The Journal of Supercomputing (2023) 79:13766–13793
https://doi.org/10.1007/s11227-023-05202-6

Estimating the energy consumption of model-view-controller applications

Daniel Guamán[1,2] · Jennifer Pérez[1] · Priscila Valdiviezo-Diaz[2]

Accepted: 15 March 2023 / Published online: 30 March 2023

Abstract

For information and communication technology to reach its goal of zero emissions in 2050, power consumption must be reduced, including the energy consumed by software. To develop sustainability-aware software, green metrics have been implemented to estimate the energy consumed by the execution of an application. However, they have a rebound energy consumption effect because they require an application to be executed to estimate the energy consumed after each change. To address this problem, it is necessary to construct energy estimation models that do not require the execution of applications. This work addresses this problem by constructing a green model based on size, complexity and duplicated lines to estimate the energy consumed by model-view-controller applications without their execution. This article defines a model constructed based on 52 applications. The results were accurate in twelve applications, which showed that the joule estimation was very close to reality, avoiding the energy consumed by the execution of applications.

Keywords Green software · Software architectures · Architectural patterns · Model-view controller (MVC) · Energy consumption estimation

Daniel Guamán and Priscila Valdiviezo-Diaz have contributed equally to this work.

✉ Jennifer Pérez
jenifer.perez@upm.es

Daniel Guamán
daguaman@utpl.edu.ec

Priscila Valdiviezo-Diaz
pmvaldiviezo@utpl.edu.ec

[1] Departamento Sistemas Informáticos, ETSI Sistemas Informáticos, Universidad Politécnica de Madrid, Alan Turing s/n, 28031 Madrid, Spain

[2] Departamento Ciencias de la Computación y Electrónica, Universidad Técnica Particular de Loja, San Cayetano Alto, 1101608 Loja, Ecuador

1 Introduction

In 2020, the carbon footprint of information and communication technology (ICT) was estimated to be 1.43 Gt (billion tonnes) of CO2e [1]. Currently, ICT generates around 2.5% of global greenhouse gas emissions, which poses a challenge in reaching the goal of zero emissions by 2050 [2]. In addition, mobile, fixed and data centre operators must meet the goal of limiting global warming to 1.5 °C by 2050 in order reduce the risks and effects of climate change [3, 4]. ICT has positive effects, such as substituting mobility and travel with teleworking and virtual meetings [2]. However, it has negative effects on sustainability [5]. The high increments of electronic devices, computers, server connections and the execution of their applications contribute to a huge energy consumption increment [6]. Therefore, ICT and energy consumption play critical roles in achieving current sustainability challenges [7] Consequently, ICT solutions must be required to support not only the sustainability BY ICT, but also the sustainability IN ICT [8].

Currently, the design of sustainable hardware and communications have been achieved by producing hardware devices and communication protocols that greatly reduce power consumption [9, 10]. However, much work is still needed in the software field [1] to reduce the additional power that the execution of software applications consumes in the hardware devices in which they are deployed [11]. Hence, in the sustainability technical dimension, it is necessary to construct new mechanisms, such as practices, criteria and metrics, to support green-aware activities during the construction and evolution of software [12, 13]. Green-aware mechanisms aim to reduce the complexity of software to improve its quality, performance and energy efficiency. Reducing software complexity is important because the way in which software is coded influences both energy consumption and energy efficiency [14], which is due to the input/output (I/O) hardware instructions it generates [15].

The reduction of software complexity, and by extension its energy-efficiency, depends on (i) the improvement of the logics of classes and methods, the data types of variables and parameters, and the control structures at the code level; and (ii) the use of good design practices and architectural patterns and styles that improve performance, scalability, maintainability and resource allocation to minimise data interchange [16]. However, in current software development, changes are rapid and continuous because of the wide adoption of agile methodologies and fast time-to-market, which bring software engineers to make sub-optimal green code and/or design decisions that may lead to less sustainable applications. This problem should be addressed by providing green-oriented development mechanisms that support sustainability-aware decisions during coding and/or design activities in the development and maintenance processes. To that end, it is necessary to guide software engineers during changes and decision-making to guarantee the construction and maintenance of green software. This guidance can be provided by green estimation models.

Green estimation models are based on green metrics that are used to estimate the energy consumed by applications when they are being executed [17].

However, the problem with green metrics is that, in estimating the energy consumed executing applications, they generate an energy consumption rebound effect [18]. This problem is even higher in agile methodologies, which are open to changes and iteration after iteration the design and code are changed [19], which exponentially increases the need to recalculate green metrics to guide the required changes and the energy consumption rebound. There are estimation models to determine the energy consumption by applications before their execution based on instructions or method calls [17], which avoids the problem of energy consumption rebound effects. However, they do not consider architectural design and the key role of software quality in green software [15], although that software architecture patterns and software quality may influence energy consumption. Therefore, it is necessary to develop quality and architectural design-based models to estimate the energy consumption of applications without the need for their execution. This work addresses this problem following the recommendation of Seo et al. [20] to construct one power consumption estimation model for each specific architectural pattern or architectural style, since the wide variability of architectural patterns and styles hampers the construction of a common power consumption estimation model. Specifically, this work presents a green model to estimate the energy consumption of applications that specialise in software architectures and that implement one of the most commonly used architectural patterns, the model-view-controller (MVC) [21–23] without the need for execution. the green model, model-view-controller-complexity and code smell energy model (MVC-CCsEM), is based on the quality metrics of complexity (Cyclomatic Complexity [24]), maintainability (Duplicated Lines) [25], and Size.

This article describes the construction process of this multiple linear regression green model for MVC applications for use by software engineers to estimate the power consumption of applications during software construction and maintenance without having to execute them and incur power consumption rebound effects. This MVC-CCsEM was constructed using a training corpus of 52 applications. In addition, this study was performed to determine how the model works in estimating the energy consumption of twelve different MVC applications. The results of the study demonstrated that the energy estimations of the model are valuable and accurate enough to conclude that the model would be useful for guiding energy-aware changes during software development and the maintenance of MVC applications.

The rest of this article is structured as follows: Sect. 2 examines related work about green metrics and estimation models. In Sect. 3, we describe the process used to construct the MVC-CCsEM based on the analysis of 52 applications. Section 4 presents a study of twelve applications in which the MVC-CCsEM energy consumption model is used to estimate energy consumption. The threats to the validity of this work are discussed in Sect. 5. Finally, Sect. 6 concludes the paper and recommends directions for future research.

2 Related work

Both industry and academia have obtained successful results in constructing sustainable hardware and communications by reducing their power consumption [9, 10]. Because ICT solutions comprise both hardware and software, not only must the hardware be sustainability-aware, but the software must also reduce its power consumption in order to maximise the sustainability of ICT solutions [11]. The evaluation of ecological software depends on the application's structure and the hardware infrastructure used for its deployment [17]. The energy consumed by an application cannot be fully isolated from the basal consumption of the hardware by which it is executed. It is possible to measure hardware without executing an application and then measure the application without measuring peripheral devices and other background processes. Hence, measuring the specific energy consumption of an application requires a calculation, which is in fact an estimation. Therefore, green metrics are considered estimations instead of measurements. Green metrics and models are used to measure different applications or versions of the same application that are executed on the same hardware to determine the greenest software solution.

2.1 Green metrics: estimation from software execution

Green metrics estimate the energy consumption of applications when they are being executed. Several metrics described in the literature are reviewed in this section.

To measure the energy consumption and hardware utilisation of software, Guldner et al. [26] defined a method for evaluating "energy consumption" and "processor utilisation". To collect measurements using green metrics, Michanan et al. [27] developed an interface based on a collection of classes using a dynamic data structure called GreenC5 to evaluate applications that used different workloads.

The collection of green metrics is supported by tools that are used to implement dynamic analysis techniques by applying calculation models to obtain only the specific power consumption of an application. Some well-known tools that measure energy consumption are RAPL [28], Microsoft Joulemeter [29], jRAPL [30], PowerAPI [31], and Jalen [32]. In this work, we used Microsoft Joulemeter to measure the energy consumption of the applications. Sehgal et al. [33] also used this tool for experimentation in their research.

Lago et al. [34] collected 66 software metrics and models to measure and estimate energy consumption, 17 of which were related to software architecture [20]. Other works have used these metrics and tools to estimate the energy consumption of a specific kind of software architecture, such as product line architectures [35] or cloud software architectures. Specifically, sustainable cloud software architectures have several open challenges [36], previous studies have attempted to define algorithms and energy-aware techniques [37, 38] that use green metrics to support sustainable solutions.

On the other hand, some previous works have used green metrics to demonstrate the relationship between the quality of a code and power consumption. Cairo et al. [39] described how smells and cyclomatic complexity influence the appearance of

errors in applications. Sehgal et al. [33] evaluated the impact of refactoring smells on total energy consumption. These previous works revealed that Cyclomatic Complexity and smells are key factors in energy consumption.

These works were based on green metrics, which required calculations to execute the software application and generate an energy consumption rebound [18]. In this work, we go beyond these previous studies by developing a solution to avoid the rebound power consumption of green metrics.

2.2 Green models: estimation from green metrics

The measurements of green metrics obtained from application execution allow for the construction of energy consumption estimation models, which are also called metrics, based on them and other kinds of metrics.

