Software Quality Journal
https://doi.org/10.1007/s11219-020-09536-9

Does maintainability relate to the energy consumption of software? A case study

Javier Mancebo[1] · Coral Calero[1] · Félix García[1]

Accepted: 28 October 2020/Published online: 06 January 2021

Abstract
Energy consumption of software has been becoming increasingly significant, since it can vary according to how the software has been developed. In recent years, developers and researchers have been interested in analyzing, among other things, how energy consumption evolves when changes occur from one version to another in any given software. Thus far, the only studies available are theoretical papers that reinforce the idea that maintainability may have an influence on energy use, but this needs to be proven empirically, which is the goal of this article. This work presents an empirical study carried out to test whether there is a relationship between the energy consumption and the maintainability of several versions of Redmine. Maintainability has been assessed by means of different measures, such as the number of lines of code, or the complexity of the software, calculated using SonarCloud, and the energy consumption measurements have been captured using the EET device. The results obtained show that the number of lines of code affects both the energy consumption of the processor and the total consumption of the computer where the software is run. It is intended that the results from this work should serve as a basis for the undertaking of new empirical studies which will enable the relationship between the software maintainability and the energy efficiency of that software to be better understood.

Keywords Software sustainability · Green software · Energy efficiency · Energy consumption software maintainability

✉ Javier Mancebo
Javier.Mancebo@uclm.es

Coral Calero
Coral.Calero@uclm.es

Félix García
Felix.Garcia@uclm.es

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

1 Introduction

There is an ever-increasing awareness in society of the importance of environmental conservation; concern for all issues related to sustainability is also growing, and software cannot be an indifferent onlooker as regards these matters. A report by Huawei Technologies (Andrae 2017) states that even the most optimistic forecast is that in 2025 ICT will consume around 2800 TWh, which is approximately 9% of the total consumption of worldwide energy use. The most pessimistic of the predictions asserts that the figure could be twice that mentioned above. According to Jones (2018), the data consumption figures are even more alarming and are projected to constitute approximately 21% of global energy consumption in 2030 .

As Chien (2019) points out, computer technologies and systems must be designed to reduce carbon emissions and environmental impact; this becomes one of the main objectives of the computer community. As an integral part of computer systems, and in the quest to make these figures less intimidating, there is a need to build software products that are more sustainable, both during their creation and in their use. Pinto and Castor (2017) remarked that, although software systems do not consume energy themselves, they affect hardware utilization, leading to indirect energy consumption. Energy consumption is a ubiquitous problem, and the years to come will require developers to be even more aware of it.

Sustainable software is defined as "software, whose impacts on economy, society, human beings, and environment that result from development, deployment and usage of the software are minimal and/or which have a positive effect on sustainable development" (Dick and Naumann 2010) and Software Sustainability is about the capability of software to last a long time by using the resources strictly needed (Calero et al. 2019). Both concepts are related to the impact of the use of software, a concern which, for some years now, has become an important research concern, where three software sustainability dimensions can be identified, as we can see in Fig. 1 (Calero and Piattini (2017):

- Human Sustainability: how software development and maintenance affect the sociological and psychological aspects of people
- Economic Sustainability: how the software lifecycle processes protect stakeholders' investments

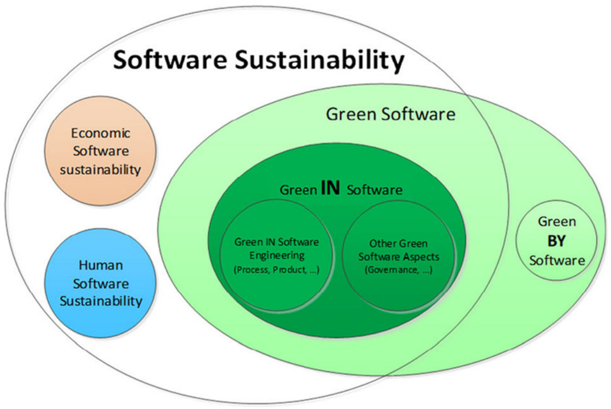
Software Sustainability
Economic Software sustainability
Human Software Sustainability
Green Software
Green IN Software
Green IN Software Engineering (Process, Product, ...)
Other Green Software Aspects (Governance, ...)
Green BY Software

Fig. 1 Software Sustainability dimension (Calero and Piattini 2017)

- Environmental Sustainability: how software product development, maintenance, and use affect energy consumption and the usage of other resources

The Environmental Sustainability dimension, also known as Green Software, which is the focus of this work, promotes improvement in the energy efficiency of software, minimizing its environmental impact and having a positive impact on the other two dimensions (Penzenstadler et al. 2014).

Green Software requirements also influence other aspects of software quality (Jagroep et al. 2017), so it is important to analyze whether there is a relationship between them and other already well-known characteristics of the quality (such as those included in the ISO 25010 standard-ISO (2011)). In fact, there are a number of studies which demonstrate that the use of good practices in software engineering can improve energy efficiency. For example, Procaccianti et al. (2016) established that the optimization of the consultation in MySQL Server, by means of the limitation of the indexation mechanism or the reduction of unnecessary operations (such as the use of the clause ORDER BY), would enable energy consumption of database consultations to be reduced. Capra et al. (2012) indicated that, depending on the particular application layer design, the energy consumption of a server could potentially increase by up to 72% more. In the study presented by García-Mireles et al. (2018), and more directly related to the object of this work, the interaction between the quality characteristics and energy efficiency is analyzed, though only in a theoretical manner. They identify a set of quality characteristics that can have a positive or negative influence on the consumption of software.

The main objective of this article is to analyze the relationship between the maintainability of a software application and its energy consumption. In our effort to achieve our goal, we will carry out a case study with different versions of Redmine software, aiming to study how its maintainability has evolved, and if there is any relationship between that maintainability and the energy required when the application is run. The software's maintainability will be assessed with the SonarCloud[1] tool, using as maintainability measures the total number of lines of code, the cyclomatic complexity, the number of comments, or duplicated lines in the source code. The energy consumption will be obtained from a representative set of test scenarios using EET, a device that measures the consumption of software when it is run. Using EET, we can discover the energy consumption of the computer where the Redmine software is running. In addition, EET allows us to obtain the energy consumed by the processor, hard disk, and graphic card. This data will allow us to make an initial approach to assessing the possible relationship between maintainability and energy consumption.

The rest of the document is structured as follows. The empirical study carried out is presented in Section 2: its scope and objectives, setting, the protocol followed, the variables chosen, and the formulation of the hypotheses. In Section 3, the results obtained in the study are set out in detail. The most important threats to the validity of our study are presented in Section 4, and information is provided which would make the experiment verifiable and possible to replicate by other researchers. Section 5 presents the work found that is related to the topic and the main differences between them and our study. Lastly, in Section 6, the conclusions of this work are set out, and there is a presentation of the follow-up research activities which are proposed.

[1] SonarCloud. https://sonarcloud.io

2 Empirical evaluation of the impact of software maintainability on software energy consumption

To present the objective of our empirical evaluation, we are going to use the recommendations of Wohlin et al. (2012), based on the application of the Goal/Question/Metric (GQM) method. Our objective is thus defined as follows:

- Analyze the *Energy Efficiency*
- for the purposes of *Evaluating its Interaction with Maintainability*
- in terms of *Energy Consumption*
- from the point of view of *Software Developers*
- in the context of *Software Application.*

To give direction to our objective, the following research question is posed:

RQ. Is there a relationship between the energy consumption of the software when it is being run and its maintainability?

The answer to this question will allow us to discover the relationship that the different maintainability measures have with energy consumption.

To ensure the consistency and reliability of our work, we are going to follow the Green Mining methodology defined by Hindle (2012) for conducting experiments where energy consumption measurements are employed. This methodology is composed of seven activities: (1) choose the software products 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) analyze the results. Figure 2 shows the process followed, highlighting the main elements of our study which are considered in each activity.

