~,~~ where the authors have developed the SEFLab environment to study the energy footprint of software. SEFLab allows the measurement at hardware and software levels in a given environment; it can be used by researchers to analyze the relationship between the software computational resources and the power dissipation of the underlying hardware.

From the analysis of the existing software-based proposals, we can conclude that all of them are estimation software tools, which means that the consumption measurements results are not exact. With regard to related proposals which use an external hardware to measure the exact consumption (GreenSoM and GreenTracker), these are more reliable. However, only the total consumption of the PC is obtained when executing a given software; it is not possible to differentiate between the different PC hardware components, as regards the particular consumption of each.

Finally, the SEFLab environment makes it possible to measure the energy consumption of some PC hardware components when executing a piece of software. However, SEFLab measurements have to be carried out in the specific lab conditions (specific computer), and it is not possible to make the measurements on different computers.

As regards the sampling frequency, only the GreenSoM one, which is about one hertz, is reported. Table 1 summarises these proposals, comparing them with the proposed device EET.



	Measurement approach	Total Consumition measuremet	Consumpttion measurement by HW component	Sampling Frequency	Posibility to measure any PC
Jalen	Software based	Estimated	No	-	Yes
Joulemeter	Software based	Estimated	No	-	Yes
Green Tracker	Hardware based	Yes	Estimated	-	Yes
GreenSoM	Hardware based	Yes	No	1Hz	Yes
SEFLab	Hardware based	Yes	Yes	-	No
EET	Hardware based	Yes	Yes	500Hz	No

Table 1. Energy measurement device comparison

As can be observed, EET allows the software modules which consume most energy (with precise values) to be identified for each hardware component monitored by the device (processor, hard disk and graphical card). Moreover, it is possible to configure the measurement sampling frequency reaching up to 500Hz. This is a sufficient frequency, given the type of device which measures energy consumption of specific hardware components. As a result, we obtain a new measurement data every 20 milliseconds. Finally, EET can be connected to any computer; which means it is possible to measure the same software running on different computers and hardware.

3 Energy Efficiency Tester (EET)

In this section, we describe the EET design, and how we can use the device to measure the energy consumption of the hardware components used by the software evaluated.

EET supports the measurement of the consumption of three different hardware components (processor, hard disk and graphical card) of any PC, since it is portable and easy to plug in. The information obtained by means of the EET could be used to perform an analysis of the consumption of software throughout its execution, to identify those parts of the software that consume more and less energy, and to discover which particular structures/algorithms/patterns/architectures require greater consumption in the software.

Figure 2 displays the design of the device architecture. In a nutshell, EET is composed of three main components:

1. system microcontroller for data acquisition,
2. a set of sensors, and
3. the power supply.

The microcontroller is a Mega Arduino development board, whose task it is to gather the information extracted from the different sensors and store them in a MicroSD memory. It also allows the frequency with which the device will perform the measurements to be adjusted. This is done through the software that manages the microcontroller, enabling the user to specify the number of samples needed.

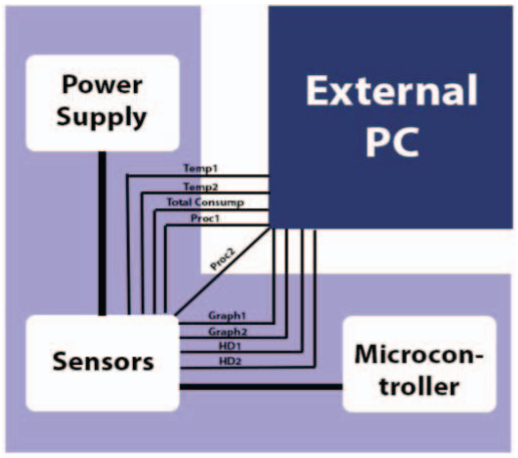


Figure 2: Design of the device architecture

The sensors are responsible for taking the consumption measurements of the PC connected to EET. EET has nine sensors:

- two sensors to measure the temperature: temp1 measures the internal temperature of the measurement device, and temp2 recovers the temperature dissipated by the processor of the device under test (DUT).