Chatzigeorgiou and Stephanides [17] proposed three metrics to estimate the energy consumption of software: executed instruction count measure (EIC), memory access count measure (MAC) and software energy (SEM). These metrics use external meter instruments to measure in runtime. Specifically, EIC measures the number of instructions of an application executed by the processor. MAC measures the number of memory accesses of an instruction, and SEM calculates the average energy cost of executing an instruction. Dufour et al. [40] proposed three "hot spot" metrics based on the methods and classes of an application: total execution frequency (TEF) of a method, class invoked frequency (CIF), and class invoked time (CITC). Energy wasting rate [41] is another metric that uses measurements during the execution, which are part of the software's logic structure (number of classes, methods, and structures for data exchange). The values obtained from the metric allow the programmer to know which Java classes or methods need to be changed or refactored to save power. These previous works revealed that the size of an application (source lines of code (SLOC), number of classes, methods, etc.) is a key factor in its energy consumption. Therefore, size is part of our estimation model. However, these models are focused on size and do not estimate considering the quality of code.

In this regard, Fu et al. [42] estimated the power consumed by an application using statistical algorithms and machine learning. However, although this work was based on quality attributes, it had the disadvantage that the application must be executed to obtain the estimation, generating a rebound energy consumption effect. Because our goal was also to reduce the power consumption of our solution, we avoided this negative effect by defining an estimation model based on Size and quality attributes without the need to execute the application.

Regarding design patterns, Feitosa et al. [43] confirmed that design decisions also affect energy consumption. The authors used crossover experiments to estimate the energy consumption of solutions that applied the GoF design patterns state/strategy and template method. To validate the estimated energy measurements, they applied statistical models and the agglomerative hierarchical clustering technique using SLOC and message passing coupling (MPC) as metrics. This work was innovative in early work on estimations because it was based on design patterns and considered the role of architectural components, not only based on Size (SLOC). They also

considered that a quality attribute was coupling (MPC). However, it is important not only to use one quality metric but also to use one that previous studies have revealed is a key factor in power consumption, such as Cyclomatic Complexity and smells [33].

Several previous studies have addressed the energy consumption estimation of software architectures. Regarding software architectures, Seo et al. [20] acknowledged the need to create specialised early estimation models that would enable different architectural patterns and styles to be precise. This capability also enables an engineer to employ energy cost predictions to determine the most appropriate architectural style for a given distributed application before the implementation of the system. This provides the opportunity to compare (i) the power consumption of different architectural patterns or architectural styles of the same application and (ii) the power consumption of different applications designed with the same architectural pattern or architectural style. In this work, Seo et al. [20] presented 17 architectural consumption metrics that estimated the energy consumption of components and connectors. They showed that, because components and connectors play diverse roles in architecture, they incur different wastages of energy consumption. Among the 17 metrics, it is important to emphasise the generic energy cost model because it measures a complete architecture by distinguishing the energy consumed by the components (i.e. computational elements) and connectors (i.e. interaction elements). The generic energy cost model specialises in other metrics, such as the client–server energy cost model and the pub–sub energy cost model, which facilitate the early energy estimation of architectural patterns. However, these estimation metrics are based on Size, and quality attributes are not included in the estimation.

2.3 Green models for estimating power consumption during software life cycle

Fast time-to-market and the wide adoption of agile methods have led software engineers to apply a fast design decision-making process that may result in sub-optimal sustainable decisions for software applications, if suitable sustainable tools are not adopted as part of the process. Ardito et al. [44] provided guidelines for correctly measuring software applications and describing the techniques, tools and models that can be used, whereas Georgiou et al. [45] drew attention to this problem throughout the entire software life cycle. They provided guidance for using tools and techniques during the requirements analysis, design, implementation, testing and maintenance of software applications. Ournazi et al. [46] also emphasised this problem and provided a solution for addressing sustainability in software requirements and measuring power consumption during software construction to fulfil green requirements. However, these tools are based on green metrics that require software execution, which leads to an undesirable power consumption rebound effect throughout the entire software life cycle.

In summary, based on the relevant literature, it may be concluded that there are no early green models that do not require the execution of a software application to estimate its energy consumption by considering its design and quality attributes. To develop an integrated solution that takes into account the needs identified in our

Table 1 Needs identified in the power consumption estimation of software applications

Software power consumption estimation needs	Related work that identifies or evidence the need
Constructing sustainable tools for supporting the entire software life cycle from requirements analysis to maintenance phases	[45, 46]
Avoiding the rebound effects of executing applications for estimating power consumption	[18, 43]
Creating specialised early estimation models for different architectural patters or architectural styles	[20, 34, 36–38, 43]
Considering the Size of an application (source lines of code (SLOC), number of classes, methods, etc.) as a key factor in the power consumption estimation	[40, 41]
Considering quality attributes, in general, and Cyclomatic Complexity and smells in particular, as key factors in the power consumption estimation	[33, 39, 43, 45]

review of related studies (see Table 1), in this work an estimation model was developed to determine the power consumption of MVC software architectures based on not only size but also quality attributes, such as complexity and smells (i.e. code smells and/or Duplicated Lines). In addition, this early estimation model can be applied as often as necessary during software construction and maintenance processes without generating rebound power consumption effects. Finally, this early estimation model was specialised in the MVC pattern to facilitate comparison of the power consumption of different applications or versions designed in the MVC pattern.

3 Construction of the MVC-CCsEM model

To construct the MVC-CCsEM, we defined and followed a rigorous process formalised in the standard of Software and Systems Process Engineering Meta-Model (SPEM) [47] to guarantee its replication and reuse for creating new estimation models based on quality and energy metrics (see Fig. 1). The process of constructing the MVC-CCsEM comprised five phases: scope definition, building, profiling, analysis and construction. Each phase included a set of tasks as well as their inputs and outputs. Tasks related to energy estimation metrics required highly rigorous testing to avoid bias and obtain precise results. Therefore, the process was based on the evaluation of energy efficiency by Mancebo et al. [48] to define energy consumption measurement tasks. Each phase, its tasks and its results are described in the following subsections.

3.1 Phase I: scope definition

This phase consists of two tasks: specifying the requirements and the goal of the model. The requirements specification consists in defining the scope and context of the model, as well as the inclusion and exclusion criteria. The goal of the model is

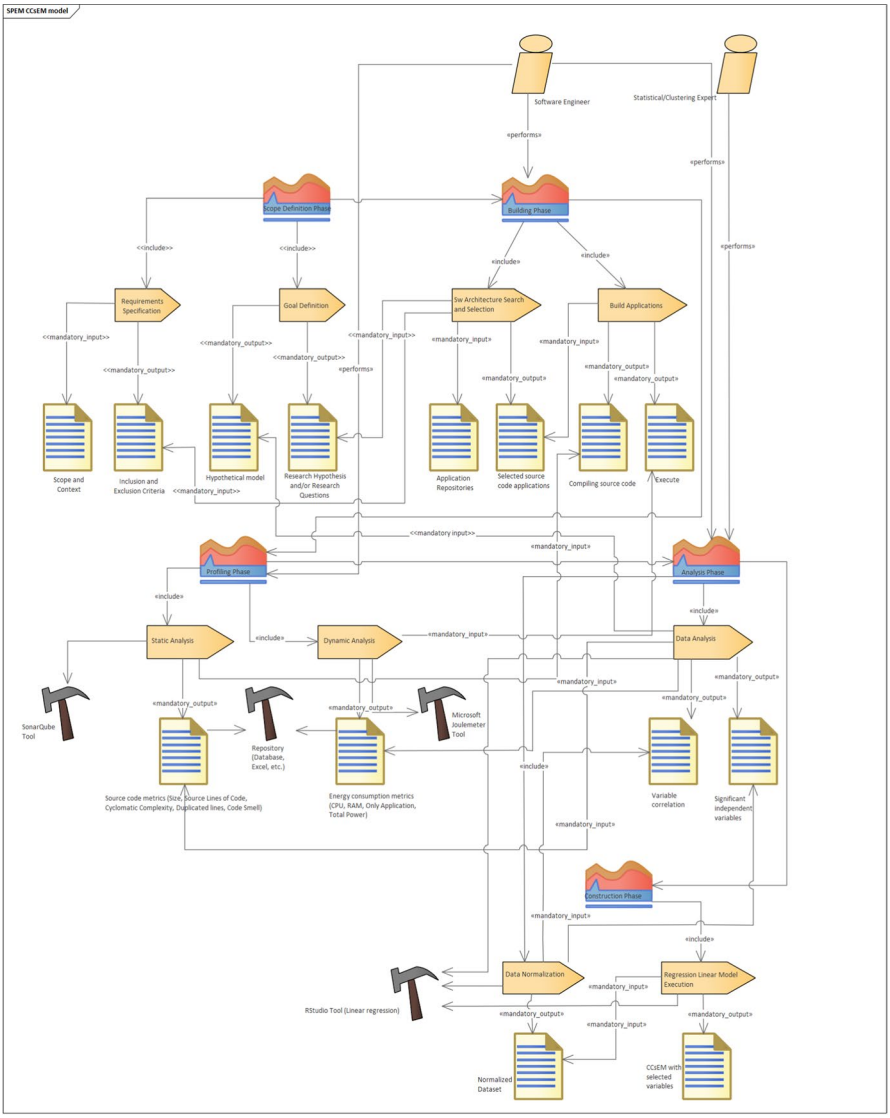

Fig. 1 The construction process of MVC-CCsEM model in SPEM

defined as a set of hypotheses and/or research questions (see Fig. 1). The "scope and context" input and the "inclusion and exclusion criteria", "hypothetical model" and "hypotheses" outcomes are described in the following subsections.

3.1.1 Scope and context

In our previous work, we defined the CCsEM model to estimate the energy consumption of software components without being executed (see Eqs. 1 and 2) [49].