2.1 Choose the software products and the context in which it should be tested

The first step is to select the software products to be analyzed in the case study. These software products must meet the following criteria: (i) each of the software products must be available for installation and execution. In addition, source code must be available for analysis; (ii) the selected software products must include at least the same functionalities. In order to achieve

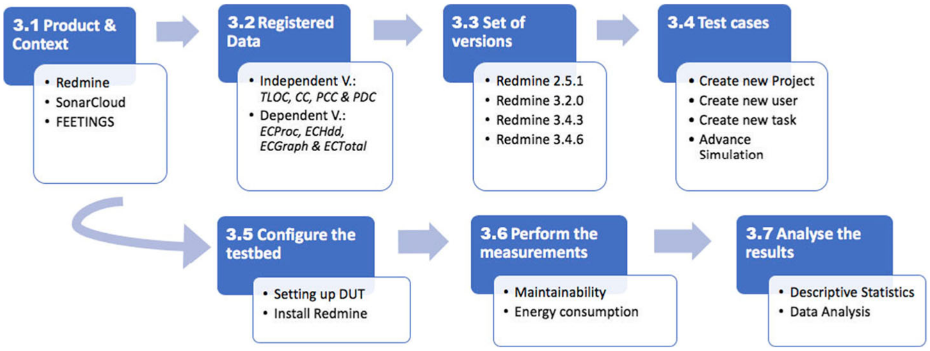

Fig. 2 Green mining methodology

the established criteria, and taking into account that we are interested in the relationship between maintainability and energy consumption, we decided to use different versions of the same software product. Thus, all versions will include at least the same functionalities. Note that each version selected will be analyzed as a standalone software product.

In our study, the chosen *software products* are different versions of Redmine,[2] a tool for software project management that allows users to follow up and control multiple projects simultaneously. It includes several functionalities, such as follow-up of incidents, calendar of activities, links to a configuration management repository, forums, wikis, and so on.

As far as the *context* in which the experiment is carried out is concerned, different tools needed to be used:

- We have used the SonarCloud[3] platform in the evaluation of the software maintainability; this platform is a cloud service for the ongoing inspection of code quality. It supports a great quantity of programming languages, offering detailed information on software maintainability measurements, among these being the ones used in our analysis and which are set out in detail later in this work.
- To evaluate the energy efficiency of the software, we have used a framework (Framework for Energy Efficiency Testing to Improve eNviromental Goals of the Software) which makes it possible to measure, analyze, and visualize the energy consumption of a software application (see Fig. 3). FEETINGS is made up of two main elements: (i) a device that allows us to measure the energy consumption of a set of hardware components (processor, hard disk, graphics card, and general consumption) when a software product is run in the DUT (device under test) (Mancebo et al. 2018) and (ii) the software application, known as Software Energy Assessment (SEA), which processes the data gathered by the EET, analyzes them, and generates an appropriate visualization of the results (depending on the nature of each).

The configuration of the DUT used to run the empirical study is shown in Table 1. As may be observed in the specifications, the DUT is a PC with a configuration that is conventional in this type of computers. A DUT without special capacities for processing or storage is chosen so that the results will be more generalizable, since the computer is similar to those in normal use.

2.2 Deciding the types of data that will be registered

The next step (*Activity 2*) is to decide on the variables to be used to carry out the experiment.

The *independent variables* are the measures used in the evaluation of the maintainability of the software analyzed, which will be obtained using the SonarCloud. The maintainability measures are *Total Lines of Code*, *TLOC* (*M1*); the *Cyclomatic Complexity*, *CC* (*M2*); the *Percentage of Comments in the Code*, *PCC* (*M3*); and the *Percentage of Duplicate Code lines*, *PDC* (*M4*).

The *dependent variable* is the energy used. To calculate it, we will use *Power Consumptions*, *PC* (*M5*), obtained with EET (see Section 2.1), and the *Execution*

[2] Redmine. https://www.redmine.org
[3] SonarCloud. https://sonarcloud.io

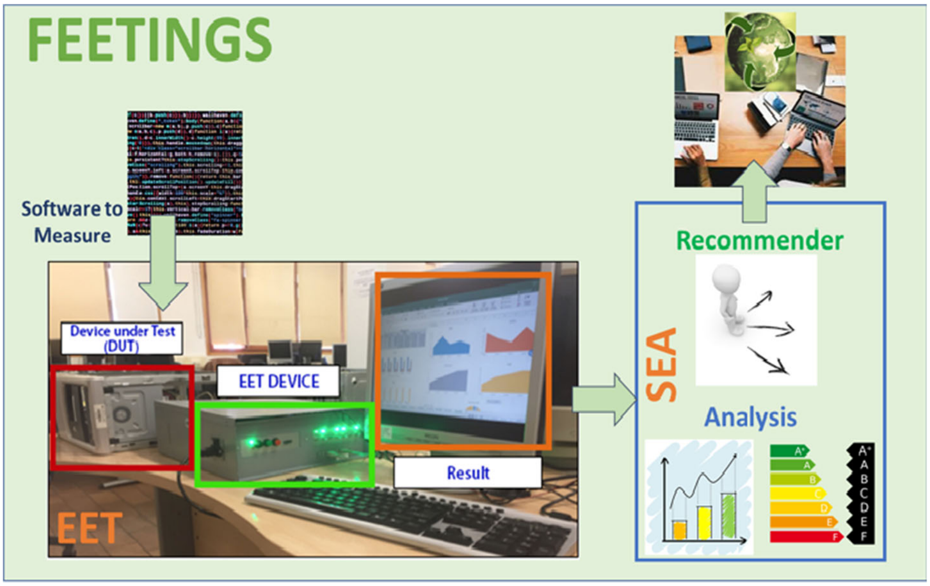

Fig. 3 FEETINGS overview

Time, *ET* (*M6*). With these two measures, we obtain the value of the dependent variable, named *Energy Consumption*, *EC* (*M7*), calculated as:

$$\mathrm{M7} = \mathrm{M5}^*\mathrm{M6}.$$

The *Energy Consumption* (M7) is going to be refined in four measures (one for each hardware device), by which EET allows us to obtain the value M5 for consumption, calculated using the ET (M5), and the PC value (M6) for each consumption sensor of the EET: *PCProc* (power consumption of the processor), P*CHdd* (power consumption of the hard disc), *PCGraph* (power consumption of the graphics disc), and P*CTotal* (total power consumption). We will therefore have the following measures associated with M7 (EC): ***ECProc*** = (*PCProc/ET*), ***ECHdd*** = (P*CHdd/ET*), ***ECGraph*** = (*PCGraph/ET*), and ***ECTotal*** = (P*CTotal/ET*).

Based on the measures selected in the design of the empirical evaluation, and in order to answer the defined research question, the following hypotheses have been defined. Each of the hypotheses will be studied for the four variables of energy consumption in which the M7 measure has been refined:

- $H1_0$, Null Hypothesis: The software Energy Consumption (M7 - *ECProc*, *ECHdd*, *ECGraph*, and *ECTotal*) *is not affected* by the maintainability measure of Total Lines of Code (M1).