- a sensor to measure the total energy consumption of the DUT (Total Comsump sensor).
- two sensors for each of the hardware components measured: hard disk (HD1 and HD2), processor (Proc1 and Proc2) and graphical card (Graph1 and Graph2). These sensors allow us to measure the electricity used by these hardware components when the software is executed.

Finally, EET has a power supply that must be connected to the device under test where the software will be executed, replacing the power supply of the DUT; the sensors are connected to the energy distribution lines from the power supply to the different hardware components.

The only thing needed to connect the measuring device to a PC is to replace the power supply; this makes EET a portable device that can be adapted to any PC, enabling us to know what the consumption is when running software on that equipment. EET does not affect the consumption of the DUT, since energy is supply independently

Figure 3 shows the measurement device (the one with the lights on) connected to a DUT (behind the device). The screen displays the measurement results of a software application running on the PC.

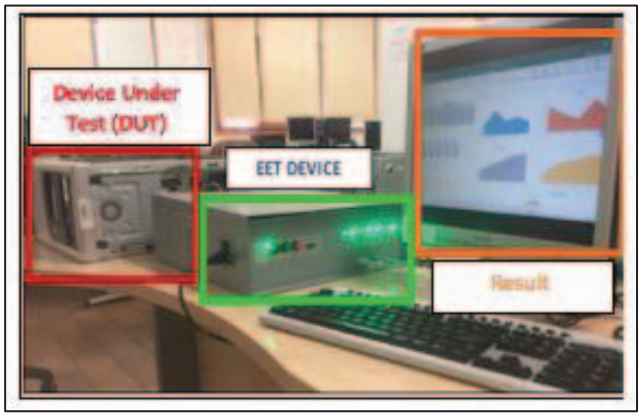


Figure 3: EET device

4 EET PROOF OF CONCEPT

4.1 Context

In this section, we will show the results of different execution tests we ran with the device. Our objective is to prove the reliability of the measurements done by EET, as well as to identify how the execution of a given piece of software can affect the energy consumption of the PC where the software runs. To carry out the validation, we have run six sorting algorithms with different levels of complexity on the PC connected to EET; the aim was to study how the complexity of the algorithm affects the energy consumption. The sorting algorithms chosen were: Insertion sort, Bubble sort, Cocktail sort, Quicksort, Heapsort and Merge sort. Each algorithm has been implemented in three different programming languages: Java, C and Python; they have been executed on two different operating systems: Microsoft Windows 7 Professional and Linux Xubuntu. The combination of these three factors (see Table 2) provided us with different scenarios of energy measurement.

	Java	C	Python
Microsoft Windows 7		Insertion sort, Bubble sort, Cocktail sort	
Linux Xubuntu		Quicksort, Heapsort, Merge sort	

Table 2. Proof Scenarios

The device under test connected to EET, where the algorithms run, had the following characteristics:

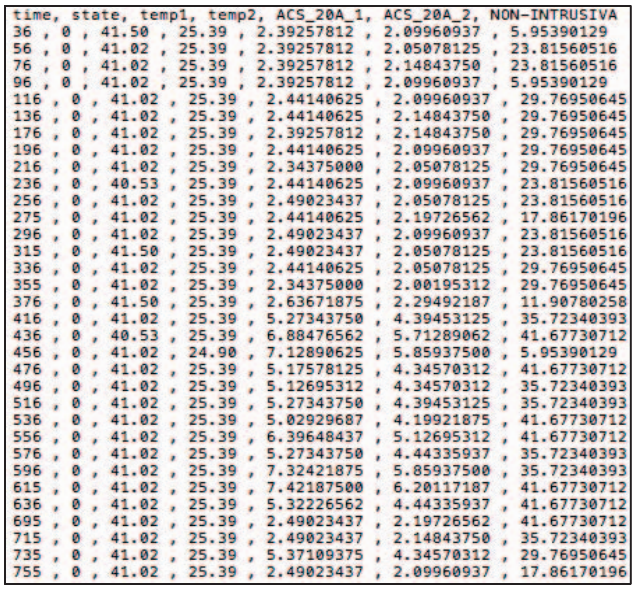
- Intel Pentium D CPU 3.00 GHz
- GigaByte GA-8I945P-G Motherboard
- 2x 1GB DDR2 533 MHz
- 1x Samsung SP2004C 200GB 7500rpm
- 1x Nvidia GForce GTS 8600

