Expert Systems With Applications 299 (2026) 130046

Contents lists available at ScienceDirect

Expert Systems With Applications

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

Energy-aware software design: A hardware and behavioural consumption scoring and labelling algorithm

Jorge Andrés Larracoechea [a,b,*], Philippe Roose [b], Sergio Ilarri [a]

[a] Department of Computer Science and Systems Engineering, I3A, Universidad de Zaragoza, C. María de Luna 3, Zaragoza, 50018, Aragón, Spain
[b] LIUPPA, E2S, Université de Pau et des Pays de l'Adour, 2 Allée du Parc Montaury, Anglet, 64600, Nouvelle-Aquitaine, France

ARTICLE INFO

Keywords:
Software behaviour
Green IT
Energy consumption labelling
Green software
Energy-aware software design
Sustainable software development

ABSTRACT

Energy labelling of electric appliances, among other goods and services, matters to both consumers and producers. It aids the consumers in gathering an expectation of how much electricity the good they are acquiring will consume, and it establishes guidelines that producers can follow to create more efficient products and services. Energy labelling is, nevertheless, nearly absent for software due to the heterogeneous hardware execution environments and unequal circumstances. Throughout this article, we provide an equal ground for software energy labelling by providing an intelligent prospective behavioural, hardware and energy-consumption patterns rating algorithm meant for the design phase of the Software-Development Life Cycle, as opposed to the existing approaches whose labelling procedure applies to the implementation phase onward. We expect our categorization method to guide software developers, helping them to decrease their energy consumption by raising their awareness of the energy footprint that their design choices produce; culminating in aware consumers who know what energy usage to expect from the software they consume.

1. Introduction

Economic growth has become an unreliable metric to assess societies' progress (Comission, 2021b), as its reliance on limited natural resources to create economic value through services and products directly influences our environment. To reduce the impact of our consumption, the global consensus is to dial down its rate and amount. Contrary to the previous goal statement, electrical devices are manufactured and introduced into our global network each year.

To address the electrical energy consumption of such devices, several design, manufacturing, and testing guidelines have been garnered within the European Union to provide an equal ground for manufacturers. One example of these guidelines is the MEErP (Methodology for Ecodesign of Energy-related Products), issued by the European Commission. The MEErP's objectives are twofold: (1) creating a common ground for the ecodesign of energy-related products (any good that has an impact on energy consumption during use) to harmonize national laws of the member states of the European Union (2) and contribute to sustainable development (Comission, 2017). The content of the MEErP establishes a framework for the ecodesign of energy-related products to ensure their free movement within the EU's internal market, and a set of requirements that such products must fulfill to be put on the market or into service. The initial version of the MEErP, issued in 2011, was last revised in 2013 to include in it new variables such as material efficiency.

In addition to the design and manufacturing guidelines, labels that categorize the electrical consumption of energy-related products have emerged to aid consumers in selecting a more frugal device. The most prominent example of an energy consumption indicator is the European energy label. The EU energy label was first introduced in 1994 and expanded in 2004 (Comission, 2024a), with a significant re-scale in 2021 to push forward energy savings and update the standards of the label (Comission, 2021c). It consists of a visual categorization of the electrical consumption of a product in a color scale and letters, from A in green to indicate the best efficiency to G in red to indicate the least efficiency.

Furthermore, the label includes the annual energy consumption of the device and iconography that corresponds to key traits of the product that should matter to the consumer (e.g., the noise level under operation). Its simplicity allows consumers to quickly identify the most frugal device in the category of their interest, saving money for the consumer with the most frugal device that satisfies their needs, and achieving the entrance of the least wasteful product

* Corresponding author.
E-mail addresses: 833814@unizar.es, jorge-andres.larracoechea@etud.univ-pau.fr (J. Andrés Larracoechea), Philippe.Roose@iutbayonne.univ-pau.fr (P. Roose), silarri@unizar.es (S. Ilarri).

https://doi.org/10.1016/j.eswa.2025.130046
Received 6 August 2024; Received in revised form 3 October 2025; Accepted 14 October 2025
Available online 26 October 2025

to households. The EU energy label is a proven success, as the special barometer 492 discovered that nearly 8 in 10 of the people surveyed said they have been influenced by the EU energy label when choosing a product (Kantar, 2019).

In the EU, computers (desktops, workstations, notebooks, and tablets among others) are also considered by the ecodesign directive and requirements, as defined in the commission regulations (EU) 2021/341 and 2013/617 (Comission, 2013, 2021a) but they are currently absent of a label. The public consultation phase for labelling computers, according to the initiative "Energy labelling requirements for computers and computers servers" (Ares 2018 770774) has recently closed, and its EU commission adoption is planned for the fourth quarter of 2025 (Comission, 2024c). The feedback that the initiative has received so far has heavily criticized the fact that a comprehensive duty cycle and power study is necessary to define a significant usage, emphasizing that proposing an energy label based on an active power metric is premature (Comission, 2024b). Similar initiatives are in development in other parts of the world, for example, the Energy Star program in the EE. UU. Star (2024), which is included in the feedback of the EU's initiative as a possible starting point.

As software rules the consumption of hardware by providing it instructions to operate on, creating a common ground for a measurement that relies heavily on the purpose of the software consumed is after all a complicated task. However, to our knowledge, no ecodesign directives on software in the EU, as the definition of an energy-related product does not cover the intangibility of software. This lack of ecodesign requirements acts as a detriment to the consumers of software, as no widespread energy label similar to EU's is applied to the consumption of software to guide them in the same way they are guided when acquiring home appliances. As observed by Guamán et al. (2023) "it is necessary to construct energy estimation models that do not require the execution of applications.".

Throughout this article, we will present the previous and current efforts at categorizing the energy consumption of software and software designs, and present our categorization algorithm aimed at the design phase of the Software Development Life-Cycle (SDLC). In particular, the contributions of the proposed work can be summarized as follows:

- A series of qualitative properties and values that describe the behaviour of software at an architectural and behavioural level.
- A series of quantitative properties that describe the magnitude of hardware consumption inflicted on a hypothetical hardware platform using a software design.
- An expert algorithm to estimate and categorize the hypothetical hardware resource consumption and behaviour of software, based on the aforementioned properties.
- A version of the expert algorithm to estimate and categorize the hypothetical hardware resource consumption of the parallel execution of hypothetical software operations.
- A series of weights that prioritize certain properties and their combinations over others, as well as the impact they represent when obtaining a categorization label for the total consumption obtained from the software unit's design.
- Thorough experimentation and comparative analysis between software-mandated consumption of hardware resources and how the differences in hardware capacities affect the results.
- Guaranteed transparency and replicability of our experiments with publicly available information and data on the setup, execution, results, and project files including a public beta of our custom tool to run them in 5 simple steps.