Table 1 Specifications of the DUT used for experiment

Hardware	Motherboard: Asus M2N-SLI Delux Processor: AMD Athom ×2 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

- *H1*$_1$, Alternative Hypothesis: The software Energy Consumption (M7 - *ECProc*, *ECHdd*, *ECGraph*, and *ECTotal*) *is affected* by the maintainability measure of Total Lines of Code (M1).
- *H2*$_{0,}$ Null Hypothesis: The software Energy Consumption (M7 - *ECProc*, *ECHdd*, *ECGraph*, and *ECTotal*) *is not affected* by the Cyclomatic Complexity measure (M2).
- *H2*$_{1,}$ Alternative Hypothesis: The software Energy Consumption (M7 - *ECProc*, *ECHdd*, *ECGraph*, and *ECTotal*) (M2) *is affected* by the Cyclomatic Complexity measure (M2).
- *H3*$_{0,}$ Null Hypothesis: The software Energy Consumption (M7 - *ECProc*, *ECHdd*, *ECGraph*, and *ECTotal*) *is not affected* by the maintainability measure of Percentage of Comments in the Code (M3).
- *H3*$_{1,}$ Alternative Hypothesis: The software Energy Consumption (M7 - *ECProc, ECHdd*, *ECGraph*, and *ECTotal*) *is affected* by the maintainability measure of Percentage of Comments in the Code (M3).
- *H4*$_{0,}$ Nu*ll Hypothesis*: *The software Ener*gy Consumption (M7 - *ECProc*, *ECHdd*, *ECGraph*, and *ECTotal*) *is not affected* by the maintainability measure of Percentage of Duplicate Code Lines (M4).
- *H4*$_{1,}$ Alternative Hypothesis: The software Energy Consumption (M7 - *ECProc*, *ECHdd*, *ECGraph*, and *ECTotal*) *is affected* by the maintainability measure of Percentage of Duplicate Code Lines (M4).

2.3 Choosing a set of software versions

In this stage (*Activity 3*), it is necessary to select the software versions, in our case, as explained in Activity 1, the different versions of Redmine.

In order to have a sample representative enough of, on the one hand, the evolution of Redmine and, on the other hand, the influence of maintainability on the energy consumption, 4 different versions of Redmine were selected, according to a series of criteria: (i) all the versions had to include at least the same functionalities; (ii) the source code had to be available for its analysis and execution; and (iii) the launch dates between the versions chosen had to be sufficiently separated for the needed evolution to take place and for there to be differences between the values of the maintainability metrics. The only exception was version 3.4.6, which was selected because it was the most up to date at the time of the study. Table 2 shows the set of versions of Redmine chosen and their main features.

As a result of this selection process, we consider that the four versions of Redmine chosen are representative enough to obtain a first insight about the relationship between maintainability and energy consumption of software; this can serve as a starting point for expanding the study in the future with new versions and other software applications.

2.4 Developing the test cases to be run

Activity 4 is the creation of the test cases that are going to be run in the DUT. The following test cases were defined for this study. With the selected test cases, we consider that we comply with a high level of code coverage, including also the main functionalities of Redmine.

a. *Creation of a new project*: With the role of administrator, create a new project, by entering the necessary data. These data are the name of the project, an identifier and a description.

Table 2 Versions of Redmine selected

ID	SW product	Last modified	Rails version	Ruby version	Database version
SW1	Redmine 2.5.1	Mar. – 14	3.2.16	1.9.3	MySQL 5.5
SW2	Redmine 3.2.0	Dec. – 15	4.1.7	2.0.0	MySQL 5.6
SW3	Redmine 3.4.3	Sep. – 17	4.1.7	2.3.5	MySQL 5.6
SW4	Redmine 3.4.6	Jun. – 18	4.1.7	2.3.6	MySQL 5.6

b. *Creation of a new user*: With the role of administrator, register a new user, filling in information such as the identifier, first name and surname and email and ticking the option of an automatic generation of a password.
c. *Creation of a new task*: As a user, create a new task in a project created previously. The task must be completed, along with the description of this task, as well as its type, its status, and the person assigned to the task.
d. *Simulation of progress*: As a user, conduct a simulation of how a task would progress, modifying the percentage of completion and the number of hours to be taken. The Gantt diagram is also consulted.

2.5 Configuring the test ground

Before the measurement is begun, the DUT where the test cases are going to be run (*Activity 5*). All the test cases were run on the same computer, with the specifications that were set out at the beginning of this section. The following steps were taken in carrying out the configuration of the DUT: (i) first of all, the different services that are included in the operative system are deactivated. These services, such as updates, virus scans, etc. are ones which run automatically, and they may add noise to the measurements; (ii) after that, the version of Redmine that is to be analyzed is installed; (iii) the software measurements are made; (iv) once the measurements for one of the versions of the selected software products are completed, the DUT is restored, such that it returns to its initial state. This procedure is repeated for the different versions of the software that are going to be assessed.

2.6 Carrying out the measurements and gathering the data registered for each software product

In *Activity 6* the measurements for each version of the software are carried out, and the data are gathered. The "Green Mining" method does not provide any protocol as to how to undertake the measurement in a way that is valid and reliable. Jagroep et al. (2016) present a measurement protocol, in which an extension of Activity 6 from the "green method" mining is carried out, setting out in detail the specific tasks that have to be carried out. For the purposes of our study, we have adopted this protocol to conduct the measurement of energy using the FEETINGS framework, such that Activity 6 is made up of the following tasks Jagroep et al. (2016):

i. Carry out the configuration of the test ground.
ii. Check that there are no other applications being run in the background.
iii. Close down all unnecessary applications.
iv. Start up the software that is to be evaluated.

v. Begin the energy consumption measurement, and run the test cases that have been defined.
vi. Gather the data.
vii. Verify the data and check whether there are outliers in the measurements; if there are atypical values, these are rejected.
viii. Return the setting to its initial state.

This will be the measurement protocol which we will follow in carrying out the measurements of energy consumption. In addition, in order to ensure the consistency of the measures undertaken, the task of *Begin the measurement and run the test cases* (*v*) will be repeated 20 times for each of the test cases that have been identified in Activity 4. As this is a controlled environment, the 20 measures are usually of a big enough sample size for the impact of the atypical values to be mitigated (as is the case, e.g., for the consumption of the energy that is dedicated to the tasks of the operative system). For this reason, unlike what Jagroep et al. propose in step VII, possible outliers have not been eliminated in this experiment, and they have been taken into account in the subsequent analysis.

The protocol used to obtain the maintainability measurements is simpler and is limited to loading the source code of the version chosen in the SonarCloud, as well as to recovering the values obtained by that tool.

2.7 Analyzing the results

Lastly, *Activity 7* is carried out, in which the results obtained for each of the selected software products are analyzed and presented in Section 3.

3 Results of empirical evaluation

This section corresponds to Activity 7 of the protocol, where the results obtained are presented. Firstly, the data extracted from the measurements of maintainability and of energy consumption (M1 to M7) are set out. We then go on to answer the hypothesis in Section 2.2. Lastly, the most significant findings are discussed.

3.1 Descriptive statistics

Table 3 shows the maintainability values that were extracted from the SonarCloud tool for each of the chosen versions of Redmine. The values that appear in brackets indicate the variation of each measurement with respect to that of the previous version. The values of the Total Lines Of Code and of Cyclomatic Complexity have increased in the latest versions. Nevertheless, the Percentage of Comments in the Code has decreased in each of the most recent versions; the only exception is version 3.4.6, where there is a slight rise. The Percentage of Duplicate Code Lines increased between version 2.5.1 and version 3.2.0, but in later versions this value was lower.

It is important to highlight that the variations between version 3.4.3 and version 3.4.6 are not very large. This is because the latter is a small-scale update of the previous version, so there are no important changes.

Table 3 Redmine maintainability measurements

	Redmine versions			
	Redmine 2.5.1	Redmine 3.2.0	Redmine 3.4.3	Redmine 3.4.6
TLOC (M1)	109,005	128,677 (+18.05%)	138,321 (+ 7.49%)	138,792 (+ 0.34%)
Cyclomatic Complexity (M2)	13,709	13,988 (+2.04%)	15,132 (+ 8.18%)	15,181 (+ 0.32%)
% of comments (M3)	12.50%	10.40% (−16.8%)	10.10% (− 2.88%)	10.11% (+ 0.1%)
% of duplications (M4)	13.10%	24.60% (+87.79%)	20.80% (− 15.45%)	20.70% (− 0.48%)

To obtain the Energy Consumption (M7), the value of the Power Consumption (M5) is used for each of the EET sensors and the Execution Time (M6). The values of Energy Consumption for each of the Redmine versions when the test cases are run can be found in Tables 4, 5, 6, and 7; each table contains the information about the energy consumption of each of the sensors available in EET (*ECProc*, *ECHdd*, *ECGraph*, and *ECTotal*). For the 20 executions carried out, the respective tables show the mean consumption (in Watts per second), the standard deviation, the median, and the maximum and minimum values of the consumption found. In addition, the difference between the mean of the energy consumption and that of the previous version is shown in brackets. Figure 4 shows, in diagram form, the results for the means from Tables 4, 5, 6, and 7.