We carried out ten executions of each ordering algorithm for each language, and on each operating system. All tests have been performed with the same data set, composed of 15000 integer type elements. This set of data has been generated randomly by a software application.

4.2 Data collection

After each execution, the measurement results were recorded in a log file (see Figure 4). The information stored on each log line was:

- Time: time since the beginning of the test (in milliseconds).
- State: binary value to know when the algorithms are running (and then the CPU working on them) or not.
- Temp1 & Temp2: the temperature measured by the corresponding sensors, measured in degrees Centigrade. Temp1 corresponds to the internal temperature of the measuring device and Temp2 is the temperature that the DUT processor dissipates.
- ACS_20A_1 & ACS_20A_2: value in watts (W) provided by Proc1 & Proc2, the sensors that capture the energy consumed by the processor.
- Non-Intrusive Sensor: value in watts (W) of the total electrical consumption of the PC, provided by the Total Consump sensor.



```
time, state, temp1, temp2, ACS_20A_1, ACS_20A_2, NON-INTRUSIVA
36 , 0 , 41.50 , 25.39 , 2.39257812 , 2.09960937 , 5.95390129
56 , 0 , 41.02 , 25.39 , 2.39257812 , 2.05078125 , 23.81560516
76 , 0 , 41.02 , 25.39 , 2.39257812 , 2.14843750 , 23.81560516
96 , 0 , 41.02 , 25.39 , 2.39257812 , 2.09960937 , 5.95390129
116 , 0 , 41.02 , 25.39 , 2.44140625 , 2.09960937 , 29.76950645
136 , 0 , 41.02 , 25.39 , 2.44140625 , 2.14843750 , 29.76950645
176 , 0 , 41.02 , 25.39 , 2.39257812 , 2.14843750 , 29.76950645
196 , 0 , 41.02 , 25.39 , 2.44140625 , 2.09960937 , 29.76950645
216 , 0 , 41.02 , 25.39 , 2.34375000 , 2.05078125 , 29.76950645
236 , 0 , 40.53 , 25.39 , 2.44140625 , 2.09960937 , 23.81560516
256 , 0 , 41.02 , 25.39 , 2.49023437 , 2.05078125 , 23.81560516
275 , 0 , 41.02 , 25.39 , 2.44140625 , 2.19726562 , 17.86170196
296 , 0 , 41.02 , 25.39 , 2.49023437 , 2.09960937 , 23.81560516
315 , 0 , 41.50 , 25.39 , 2.49023437 , 2.05078125 , 23.81560516
336 , 0 , 41.02 , 25.39 , 2.44140625 , 2.05078125 , 29.76950645
355 , 0 , 41.02 , 25.39 , 2.34375000 , 2.00195312 , 29.76950645
376 , 0 , 41.50 , 25.39 , 2.63671875 , 2.29492187 , 11.90780258
416 , 0 , 41.02 , 25.39 , 5.27343750 , 4.39453125 , 35.72340393
436 , 0 , 40.53 , 25.39 , 6.88476562 , 5.71289062 , 41.67730712
456 , 0 , 41.02 , 24.90 , 7.12890625 , 5.85937500 , 5.95390129
476 , 0 , 41.02 , 25.39 , 5.17578125 , 4.34570312 , 41.67730712
496 , 0 , 41.02 , 25.39 , 5.12695312 , 4.34570312 , 35.72340393
516 , 0 , 41.02 , 25.39 , 5.27343750 , 4.39453125 , 35.72340393
536 , 0 , 41.02 , 25.39 , 5.02929687 , 4.19921875 , 41.67730712
556 , 0 , 41.02 , 25.39 , 6.39648437 , 5.12695312 , 41.67730712
576 , 0 , 41.02 , 25.39 , 5.27343750 , 4.44335937 , 35.72340393
596 , 0 , 41.02 , 25.39 , 7.32421875 , 5.85937500 , 35.72340393
615 , 0 , 41.02 , 25.39 , 7.42187500 , 6.20117187 , 41.67730712
636 , 0 , 41.02 , 25.39 , 5.32226562 , 4.44335937 , 41.67730712
695 , 0 , 41.02 , 25.39 , 2.49023437 , 2.19726562 , 41.67730712
715 , 0 , 41.02 , 25.39 , 2.49023437 , 2.14843750 , 35.72340393
735 , 0 , 41.02 , 25.39 , 5.37109375 , 4.34570312 , 29.76950645
755 , 0 , 41.02 , 25.39 , 2.49023437 , 2.09960937 , 17.86170196
```