Equation 1 presents the final model in terms of variables that represent size (SIZE and SLOC), complexity (CC) and maintainability (CS), and the model error (e) (see the variables description, Sect. 3.1.3). However, it is important to consider that the obtained results are normalised by a logarithmic transformation. Therefore, to obtain the estimation value without normalisation, it is necessary to apply the inverse function of the log (CCsEM), that is, the exponential function. Therefore, the power consumption estimation of the model CCsEM is obtained by applying the exponential function, as shown in Eq. 2.

$$\begin{aligned}\log(\text{CCsEM}) = 4.187 - 0.5925\log(\text{SIZE}) - 0.84031\log(\text{SLOC}) \\ + 0.3332\log(\text{CC}) + 0.4084\log(\text{CS}) + e\end{aligned} \tag{1}$$

$$\text{CCsEM} = \exp(\log(\text{CCsEM})) \tag{2}$$

This CCsEM model is based on the metric Generic Energy Cost Model [20], which estimates the energy consumption of software architectures in terms of the energy costs of components and connectors. The CCsEM model was narrowed to facilitate the energy consumption estimation of components and simple interactions to avoid the uncertainty generated by complex connectors. They vary from simple interactions [50] between components to complex orchestrators among components that implement coordination protocols [51]. In fact, the energy consumption of complex connectors that orchestrate a high number of components with complex policies may have a high impact on energy consumption even higher than the components.

Complex connectors are applied by a wide variety of architectural patterns [23] and technologies [52–54] which influence energy consumption to different degrees. Therefore, they require a separate specific study to define an appropriate estimation model and determine their energy consumption, depending on their role, technology, protocol and number of orchestrated components, among other properties. In the present work, we went beyond the CCsEM model by considering both components and connectors, that is, the complete architecture. However, because the complex connector variability and their derived uncertainty must be avoided, this work was constrained to a specific architectural pattern, following the recommendations of Seo et al. [20] to construct one power consumption estimation model for each specific architectural pattern or style. In this work, the CCsEM model is specific to software architectures that implement the MVC pattern. Therefore, the training of the model is also constrained by the execution of applications that implement MVC software architectures.

Guaman et. al [55] revealed the metrics that are the most frequently used as static analysis tools to examine software architectures and applications. In addition to size, these metrics include complexity and maintainability. Maintainability is usually supported by tools that provide technical debt (TD) analysis. These TD tools measure smells, such as code smells and duplicated lines. In addition, Sehgal et al. [33] found that cyclomatic complexity and smells are vital factors in power consumption. This base of knowledge was used to construct a quality-aware energy estimation model to predict energy consumption before execution. In this work, the metrics that were measured to construct the MVC-CCsEM are as follows:

- Size: Size was measured by the application Size and SLOC.
- Complexity: Complexity was measured by Cyclomatic Complexity.
- Maintainability: Maintainability was measured by smells, Duplicated Lines and code smells.

These decisions allowed the model to fulfil the needs identified in the software power consumption estimation field (see Table 1). In addition, it is important to emphasize the application context of the model. The MVC-CCsEM model was defined to support a sustainable software construction and maintenance without the need for execution. This means that the model can be calculated as many times as the engineer requires without incurring additional power consumption. This calculation during the software construction and maintenance is locally performed by an engineer on a computer; obtaining the measurements of the quality and size from tools such as SonarQube [56, 57]. Therefore, the model is independent of the computer or IoT device on which it is deployed and an MVC-CCsEM-improved greener code during software development fosters greener ICT solutions if we deployed it on power-efficient computers and IoT devices in the real setting. Then, these greener ICT solutions can be additionally measured using green metrics and models (see Sects. 2.1 and 2.2) in their real setting to measure different devices, communications and user-connections.

3.1.2 Inclusion and exclusion criteria

The criteria are based on the fact that the model only is going to estimate the energy consumption of MVC applications in order to estimate applications with complex connectors constraining their variability to only one architectural pattern. Based on this premise, we selected applications by applying the following inclusion criteria (IC) and exclusion criteria (EC):

- *IC1*: Executable Java or C# MVC applications in which the SonarQube tool can perform a static analysis to extract the metrics size, source lines of code, cyclomatic complexity, duplicated lines and code smells.
- *EC1*: Non-MVC applications.
- *EC2*: Java and C# MVC applications with compilation errors.
- *EC3*: Java and C# MVC applications with execution errors.
- *EC4*: Applications not included in IC1.

3.1.3 Goal

The main goal of this work was to construct a green model for estimating the energy consumption of the MVC software architectures in terms of size, maintainability and complexity to avoid software execution (see Table 1). To address this goal, the following two hypotheses (H1 and H2) and null hypothesis were defined (H0):

- *H1*. The power consumption of the applications that implement a MVC software architecture can be estimated accurately without their execution from their Size, Complexity and Maintainability.
- *H2*. The complexity and maintainability cannot be depreciated in the power consumption estimation of applications that implement an MVC software architecture without being executed because both are significant.
- *H0*. ¬ (H1 and H2).

To validate these hypotheses, this work also defined an initial hypothetical model named Model-View-Controller Complexity and Code smells Energy Model(MVC-CCsEM). This model is based on the metrics (variables) defined in the "Scope and Context" input of the "requirements specification" task of the process (see Fig. 1 and Sect. 3.1.1), i.e. Size, Maintainability and Complexity. Because the model required several variables to be considered, the MVC-CCsEM was specified as a multiple linear regression model. The MVC-CCsEM is the value that must be calculated, that is, the dependent variable of a multiple linear regression model, whereas the variables used to calculate the dependent variable (MVC-CCsEM) are independent variables in a multiple linear regression model. As a result, the validation of the multiple linear regression model allowed for the determination of significant independent variables for its calculation. The selected variables are detailed as follows:

- *SIZE*[1]: SIZE is a categorical variable with k categories (XS: <1K, S:1K-10K, M: 10K – 100K), L (100k-500k), XL (>500k)) [56].
- *SLOC*: Source Lines of Code.
- *CC*: Cyclomatic Complexity.
- *DL*: Duplicated Lines.
- *CS*: Code Smells.

Based on these initial variables, the formula for the multiple linear regression model was defined, as shown in Eq. 3).

$$\begin{aligned} \text{MVC} - \text{CCsEM} = \beta_0 + \beta_{\text{SIZE}}\text{SIZE} + \beta_{\text{SLOC}}\text{SLOC} \\ + \beta_{\text{CC}}\text{CC} + \beta_{\text{CS}}\text{CS} + \beta_{\text{DL}}\text{DL} + e \end{aligned} \tag{3}$$

Let be β_0: the value of the dependent variable when the rest of the variables are set to zero; β_i: the regression coefficient of its independent variable, that is, the average effect of a unit increment of the independent variable on the dependent variable, being i=SIZE, SLOC, CC, CS or DL; and e: the model error.

The *H2* is validated during the construction of the model. If the two variables of maintainability are not significant or the CC is not significant, the hypothesis *H2* is rejected. If the *H2* is accepted, then the hypothesis *H1* is validated with the model execution by determining if the estimation results of MVC-CCsEM are accurate

[1] SIZE in upper case is only refereeing to this variable, whereas size without upper case is refereeing to the complete category, i.e. SIZE + SLOC.

for its adoption. To address our goal both hypotheses *H1* and *H2* must be fulfilled, which means the rejection of *H0*.

3.2 Phase II: building

To construct the model, a training corpus data set is required. It must be composed of a set of MVC applications and their quality and energy measurements. This data set is created in the building phase. It consisted of two tasks: "SW architecture search and selection" and "Build applications" (see Fig. 1). The tasks are described in the following subsections.

3.2.1 SW architecture search and selection

This first task aims to search the training corpus of MVC applications in public and accessible repositories. By taking into account the inclusion criteria (IC) and exclusion criteria (EC), a search of the C# and Java MVC applications was performed in GitHub (https://github.com/), where the software code is accessible and was written by different programmers to avoid human and programming bias. The applications were then downloaded from GitHub and stored locally on the computer. In this search, we obtained 74 applications. The second step consisted of confirming that none of the downloaded applications fulfilled EC1. Therefore, the applications were opened in their corresponding integrated development environments (IDE), and it was confirmed that their architecture implemented an MVC pattern. When it was confirmed that our data set was composed of MVC applications, it was possible to address the second task. At this point, we excluded 19 applications that applied EC1, and 55 applications were preserved for the data set.

3.2.2 Build applications

This task requires two actions (1) compile and execute the MVC applications and (2) determine that they did not fulfil EC2 and EC3. To that end, the applications must be compiled and executed using the corresponding Integrated Development Environment (IDE). This execution could include pre-processing the code, such as updating dependencies or preparing the JAR files for Java applications. In this task, three applications were removed from the data set by applying EC2, EC3 and EC4. The resulting data set of 52 MVC applications was obtained (see Tables 2 and 3). These applications encompass a range of sizes in percentages of 36.5 % XS, 34.6% S, 1.9 % M, 25% L, and 1.9%XL. Specifically, 73.1% are codified in Java, and 26.9% are codified in C#, and each addresses different domains, as shown in Table 2 and Fig. 2.

3.3 Phase III: profiling

The Profiling phase consists of two tasks: "Static Analysis" and "Dynamic Analysis" (see Fig. 1). The Static Analysis is conducted to calculate the size, complexity

Table 2 Applications classified by size, domain and programming language

	Number of applications
Size	
XS	20
S	18
L	13
XL	1
Programming language	
Java	38
C#	14
Domain	
Entertainment	6
Business	17
Tourism	10
Leisure	8
Reservation	6
Utilities	5

and maintainability metrics from the selected applications. The Dynamic Analysis is conducted to measure the energy consumption of the applications during their execution. These measurements constitute the training corpus of the model. These two tasks are described in the following subsections.

3.3.1 Static analysis

The static analysis was performed using the SonarQube tool [57]. Each application was loaded in SonarQube, and its SIZE, SLOC, CC, DL and CS were collected by executing the sonar-scanner command. The collected data were then stored in a MySQL repository by SonarQube. We also stored the data in an Excel file for processing.