From the results obtained for the maintainability measurements of the four versions (Table 3) and the consumption of these versions (Tables 4, 5, 6, and 7), we can observe that:

- The amount of Total Lines Of Code (M1) and of Cyclomatic Complexity (M2) increases in each version. However, the Percentage of Comments in the Code (M3) and of Percentage of Duplicate Code Lines (M4) do not follow this pattern. The value of M3 has a lower value in all versions except in the latest one; the M4 values increase in the second version analyzed but decrease in the following ones.
- As regards the energy consumption (M7), the *ECTotal* increased in each version (between 7% and 20% more). The same thing happens if we observe the energy consumption values of the processor (*ECProc*), where it can be seen that it increased in each of the most recent versions (a variation of between 4% and 24%). In addition, the energy consumption of the hard disk (*ECHdd*) and of the graphics card (*ECGraph*) did not follow the same pattern of increasing in the latest versions. *ECHdd* has been increasing right up to version 3.4.6, where the value in this case is lower: it is around 4%, the same as in version 3.4.3. In the case of *ECGraph*, there was a large increase (more than 13%) between version 2.5.1 and

Table 4 Descriptive statistics of ECProc (in Watts per second)

	Redmine Versions			
	Redmine 2.5.1	Redmine 3.2.0	Redmine 3.4.3	Redmine 3.4.6
Mean	393.08	491.01 (+ 24.91%)	518.54 (+ 5.61%)	538.81 (+ 3.91%)
Standard deviation	25.51	20.06	21.03	31.86
Median	389.90	488.22	515.07	565.98
Max	429.21	535.25	563.55	630.59
Min	356.03	457.75	479.69	516.44

Table 5 Descriptive statistics of ECHdd (in Watts per second)

	Redmine versions			
	Redmine 2.5.1	Redmine 3.2.0	Redmine 3.4.3	Redmine 3.4.6
Mean	1549.18	1627.45 (+ 5.05%)	1814.46 (+ 11.49%)	1740.91 (− 4.05%)
Standard deviation	106.81	69.86	95.71	98.04
Median	1531.08	1633.47	1826.13	1730.38
Max	1700.01	1800.57	1996.63	1938.21
Min	1375.97	1501.39	1648.15	1576.27

3.2.0, but in the following versions, the consumption of the graphics card decreased by around 3% compared to its previous version.

As has been indicated, the results of the energy consumption were evaluated together for the four test cases defined in Section 2.4. We also believe it to be of interest to analyze the impact on the energy consumption registered by the EET sensors for each of the functionalities demonstrated in the test cases.

In Tables 8, 9, 10, and 11, this information is set out, along with the differences in the mean energy consumption that exist between each of the versions.

With the results of the energy consumption listed for each of the four test cases that had been defined, we can see that:

- The energy consumption of the processor (*ECProc*) increased in each version for the four test cases proposed. It was between version 2.5.1 and version 3.2.0 that there was most variation of the *ECProc*; it reached more than 78% in Test Case A.
- The values of energy used by the hard disk (*ECHdd*) and the graphics card (*ECGraph*) do not seem to follow any pattern, since the values do not increase in any of the versions; in others they decrease.
- The total energy consumption (*ECTotal*) increased in all the versions of the four test cases (between 5% and 58% more), except in Test Case D, where the energy consumption in the latest version fell (almost 15% less) with respect to the one that preceded it.
- Another aspect to bear in mind is that in Test Case C (*Create a new task*), the consumption values of *ECProc* and *ECTotal* suffered a greater variation (of + 22% and + 36%, respectively) between version 3.4.6 and 3.4.3 than there was between the other versions, in spite of the fact that, if we look at the data obtained by SonarCloud, the differences between both versions were actually quite small.

Table 6 Descriptive statistics of ECGraph (in Watts per second)

	Redmine versions			
	Redmine 2.5.1	Redmine 3.2.0	Redmine 3.4.3	Redmine 3.4.6
Mean	133.17	151.31 (+ 13.62%)	146.76 (− 3.01%)	143.16 (− 2,45%)
Standard deviation	9.08	9.23	7.15	10.42
Median	131.71	151.34	148.72	139.27
Max	147.52	173.71	159.32	170.70
Min	114.64	135.66	134.05	128.41

Table 7 Descriptive statistics of ECTotal (in Watts per second)

	Redmine versions			
	Redmine 2.5.1	Redmine 3.2.0	Redmine 3.4.3	Redmine 3.4.6
Mean	11,493.27	13,852.33 (+ 20.53%)	15,560.97 (+ 12.33%)	16,697.93 (+ 7.31%)
Standard deviation	757.58	613.26	565.96	881.87
Median	11,360.33	13,813.39	15,663.27	16,459.58
Max	12,885.66	15,469.07	16,556.68	18,748.81
Min	10,346.38	12,759.73	14,545.35	15,325.52

3.2 Data analysis

In this section, the hypotheses posed in Section 2.2 are tested. To that end, we are going to study whether the energy consumption (M7) obtained for each of the EET measurement sensors (*ECProc*, *ECHdd*, *ECGraph*, and *ECTotal*) is affected or not by the maintainability measurements (M1, M2, M3, and M4) whose values were obtained by using SonarCloud.

First of all, the mean value of energy consumed was calculated using the data of Power Consumption (M5) of each of the sensors and the Execution Time (M6) for each of the chosen Redmine versions.

Subsequently, aiming to find out whether the energy consumption data followed a normal distribution, a normal Q-Q plot analysis was carried out, as shown in Fig. 5.

Once it was demonstrated that the data did indeed have a normal distribution, a hypothesis check was conducted, using the Pearson correlation coefficient. The aim was to check whether the maintainability measurement values can be significant with respect to the energy consumption values. In our hypothesis test, we worked with a significance value of p-level = 0.05.

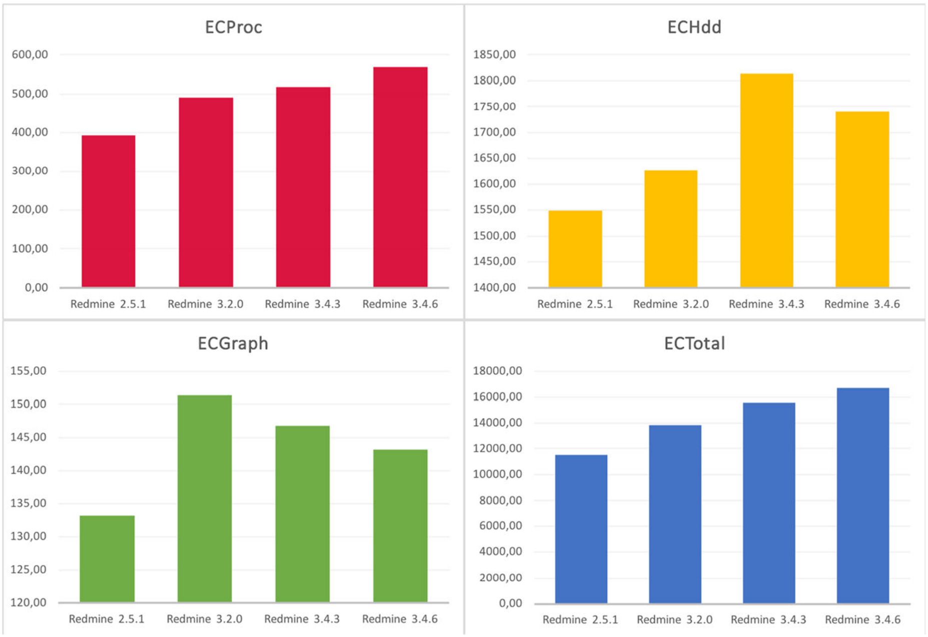

Fig. 4 Energy consumption for each Redmine version according to the hardware component