In Section 2, we delve into the existing approaches and propositions for categorizing or labelling software's hardware consumption. In Section 3 we explain the previous research artifacts that support the expert algorithm proposed in Section 4. In Section 5, we present the experiments that test our algorithm proposal under diverse circumstances, explain the methods employed for their execution, and their results. The closure of this paper is performed in Section 6 discussing the results in the previous section, and closing our article with our future work.

2. Related work

The idea of categorizing software with labels that reflect its consumption of energy or hardware resources is not new. In the work of Deneckère and Rubio (2020), software criteria are classified according to the three phases of the SDLC, which the Greensoft model also follows: development, usage, and end-of-life (Naumann et al., 2011). In contrast to Greensoft, they concentrate on two software aspects: sustainability by software and sustainability in the software (Calero & Piattini, 2015). Even though the criteria they proposed provide a good starting point for understanding the elements to consider in labelling a software product, they did not propose a concrete form of categorization. Moreover, to our knowledge, the authors did not pursue any pragmatic implementation of their proposal. By contrast, the GreenSoft Model Methodology (GSMM) has been further developed into a practical testing framework that is now applied in much of the green coding community (Naumann et al., 2021). The GSMM defines measurement points for energy and resource consumption across the software life cycle, with a strong focus on runtime efficiency during the usage phase. In practice, this involves standardized benchmark environments, monitoring of CPU, memory, storage, and network demand, as well as the evaluation of scaling behaviour under different workloads (Guldner et al., 2024). These methodological foundations directly inform ecolabel schemes such as the Blue Angel, which adopts comparable measurement proposals for mobile, desktop, and server platforms.

The "Blue Angel for Software Development and Components" (Angel, 2023), commissioned by the German Federal Environment Agency, is the German equivalent of the EU's energy label. Blue Angels have existed for more than 45 years without any mandatory aspect, as they are completely voluntary and independent of business interests. There are currently more than 12,000 products and services that bear the Blue Angel, representing the best in their respective product group. As explained by Naumann et al. (2021), up to 2024 the scope of the award criteria was limited to software that has a user interface and can be run on a benchmark desktop system. In the same year, the organization updated its standard to also include mobile applications, desktop software, and server–client platforms. In the revised Version 4 of the criteria (June 2024), the Blue Angel additionally introduced concrete measurement proposals that are applicable across these device classes, thereby extending the methodology beyond its earlier desktop-only focus. The main components of the award criteria are grouped as follows:

- **Resource and energy efficiency:** definition of minimum system requirements, transparency of energy and resource demand, support for operating system energy-saving modes, avoidance of unnecessary background processes, and the use of efficient algorithms and data structures. The software must not place disproportionate demands on CPU, RAM, storage, or network resources relative to its functionality.
- **Potential hardware lifetime:** compatibility with older hardware and operating systems, adaptability to upgraded systems, and avoidance of artificial obsolescence. The software must maintain efficiency even as functionality expands, and vendors must commit to update policies that extend rather than curtail the useful life of hardware.
- **Autonomy of use:** provision of standardised, interoperable, and openly documented data formats, guaranteed uninstallability without residue, ability to run without permanent internet connectivity, and absence of advertising, forced cloud connections, or hidden costs. The criteria also emphasize user control over the configuration, update installation, and data portability.
- **Transparency and user rights:** clear and publicly available documentation of functionalities, system requirements, data collection,

and update cycles. Vendors must disclose whether telemetry or analytics are collected, avoid vendor lock-in strategies, and respect user privacy by default.
- **Quality assurance and robustness:** the criteria require that software undergo systematic quality testing, maintain backward compatibility where possible, and demonstrate resilience against faulty input or unexpected system conditions. These aspects ensure that efficiency and usability are not only theoretical but also practically verifiable.

Concerning green-oriented tools, whose specific mission statement is to aid in the creation of green software, several have their own method of labelling the hardware consumption, energy consumption, or greenhouse emissions. For instance, Wilke et al. (2012) intended to establish a framework that allows profiling the power consumption of mobile applications so that a long-run approximation of their energy consumption can be generated. In addition, they considered that usage profiles should be used together with the profiled power consumption. The authors proceeded to describe a case study of two different email clients for Android and discussed the results and the variables that impacted their study. In the end, no actual energy label was proposed. Even though the authors remain active in the topic of sustainable software, further research built on this proposal appears to be halted, and the specific algorithm employed to generate a label was not disclosed.

Carat (Oliner et al., 2013) constructs energy models of Android applications using a crowd-sourced approach, and later assigns a J-score to the results of the learned energy consumption patterns of the device running it. The J-score is a percentile, meaning that a score of 75 would mean that the device has a better battery life than 3 in 4 devices.

ecoCode (Le Goaer & Hertout, 2022) provides multiple plugins for SonarQube, a code evaluation platform. EcoCode currently supports Java, JavaScript, PHP, Python and C#. It also includes plugins for Android and iOS, which use a set of *smells* to detect “bad” (energy-consuming) code patterns. The labelling mechanism employed in EcoCode is the green quality gate, a SonarQube quality gate with a threshold of four issues, meaning a project is not publishable if four or more issues are present.

EcoDroid (Behrouz et al., 2015) employs static and dynamic analysis of applications in the same category to obtain their energy consumption. It is capable of generating an alphabetic label from A (most efficient) to E (least efficient) from an e_{index} obtained by dividing the energy expenditure of API calls per path possible in an application’s call graph by the total number of visited nodes out of all the nodes available in the call graph.

EcoIndex (Bordage, 2024) uses the information on the user’s hardware, the data transferred over the network, and the number of server requests to create a score of a website. These three dimensions are used to obtain a weighted average that concludes in an alphabetic label from A (most frugal) to G. EcoGrader (Mightybytes, 2024) uses CO2.js to obtain available metrics of energy consumption in data transfer and content load time, among others, to rate websites. EcoGrader has two different labels: the EcoGrader score based on the Sustainable Web Design Model (Design, 2024), and the Digital Carbon Rating based on HTTP Archive Transfer Size (kb) and grams of CO2 emitted per pageview. Globemallow (2024) is similar to EcoGrader and EcoIndex, but it uses the American grading system to rate webpages from A+ (most efficient) to F.

Kastor (Spécinov, 2024) is another tool similar to EcoIndex, EcoGrader and Globemallow. Kastor gives scores for accessibility, ecodesign, and also a global score, all based on a weighted average according to a study of good practices. A leaderboard of webpages based on the global score is also available. The Experiment Impact Tracker framework (Henderson et al., 2022) proposes (but is not limited to) carbon-friendly cloud-provider regions to be used for Machine Learning (ML) training, and a proposal of a leaderboard of frugal ML approaches as an incentive for more frugal ML. The leaderboard is based on the performance return per kWh consumed by the ML model. Li10 (2024) reports the estimated CO_2 emissions of hardware consumption on different cloud computing providers. Li10 assigns a Cloud Sustainability Score to the user’s emissions, its precise algorithm is, to our knowledge, proprietary.