3.3.2 Dynamic analysis

The dynamic analysis was performed using Microsoft Joulemeter, which measures the energy consumed by the execution of applications and stores the measurements in a CSV file. Specifically, the measurements provided by Microsoft Joulemeter are CPU, MONITOR, RAM, BASE, Application Power (CPU only), and Total Power (see Fig. 3). To obtain valuable data from the dynamic analysis, it is required that the context of execution and the collection procedure be the same for all the applications under measurement [48]. Because all the applications are measured under the same execution conditions, Joulemeter based their calculation on the same basal consumption, which guaranteed that we compare only the power consumption of the application, and the measurement is independent of the computer. Next, we define

Application ID	Name	Domain	Programming language	Size	Functionality
App1	gmagil6 - Teletickets	Academic-Entertainment	C#	L	Application to manage the sport and cultural ticket sales for concerts, musical, sport events, museums, etc. Modules and functionalities to manage the different kinds of tickets.
App2	gmagil14	Academic-Entertainment	C#	XL	Application to manage the sport and cultural ticket sales for concerts, musical, sport events, museums, etc. Modules and functionalities to manage the different kinds of tickets.
App3	gmagil15	Academic-Entertainment	C#	L	Application to manage the sport and cultural ticket sales for concerts, musical, sport events, museums, etc. Modules and functionalities to manage the different kinds of tickets.
App4	IPSFlix	Academic-Leisure	C#	L	Application to manage the supply of online leisure resources. Modules and functionalities to manage the different kinds of leisure resources.
App5	ProTicket	Academic-Entertainment	C#	L	Application to manage the sport and cultural ticket sales for concerts, musical, sport events, museums, etc. Modules and functionalities to manage the different kinds of tickets.
App6	proyecto6 - VirtualDreams	Academic-Leisure	C#	L	Application to manage the supply of online leisure resources. Modules and functionalities to manage the different kinds of leisure resources.
App7	ProjectIPSeFunG4	Academic-Leisure	C#	L	Application to manage the supply of online leisure resources. Modules and functionalities to manage the different kinds of leisure resources.
App8	TicketNation	Academic-Entertainment	C#	L	Application to manage the sport and cultural ticket sales for concerts, musical, sport events, museums, etc. Modules and functionalities to manage the different kinds of tickets.
App9	TravelTester	Academic - Tourism	C#	L	Application to manage travels. Modules and functionalities to manage the accomodation, transport and tourism activities.
App10	VR_Traveling	Academic - Tourism	C#	L	Application to manage travels. Modules and functionalities to manage the accomodation, transport and tourism activities.
App11	Ticketstips	Academic-Entertainment	C#	L	Application to manage the sport and cultural ticket sales for concerts, musical, sport events, museums, etc. Modules and functionalities to manage the different kinds of tickets.
App12	FinalProject	Academic - Tourism	C#	L	Application to manage travels. Modules and functionalities to manage the accomodation, transport and tourism activities.
App13	Teletravelling	Academic - Tourism	C#	L	Application to manage travels. Modules and functionalities to manage the accomodation, transport and tourism activities.
App14	Projectagiles5	Academic - Tourism	C#	L	Application to manage travels. Modules and functionalities to manage the accomodation, transport and tourism activities.
App15	Tickets	Academic-Reservation	Java	S	Application to manage transportation tickets. Modules and functionalities: tickets, employees, passengers, routes.
App16	Library	Academic-Reservation	Java	S	Application to manage library data borrowing books, magazines, etc. Modules and functionalities: student, teacher, bibliographic material, loans.
App17	BoulderDash	Academic-Leisure	Java	S	Playful game of strategies to play using the up, down, left, right arrows.
App18	Accounting System	Academic-Business	Java	S	Application to manage accounting data. Modules and functionalities: Audit, Catalog, Company, Users, Roles
App19	Test	Academic-Reservation	Java	S	Application to manage library data borrowing books, magazines, etc. Modules and functionalities: student, teacher, bibliographic material, loans
App20	java-swing -contact	Academic-Business	Java	XS	Application to manage people's contact information. Modules and functionalities: Registration, Update, Deletion
App21	AlarmClock	Academic-Utilities	Java	XS	Application to configure the operation of alarms
App22	DesktopMVC	Academic-Business	Java	XS	Application for the management of personal data of people
App23	mvc-Customers TravelsAgency	Academic - Tourism	Java	XS	Application to manage travel agency data. Modules and functionalities: Passengers, routes and tickets.
App24	Mvc_Calc_Pro	Academic-Utilities	Java	XS	Application that performs the 4 basic mathematical operations using different algorithms.
App25	LibraryManagement1	Academic-Reservation	Java	S	Application to manage library data according to a type (books, magazines). Modules and functionalities: student, teacher, bibliographic material, loans.

Fig. 2 Data set: description of applications

the context of execution and the collection procedure followed during the dynamic analysis:

- *Context of Execution*: It is important to include both hardware and software configurations.

Application ID	Name	Domain	Programming language	Size	Functionality
App26	BrickBreaker	Academic-Leisure	Java	S	Strategy game break bricks, use the keyboard
App27	java-swing -mvc-master	Academic-Business	Java	XS	Application to manage personal data. Modules and functionalities: register, update, search and delete.
App28	java-tetris-mvc-master	Academic-Leisure	Java	XS	Tetris game, use the keyboard
App29	HotelSystem JavaSwingMVC	Academic-Reservation	Java	S	Hotel management application. Modules and functionalities: clients, booking, rooms, reservations
App30	Electronic Calendar	Academic-Business	Java	XS	Application to manage personal data. Modules and functionalities: register, update, search and delete.
App31	StudentGrades	Academic-Business	Java	XS	Application to manage student grades. Modules and functionalities: register, edit, update and delete students, subjects and grades
App32	Registers	Academic-Business	Java	XS	Application to manage personal data. Modules and functionalities: register, update, search and delete.
App33	CurrencyConversorGUI	Academic-Utilities	Java	XS	Application to perform the conversion of a numerical value to different currencies
App34	loginmvc19	Academic-Business	Java	XS	Application for user registration and authentication. Module and functionality for user creation.
App35	MVC_demo	Academic-Utilities	Java	XS	Application that performs the 4 basic mathematical operations using different algorithms.
App36	CRUDMVC	Academic-Business	Java	XS	Application to manage product data. Modules and functionalities: register, update, search and delete products.
App37	Libraryjava-master	Academic-Reservation	Java	S	Application to manage library data borrowing books, magazines, etc. Modules and functionalities: student, teacher, bibliographic material, loans
App38	FinalProjectJava	Academic-Business	Java	S	Application to manage roles and users. Modules and functionalities to create, edit, update, delete people data and associate roles and users.
App39	07_SimpleDrawMVC Visitor-master	Academic-Leisure	Java	XS	Drawing lines game
App40	APP-JAVAswing-GYM-master	Academic-Business	Java	XS	Application to manage data from a gym. Modules and functionalities for user registration and monitoring of sports routines.
App41	Race_express	Academic-Tourism	Java	XS	Application to manage travel agency data. Modules and functionalities: Passengers, routes and tickets.
App42	Company	Academic-Tourism	Java	XS	Application to manage travel agency data. Modules and functionalities: Passengers, routes and tickets.
App43	Ideal Weight Calculator	Academic-Utilities	Java	S	Application that calculates the weight in kilograms and pounds of people
App44	Registration	Academic-Business	Java	S	Application to manage student registration for courses. Modules and functionalities: registration of people, courses and subjects.
App45	ProgrammingTest	Academic-Business	Java	S	Application to manage personal data. Modules and functionalities: register, update, search and delete.
App46	FirstTwoMothsPeriod	Academic-Tourism	Java	S	Application to manage travel agency data. Modules and functionalities: Passengers, routes and tickets.
App47	UtplTest	Academic-Business	Java	S	Application to manage library data according to a type (books, magazines). Modules and functionalities: student, teacher, bibliographic material, loans.
App48	TestTwoMonthsPeriod	Academic-Tourism	Java	S	Application to manage travel agency data. Modules and functionalities: Passengers, routes and tickets.
App49	CrudPackages	Academic-Business	Java	XS	Application to manage roles and users. Modules and functionalities to create, edit, update, delete people data and associate roles and users.
App50	LibraryJava	Academic-Business	Java	S	Application to manage library data according to a type (books, magazines). Modules and functionalities: student, teacher, bibliographic material, loans.
App51	Dungeons-master	Academic-Leisure	Java	S	Strategy game
App52	SwingMvc	Academic-Business	Java	XS	Application for user registration and authentication. Module and functionality for user creation.