Figure 4: Excerpt of an execution log file generated by the EET device

As may be noted, during the proof of concept we have not measured the data corresponding to the hard disk consumption nor the graphical card; we have worked only with the information related to the processor and the general energy consumption.

From the information recovered it was possible to represent the information obtained from the measurements graphically and analyse them easily.

An excerpt of the results obtained is shown in Figure 5, in which the visualization of a log derived from the execution of the set of algorithms developed in Java and running on Windows is presented.

As a result of the analysis of the graphical information generated from the obtained measurements for all the scenarios, we can conclude that:

- The behavior at the processor consumption level of the different algorithms in each execution is similar. The duration of each sorting algorithm execution is the only appreciable difference; this can be observed in Figure 5, for the Java+Windows scenario.
- The energy consumed is largely determined by the execution time of each sort algorithm. As the authors of [13] claimed, there is a lineal regression between the energy consumed and the execution time. For that reason, we can conclude that if we change the OS (Windows – Linux) or the programming language (Java – C – Python), the energy consumed by the processor is affected depending on the execution time; but our results suggest that there is no variation in the maximum consumption peak of the algorithms depending on the programming language or the OS.
- The consumption in the processor is not linear. See Figure 5, for the Java+Windows scenario.
- The consumption of the processor is equivalent to approximately 25% of the total consumption of the PC during the execution of the algorithms (Total Consump, Proc1 and Proc2 in Figure 5). Nevertheless, the total consumption remains at the same values when no algorithm is running.

It is also remarkable the fact that the 25% of the total consumption is due to the CPU. Therefore, it is important to improve the energy consumption by finding ways to reduce this value, for example, by reducing the CPU usage resulting from software execution.