While existing labels and tools like Carat, EcoCode, and EcoGrader provide valuable insights into software efficiency, they all rely on analyzing a functioning application. In difference to other planned energy-saving strategies that can be created before software deployment, such as scheduling algorithms (Fernández-Montes et al., 2012; Sun et al., 2023), reactive software categorization approaches miss the opportunity to incorporate sustainability into the software design from the very beginning. Recent community studies, such as Potentials of Green Coding (Junger et al., 2024), highlight this same limitation: most tools operate at runtime, monitoring CPU, memory, storage, or network consumption under specific workloads, and then translating these observations into scores or labels. While this provides valuable transparency for deployed systems, it does not directly guide architectural or design choices. A proactive approach would empower developers to make informed decisions early on, ultimately leading to greener software and a more sustainable future.

Recent community studies, such as Potentials of Green Coding (Junger et al., 2024), highlight this same limitation: most tools operate at runtime, monitoring CPU, memory, storage, or network consumption under specific workloads, and then translating these observations into scores or labels. While this provides valuable transparency for deployed systems, it does not directly guide architectural or design choices. As noted by Green Software: The Curse of Methodology (Hindle, 2016), precise measurement is inherently limited by the constant evolution of software and hardware, and progress requires bridging multiple disciplines toward a common goal. **Our goal is therefore to create a software sustainability label applicable during the design phase, even before any implementation effort, as opposed to existing approaches that require functional code to work and fine-grained metrics that can quickly become obsolete.** A proactive, high-level framework empowers developers to make informed choices early on, leading to greener software and a more sustainable future.

3. Research artifacts employed

Before introducing our proposal for the algorithm, we believe there are several important research artifacts that need to be briefly introduced to provide a better picture of its role within a custom toolset for green software design we created. The first artifact is the Behavior-Based Consumption Profiles (BBCP) (Larracoechea et al., 2024a), a Domain Specific Language (DSL). The objective of the BBCP is to model the behaviour of software and the evolution of its hardware consumption over time. In contrast to other approaches, the BBCP allows software designers to embed an estimation of the intensity of the usage of the software in its design, meaning that the resulting software design considers dynamic variations that cannot be controlled but should be accounted for, such as volatile user behaviour and stochastic events.

Later, the aforementioned intensity can be translated to an estimation of hardware resource usage and, therefore, energy consumption. The BBCP consists of properties at 2 levels of abstraction: the high level and the low level. The high level is constituted by qualitative properties that describe the non-functional behaviour of the software. The low level consists of quantitative properties that assert how the behaviour of software is translated into hardware consumption.

The algorithm explained throughout this paper employs 3 sets of high-level properties and 1 set of low-level properties of BBCP. The high-level properties are distributed into 3 categories, which are available in Table A.1 of Appendix A. The categories, which are similar to the ones proposed by Forward and Lethbridge (2008), are used to generalize each group of properties. Properties in the *computation-centric* category are responsible for the description of how the software product is expected

Table 1
The qualitative properties included to provide data to the algorithm on the consumption of hardware sensors.

Sensor	Possible values
NIC	On/Off/Circumstancial/Absent
Bluetooth	On/Off/Circumstancial/Absent
GPS	On/Off/Circumstancial/Absent
Camera	On/Off/Circumstancial/Absent
Display (brightness level)	Off/Low/Medium/High/Auto/Absent

Table 2
Quantitative properties fed to the algorithm to account for hardware resources and the programming language used to execute the demands of software.

Resource	Unit
CPU	GHz
RAM	GB
Network (NIC)	Mbps
Storage	GB
Programming Language	Rank in Pereira et al. (2021) re-ranked version of rosetta code

to perform its computation. *Data-centric* properties describe the relationship of the software with data it creates or receives. Finally, the *conduct-centric* category contains properties that describe the behavioural traits of the software. A more in-depth explanation of each property is available in previous related publications (Larracoechea et al., 2021, 2022, 2024a).

In contrast to previous approaches, the BBCP categories reflect a coalition of three perspectives: the Behavioural-Software Engineering (BSE), which describes how the business decisions affect the software's design; Human-Computer Interaction (HCI), which describes how the user interacts with software and vice-versa; and the Software Architecture (SA). For instance, the properties in the Computation-Centric category reflect the needs of the business and the software architecture. For example: a computation criticality value of *high* can reflect the necessity to comply with a pre-defined quality-of-service response time stipulated in service-level agreements (SLAs), while a computational complexity with a *high* value can reflect that the software architecture must be tailored to prioritize response time (the computation criticality), at the cost of a greater computational expense.

In addition, the properties in the behaviour-centric category describe behavioural characteristics that could be attributed to the user. For example, a *definite* consumption rate could be used to describe a video streaming service, as the activity's duration is equal to the content's length. For example, a 1 h-long movie has a definite consumption rate of 1 h. The importance of the properties selected for the algorithm is that they allow us to better understand the intent of the software designer, and take the qualitative characteristics of the software's design into account as a possible source of power expenditure.

To complement the aforementioned properties, we include some sensor-related properties, which are available in Table 1. The purpose of these properties is to define the state of a specific sensor at the time of software activation, participating in our method as a dependent variable that could significantly impact the final categorization of a software design. As for the low-level properties, we are limiting the algorithm to five of them, which are available in Table 2. They represent the basic hardware resource consumption that software usage requires in any computing hardware platform, and they are the most common metric to specify hardware requirements that software developers propose for the execution of their software products.

The last elements that are directly related to the low-level properties and play a significant role in making our algorithm useful during the design stage of software development are *consumption guides*. Consumption guides are databases meant to guide software designers through the process of selecting an approximate amount of hardware consumption per available hardware resource. They are a by-product of the process we followed during the development of the BBCP, in which we performed a taxonomical study of software-induced hardware consumption. Throughout the process, we collected multiple samples of software-hardware requirements that were made publicly available by their software developers, based on several horizontal slices of software types and categories we selected from the taxonomy created by Forward and Lethbridge (2008).

The second research artifact involved in the creation of our algorithm is RADIANCE+, formerly known as "RADIANCE" (Larracoechea et al., 2024b). RADIANCE+ is a custom-built tool designed with three main objectives in mind: (1) assist in the creation of BBCPs, (2) interpret the BBCPs with a custom execution engine, and (3) categorize and label software designs based on the BBCP DSL according to user-customizable algorithms. The proposal in this paper addresses the third objective.