Fig. 2 (continued)

- *Hardware*: All applications were executed on a laptop with the same hardware configuration: Processor: ADM Ryzen 5 4600 H with Radeon Graphics 3.GHz, RAM: 16.0 GB.
- *Software*: All applications were deployed on the same software stack and configured with the same parameters. The operating system, the Integrated Development Environment (IDE), and the database for executing the Java and C# applications are the following: Operating System: Windows

		SonarQube				Joulemeter (W)					
Application ID	SIZE	Lines of Code (LOC)	Cyclomatic Complexity (CC)	Code Smell (CS)	Duplications Lines (DL)	Total Power	CPU	Monitor	Disk	Base	Application Power (CPU Only)
App1	L	366201	15.04	64	60592	25.054386	0.87894737	9.1	0.05877193	15	0.01666667
App2	XL	527698	15.04	64	60592	24.87688442	0.761809045	9.1	0.008542714	15	0.053030303
App3	L	442863	11.588	45	43768	24.69736842	0.592105263	9.1	0	15	0.04
App4	L	429816	15.222	64	61512	24.93859649	0.823245614	9.1	0.008333333	15	0.043421053
App5	L	36816	15.04	64	60592	24.93776824	0.825321888	9.1	0.006008584	15	0.048275862
App6	L	369751	15.222	64	61512	24.82197802	0.705128205	9.1	0.011721612	15	0.041911765
App7	L	369779	15.222	64	61512	24.6	0.490140845	9.1	0.001408451	15	0.005633803
App8	L	373814	16.533	69	66844	26.10855856	1.087387387	10	0.015765766	15	0.041176471
App9	L	371146	15.222	64	61512	26.09893617	1.044680851	10	0.046808511	15	0.086096257
App10	L	429880	15.222	64	61512	23.95760234	0.741520468	8.2	0.009064327	15	0.037941176
App11	L	368160	15.04	64	60592	24.00767123	0.783835616	8.2	0.016164384	15	0.043406593
App12	L	403301	24.841	77	114289	23.98247126	0.764655172	8.2	0.011781609	15	0.038505747
App13	L	369751	15.222	64	61512	24.07163121	0.856382979	8.2	0.011702128	15	0.040070922
App14	L	369751	15.222	64	61512	24.05522124	0.821415929	8.2	0.028672566	15	0.041207815
App15	S	3953	350	460	669	23.76114943	0.552873563	8.2	0.003908046	15	0.086635945
App16	S	2957	455	369	791	23.64609572	0.441561713	8.2	0.002015113	15	0.052392947
App17	S	2842	568	202	86	24.00542636	0.803100775	8.2	0.000775194	15	0.272868217
App18	S	7834	721	894	4207	24.07664474	0.859539474	8.2	0.007236842	15	0.065676568
App19	S	1751	282	212	202	23.89899666	0.688294314	8.2	0.001672241	15	0.041275168
App20	XS	435	42	31	0	23.7754717	0.546361186	8.2	0.02722372	15	0.093243243
App21	XS	851	127	121	0	23.82836364	0.624727273	8.2	0.000363636	15	0.111272727
App22	XS	652	80	78	0	23.89961538	0.693076923	8.2	0.001923077	15	0.164230769
App23	XS	231	36	11	0	29.01935484	3.969892473	10	0.040860215	15	0.034408602
App24	XS	417	39	40	32	31.45769231	6.455769231	10	0	15	0.082692308
App25	S	3773	376	557	1457	35.92056075	10.91775701	10	0.000934579	15	0.801869159
App26	S	9114	1590	733	1273	23.70888889	0.505079365	8.2	0	15	0.07047619
App27	XS	231	29	13	0	23.7984456	0.573056995	8.2	0.026943005	15	0.120725389
App28	XS	407	93	18	0	23.65466102	0.453813559	8.2	0	15	0.043829787
App29	S	5475	517	121	424	23.66751412	0.462711864	8.2	0.001977401	15	0.041076487
App30	XS	646	71	123	26	23.76529851	0.557089552	8.2	0.00261194	15	0.118656716
App31	XS	378	59	46	0	23.71107143	0.507857143	8.2	0.000357143	15	0.090357143
App32	XS	694	58	79	0	23.76542056	0.514953271	8.2	0.046105919	15	0.053894081
App33	XS	331	14	50	0	23.72011331	0.5101983	8.2	0.007648725	15	0.079603399
App34	XS	1306	115	101	0	23.65420561	0.447975078	8.2	0.004049844	15	0.066978193
App35	XS	311	21	18	0	23.88700565	0.666101695	8.2	0.01920904	15	0.162429379
App36	XS	384	23	32	0	23.79256506	0.574349442	8.2	0.011524164	15	0.076951673
App37	S	3452	319	726	788	23.70216718	0.483591331	8.2	0.011764706	15	0.082352941
App38	S	4852	577	376	1803	23.9515625	0.723125	8.2	0.0240625	15	0.1521875
App39	XS	808	100	36	0	23.75935252	0.552517986	8.2	0.002877698	15	0.087364621
App40	XS	3203	335	367	0	23.90581818	0.701454545	8.2	0	15	0.1
App41	XS	3528	414	556	317	24.03626374	0.831868132	8.2	0	15	0.158088235
App42	XS	4891	415	787	1156	23.69121622	0.477702703	8.2	0.0125	15	0.058644068
App43	S	1448	62	198	30	28.94210526	3.934210526	10	0.002631579	15	0.048684211
App44	S	3083	306	285	451	30.025	8.023913043	7	0.001086957	15	0.05326087
App45	S	3083	333	529	2292	29.83188406	7.828985507	7	0.001449275	15	0.043478261
App46	S	1632	147	305	154	36.47966102	11.4779661	10	0	15	0.113559322
App47	S	1178	122	270	23	29.24363636	9.341818182	4.9	0	15	0.123636364
App48	S	5993	719	885	4087	32.05405405	7.040540541	10	0.010810811	15	0.044594595
App49	XS	574	55	87	0	31.83606557	6.832786885	10	0	15	0.057377049
App50	S	3848	367	596	438	38.11014493	13.10434783	10	0.002898551	15	1.314492754
App51	S	4414	420	698	1514	39.07777778	14.07	10	0	15	0.074074074
App52	XS	435	42	31	0	23.65730028	0.449586777	8.2	0.004132231	15	0.062983425

Fig. 3 Training corpus: measurements

11 Home 64 bits. IDE: NetBeans 8.6 (Java) and Microsoft Visual Studio Enterprise 2019 (C#), DBMS: MySQL 5.6 (Java) and SQL Server (C#).

- *Collection procedure*: This is composed of several steps, which are detailed as follows:

1. *Calibrating*: Microsoft Joulemeter was calibrated running on batteries with the defined software stack (power chord disconnected). Subsequently, it was not possible to update the hardware and software established in the context of execution until the collection procedure was finished.
2. *Determining the measuring time of applications*: During the execution of an application, energy consumption fluctuated, depending on the functionality that was being executed at that moment. Hence, to ensure that the measurement of an application is representative, the execution time must allow the

execution of all its functionalities. To determine this time systematically, we defined the following steps:

(a) The applications categorized with the two highest SIZE of the training corpus are selected in order to determine the number of functionalities provided for execution. In this case, we selected applications that were categorised as XL and L, and we calculated the number of functionalities. For example, if the application provided the create-read-update-delete (CRUD) of a specific element of the software system, it was recorded as having four functionalities.
(b) Applications with a high number of functionalities in the previous step are selected. We selected all applications that had the three highest numbers of functionalities.
(c) The applications selected in the previous step are run by executing all their functionalities and timing the duration of the entire execution. We run the applications and their execution times were recorded.
(d) From the times extracted in the previous step, the highest time is selected and rounded to minutes. In our case, the time was 10 min.

3. *Establishing the execution context*: The unnecessary peripheral devices and background processes were disconnected and stopped, respectively. Only Microsoft Joulemeter and the software required to execute the application were left running. In addition, to begin the measurement at the same starting point in applications that managed databases, all databases were emptied beforehand.
4. *Measuring power consumption*: Each application was measured during the time determined in step 2, which was 10 min. It is recommended to follow the same interaction pattern to avoid neglecting any functionality during execution. Small applications in which the execution of all functionalities was finished before the established time were executed until the time was completed by repeating the interaction pattern as many times as necessary.
5. *Storing power consumption measurements*: After completing step 4 in each application, the obtained measurements were stored.

Steps 4 and 5 were performed iteratively. When they were executed for all applications, the training corpus of the model was obtained, which in this case was composed of 52 MVC applications (see Fig. 3).

3.4 Phase IV: analysis

The Analysis phase consists of two tasks: “Data Analysis” and “Data Normalization” (see Fig. 1). They are described in the following subsections.

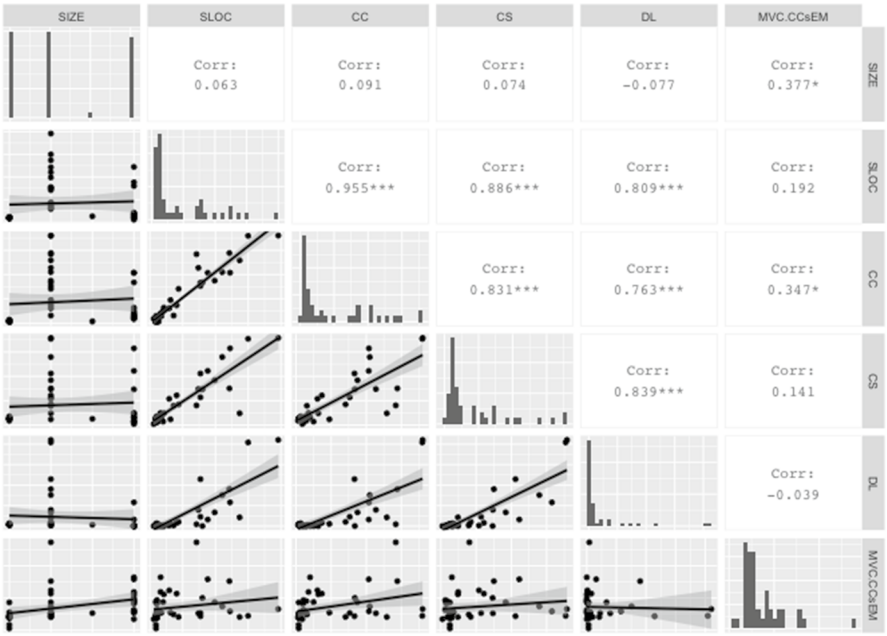

Fig. 4 Histogram, dispersion and correlation matrix of the normalized data extracted from the metrics of the 52 applications

3.4.1 Data analysis

In this task, we prepared and selected the energy consumption values provided by Microsoft Joulemeter and used them to train the regression model (see Fig. 4). Microsoft Joulemeter calculates power consumption using its estimation mathematical model, which is based on CPU, screen, memory and storage of total power [58]. Total power and CPU metrics cannot be used as estimation variables because they include measurements not only from the execution of the application under study but also from other processes that are being executed at the same time. However, in Joulemeter, the power consumption estimation model also calculates the "application power (CPU only)", that is, it subtracts the total CPU and the basal CPU consumption of the computer (see Eq. 4). Hence, the representative metric is "application power (CPU only)" because it stores the energy consumption of the MVC application during its execution-that is, the strict power consumption of the applications under evaluation. In addition, the "data analysis" task (see Fig. 1) requires analysing the relationship between the variables to identify two possible problems: (**P1**) there are independent variables that present non-linear relationships with the dependent variable, and (**P2**) there are collinearity problems between the independent variables.

$$\text{ApplicationPower(CPUonly)} = \text{CPU} - \text{Basal CPU Consumption} \tag{4}$$

To analyse the data distribution, we used the GGally package [59], which is able to plot a data set with multiple variables. Figure 2 shows the GGally results of the data set in histograms and scatter plots. They show a linear relationship among the variables because the values of the scatter plots between two independent variables are close to the line; thus, problem **P1** regarding non-linear relationships was avoided. The data distribution showed that some variables were linearly dependent (see Fig. 2).