Table 8 Energy consumption, in watts per second, of Test Case A (Create a new product)

	Redmine versions			
	Redmine 2.5.1	Redmine 3.2.0	Redmine 3.4.3	Redmine 3.4.6
ECProc	67.27	120.11 (+78.55%)	122.74 (+2.19%)	128.04 (+4.32%)
ECHdd	383.68	415.30 (+8.24%)	416.28 (+0.24%)	443.21 (+6.47%)
ECGraph	31.59	39.70 (+25.66%)	34.76 (−12.43%)	34.09 (−1.94%)
ECTotal	2215.20	3507.39 (+58.33%)	3894.39 (+11.03%)	3964.20 (+1.79%)

We will now set out the results obtained for each set of hypotheses:

H1: *Total Lines Of Code (M1) – Energy Consumption (M7)*

The results obtained from calculating the Pearson correlation coefficient (see Table 12) between the Total Lines Of Code (M1) and the energy consumption measured by each of the sensors (*ECProc*, *ECHdd*, *ECGraph*, and *ECTotal*) enable us to establish that there is a significant correlation between the consumption of the processor (*ECProc*) and Total Lines Of Code, with a *p*-level of 0.0357.

There is a significant correlation between the total consumption of the DUT (*ECTotal*, with a *p*-level of 0.0317), allowing us to be certain that the TLOC for both components are significant as far as the energy consumption of the application is concerned.

In addition, in both cases, the correlation value is close to 1. This tells us that there is a strong positive relationship between the variables *ECTotal* and *ECProc* and TLOC; in other words, the more lines of code (M1) there are, the greater the increase in energy consumption (M7), both of the processor and of the computer in which the software is being run.

It therefore seems that the TLOC (M3) maintainability measurement affects the energy consumption of the processor (*ECProc*), and the total consumption of the computer (*ECTotal*), meaning that the alternative $H1_1$ can be accepted for these cases.

H2: *Cyclomatic Complexity (M2) – Energy Consumption (M7)*

Having calculated the Pearson coefficients between Cyclomatic Complexity (M2) and the Energy Consumption (*ECProc*, *ECHdd*, *ECGraph*, and *ECTotal*), which are shown in Table 13, it cannot be concluded that there is any significant correlation between CC and the energy consumption measurements. Taking these data into account, it is not possible to reject the null hypothesis $H2_0$, and we can therefore not conclude that energy consumption may be affected by the Cyclomatic Complexity in any of the measurements of the sensors.

H3: *Percentage of Comments in the Code (M3) – Energy Consumption (M7)*

Table 9 Energy consumption, in watts per second, of Test Case B (Create a new user)

	Redmine versions			
	Redmine 2.5.1	Redmine 3.2.0	Redmine 3.4.3	Redmine 3.4.6
ECProc	97.01	109.27 (+12.63%)	112.84 (+3.27%)	122.13 (+8.23%)
ECHdd	356.78	365.62 (+2.48%)	356.01 (−2.63%)	375.33 (+5.43%)
ECGraph	29.18	34.36 (+17.75%)	28.04 (−18.40%)	32.10 (+14.48%)
ECTotal	2831.50	3195.66 (+12.86%)	3352.10 (+4.90%)	3412.64 (+1.81%)

Table 10 Energy consumption, in watts per second, of Test Case C (Create a new task)

	Redmine versions			
	Redmine 2.5.1	Redmine 3.2.0	Redmine 3.4.3	Redmine 3.4.6
ECProc	128.72	137.02 (+ 6.45%)	155.95 (+ 13.82%)	190.80 (+ 22.34%)
ECHdd	463.15	448.92 (− 3.07%)	628.12 (+ 39.92%)	527.73 (− 15.98%)
ECGraph	41.59	41.49 (− 0.22%)	49.89 (+ 20.24%)	43.88 (− 12.05%)
ECTotal	3631.32	3776.43 (+ 4.00%)	4358.03 (+ 15.40%)	5951.85 (+ 36.57%)

Table 14 displays the results obtained from calculating the Pearson correlation coefficient for the Percentage of Comments in the Code (M3) as regards the energy consumption of each of the components analyzed (*ECProc*, *ECHdd*, *ECGraph*, and *ECTotal*). On analyzing the results, we cannot affirm that the PCC correlates significantly with energy consumption, so the null hypothesis $H3_0$ would be accepted, since there are no indications that this quality characteristic (M3) affects the energy consumption (M7).

H4: *Percentage of Duplicate Code Lines (M4) – Energy Consumption (M7)*

The results obtained from the Pearson correlation between the Percentage of Duplicate Code Lines (M4) and the energy consumption for each of the sensors (*ECProc*, *ECHdd*, *ECGraph*, and *ECTotal*), as shown in Table 15, enable us to establish that in all cases the *p*-level is above the level adopted as significant. These data do not allow us to affirm that there is any relationship between both measurements, so we cannot reject the null hypothesis $H4_0$ in any of the cases. We can therefore not certify that the PCD (M4) may affect the energy consumption.

4 Discussion

When all the data of the correlation between the different maintainability measurements chosen have been analyzed, together with the measurements of consumption obtained by the different sensors of the measurement device, we can determine that the consumption values do not follow a pattern between the different software versions. We can thus affirm that the maintainability measurements do not affect the consumption of each of the hardware components to the same extent.

This makes it necessary to analyze the consumption of each hardware component independently, so that their correlation with the measurement measures evaluated by the SonarCloud can be seen. The findings are that:

Table 11 Energy consumption, in watts per second, of Test Case D (Simulate progress)

	Redmine versions			
	Redmine 2.5.1	Redmine 3.2.0	Redmine 3.4.3	Redmine 3.4.6
ECProc	100.07	124.61 (+ 24.52%)	127.00 (+ 1.92%)	127.84 (+ 0.66%)
ECHdd	345.56	397.61 (+ 15.06%)	414.04 (+ 4.13%)	394.64 (− 4.69%)
ECGraph	30.81	35.75 (+ 16.05%)	34.07 (− 4.71%)	33.09 (− 2.88%)
ECTotal	2815.26	3372.84 (+ 19.81%)	3956.45 (+ 17.30%)	3369.25 (− 14.84%)

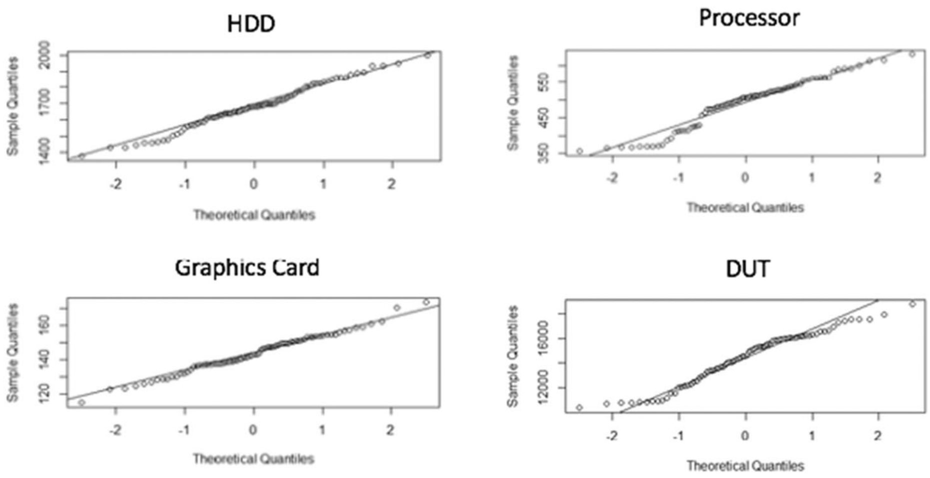

Fig. 5 Q-Q plots of the distribution of energy consumption measurements

- There seems to be no correlation between the energy consumption of the graphics card (*ECGraph*) and the classic maintainability measurements. This result could be expected, since the Redmine software does not make intensive use of graphics, given its nature.
- There appears to be no correlation between the energy consumption of the hard disk (*ECHdd*) and the classic maintainability measurements. The main functionalities of the Redmine tool do not stress hard disk storage. This provides justification for the fact that the results do not show a correlation between the maintainability measures and the energy consumed by the hard disk.
- There *does* seem to be a relationship between the energy consumption of the processor (*ECProc*) and the maintainability measurement of TLOC (M1).
- There *does* appear to be a relationship between the energy use of the total DUT consumption (*ECTotal*) and the maintainability measurement of the TLOC (M1).
- Although it was not possible to prove that there is a correlation between the cyclomatic complexity (M2) and the consumption, values that were close were obtained for the processor and the graphics card. Taking these results into account, new measurements with a greater number of samples need to be taken in order to confirm if this relationship exists, to see if there really is an influence between both variables.