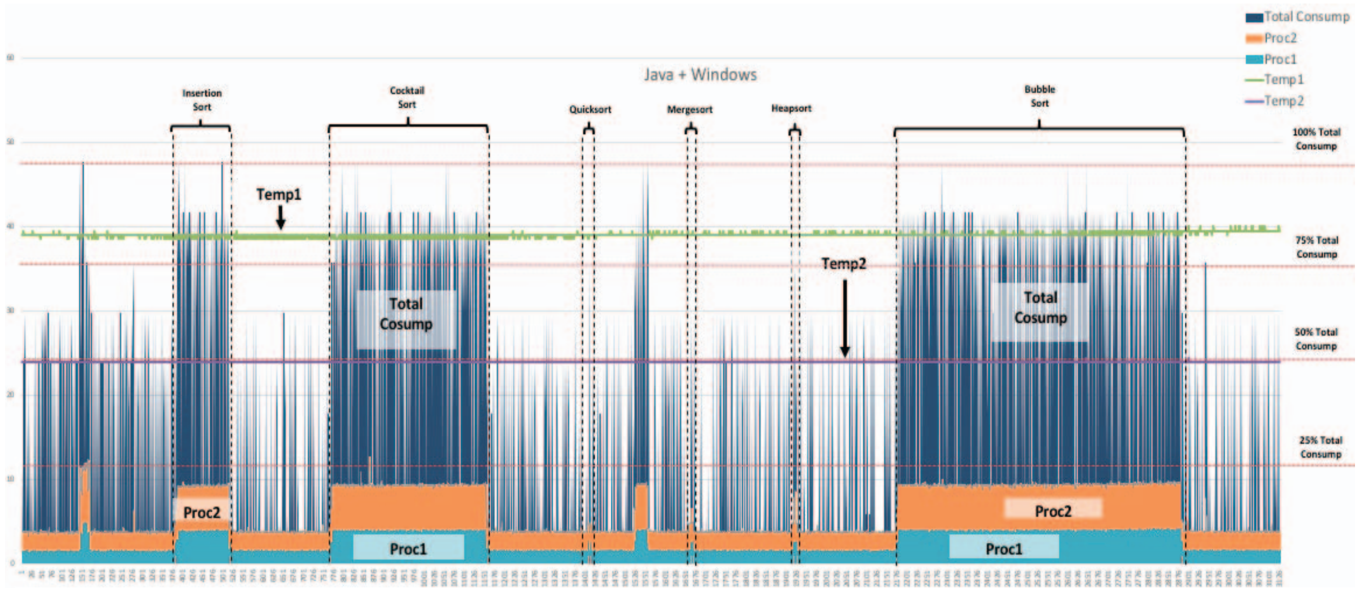


Figure 5: Example of graphical measurement results

4.3 Reliability of the measurement device

In order to verify that there are no large variations in the device measurements, we carried out a SPC (Statistical Process Control) analysis to evaluate the reliability of EET measurements.

SPC [14] is an industry standard for measuring quality during the manufacturing process. SPC makes it possible to identify when a process is working correctly and to drive continuous improvement. By using SPC we are thus able to identify possible instabilities in the EET measurements. We have chosen one scenario to apply SPC, which includes the same hardware as specified in section 4.1, namely: Microsoft Windows 7 Professional as operating system and the Simple Bubble sorting algorithm programmed in Java.

In the tests carried out, we used X-bar and R charts with limit control calculated from the samples. The number of samples used was around 130, using all the measurement points obtained by our device for the simple bubble algorithm.

For each execution of the indicated scenario, 2 graphs were generated to control possible variations, one with the measured values of the real electrical consumption of the processor (A), and another (range) with the difference between a measurement and the previous one (B). Figure 6 shows an example of these graphs obtained for the cases.

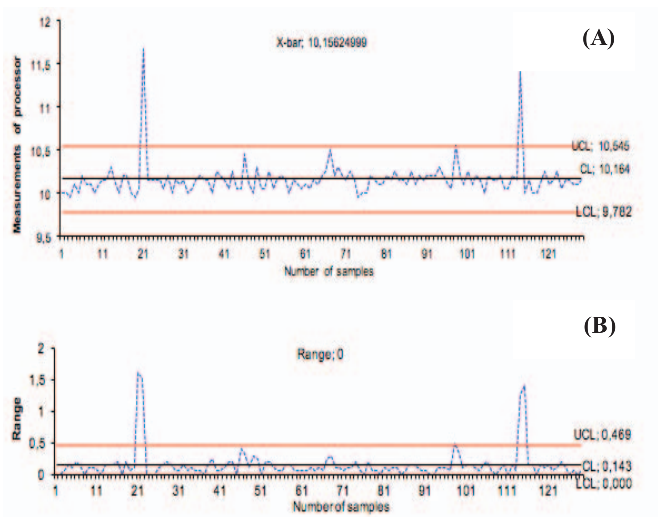


Figure 6: Graphics of measured samples

After an analysis of all the graphs obtained in the ten executions, the following conclusions can be drawn:

- Most of the measurement points of the samples are within the established limits.
- In the graphs, some peaks of consumption are observed above the upper control limit (UCL), but these are not maintained during two consecutive samples; this makes us think that these peaks are due to external factors, and not to the execution of the algorithm in itself.
- There are no samples below the lower control limit (LCL).
- Energy consumption does not have large variations.
- No pattern is observed that relates a greater consumption to parts of the execution of the software.