In addition, the correlation coefficient of each pair of variables was calculated. The results showed that some variables were highly correlated, such as source lines of code (SLOC) and cyclomatic complexity (CC), with a correlation of 0.955, and SLOC and code smells (CS), with a correlation of 0.886 (see Fig. 4). Therefore, when SLOC increased, CC and CS also increased, and vice versa. In addition, this implied that when SLOC had a high value, CC and CS also had high values. This indicated that collinearity problems may exist and some independent variables may be depreciated in the final model. In addition, as shown in Fig. 4, the results revealed a low positive correlation between the MVC-CCsEM and DL variables, which may have implied a weak relationship between both variables.

3.4.2 Data normalisation

This step is conducted to determine whether the data required normalisation to be processed correctly and to ensure that valid results is obtained. This task is optional, depending on the kind of variables used and the data distribution obtained from the data analysis. If there are variables without a linear-relationship, the normalization is required, whereas the normalization is not performed if the variables have linear relationships. In this case, data normalisation was not required because there were no variables with non-linear relationships (see Fig. 4). On the one hand, the categorical variable SIZE must be transformed into a quantitative variable by assigning continuous values [60]. Hence, the k categories were coded as the following continuous values: 1 (XS), 2(S), 3 (M), 4 (L) and 5(XL).

3.5 Phase V: construction

In the final phase, the model is constructed. The regression model is applied to estimate the value of the dependent variable and to evaluate the influence of the predictors for use as an early estimation model (see Eq. 3). In our case, the MVC-CCsEM model was applied to predict the expected MVC-CCsEM of a new MVC software architecture, which energy consumption is not known.

Through the generation of the MVC-CCsEM regression model, hypothesis H2 was validated by determining whether the coefficients of the independent variables were statistically significant. They are significant, with a p-value of less than 5% (0.05) [61, 62].

To construct the MVC-CCsEM regression model that validated our hypotheses, we used the least squares method [63]. This method checks for the potential problem of multicollinearity (**P2**), and it allowed us to determine that this problem did

Table 3 Coefficients of the initial regression model with the dependent variable MVC-CCsEM

	Coefficients (β)	Std. Error	t value	p-value
Intercept (β_0)	0.03670	0.01209	3.036	0.00426**
SIZE (β_1)	0.009194	0.004502	2.042	0.04794*
SLOC (β_2)	− 0.00003247	0.00001154	− 2.814	0.00764**
CC (β_3)	0.0003934	0.00008613	4.567	0.0000485***
CS (β_4)	0.00004994	0.00005506	0.907	0.37000
DL (β_5)	− 0.00002569	0.00001109	− 2.317	0.02584*
F-statistic: p-value:0.00003114				

not exist in the MVC-CCsEM model. In this method, an R^2 value (0–1) close to 1 indicates multicollinearity. Therefore, we calculated the R^2 coefficient of the MVC-CCsEM model to discard that it was close to 1. To obtain the first version of the regression model and determine its coefficient values, the "lm" function of RStudio was used to fit linear models. The values of the metrics from the 52 applications were included in the R-Studio Tool as the training corpus of the model. In addition, the dependent variable (MVC-CCsEM) and the independent variables (SIZE, SLOC, CC, CS, DL) were specified in the R-Studio Tool.At this point, the "lm" was executed to generate the linear regression model using the least squares method. This execution provided us with the first version of the MVC-CCsEM model, which is presented in Eq. 5, showing the coefficients of the MVC-CCsEM regression model:

$$\begin{aligned} \text{MVC} - \text{CCsEM} = {} & 0.03670 + 0.009194\text{SIZE} - 0.00003247\text{SLOC} \\ & + 0.0003934\text{CC} + 0.00004994\text{CS} - 0.00002569\text{DL} + e \end{aligned} \tag{5}$$

This calculation obtained the values of R-square (0.5043) and adjusted R-square (0.4408). Since their values were not close to 1, they indicated that a multicollinearity problem was not present and that the data fit of the model was representative of energy estimation [64]. As shown in Table 3, the coefficients β_{SIZE}, β_{SLOC}, β_{CC} and β_{DL} were significant because their p-values were less than 0.05. Those with the highest number of asterisks were the most representative. β_{SIZE} represents the relationships between MVC-CCsEM and SIZE with a p-value= 0.04794; β_{SLOC} is the relationship between MVC-CCsEM and SLOC with a p-value=0.00764; and β_{DL} represents the relationships between MVC-CCsEM and DL with a p-value= $0.02584 < 0.05$. β_{CC} was the most significant coefficient with the lowest p-value of 0.0000485, which confirmed the high correlation between cyclomatic complexity and energy consumption. This indicated that the coefficients β_{SIZE}, β_{SLOC}, β_{CC} and β_{DL} were significant because their p-values were less than 0.05, being the most representative for the MVC-CCsEM model.

Moreover, the statistic F model had an acceptable p-value of 0.00003114. However, it was possible to conclude that with a p-value of 0.37000, the coefficient β_{CS} was not significant, and it could be removed from the model. However, insignificant variables in the initial model could be significant in other models in which one of the

Table 4 Coefficients of the final regression model with the dependent variable MVC-CCsEM

	Coefficients (β)	Std. error	t value	p-value
Intercept (β_0)	0.03730	0.01204	3.096	0.00357**
SIZE (β_1)	0.009875	0.004429	2.230	0.03145*
SLOC (β_2)	− 0.00002800	0.00001041	− 0.00002689	0.01039*
CC (β_3)	0.0003853	0.00008548	4.508	0.000056***
DL (β_5)	− 0.00002096	0.000009762	− 2.147	0.03790*
F-statistic: p-value:0.00001324				

variables was removed. Hence, before definitively depreciating β_{CS}, we applied the Stepwise strategy to the model, following Akaike's criterion (AIC) [65] of selecting only significant variables for its construction. This strategy consists of applying the square method to the different combinations of our model by removing one or multiple variables from the initial model and determining the best result. By applying this strategy, we determined that the significant variables were SIZE, SLOC, CC and DL (see Table 4). They conformed to the MVC-CCsEM model, obtaining the values of R-square (0.4939) and adjusted R-square (0.4432), which determined that the fit of the data to the model was representative of energy estimation [64]. In addition, this model highly improved the initial model by increasing the significance of the statistic F model with a p-value = 0.00001324 and determining the following p-values of the coefficients: β_{SIZE}=0.03145, β_{SLOC}=0.01039, β_{CC}=0.000056 and β_{CS}=0.03790. These values corresponded to the best model resulting from the AIC-based variable selection process. Based on this statistical evidence, hypothesis *H2* was validated. In addition, it was demonstrated that size (SIZE, SLOC), complexity (CC) and maintainability (DL) are significant predictors of estimating energy and they cannot be depreciated. Based on these results, the final model is presented in Eq. 6.

$$\begin{aligned} \text{MVC} - \text{CCsEM} = {} & 0.03730 + 0.009875\text{SIZE} - 0.00002800\text{SLOC} \\ & + 0.0003853\text{CC} - 0.00002096\text{DL} + e \end{aligned} \tag{6}$$

4 Estimating the energy consumption of MVC applications using the MVC-CCsEM

Once the MVC-CCsEM model was constructed and *H2* was validated (see Phase I), it was necessary to validate *H1* by evaluating the model's estimation capabilities and accuracy. To that end, an experiment was conducted in which the energy consumed by twelve MVC applications was estimated using the model MVC-CCsEM, and it was compared with the power consumption value measured by Joulemeter during the execution of the MVC application (i.e. application power (CPU only)). The characteristics of these twelve applications are described in detail in Table 5. To conduct this experiment, the Profiling phase was performed for all twelve applications according to the construction process of the model. The metrics SIZE, SLOC,

Table 5 Applications Used to Validate the MVC-CCsEM Model

Application ID	Size	Programming language	Domain	Joulemeter energy consumption measurement (Joules)	MVC-CCsEM (Joules)	‖error‖
App1	L	C#	Tourism	0.05545455	0.04338008	0.0121
App2	L	C#	Tourism	0.04553073	0.04140903	0.0041
App3	XS	Java	Entertainment	0.04262735	0.10360179	0.0610
App4	S	Java	Reservation	0.05463576	0.1027345	0.0481
App5	M	Java	Business	0.0748062	0.15277697	0.0780
App6	XS	Java	Entertainment	0.0447458	0.08117975	0.0364
App7	S	Java	Business	0.062983425	0.0818026	0.0188
App8	XS	Java	Business	0.035526316	0.06674008	0.0566
App9	S	Java	Entertainment	0.034693878	0.0243309	0.0467
App10	XS	Java	Entertainment	0.04375	0.0600897	0.0361
App11	XS	Java	Entertainment	0.025423729	0.0601804	0.0545
App12	XS	Java	Entertainment	0.040804598	0.0601244	0.0391