To answer the research question posed, and taking as a basis the results and analyses carried out, we can conclude that the TLOC (M1) maintainability measurement, when compared with empirical evidence, affects the energy consumption of the processor (ECProc) and of the DUT (ECTotal).

Table 12 Pearson correlation analysis for the H1 hypothesis

HW component	T	df	*p*-level	Coefficient
Processor	5.149	2	**0.0357**	0.9643
Hard disk (HDD)	3.1272	2	0.0888	0.911
Graphics card	1.4559	2	0.2827	0.7173
Total DUT	5.48	2	**0.0317**	0. 968

Table 13 Pearson correlation analysis for the H2 hypothesis

HW component	T	df	*p*-level	Coefficient
Processor	1.4606	2	0.0595	0.705
Hard disk (HDD)	0.768	2	0.5225	0.4774
Graphics card	7.5714	2	0.0617	0.9827
Total DUT	1.097	2	0.387	0.61

4.1 Limits of the empirical evaluation

This section sets out the threats to the validity of the study carried out, which have to be taken into account so that it can be understood to what extent the results are valid (Wohlin et al. 2012).

4.2 Threats to construct validity

The main threat to construct validity is related to whether the measurements that we are using both for consumption and for maintainability do actually measure those factors.

To measure consumption, we have used EET, a device that has been validated and used previously in other measurements of this type.

We have used SonarCloud to measure maintainability, which is a tool that is widely recognized and accepted by professionals and academics from Software Engineering and which can also be considered trustworthy.

We believe that we are indeed obtaining values for all the measures defined, which is what we are pursuing.

4.3 Threats to internal validity

The threats to internal validity are those uncontrolled factors that may affect the results of the experiment. In our case, we are referring to threats related mainly to the DUT in which the measurements are conducted. As already mentioned, as our DUT, we try to use a computer with similar characteristics to one that can be bought on the market. In this study, measures have been taken to ensure that the DUT was always in the same conditions for the running of each different test scenario. One factor that could have affected the internal validity was the operating system used, since in the measurements of the experiment, the consumption generated by the operating system is included in the energy used by the software.

Nonetheless, as it was run on the same computer and under the same conditions, this consumption produced by the operating system is always present. In addition, given the possibility that some of the operative system processes (as updates) will be activated during the measurement, each one of these is carried out 20 times, thereby mitigating the impact of the atypical values.

Table 14 Pearson correlation analysis for the H3 hypothesis

HW component	T	df	*p*-level	Coefficient
Processor	− 3.707	2	0.0657	− 0.9343
Hard disk (HDD)	− 2.065	2	0.1749	− 0.825
Graphics card	− 2.2583	2	0.1525	− 0.8475
Total DUT	− 3.05	2	0.092	− 0.907

Table 15 Pearson correlation analysis for the H4 hypothesis

HW Component	T	df	*p*-level	Coefficient
Processor	2.5273	2	0.1273	0.705
Hard disk (HDD)	4.3603	2	0.0687	0.9512
Graphics card	0.5104	2	0.6605	0.3394
Total DUT	3.9	2	0.0598	0.94

4.4 Threats to external validity

The external validity is related to the capacity to be able to generalize the results obtained in the experiment so that they will be relevant for other cases.

- *Software selection*: We have worked with four versions of Redmine in our analysis. It is obvious that the selection of a greater number of applications and versions would have meant that better results could have been extracted, but we believe that four versions are enough for us to be able to have indications of the interaction between the maintainability and the energy consumption. In addition, in the endeavor to mitigate possible effects, the selection of the versions was undertaken following criteria of functionality and availability of the source code. There also had to be a time separation that was great enough for there to be notable changes, since it is impossible to know about the changes carried out in each version in any detail. Nevertheless, as only four versions of Redmine have been considered, the results presented in this article should be considered promising introductory results. Another aspect to take into account is that each of the versions analyzed uses a different version of the programming language. This may influence the results, but we believe that when studying how the evolution of a software affects its energy expenditure, it is logical that the versions used may vary in their maintenance.
- *Empirical validation setting*: Our experiments were conducted in a controlled setting, so they might have been different in different settings. One of the main factors that could have had an influence on the measurements of energy consumption is the configuration of the DUT in which the software being evaluated is run. We do believe, however, that although we might have obtained differences in the values of the measurements, the correlations that exist between these values will be maintained, thereby obtaining the same results.
- *Measurement hardware device (EET)*: The EET device was used to carrying out the measurement of the energy consumption. EET enables there to be exact measurements of the energy consumed by the different hardware components in a very small interval of time (approximately 90 samples per second). Obviously, the measurements obtained are specific to EET and may differ if we use other mechanisms as an estimate, or if we employ other devices (where they exist). This device has been validated and compared with another measuring device in Mancebo et al. (2018). The energy consumption results obtained by both devices were similar. In addition, EET has previously been used in other similar measurements
- Verifiability

In this work, a study was conducted to find out if there is a correlation between the maintainability and the energy consumption of the software. It is our belief that this work provides empirical evidence for the field of software energy efficiency, a field that still requires more evidence if solid bases and principles are to be provided (Verdecchia et al. 2017)[4]. In our effort to make our work more robustly replicable, all of the raw data obtained, the scripts used, and the analyses performed have been made available to whoever should need to consult them or anyone who might wish to replicate the study.[4]

5 Related work

The relationship between energy consumption and other quality characteristics is an area that is barely explored in the literature, and this is also true regarding the direct relationship of energy consumption with maintainability; however, a number of relevant pieces of work can be found. García-Mireles et al. (2018) present a systematic mapping study (SMS) in which, from a theoretical point of view, they identify and classify the potential interactions between the quality characteristics of the software product (set out in the ISO25010) and sustainability in the software context. The authors highlight various quality characteristics that may affect energy efficiency either positively or negatively. For example, increasing or improving the security or the modularity of the source code (*maintainability* characteristic) produces a negative interaction with energy efficiency and an ensuing increase in the software energy consumption. These authors also point to characteristics that have a potentially positive impact on sustainability (reusability, reliability, and usability) and highlight characteristics that may have a positive or negative influence on consumption (functional suitability, performance efficiency). In any case, as has already been commented, this work was carried out from a merely theoretical point of view.

In Calero et al. (2013), a systematic literature review is developed, in an effort to find out the current state of the art in measurements of software sustainability. They discovered that the measures that focused on sustainability also dealt with other quality characteristics from the 25,010 + S product quality model, such as performance efficiency, maintainability, portability, usability, and reliability.

As far as maintainability is concerned, García-Rodríguez de Guzmán et al. (2015) present the term "Ecological Debt," referring to the impact that some of the typical defects of software have on sustainability. There are also studies which look at the impact that the use of code refactoring patterns has on the energy consumption. However, there are no consistent results, since some of the articles identify a positive interaction with energy efficiency (Sahin et al. 2014), while others conclude that refactoring techniques can increase consumption (Park et al. 2014; Pérez-Castillo and Piattini 2014). Also related to the refactoring of the code smell is the work of Verdecchia et al. (2018), where they conclude that refactoring code smells could be a viable process to significantly improve the energy efficiency of the software but emphasize the need for further in-depth research on this issue.