To confirm that the device is stable and that there were no large variations in the measurements, two new charts were generated (Figure 7). These charts represent the mean and standard deviation of the values from the previous 10 executions.

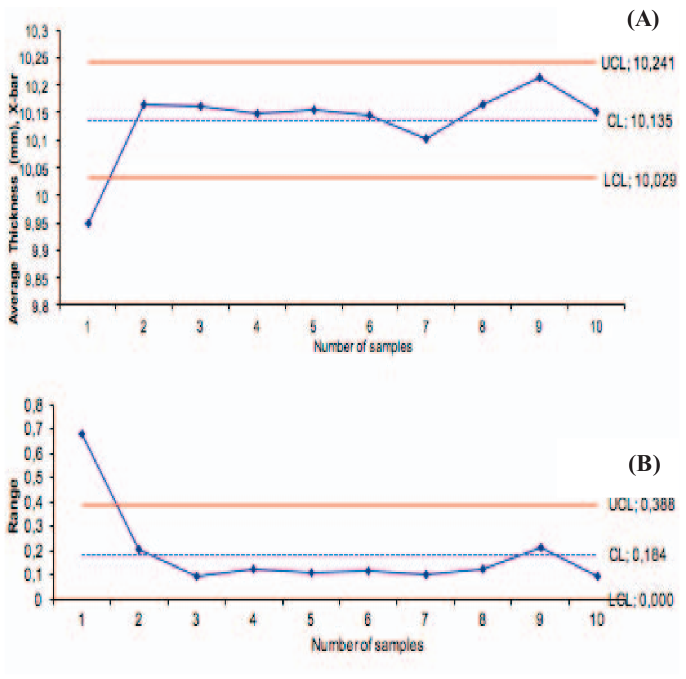


Figure 7: Representation of values for calibration

From both graphs it is possible to observe that the only deviation produced is in the first sample, being the rest of the samples within the correct parameters. This is because after the first sample, and taking into account the inconsistencies detected, some changes in the configuration of the software sensors were applied. The stable results obtained for the rest of the samples allow us to conclude that the changes made were correct.

From all these analyses, we can conclude that EET performs stable measurements, within correct parameters.

5 CONCLUSIONS

Software technology can play a highly important role in global energy consumption. In this paper, the *Energy Efficiency Tester* (EET) has been presented. EET is the hardware device which is the core component of the *Framework for Energy Efficiency Testing to Improve eNvironmental Goals of the Software* (FEETINGS).

EET is a complete device which, when connected to several components of an device under test (e.g. CPU, hard disk, etc.) is able to keep track both of energy consumption and of the temperature of such components.

EET can play a very important role in the quest to improve energy consumption, offering continuous measurement of the energy consumption of the system while a given software is executed in the DUT. Such information will then be analysed in several ways, in an effort to detect parts, or software components, with excessive energy consumption. Far from giving a global consumption measurement (which could also be measured by other devices in the industry), EET gives the chance to obtain detailed measurements taken from different components of the DUT. These in turn let researchers know if the software execution implies an excessive consumption in the CPU, in the graphical components, or in the massive storage units. Such information leads to a very accurate analysis of the software system, flagging up the particular parts that cause such high consumption, which can help developers to understand the energy consumption effects of their software engineering decisions, and therefore promote energy-efficient software development. An additional advantage of EET is the sampling frequency, which is around 500Hz; this offers very realistic consumption information.

EET has been tested by measuring the same family of algorithms developed, using the Java, C and Python programming languages, under the Windows and Linux operating systems. The results obtained from the case studies conducted reveal that the measurements of EET, submitted to *Statistical Process Control*, are totally stable, with no outlier values.