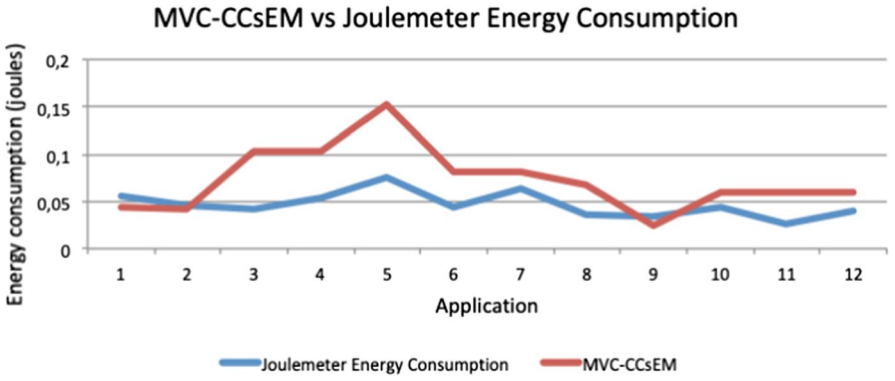

Fig. 5 Comparison of the estimation of the MVC-CCsEM and real consumption (joules)

CC and DL were collected by the SonarQube tool to determine the values of the independent variables of the MVC-CCsME model. In addition, to collect data on the real consumption of the MVC applications, they were executed in the same context and pattern to extract the application power (CPU only) using Microsoft Joulemeter. The results are presented in Table 5.

Figure 5 and Table 5 show the comparison in joules between real energy consumption and the estimation of the MVC-CCsEM model. To measure the model's performance and determine the precision with which the model predicted the dependent variable, the root mean squared error (RMSE) was used [66], which was calculated using R-Studio. In this case, the prediction of these twelve applications was RMSE = 0.2861, which was an acceptable value for the estimation [66–68]. The estimation of Application 2 was the most accurate, with a difference of $e = 0.0041$, whereas Application

5 had the highest error of estimation, $e = 0.0780$ (see Table 5). Based on the results of this analysis, we concluded that the estimation of the model was sufficiently accurate to be a valuable tool for software engineers. The joule estimation was very close to the reality in most of the applications. Thus, the need to execute the application to determine its power consumption and generate extra consumption was avoided. Based on these results, the MVC-CCsEM model could effectively support software engineers during software construction and maintenance by informing them about variations in power consumption due to the code changes. Software engineers can therefore avoid performing explicit measurements, such as new tasks in the software life cycle, executing an application to obtain information and therefore generating power consumption rebound effects.

5 Threats to validity

To improve the internal validity of our results, we selected applications based on evidence described in the code or documentation of GitHub or academic repositories. More than one author checked the architecture to select applications that implemented the MVC pattern. Furthermore, to collect the metrics, we configured a local computer dedicated to collecting and extracting the metrics through the tool. Each application was downloaded from the repository, built and executed under the same context, even the database, which was empty in all cases. As a result, personal bias was avoided in automatic information management.

Construct validity was addressed following Lago et al. [33] who proposed a set of metrics that can be used to estimate energy consumption. We also followed the guidelines proposed by the PowerAPI Reference Architecture [34] to collect, measure, extract and export metrics. The model was developed by considering the variables identified in the literature as essential in software power consumption estimation (see Table 1), since our aim was to provide a simple model that could be quickly adopted by software engineers. Regarding the regression model, we applied statistical methods to avoid bias and followed Kiers et al. [69] to evaluate correlations between different variables by assigning continuous values to translate qualitative variables into quantitative variables.

Finally, to ensure external validity, the model was constructed using 52 MVC applications programmed in C# and Java with different sizes and quality features (see Table 2 and Fig. 3). Hence, this model is useful for Java and C# MVC applications that use MySQL and SQL Server databases. So, to generalise the applicability of the MVC-CCsEM model, it will be necessary to increase the training corpus of applications, including applications in other languages, database management systems and non-SQL databases.

6 Conclusion

In this work, we developed a green estimation model, MVC-CCsEM, which provides an integrated solution for the identified needs of software power consumption estimation throughout the software lifecycle. MVC-CCsEM estimates the energy consumption of MVC applications in terms of Size, Source Lines Of Code, Cyclomatic Complexity and Duplicated Lines without the need for execution and the resulting generation of power consumption rebound effects. The adoption of this model is feasible during the construction and maintenance decision-making of C# and Java applications of any size in which the architecture implements an MVC pattern that uses MySQL and SQL server databases. The model was constructed by analysing 52 applications and validated by estimating twelve applications. The results showed that this model is an advancement in the field. The results for the twelve applications showed an RMSE = 0.2861, indicating that the joule estimation was very close to reality in avoiding the extra energy consumed by application execution. Therefore, MVC-CCsEM can assist software engineers saving their time during their development tasks and the power consumption of their applications.

In addition, this work formalises in SPEM the process that was defined and followed to construct the MVC-CCsEM model, which is a reusable asset for the research community, by emphasising optional and mandatory tasks as well as defining how to perform them to avoid bias. In future work, we will extend and improve the MVC-CCsEM model by extending the training corpus with a greater number of applications, examining the ways in which a population and the use of MVC scaffolding mechanisms may influence power consumption, and measuring each application several times to increase the accuracy of the measurements. In addition, we plan to automatise the calculation using Sonarqube to construct a tool. We will examine other variables that can be extracted from a static analysis to determine whether they influence the energy consumption of software applications. In addition, we will determine how new languages and databases influence the model to determine whether it is necessary to specialise in programming languages or data management systems. Finally, we will determine the success of adopting the model during the software life cycle for the benefit software engineers.

Acknowledgements This work is partially supported by Universidad Técnica Particular de Loja Computer Science Department) and the Spanish Ministry of Economy and Competitiveness (MINECO) through the project SIoTCom: Sustainability-Aware IoT Systems Driven by Social Communities (PID2020-118969RB-I00). We would like to specially thank to Andres Alejandro Jimenez and Joselito Miguel Ordoñez for supporting the collection of metrics.

Author contributions DG and JP involved in conceptualization; took part in methodology and involved in collected the data. DG, JP and PV performed the analysis and took part in the writing.

Funding Open Access funding provided thanks to the CRUE-CSIC agreement with Springer Nature. The Spanish Ministry of Economy and Competitiveness (MINECO) through the project SIoTCom: Sustainability-Aware IoT Systems Driven by Social Communities (PID2020-118969RB-I00).

Data availability The information about the data set is provided in the article.

Declarations

Conflict of interest The authors declare no conflict of interest.

Ethical approval Not applicable.

References

1. Group TC (2008) Smart 2020: enabling the low carbon economy in the information age. GeSi, The Climate Group
2. The International Telecommunication Union (ITU), ICTs and Energy Efficiency ((Last acceded: march 2023)). https://www.itu.int/en/action/environment-and-climate-change/Pages/energy_efficiency-BAK.aspx
3. The International Telecommunication Union (ITU), Global Enabling Sustainability Initiative (GeSI), Global System for Mobile Communications Association (GSMA), and Guidance for ICT Companies Setting Science Based Targets Mobile Networks Operators, Fixed Networks Operators and Data Centres Operators, Report (2022, Last acceded: march 2023). https://gesi.org/research/guidance-on-setting-sbts-for-ict-companies
4. Intergovernmental Panel Climate Change IPCC. An IPCC Special Report on the impacts of global warming of 1.5°C above pre-industrial levels and related global greenhouse gas emission pathways, in the context of strengthening the global response to the threat of climate change, sustainable development, and efforts to eradicate poverty. Cambridge University Press, Cambridge, UK and New York, NY, USA (2018). https://doi.org/10.1017/9781009157940
5. Hilty LM (2008) Information technology and sustainability: essays on the relationship between ict and sustainable development, pp. 1–160. Books On Demand, Herstellung und Verlag
6. Global System for Mobile Communications Association (GSMA), Mobile Industry Impact Report: Sustainable Development Goals. GSMA (2022, Last acceded: march 2023). https://www.gsma.com/betterfuture/wp-content/uploads/2022/11/2022-SDG-Impact-Report.pdf
7. Digitalization and energy consumption (2020) Does ict reduce energy demand? Ecol Econ. 176:106760. https://doi.org/10.1016/j.ecolecon.2020.106760
8. Hilty LM, Aebischer B (2015) Ict for sustainability: an emerging research field. In: Hilty LM, Aebischer B (eds) ICT innovations for sustainability. Springer, Cham, pp 3–36
9. Madhura S (2022) A review on low power vlsi design models in various circuits. J Electron Inform 4(2):74–81. https://doi.org/10.36548/jei.2022.2.002
10. Kiran WS (2022) Performance analysis of multi-layered clustering routing protocol for wireless sensor networks. IRO J Sustain Wireless Syst 4(1):11–22. https://doi.org/10.36548/jsws.2022.1.002
11. Capra E, Formenti G, Francalanci C, Gallazzi S (2010) The impact of mis software on it energy consumption. In: European Conference on Information Systems
12. Becker C, Chitchyan R, Duboc L, Easterbrook S, Penzenstadler B, Seyff N, Venters CC (2015) Sustainability design and software: The karlskrona manifesto. In: 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering. 2:467–476
13. Amsel N, Ibrahim Z, Malik A, Tomlinson B (2011) Toward sustainable software engineering: Nier track. In: 2011 33rd International Conference on Software Engineering (ICSE), pp. 976–979