A multistep process incorporated in RADIANCE+'s workflow begins by providing the users with a selection of choices per four core properties: software category, software type, target hardware platform (desktop, mobile, and laptop), and programming language, all at a per-operation level and within the context of a BBCP profile. Additionally, consumption guides pre-built consumption guides are incorporated into RADIANCE+ to improve the convenience of the design process. For the programming language, the user is presented with a list of options to choose from, which originates from a study performed by Pereira et al. (2021) on the energy consumption of programming languages. It is important to note that subsequent research has questioned the conclusions of the previous study. Thus, van Kempen et al. (2025) argue that, when controlling for factors such as the programming language implementation, application implementation, number of active cores, and memory activity, the choice of programming language has no significant impact on energy consumption beyond the execution time. This suggests that the energy efficiency rankings proposed by Pereira et al. may not be as definitive as previously thought. In light of this, RADIANCE+ offers a flexible framework where users can update and refine the consumption guides as new research emerges with better consumption guides. This adaptability ensures that the tool remains relevant and accurate, allowing the community to contribute to the continuous improvement of energy consumption models.

4. The expert algorithm

Our algorithm consists of 3 core parts: the Hardware Consumption Score (HCS), the Sensor Consumption Score (SCS), and the Behavioural Consumption Score (BCS). This compartmentalization was selected as a representation of our aforementioned interest in the BSE, HCI, and SA perspectives, explained in Section 3, so that each component can be characterized and studied independently from each other.

The HCS is a metric that quantifies the consumption of various hardware resources on a computing system. More specifically, the HCS is a weighted usage ratio of different hardware resources, which include the CPU, RAM, network, and storage devices. It uses the properties in Table 2 and considers a hypothetical resource availability to approximate the usage ratios per hardware resource:

- CPU efficiency ratio $(\frac{TDP}{Ca}) \cdot Cc$: This is calculated by dividing the CPU's Thermal Design Power (TDP, in watts) by the CPU base frequency reported by the hardware manufacturer (Ca, in GHz) to obtain the power-to-cycle ratio, which is later multiplied by the hypothetical consumption of the CPU (Cc) to obtain an approximation of the power consumption and heat output to dissipate. Moreover, this method punishes raw CPU cycle capacity and fosters efficiency. While this method of deriving a metric for CPU consumption may not offer absolute precision, it invites a broader discussion on how alternative approaches risk conflating the characterization of software behaviour with that of hardware performance.

Table 3
The weight of each hardware resource.

Tag	Component	Weight
$NETw$	Network Interface Card (NIC)	0.5
$CPUw$	CPU	0.3
$RAMw$	RAM	0.1
STw	Storage	0.1

- RAM usage ratio ($\frac{Rc}{Ra}$): Similarly, this is calculated by dividing the RAM consumption (in GB) by the RAM availability (also in GB).
- Network Interface Card (NIC) usage ratio ($\frac{Nc}{Na}$): This is calculated by dividing the network card consumption (in Mbps) by the network card availability (also in Mbps). The consumption is calculated by obtaining the average between the transmission and reception.
- Storage usage ratio ($\frac{Sc}{Sa}$): Finally, this is calculated by dividing the storage consumption (in GB) by the storage availability.

As previous research has found, the programming language plays an important role in the final power consumption of software. In addition to the previous ratios, we employed the global Rosetta Code ranking of programming languages (Code, 2025), which was re-ranked according to energy consumption by Pereira et al. (2021). To do so, the position in the aforementioned rank of the selected language is multiplied by a weight. The weight value employed was obtained by calculating the Inter-Quartile Range (IQR) of the rankings and dividing the result by 1000 to tone down the relevance of the result in the HCS. We are aware that during the design phase the values of the aforementioned properties are difficult to predict to produce accurate ratios. Nevertheless, we propose that software designers and architects use the system requirements of a similar application to the one being built as a frame of reference. Moreover, a market analysis of the target users and target hardware platform should be included in the process of obtaining the ratios to facilitate the decision-making.

In addition to the ratios, a system of weights whose individual values range from 0 to 1 is included to prioritize the consumption of certain hardware components over others. As seen in Table 3, we distributed the weights so that the network card carries the heaviest weight, with the CPU following it in second place and the RAM and storage in the last place. This hierarchy is backed by the results of previous research (buildcomputers.net, 2023; Carroll et al., 2010; Chiaravalloti et al., 2011) where the consumption of hardware components is assessed under diverse circumstances to analyze the portion of the battery that each one consumes. The final HCS is given by Eq. (1).

$$HCS = \left(\sqrt{\frac{TDP}{Ca}} \cdot Cc\right) \cdot CPUw + \frac{Rc}{Ra} \cdot RAMw + \frac{Nc}{Na} \cdot NETw + \frac{Sc}{Sa} \cdot STw + Pr \cdot Pw \quad (1)$$

where:
TDP = the TDP value of the CPU model of the target hardware
Cc = a deliberate CPU consumption value from a BBCP
Ca = a deliberate CPU availability value
CPUw = the weight value for the CPU in Table 3
Rc = a deliberate RAM consumption value from a BBCP
Ra = a deliberate RAM availability value
RAMw = the weight value for the RAM in Table 3
Nc = a deliberate NIC from a BBCP
Na = a deliberate NIC availability value
NETw = the weight value for the NIC in Table 3
Sc = a deliberate Storage consumption
value from a BBCP
Sa = a deliberate Storage availability value
STw = the weight value for the Storage inTable 3
Pr = the rank of the programming language in
Pereira et al. (2021)
Pw the IQR of the rankings divided by 1000

Table 4
Available sensors taken into account in the Sensor Consumption Score.

Sensor	Weight
Network Interface Card (NIC)	0.25
Bluetooth	0.25
GPS	0.2
Camera	0.1
Display brightness	0.2

Table 5
State-dependent values per sensor.

Sensor	State	Value (0 to 1)
NIC	Absent	Redistribute weights and remove from the HCS
NIC	Off	0
NIC	On	1
NIC	Circumstancial	0.5
Bluetooth	Absent	Redistribute eights
Bluetooth	Off	0
Bluetooth	On	1
Bluetooth	Circumstancial	0.5
GPS	Absent	Redistribute weights
GPS	Off	0
GPS	On	1
GPS	Circumstancial	0.5
Camera	Absent	Redistribute weights
Camera	Off	0
Camera	On	1
Camera	Circumstancial	0.5
GSM	Absent	Redistribute weights
GSM	Off	0
GSM	On	1
GSM	Circumstancial	0.5

Table 6
Possible values for the display's brightness level.