Hindle (2015) presents a study that is applied to different versions of three applications which deals with the impact that the changes in software have on energy consumption (Firefox, Vuze, rTorrent). Although the results showed variation in energy consumption between the versions, it was not possible to establish that these variations were in fact related to changes in the software.

[4] http://alarcos.esi.uclm.es/sustainabilityandmaintainability

Radu (2018) describes the ecological benefits brought about through the reuse of software. Jelschen et al. (2012), for their part, expose that some reengineering techniques improve the energy efficiency of mobile applications.

Pereira et al. (2020) propose a spectrum-based energy leak detection technique to identify inefficient energy consumption in the source code of software systems. This technique uses a statistical method to associate the energy consumed with the different components of the source code of a software system, thus drawing the developer's attention to the most critical red points in his code.

In Pereira et al. (2017), the authors analyze the performance of twenty-seven software languages, providing a list of the most efficient programming languages, so that software engineers can decide which language to use when energy efficiency is a concern. A similar work is proposed by Lima et al. (2019), where they carry out a study of the energy efficiency of software programs developed in the Haskell programming language. In addition, they provide a set of guidelines to help Haskell developers save energy.

Linking maintainability to the energy efficiency of Android software is the work of Cruz et al. (2019). In this article, the authors study the impact of changes designed to improve energy efficiency on the maintainability of Android applications. Their results show that improved energy efficiency is accompanied by a significant decrease in maintenance capacity. Palomba et al. (2019) propose a study of the relationship between code smells in Android applications and energy efficiency.

Table 16 summarizes the works mentioned above, from the point of view of the relationship between energy consumption and software maintainability, and compares them with our study. For each work, it is indicated whether the approach is theoretical or practical, which characteristics are analyzed, the measurements the measurement method used, and the hardware components evaluated.

As can be observed in Table 16, only five pieces of work relate energy efficiency to software quality (maintainability) measures. The remaining articles focus on how code refactoring, or the use of reengineering techniques, or programming languages, affect energy consumption. Another aspect to highlight is that only 20% of the studies ([6], [7], and [8]) deal with the real measurement of energy efficiency. The rest of the articles estimate energy consumption or present only theoretical proposals.

Namely, the studies [1], [2], [3] have tackled the relationship between maintainability and energy consumption under a theoretical perspective, without performing empirical analysis. Other works have studied in an empirical perspective the relationship between some software features, such as the programming language, or refactoring and bad smell techniques, with the energy consumption ([4], [5], [6], [7], [10], [11], [12], [13], [15]). To the best of our knowledge, the studies [8] and [14] are the ones more directly related with our current work, as they attempt to evaluate the relationship between maintainability measures and energy consumption. However, Hindle (2015) focused on only one measure (lines of code) for the evaluation and the study by Cruz et al. (2019) is focused on a different domain (Android applications).

Furthermore, for our study, we used a hardware device (hardware-based approach) to measure the power of a specific component or the overall system. The measuring instrument employed was EET, which allows us to obtain detailed measurements from different components of the DUT (processor, hard disk, graphic card), along with the total energy consumption of the PC. In addition, this device has a higher sampling rate than other devices used in other articles. This means that the energy measurements provided by EET are more precise than the

Table 16 Comparison between other pieces of work

ID	Papers	Approach	Characteristics evaluated	Measures	Energy measurement method	Hardware component measured
[1]	García-Mireles et al. (2018)	Theoretical	Maintainability, security, usability, performance efficiency	–	No measurements	–
[2]	Calero et al. (2013)	Theoretical	Performance efficiency, maintainability, portability, usability, reliability	–	No measurements	–
[3]	García-Rodríguez de Guzmán et al. (2015)	Theoretical	Maintainability, bad smell. Anti-pattern	–	No measurements	–
[4]	Sahin et al. (2014)	Practical	Code refactoring techniques	Number of Classes, Number of Methods, TLOC	Hardware	CPU, HDD, memory
[5]	Park et al. (2014)	Practical	Code refactoring techniques	–	Software	CPU
[6]	Pérez-Castillo and Piattini (2014)	Practical	Code refactoring techniques	Number of Classes, Number of Methods, TLOC, Cycle complexity	Hardware	Total EC
[7]	Verdecchia et al. (2018)	Practical		Afferent Couplings,	Hardware	Total EC

Table 16 (continued)

ID	Papers	Approach	Characteristics evaluated	Measures	Energy measurement method	Hardware component measured
			Code refactoring techniques	Efferent Couplings, Depth of Inheritance Tree, Number of Classes, Weighted Methods per Class, TLOC		
[8]	Hindle (2015)	Practical	Maintainability	TLOC.	Hardware	Total EC, CPU, and memory utilization
[9]	Radu (2018)	Theoretical	Software reuse	–	No measurements	–
[10]	Jelschen et al. (2012)	Theoretical	Software reengineerin-g techniques	–	No measurements	–
[11]	Pereira et al. (2020)	Practical	Power leaks in the source code	Number of Classes, Number of Methods, TLOC	Software	CPU, DRAM, GPU, total EC
[12]	Pereira et al. (2017)	Practical	Programming languages	Energy used by 10 program-ming languages	Software	CPU, DRAM, GPU, total EC
[13]	Lima et al. (2019)	Practical	Programming languages	Energy used by Haskell	Software	CPU, DRAM, GPU, total EC
[14]	Cruz et al. (2019)	Practical	Maintainability Android	Design pattern, TLOC	Software	Total EC
[15]	Palomba et al. (2019)	Practical	Code refactoring	–	Software	Total EC

Table 16 (continued)

ID	Papers	Approach	Characteristics evaluated	Measures	Energy measurement method	Hardware component measured
			patterns Android			
[16]	Our proposal	Practical	Maintainability in software application	TLOC, Cyclomatic Complexity, % of Comments, % of Duplicate Code Lines	Hardware	Total EC, CPU, HDD, Graphic card

other proposals, giving more accurate information about the relationship between the energy consumption of the software and the characteristic studied in each case (in our case, the maintainability).

6 Conclusions and future work

In this work, an empirical study has been presented which aims to discover whether the energy consumption of certain software when run is affected by different maintenance measures (Total Lines Of Code, Cyclomatic Complexity, Percentage of Comments in the Code, and Percentage of Duplicate Code Lines).

The presented work is a novel study where an empirical analysis of how some software maintainability measurements can affect the energy consumption has been carried out, using a measuring instrument that allows us to obtain accurate results of the energy consumption of different hardware components. To carry out this study, four different versions of the Redmine software were chosen, and four test cases were launched, measuring the energy consumption of different hardware components of the computer it is run on (processor, HDD, graphics card, and total DUT).

Our results show that:

- The amount of Total Lines Of Code (M1) and of Cyclomatic Complexity (M2) has grown in each successive version.
- The values of Percentage of Comments in the Code (M3) and the Percentage of Duplicate Code Lines (M4), in contrast to M1 and M2, have not increased in each successive version. In some versions, the values of these measurements have even diminished in comparison with the previous version.
- The consumption of the energy of the processor (*ECProc*) and the total energy consumed by the DUT (*ECTotal*) increase in each of the successive versions, following a very familiar pattern.
- Nevertheless, the energy consumption of the hard disk (*ECHdd*) and of the graphics card (*ECGraph*) did not follow the same pattern as the previous consumption measurements. The latest version is not always the one that has greater energy consumption than the one before it.
- There seems to be a correlation only between the measurement of Total Lines Of Code (M1) and the energy consumption of the processor (*ECProc*) and of the DUT (*ECTotal*). We may thus conclude that only the measure of the TLOC can affect the energy consumption.

For the rest of the quality measurements, no significant correlations with energy consumption were found that would indicate that the energy consumption might be affected by them.