14. Calero C, Piattini M (2017) Puzzling out software sustainability. Sustain Comput Inform Syst 16:117–124
15. Procaccianti G (2015) Energy-efficient software. PhD thesis, VU University Amsterdam
16. Noureddine A, Rouvoy R, Seinturier L (2013) A review of energy measurement approaches. SIGOPS Oper Syst Rev 47(3):42–49
17. Chatzigeorgiou A, Stephanides G (2002) Energy metric for software systems. Software Qual J 10(4):355–371
18. Hilty LM, Arnfalk P, Erdmann L, Goodman J, Lehmann M, Wäger PA (2006) The relevance of information and communication technologies for environmental sustainability: a prospective simulation study. Environ Model Softw 21(11):1618–1629. https://doi.org/10.1016/j.envsoft.2006.05.007. (**Environmental Informatics**)
19. Beck K (2001) e.a Manifesto for agile software development. Acceded March 2023. http://www.agilemanifesto.org/
20. Seo C, Edwards G, Popescu D, Malek S, Medvidovic N (2009) A framework for estimating the energy consumption induced by a distributed system's architectural style. In: Proceedings of the 8th International Workshop on Specification and Verification of Component-Based Systems. SAVCBS '09, pp. 27–34. Association for Computing Machinery, New York, NY, USA
21. Burbeck S (1992) Applications programming in smalltalk-80: how to use model-view-controller (mvc)
22. Fowler M (2006) GUI architectures: model view controller. https://martinfowler.com/eaaDev/uiArchs.html#ModelViewController
23. Taylor RN, Medvidovic N, Dashofy EM (2009) Software architecture: foundations, theory, and practice. Wiley Publishing, United States
24. Ebert C, Cain J, Antoniol G, Counsell S, Laplante P (2016) Cyclomatic complexity. IEEE Softw 33(6):27–29
25. Fowler M (1999) Refactoring: improving the design of existing code. Addison-Wesley, Boston
26. Guldner A, Garling M, Morgen M, Naumann S, Kern E, Hilty L (2018) Energy consumption and hardware utilization of standard software: methods and measurements for software sustainability, pp. 251–261
27. Michanan J, Dewri R, Rutherford MJ (2017) Greenc5: an adaptive, energy-aware collection for green software development. Sustain Comput Inform Syst 13:42–60
28. David H, Gorbatov E, Hanebutte UR, Khanna R, Le C (2010) Rapl: memory power estimation and capping. In: Proceedings of the 16th ACM/IEEE International Symposium on Low Power Electronics and Design. ISLPED '10, pp. 189–194. Association for Computing Machinery, New York, NY, USA
29. Microsoft: Microsoft Joulemeter (2010). https://www.microsoft.com/en-us/research/project/joulemeter-computational-energy-measurement-and-optimization/
30. jRAPL: jRAPL: A framework for profiling energy consumption of Java programsr (2022). https://www.microsoft.com/en-us/research/project/joulemeter-computational-energy-measurement-and-optimization/
31. Bourdon A, Noureddine A, Rouvoy R, Seinturier L (2013) Powerapi: a software library to monitor the energy consumed at the process-level. ERCIM News 2013(92)
32. Noureddine A, Rouvoy R, Seinturier L (2014) Unit testing of energy consumption of software libraries. In: Proceedings of the ACM Symposium on Applied Computing 1200–1205
33. Sehgal R, Mehrotra D, Nagpal R, Sharma R (2020) Green software: refactoring approach. J King Saud Univ Comput Inf Sci. https://doi.org/10.1016/j.jksuci.2020.10.022
34. Bozzelli P, Gu Q, Lago P (2013) A systematic literature review on green software metrics. VU University, Amsterdam
35. Guégain E, Quinton C, Rouvoy R (2021) On reducing the energy consumption of software product lines. In: Proceedings of the 25th ACM International Systems and Software Product Line Conference—Volume A. SPLC '21, pp. 89–99. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3461001.3471142
36. Anjum Mohd Aslam MK (2018) A review on energy efficient techniques in green cloud: open research challenges and issues. Int J Sci Res Comput Sci Eng 6:44–50. https://doi.org/10.26438/ijcse/v6i3.4450
37. Mojarad M, Hosseini LT, NS, (2021) Optimal task assignment to heterogeneous cores in cloud computing using particle swarm optimization. Int J Sci Res Comput Sci Eng 9(3):1

38. Chingrace Guite KKM (2018) A study on energy efficient vm allocation in green cloud computing. Int J Sci Res Comput Sci Eng 6:37–40. https://doi.org/10.26438/ijcse/v6i4.3740
39. Cairo A, Carneiro G, Monteiro M (2018) The impact of code smells on software bugs: a systematic literature review. Information 9:273
40. Dufour B, Driesen K, Hendren L, Verbrugge C (2003) Dynamic metrics for java. In: Proceedings of the 18th Annual ACM SIGPLAN Conference on Object-Oriented Programing, Systems, Languages, and Applications. OOPSLA '03, pp. 149–168. Association for Computing Machinery, New York, NY, USA
41. Rocheteau J (2015) Energy wasting rate as a metrics for green computing and static analysis, p. 10
42. Fu C, Qian D, Luan Z (2018) Estimating software energy consumption with machine learning approach by software performance feature. In: 2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), pp. 490–496
43. Feitosa D, Alders R, Ampatzoglou A, Avgeriou P, Nakagawa E (2017) Investigating the effect of design patterns on energy consumption. J Softw Evol Process 29:e1851
44. Ardito L, Coppola R, Morisio M, Torchiano M, Risi M (2019) Methodological guidelines for measuring energy consumption of software applications. Sci Progr. https://doi.org/10.1155/2019/5284645
45. Georgiou S, Rizou S, Spinellis D (2019) Software development lifecycle for energy efficiency: techniques and tools. ACM Comput Surv. https://doi.org/10.1145/3337773
46. Ournani Z, Rouvoy R, Rust P, Penhoat J (2020) On reducing the energy consumption of software: From hurdles to requirements. In: Proceedings of the 14th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM). ESEM '20. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3382494.3410678
47. Object Management Group (2008) SPEM: software & systems process engineering meta-model specification
48. Mancebo J, García F, Calero C (2021) A process for analysing the energy efficiency of software. Inf Softw Technol 134:106560
49. Guamán D, Pérez J, Valdiviezo-Diaz P, Canas N (2022) Estimating the energy consumption of software components from size, complexity and code smells metrics. In: Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing. SAC '22, pp. 1456–1459. Association for Computing Machinery, New York, NY, USA
50. Perry D, Wolf A (2000) Foundations for the study of software architecture. ACM SIGSOFT Softw Eng Notes 17:40–52
51. Shaw M (1996) Procedure calls are the assembly language of software interconnection: connectors deserve first-class status. In: Lamb DA (ed) Studies of software design. Springer, Berlin, Heidelberg, pp 17–32
52. Mell P, Grance T, et al (2011) The NIST definition of cloud computing
53. Soldani J, Tamburri DA, Van Den Heuvel W-J (2018) The pains and gains of microservices: a systematic grey literature review. J Syst Softw 146:215–232
54. El Malki A, Zdun U (2019) Guiding architectural decision making on service mesh based microservice architectures. In: Bures T, Duchien L, Inverardi P (eds) Software architecture. Springer, Cham, pp 3–19
55. Guamán D, Pérez J, Garbajosa J, Rodríguez G (2020) A systematic-oriented process for tool selection: the case of green and technical debt tools in architecture reconstruction. In: Morisio M, Torchiano M, Jedlitschka A (eds) Product-focused software process improvement. Springer, Cham, pp 237–253
56. SonarQube: Metrics Definitions (2008-2022). https://docs.sonarqube.org/latest/user-guide/metric-definitions/
57. SonarQube: SonarQube (2008-2022). https://www.sonarqube.org/
58. Kansal A, Zhao F, Liu J, Kothari N, Bhattacharya AA (2010) Virtual machine power metering and provisioning. In: Proceedings of the 1st ACM Symposium on Cloud Computing. SoCC '10, pp. 39–50. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/1807128.1807136
59. Larmarange J (2016) Package GGally: an extension to ggplot2. R package version, 1(0)
60. Revelle W, Revelle MW (2015) Package 'psych'. The comprehensive R archive network 337:338
61. Benoit K (2011) Linear regression models with logarithmic transformations. London School Econ 22(1):23–36

62. Andrade C (2019) The p value and statistical significance: misunderstandings, explanations, challenges, and alternatives. Indian J Psychol Med 41(3):210–215
63. Miller SJ (2006) The method of least squares
64. Capra E, Francalanci C, Slaughter SA (2012) Is software "green''? Application development environments and energy efficiency in open source applications. Inf Softw Technol 54(1):60–71
65. Thompson ML (1978) Selection of variables in multiple regression: part i. a review and evaluation. Int Stat Rev 46(1):1–19
66. Silhavy R, Silhavy P, Prokopova Z (2017) Analysis and selection of a regression model for the use case points method using a stepwise approach. J Syst Softw 125:1–14
67. Touzani S, Granderson J, Fernandes S (2017) Gradient boosting machine for modeling the energy consumption of commercial buildings. Energy Build 158:1533
68. Vu N, Lahmer T, Zhuang X, Rabczuk T (2016) A software framework for probabilistic sensitivity analysis for computationally expensive models. Adv Eng Softw 100:19–31
69. Rasson J-P, Kiers H, Groenen P, Schader M (2000) (eds.): Proceedings: Data Analysis, Classification and Related Methods. Springer Verlag. Springer, Germany