Display brightness (Db)	Value (0 to 1)
Absent	Redistribute weights
Off	0
Low	0.4
Medium	0.6
High	1
Automatic brightness control using an external mechanism or sensor	0.5

The next part of our algorithm, the Sensor Consumption Score, considers the array of sensors in Table 1, which are most frequently present in smartphones. The only purpose-specific sensors we decided to include are the GPS and the camera, due to the relevant energy consumption of the GPS and the relevant data generation of the camera. A similar weight system to that in the HCS is also included, with the Wifi/4G (NIC) and Bluetooth considered the "heaviest", followed by the GPS and, finally, the camera. The weights are available in Table 4, and we distributed them to reflect the results of the research performed by Khan et al. (2016). There are four possible states per sensor: *Absent*, *off*, *on*, and *circumstantial*, where a *circumstantial* value attaches the sensor state to the software state (the sensor value applies as long as the software is directly in use), and an *absent* sensor means that the weights should be modified to account for a lack of a specific sensor by distributing its weight evenly across the available sensors. Not distributing the weight of an absent sensor would lead to skewed results, where the profiles whose target platform lacks a sensor would receive an unfair advantage. A quantitative value for each sensor state is available in Table 5.

The last addition to the SCS is the display brightness (Db), whose state-dependent value is available in Table 6. We decided to include the display brightness in the SCS as the research performed by Chen et al. (2013) reveals an important consumption of electrical energy generated by the display, especially when paired with the camera. The total SCS

is given by Eq. (2), using the values in Tables 5 and 6 as an input for the variables, and the values in Table 4 for the weights.

$$SCS = Nv \cdot Nw + Bv \cdot Bw + Gv \cdot Gw + Cv \cdot Cw + Db \cdot Dw \tag{2}$$

where:

Nv =	a value used for the NIC in Table 5
Nw =	the weight used for the NIC in Table 4
Bv =	a value for Bluetooth in Table 5
Bw =	the weight used for Bluetooth in Table 4
Gv =	a value for GPS in Table 5
Gw =	the weight used for GPS in Table 4
Cv =	a value for camera in Table 5
Cw =	the weight used for camera in Table 4
Db =	a value for the display brightness in Table 6
Dw =	the weight used for display brightness in Table 4

The last component in the algorithm is the BCS. The BCS is created by taking into account the selected value for each property of the BBCP and giving it a *property score value*. The equivalences from BBCP values to property score values are available in Table A.2 of Appendix A. A table that assigns a *combinational score values* to the combination of specific properties' values is also available in Table A.3 of Appendix A.2. Combinations are an important part of the algorithm, as they help us to detect peculiarly detrimental behavioural patterns that could be prevented. For instance, a task distribution with a lack of a strategy would be detrimental to energy consumption. Therefore, it is of the essence that the label should reflect the danger of this pattern. In addition, detecting these "energivore" (higher than usual energy consumption) combinations aid us to generate an "eco-report" to elaborate on why such combinations are detrimental, guiding software designers with it. The final BCS is given by Eq. (3), where the variables labeled from A to K represent the pertinent property score value, whose value can be found in Table A.2. Co represents the total combinational score obtained with the possible combinations of properties values in Table A.3.

$$BCS = \lceil (Co + \sum_{i=A}^{K} S_i) \rceil \tag{3}$$

where:

S_i =	property's score value in Table A.2
Co =	valid property combinational score value in Table A.3 of Appendix A.2

The algorithm includes the components explained so far to create a final Consumption Score (CS). The CS is given by Eq. (4), which produces an estimated percentage of a profile's hardware consumption, including the effect of its behavioural traits.

$$CS = (HCS \cdot HCSw + SCS \cdot SCSw) \cdot 100 + BCS \tag{4}$$

where:

HCSw =	a weight value $0 \le HCSw + SCSw \le 1$
SCSw =	a weight value $0 \le HCSw + SCSw \le 1$

The pseudo-code of the algorithm for a single profile is available in Algorithm 1. Once the three core components (HCS, SCS, and BCS) are obtained from the variables in the profile, the CS is obtained. The last step is to categorize (label) the estimated total percentage of hardware consumption. The categorization consists of labels from A to G, where A represents the lower hardware resource usage and G is the highest, similar to the EU's energy label, using that similarity to our advantage as people already react positively to this categorization scheme. With respect to the almost equidistant ranges that make each label eligible, we are convinced that at the current stage they suffice for providing some guidance but we do not consider these ranges to be permanently applicable. In the same fashion in which the EU's energy label has been rebalanced multiple times to close holes in their algorithms, with sufficient data a better distribution of the values for each threshold that make a label applicable could be considered. A comparison of results among our linear, Pareto, and Gaussean scales is later included in the experimental evaluation section of this paper.

Algorithm 1 Obtaining a consumption score and a label.

```
Require: $BBCP$, a profile containing all the values required to generate the $CS$
1: $PCS \leftarrow$ the consumption score of the profile
2: $PCS = RateProfile(BBCP)$
3: procedure RATEPROFILE($BBCP$)
4:     HCS, SCS, BCS ← values from the respective profile
5:     Consumption Score ← $(HCS \cdot HCSw + SCS \cdot SCSw) \cdot 100 + BCS$
6:     if Consumption Score $\le 20\,\%$ then
7:         label ← "Category A"
8:     else if $20\,\% >$ Consumption Score $\le 30\,\%$ then
9:         label ← "Category B"
10:    else if $30\,\% >$ Consumption Score $\le 40\,\%$ then
11:        label ← "Category C"
12:    else if $40\,\% >$ Consumption Score $\le 50\,\%$ then
13:        label ← "Category D"
14:    else if $50\,\% >$ Consumption Score $\le 60\,\%$ then
15:        label ← "Category E"
16:    else if $60\,\% >$ Consumption Score $\le 70\,\%$ then
17:        label ← "Category F"
18:    else if $70\,\% >$ Consumption Score then
19:        label ← "Category G"
20:    end if
21:    return Consumption Score, label
22: end procedure
```

As the BBCP allows software designers to describe multiple operations per profile, we created a variation of the algorithm that obtains the CS of each operation. Furthermore, a CS for each operation allows us to label the *intensity* of the hardware consumption and the behaviour characteristics of profiles. For instance, an elevated amount of "A" rated operations running simultaneously could severely impact energy consumption (a higher intensity), similarly to a G labeled operation. To label intensity, the algorithm performs a step-by-step analysis of the state of each operation, its prerequisites, and its results; all as defined by the BBCP. The evolution of the activity per step is rated with the average CS for all the active operations in each step, which is later labeled. In Fig. 1, a visual representation of a step diagram consisting of 4 operations and 3 steps can be observed. In the diagram the activity or lack of activity per operation is shown as "on" and "off" states respectively.

The variation of Algorithm 1 that supports step rating is available in Algorithm 2. This is of vital importance, as software architecture requires that the designers understand *when* each operation will take place. Such necessity is an opportunity to provide software designers with an estimation of the parallel execution of their architectures, identifying the most resource-consuming sequences and points in time. As the $StepRating$ consists of the average hardware consumption of the operations at a given step, the rating can become unreliable in scenarios where the difference between the consumption of operations is too great. To address the issue, we decided to give steps where the aforementioned scenario is present the rating of its more demanding operation; this is performed by $WeightStepScore()$ in Algorithm 2.