The results obtained provide us with some evidence that the energy consumption of a given software product can be affected by software quality measures. This evidence can allow us to find software measurements that could be employed as indicators of the impact on the energy consumption of the software. This study lays the foundations to research in greater depth into the relationship of maintainability and software consumption, selecting more software applications and a greater number of versions for analysis. We would thus be able to establish whether the result obtained from this analysis can be extrapolated to other software applications.

This article also throws into relief the importance of using measurement devices such as EET in similar empirical studies. They allow us to obtain exact measurements of the power consumed in the different hardware components and of the software when it is run on a computer.

In conclusion, there are some initial signs about how the amount of Total Lines Of Code can affect the energy consumption of the software. We intend to use these results as a basis for more in-depth research about the impact of different quality measures on the energy consumption. We believe that this type of research can be very useful from the developer's point of view; the professionals will be able to take the results obtained into account in the development of software that is more efficient as far as its energy use is concerned.

Funding 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

Andrae, A. (2017). Total consumer power consumption forecast. In: Nordic Digital Business Summit. Huawei Technologies.

Calero, C., & Piattini, M. (2017). Puzzling out software sustainability. *Sustainable Computing: Informatics and Systems, 16*, 117–124.

Calero, C., Bertoa, M. F., & Moraga, M. Á. (2013). A systematic literature review for software sustainability measures. In: *2013 2nd international workshop on green and sustainable software* (GREENS). IEEE, pp 46-53. https://doi.org/10.1109/GREENS.2013.6606421.

Calero, C., Mancebo, J., García, F., Moraga, M. Á., Berná, J. A. G., Fernández-Alemán, J. L., & Toval, A. (2019). 5Ws of green and sustainable software. *Tsinghua Science and Technology, 25*, 401–414.

Capra, E., Francalanci, C., & Slaughter, S. A. (2012). Is software "green"? Application development environments and energy efficiency in open source applications. *Information and Software Technology, 54*(1), 60–71.

Chien, A. A. (2019). Owning computing's environmental impact. *Communications of the ACM, 62*(3), 5–5. https://doi.org/10.1145/3310359.

Cruz, L., Abreu, R., Grundy, J., Li, L., & Xia, X. (2019). Do energy-oriented changes hinder maintainability? In: *IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, pp 29-40. https://doi.org/10.1109/ICSME.2019.00013.

Dick, M., & Naumann, S. (2010). Enhancing software engineering processes towards sustainable software product design. In: EnviroInfo. pp 706–715.

García-Mireles, G. A., Moraga, M. Á., García, F., Calero, C., & Piattini, M. (2018). Interactions between environmental sustainability goals and software product quality: A mapping study. *Information and Software Technology, 95*, 108–129.

García-Rodríguez de Guzmán, I., Piattini, M., & Pérez-Castillo, R. (2015). Green software maintenance. In C. Calero & M. Piattini (Eds.), *Green in Software Engineering* (pp. 205–229). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-319-08581-4_9.

Hindle, A. (2012). Green mining: A methodology of relating software change to power consumption. In *Proceedings of the 9th IEEE Working Conference on Mining Software Repositories* (pp. 78–87). IEEE Press.

Hindle, A. (2015). Green mining: A methodology of relating software change and configuration to power consumption. *Empirical Software Engineering 20*. pp 374–409 https://doi.org/10.1007/s10664-013-9276-6.

ISO (2011). *ISO-IEC 25010: 2011 Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models*.

Jagroep, E., van der Werf, J. M., Brinkkemper, S., Procaccianti, G., Lago, P., Blom, L., van Vliet, R. (2016). Software energy profiling: Comparing releases of a software product. In: *Proceedings of the 38th International Conference on Software Engineering Companion*. ACM, pp 523–532

Jagroep, E., Procaccianti, G., van der Werf, J. M., Brinkkemper, S., Blom, L., & van Vliet, R. (2017). Energy efficiency on the product roadmap: An empirical study across releases of a software product. *Journal of Software: Evolution and Process 29*. p 1852 https://doi.org/10.1002/smr.1852.

Jelschen, J., Gottschalk, M., Josefiok, M., Pitu, C., Winter, A. (2012). Towards applying reengineering services to energy-efficient applications. In: *2012 16th European Conference on Software Maintenance and Reengineering*. IEEE, pp 353–358. https://doi.org/10.1109/CSMR.2012.43.

Jones, N. (2018). How to stop data centres from gobbling up the world's electricity. *Nature, 561*(7722), 163–167.

Lima, L. G., Soares-Neto, F., Lieuthier, P., Castor, F., Melfe, G., & Fernandes, J. P. (2019). On Haskell and energy efficiency. *Journal of Systems and Software, 149*, 554–580.

Mancebo, J., Arriaga, H. O., García, F., Moraga, M., de Guzmán, I. G. -R., & Calero, C. (2018). EET: A device to support the measurement of software consumption. In: *Proceedings of the 6th International Workshop on Green and Sustainable Software, Gothenburg, Sweden.* ACM, pp 16–22. https://doi.org/10.1145/3194078.3194081.

Palomba, F., Di Nucci, D., Panichella, A., Zaidman, A., & De Lucia, A. (2019). On the impact of code smells on the energy consumption of mobile applications. *Information and Software Technology, 105*, 43–55.

Park, J. J., Hong, J.-E., & Lee, S.-H. (2014). *Investigation for software power consumption of code refactoring techniques* (pp. 717–722). In: SEKE.

Penzenstadler, B., Raturi, A., Richardson, D., Calero, C., Femmer, H., & Franch, X. (2014). Systematic mapping study on software engineering for sustainability (SE4S). In: *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering (EASE)*, London, United Kingdom, vol 14. ACM, pp 1-14. https://doi.org/10.1145/2601248.2601256.

Pereira, R., Couto, M., Ribeiro, F., Rua, R., Cunha, J., Fernandes, J. P., & Saraiva, J. (2017). Energy efficiency across programming languages: How do energy, time, and memory relate? In: *Proceedings of the 10th ACM SIGPLAN International Conference on Software Language Engineering*, Vancouver, Canada. ACM, pp 256–267. https://doi.org/10.1145/3136014.3136031.

Pereira, R., Carção, T., Couto, M., Cunha, J., Fernandes, J. P., & Saraiva, J. (2020). Spelling out energy leaks: Aiding developers locate energy inefficient code. *Journal of Systems and Software, 161*, 110463.

Pérez-Castillo, R., & Piattini, M. (2014). Analyzing the harmful effect of god class refactoring on power consumption. *IEEE Software, 31*(3), 48–54.

Pinto, G., & Castor, F. (2017). Energy efficiency: A new concern for application software developers. *Communications of the ACM, 60*(12), 68–75.

Procaccianti, G., Fernández, H., & Lago, P. (2016). Empirical evaluation of two best practices for energy-efficient software development. *Journal of Systems and Software, 117*, 185–198.

Radu, L. (2018). An ecological view on software reuse. *Informatica Economica, 22*, 75–85. https://doi.org/10.12948/issn14531305/22.3.2018.07.

Sahin C, Pollock L, Clause J (2014) How do code refactorings affect energy usage? In: Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, Torino, Italy, vol 36. ACM, pp 1–10. https://doi.org/10.1145/2652524.2652538.

Verdecchia, R., Ricchiuti, F., Hankel, A., Lago, P., & Procaccianti, G. (2017). Green ICT research and challenges. In *Advances and New Trends in Environmental Informatics* (pp. 37–48). Springer.

Verdecchia, R., Saez, R. A., Procaccianti G, Lago P (2018) Empirical evaluation of the energy impact of refactoring code Smells. In: *ICT4S2018. 5th International Conference on Information and Communication Technology for Sustainability*. pp 365-383. https://doi.org/10.29007/dz83.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M., Regnell, B., & Wesslén, A. (2012). *Experimentation in software engineering*. Springer Science & Business Media.

Javier Mancebo is a PhD student in computer science by 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.

Coral Calero is a 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.

Felix Garcia is a 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.