As future work, we will continue working on the FEETINGS framework, seeking to detect what the best techniques for analysing the information are thereby offering opportunities to improve software consumption and enhance software sustainability as a result. In addition, future work must be focused on evaluating the accuracy of the device by comparing the obtained measurements with the output results of other validated instruments.

ACKNOWLEDGMENTS

The authors would like to thank Ramon Tabaco, who actively participated in the construction of the EET device.
This work is part of the GINSENG-UCLM (TIN2015-70259-C2-1-R) projects, supported by the Spanish Ministry of Economy, Industry and Competitiveness and European FEDER funds, and within the Software Sustainablity (SOS-SBPLY/17/180501/000364) project funded by "Consejería de Educación, Cultura y Deportes de la Dirección General de Universidades, Investigación e Innovación de la JCCM".

REFERENCES

1. Murugesan, S. 'Harnessing Green IT: Principles and Practices', IT professional, 2008, 10 (1)
2. Lago, P., Kazman, R., Meyer, N., Morisio, M., Müller, H.A., and Paulisch, F. 'Exploring initial challenges for green software engineering: summary of the first REENS workshop, at ICSE 2012', ACM SIGSOFT Software Engineering Notes, 2013, 38 (1), 31-33.
3. Ardito, L., Procaccianti, G., Torchiano, M., and Vetro, A. 'Understanding Green Software Development: A Conceptual Framework', IT Professional, 2015, 17 (1), 44-50.
4. Capra, E., Francalanci, C., and Slaughter, S.A. 'Measuring application software energy efficiency', IT Professional, 2012, 14 (2), 54-61.
5. Pinto, G. and Castor, F. 'Energy efficiency: a New Concern for Application Software Developers'. Commun ACM, 2017, 60, 12, 68-75. DOI: https://doi.org/10.1145/3154384
6. Jagroep, E., van der Werf, J.M.E., Jansen, S., Ferreira, M., and Visser, J. 'Profiling energy profilers'. In Proceedings of the 30th Annual ACM Symposium on Applied Computing, ACM, 2015, 2198-2203.
7. Jagroep, E.A., van der Werf, J.M., Brinkkemper, S., Procaccianti, G., Lago, P., Blom, L., and van Vliet, R. 'Software Energy Profiling: Comparing Releases of a Software Pproduct'. In Proceedings of the 38th International Conference on Software Engineering Companion, ACM, 2016, 523-532.
8. Procaccianti, G., Fernandez, H., and Lago, P. 'Empirical evaluation of two best practices for energy-efficient software development', Journal of Systems and Software, 2016, 117, 185-198.
9. Noureddine, A., Rouvoy, R., and Seinturier, L. 'A Review of Energy Measurement Approaches', ACM SIGOPS Operating Systems Review, 2013, 47 (3), 42-49.
10. Amsel, N., and Tomlinson, B. 'Green tracker: a tool for estimating the energy consumption of software'. In CHI'10 Extended Abstracts on Human Factors in Computing Systems, ACM, 2010, 3337-3342.
11. Cordero, V., de Guzmán, I.G.-R., and Piattini, M. 'A first approach on legacy system energy consumption measurement'. In Global Software Engineering Workshops (ICGSEW), 2015 IEEE 10th International Conference on, IEEE, 2015, 35-43.
12. Ferreira, M.A., Hoekstra, E., Merkus, B., Visser, B., and Visser, J. 'Seflab: A lab for measuring software energy footprints'. In Green and Sustainable Software (GREENS), 2013 2nd International Workshop on, IEEE, 2013, 30-37.
13. Rashid, M., Ardito, L., and Torchiano, M. "Energy consumption analysis of algorithms implementations." Empirical Software Engineering and Measurement (ESEM), 2015 ACM/IEEE International Symposium on. IEEE, 2015
14. Oakland, J.S. 'Statistical Process Control' Routledge. 2007