The specific steps followed by $WeightStepScore()$ are the following:

(i) We obtain the *Step Variance* (variance) among the consumption scores (Ocs) of the operations in a given step using its array of operations' consumption scores:

$$StepVariance = \frac{\sum_{k=1}^{R_l} (O_{cs}[k] - S_{cs})^2}{R_l} \tag{5}$$

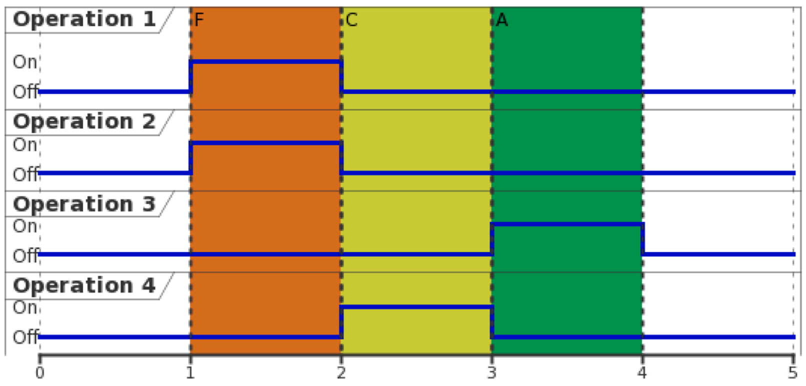

Fig. 1. An example of a step diagram created with PlantUML using the results of the algorithm.

Algorithm 2 Obtaining a consumption score and a label for a profile step using RADIANCE+.

```
Require: $BBCP$, a populated BBCP with all the values necessary for generating a $CS$
Require: $Os$, the state of every operation for a given step
 1: $StepRating = RateStep(RateOperationsInStep(BBCP, O_s))$
 2: $StepScore = ScoreStep(StepRating)$
 3: procedure RATEOPERATIONSINSTEP($BBCP$, $Os$)
 4:     $O \leftarrow$ all the operations in the $BBCP$
 5:     $A_{ocs} \leftarrow$ an array with the $CS$ of each operation
 6:     for each $o \in O$
 7:     if $Os[o]$ is On then
 8:         We obtain:
 9:         HCS, SCS, BCS $\leftarrow$ of operation $[o]$
10:         $Ocs \leftarrow$ the $CS$ of operation $[o]$
11:         We push the tuple ($Ocs$,$[o]$) into $A_{ocs}$
12:     end if
13:     end for
14:     return $A_{ocs}$
15: end procedure
16:
17: procedure RATESTEP($A_{ocs}$)
18:     $Rl \leftarrow$ length of the array $A_{ocs}$
19:     $StepConsumption = \frac{(\sum_{k=1}^{Rl} A_{ocs}[k][Ocs])}{Rl}$
20:     $WeightedStepRating = WeightStepScore(StepConsumption, A_{ocs})$
21:     return $WeightedStepRating$
22: end procedure
23:
24: procedure SCORESTEP(StepRating)
25:     $label \leftarrow$ obtained with the criteria of Algorithm 1
26:     return $label$
27: end procedure
```

where:

Rl = the number of operations in the step
O_{cs} = the operations' consumption score
S_{cs} = the mean consumption score of the step

(ii) We obtain the standard deviation of the variance, $StepDeviation$ (SD) using the $StepVariance$ (SV), of the previous step:

$$StepDeviation = \sqrt{SV} \tag{6}$$

(iii) We decide the final $StepRating$ based on the SD:

$$StepRating = \begin{cases} \text{highest score in } A_{ocs}, & \text{if } SD \geq SR_t \\ \text{average } O_{cs} \text{ for the step}, & \text{if } SD < SR_t \end{cases} \tag{7}$$

where:

SR_t = a deliberate threshold amount

After obtaining a *weighted* Step Score through the steps above, the step is assigned a label, and the step rating and categorization procedure is finished. In the following section, we present a series of experiments where we profile and rate (Discord, 2024), a voice, chat, and video call platform, under diverse circumstances, to observe how the current weight system's values and each component of the algorithms categorize it. In addition, we also compare the label generated with our algorithm when profiling a 3D rendering process on two different machines.

5. Experimental evaluation

The experimental evaluation of the expert algorithm consisted of 2 different sets of experiments: a set of test cases using a voice/text/chat communication app and a test case running a 3D modeling app. There were 3 main objectives to our experiments: (1) test the expert labelling algorithm with preexisting software by obtaining hardware consumption metrics under diverse circumstances; (2) test if the properties of software behaviour proposed with the BBCP appropriately affect the label, and aid the user to infer the behaviour of an existing software; and (3) determine if transferring the consumption of resources demanded by software in one device produced a different label in other hypothetical machines, and was meaningful. The premise of these experiments is that labelling a similar application to the application that software designers and developers wish to create, using our algorithm, will tell them:

- Which features of the application could consume power the most.
- What behaviours of the application can increase power consumption.
- How could the features of their application be labelled and how much do each of them matter.
- How different combinations of software behaviours can affect power consumption.
- How efficient their designs could be when executed on hardware platforms with varying configurations.

5.1. Specific tools and methods

Democratizing the adoption of software design labelling methods as our goal implies that the methods of collecting sample data should remain easy to replicate. Due to the scope of our work, precision was not of the outmost importance to us, but obtaining a baseline with a simple method of data recollection remained necessary to generate a frame of comparison that we could use to test our proposal. Specifically, the tools and methods used during the collection of data for each test case and their variation are as follows:

Table 7
Experimental test cases' samples.

Tag	Sample Name	Execution Place	Noise Suppression	Noise Level (dB)	Network Connection	Frame Rate	Image Resolution
TC1S1	Idle Sample 1	Foreground	N/A	N/A	N/A	N/A	N/A
TC1S2	Idle Sample 2	Background	N/A	N/A	N/A	N/A	N/A
TC2S1	Voice Call Sample 1	Foreground	Off	60	N/A	N/A	N/A
TC2S2	Voice Call Sample 2	Foreground	On	60	N/A	N/A	N/A
TC2S3	Voice Call Sample 1	Background	Off	60	N/A	N/A	N/A
TC2S4	Voice Call Sample 2	Background	On	60	N/A	N/A	N/A
TC3S1	Video Call Sample 1	Background	Off	60	WAN	N/A	N/A
TC3S2	Video Call Sample 2	Background	On	60	WAN	N/A	N/A
TC3S3	Video Call Sample 1	Foreground	Off	60	WAN	N/A	N/A
TC3S4	Video Call Sample 2	Foreground	On	60	WAN	N/A	N/A
TC4S1	Live Stream Sample 1	Background	N/A	N/A	LAN	30	720p
TC4S2	Live Stream Sample 2	Background	N/A	N/A	LAN	60	1080p
TC4S3	Live Stream Sample 3	Background	N/A	N/A	WAN	30	720p
TC4S4	Live Stream Sample 4	Background	N/A	N/A	WAN	60	1080p

- Power consumption recording: a smart power meter was employed during each sample to record the average system-wide power consumption of each laptop. More precisely, we used a Xiaomi Smart Plug 2. We chose this smart meter because it is available in multiple regions of the world, has and affordable price, and is consumer friendly, reducing the need of specialized hardware, which is frequently identified as a hurdle in the adoption of green software methods (Hindle, 2016). Besides, the laptops employed during the experiments ran solely on A/C power to enhance the precision of our power consumption readings.
- Hardware recording: the laptops employed during the test cases ran Windows 11 build 23H2, and PerfMon was used to record hardware consumption with a sampling rate of one sample every 15 s ($\frac{4}{60}Hz$).
  - The network consumption employed in each profile per variation of each test case is the average of the sum of the transmission and the reception rates in Mbits.
- Repeatability and precision: 15 samples of each test-case variation were recorded to ensure repeatability and enhance the precision of our reading. Data were statistically processed to remove outliers using a Z score method. The test cases which involved user input were performed using a lightweight macro-recording software called TinyTask, ensuring repeatability of the input commands and a minimum overhead.

Although these tools and methods used in our experimental evaluation are not extremely sophisticated, they serve our purpose and facilitate the replication of the experiments.

5.2. Labelling an internet-based communication application

For the first set of experiments, consisting of 4 test cases with multiple variations, we selected Discord (2024) (a popular voice, chat, and video call application) as the test application. The purpose of the four tests was to observe the labels generated with our algorithm in relation to the consumption generated by each feature of Discord. The starting parameters and variations for each test case sample are available in Table 7. Discord was selected due to its popularity in the age groups 18 to 34 (Ceci, 2023), which are heavily engaged in video game playing; a relatively energy-expensive activity when compared to other computer-based activities (Mills et al., 2019). The test cases defined are suitable for our purposes, as we want to illustrate how our approach can be used to easily help developers to estimate software consumption from the design phase. Whereas other possible test cases could be defined, and more demanding experimental procedures could be used, it should be noted that we do not focus on hardware assessment.

The first test case (TC1), *idle consumption*, involved recording the baseline hardware consumption of Discord in a source machine running Windows 11, using perfmonitor and a sampling rate of 15 s, without any interference or usage other than Discord's home screen. Two samples of this test case were created with Discord executed in the foreground (TC1S1 in Table 7) and then in the background (TC1S2 in Table 7), to observe if a difference in consumption existed, as the algorithm does differentiate between foreground and background usage. The hardware specifications of the source machine used in all test cases are available in Table 8, the complete experimental setup for each sample and their results are publicly available as supplementary material (Larracoechea et al., 2025). The steps for carrying out the first test case consisted of the following:

Table 8
Specifications of the portable computer (source machine) used for most of the experiments (Lenovo Legion Go).

Component	Specification
CPU	3.3 GHz AMD Z1 Extreme
TDP	15 W (manually set)
Storage	516 GB
RAM	16 GB
Network	2400 Mbps

1. Discord was executed and left on its home screen first in the foreground (TC1S1) and then in the background (TC1S2).
2. Windows 11 hardware resource monitor (perfmon) was left recording Discord's hardware usage in the source machine for 5 min.
3. The hardware resource consumption reported by the monitor during the test, and the behavioural traits of the application were used to characterize an operation in a BBCP profile.

The second test case, *voice call* (TC2), involved recording the hardware consumption of Discord during a 10 min voice call between 2 participants connected through different networks. Only the metrics of our source machine were recorded. Two samples of this test case were performed: one with Discord running in the background and another with Discord in the foreground. Additionally, both samples were executed with noise suppression either on or off (TC2S1 to TC2S4). The purpose of sampling the differences in noise suppression "on" and "off" was to observe if it had an impact on the hardware consumption. The complete experimental setup for each sample and their results are available as well in the supplementary material (Larracoechea et al., 2025). The steps for each sample of the test case were as follows:

1. Discord was executed and left in a voice call in either the foreground (TC2S1 and TC2S2) or the background (TC2S3 and TC2S4) and either noise suppression off (TC2S1 and TC2S3) or on (TC2S2 and TC2S4) depending on the test case sample.
2. The hardware resource monitor was left recording Discord's hardware usage for 10 min.
3. A vacuum cleaner with an approximate sound level of 60 dB was left running in the background during the test while a text script was read

Table 9
Minimum system requirements for executing Dusk.

Component	Specification
Processor	2.4 GHz
RAM	2 GB
Storage	2 GB
Network	0 Mbps (offline)

aloud by a person. The vacuum cleaner served as a noise baseline to test noise suppression.

4. The hardware resource consumption reported by the monitor during the test was used to characterize an operation in a BBCP.

The third test case, *video call* (TC3), involved recording the hardware consumption of Discord throughout a 10 min video call between 2 participants in separate networks, where only the source machine's consumption was recorded. Similar patterns with slight variations (TC3S1 and TC3S2) as the previous test case were conducted using similar steps. The only difference was the use of a 1080p external camera for the video call, capturing our research member reading a script.

The fourth and final test case, *video game live stream*, (TC4S1 through TC4S4) involved recording the hardware consumption of Discord for a 10 min video game stream between 2 participants, recording the consumption of the source machine only, while also executing the game being streamed. The game chosen for this test is "Dusk" a 3D shooter game. Dusk was chosen due to its low system requirements and its availability. Dusk's system requirements for Windows are available in Table 9.

There are 4 samples of this test case (TC4S1 to TC4S4), each considering different live stream image resolutions and frame rates, with comparisons between 2 machines on a LAN (Local-Area Network) and two machines on different networks. The steps followed for each variation of this test case are as follows:

1. Discord was executed and left in a video call with the second machine, both connected to the same LAN (TC4S1 and TC4S2) or separate networks (TC4S3 and TC4S4) depending on the test variation.
2. Dusk was started and Discord was switched to live game streaming mode, with either 720p at 30 FPS (TC4S1 and TC4S3) or a 1080p resolution and 60 FPS (TC4S2 and TC4S4). The integrated webcam of the source machine was turned off.
3. A member of our research team played the same level of Dusk for 5 min while the hardware resource monitor recorded Discord's consumption in the source machine.
4. The hardware resource consumption recorded by the monitor was used to characterize an operation in a BBCP profile.

It should be noted that the properties *Operation Depth*, *Operation Task Distribution*, and *Operation Data Handling* override the values of *Depth*, *Data Handling*, and *Task Distribution*, as the feature sampled can change the way the application is perceived. Furthermore, disparities among samples reflect their intrinsic differences e.g.: video call samples' screen brightness is used in the foreground (TC3S1 and TC3S2) but not in the background (TC3S3 and TC3S4).

Taking TC4S1 (720p 30fps LAN live stream) as an example, the logic followed to characterize each BBCP profile for the test cases was similar to the following:

1. **Task Distribution:** *Centralized*, as the hardware consumption and monitoring occurs in the same machine.
2. **Computational criticality:** *High*, as the live stream is meant to be "live"; as close to "real time" as possible.
3. **Computational complexity:** Set to *High*, as we expect the constant video feed collection, compression and network casting to be a taxing operation.
4. **Distribution strategy**: *No*, as there is no need for it (equivalent to off, as the task distribution is set to centralized).
5. **Consumption rate:** *Indefinite*, as normally a live stream's duration depends entirely on user's discretion.
6. **Data flow behaviour:** *Regular*, as we have a clearly defined expectation of the rate at which data will be shown (30 fps); live video interruption is considered a detriment to the user.
7. **Data flow direction:** *Bi-directional* as the source machine was also receiving data from the "spectating" machine (audio feed).
8. **Data handling:** *Destroy*, as the live stream was not meant to be stored for further reproduction.
9. **Access Frequency:** *Irregular*, as the live stream occurs at the user's discretion.
10. **Depth:** *Background* as the user sees and interacts with Dusk, not with Discord.
11. **Dependence:** *Dependent* as it relies on Dusk to gather the data for the feed.
12. **Operation data handling:** *Destroy* as there are no differences between the operation and the profile it exists in.
13. **Operation task distribution:** *Centralized* for the same reasons as data handling.
14. **Operation depth:** *Background* for the same reason as the profile's depth value.
15. **CPU usage:** Obtained from the resource monitor as an average percentage of CPU consumption in the last 60 s of execution. Re-scaled to actual consumption in GHz.
16. **RAM usage:** Obtained from the resource monitor as a total working consumption in KB, re-scaled to GB; RAM consumption without the total RAM allocation.
17. **Storage usage:** Obtained from the resource monitor as a total in KB, re-scaled to GB.
18. **Network usage:** The average total bytes sent and received per second reported by the resource monitor.
19. **Camera:** *Off*, as it is not used during the sample.
20. **Screen brightness:** Set to *off* as Dusk was in the foreground.
21. **GPS:** *Off* as it is not required.
22. **Bluetooth:** *Off* as it was not required.
23. **NIC (WIFI):** *On* to perform the transmission of the live stream.
24. **Programming language:** *JavaScript* as Discord's front end is mostly programmed in TypeScript through React (Discord Blog, 2023).

As it can be seen, such a procedure can be used to infer the value for each property during the design phase of a software application by taking inspiration from existing applications of a similar type.

5.2.1. Results

We obtained a sample for each of the test cases by creating a test case profile using the BBCP and filling it with operations indicative of the setup for each test case sample; the details are available in Larracoechea et al. (2025). Later, Algorithm 1 was implemented in RADIANCE+. Finally, we created a profile that held the hardware specifications of the source and guest machines (Tables 8 and 12). As previously mentioned, the setup and results for each sample are available in Larracoechea et al. (2025) and system-wide power consumption for every sample was recorded with a smart meter to determine the power consumption of our source machine (see Table 11). We modified the algorithm to repeat each sample with each laptop specification that we made available for it; this means that, once we captured the resource consumption of Discord and Dusk in the source machine, the same data were reutilized to test the resilience of the algorithm and its utility in comparative hypothetical hardware. We also increased our subject size from the source machine to 12 laptop specifications: the source machine operating at 3 different TDP settings, and the specifications of some models among the most sold laptops in the 4th quartile of 2024 (Laptop Media, 2025).

The results of the complete procedure produced relevant labels and insights into how our algorithm pondered every characteristic and variable, giving us the opportunity to include a 3D scatter plot in

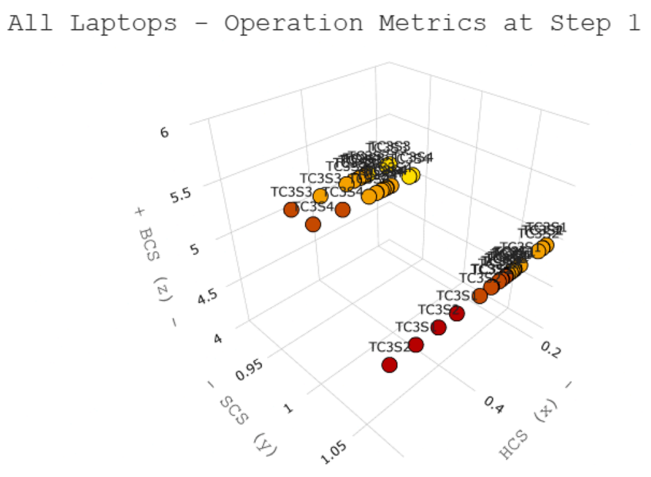

Fig. 2. Scatter plot of all the results of all samples in TC3 for all laptops.

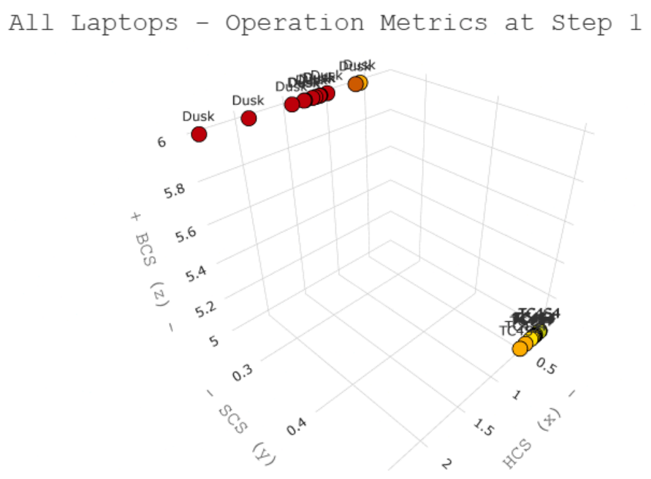

Fig. 3. Scatter plot for the simultaneous sampling (same step) of Dusk and TC4S4.

RADIANCE+ to expose how the results of each sample per laptop spread out according to their HCS, SCS, and BCS. A sample of such scatter plots belonging to TC3 is available in Fig. 2.

Algorithm 2 was also implemented in RADIANCE+ to categorize the simultaneous execution of TC4S4 and Dusk and obtain the score of a step with a step rating threshold (SR_t) of 15. We chose $SR_t = 15$ so that a deviation among operations' scores (SD) of 10 points (a threshold of 10) would automatically scale up the step's score to the heaviest (highest O_{cs}) operation in it. We decided on a value of 15 as a round-up if the deviation among operations' scores was close to shifting into a higher label. The resulting scatter plot for the results for this algorithm is available in Fig. 3. Regarding the results, most of the laptops inherited a label comparative to the Ocs of Dusk. This was due to the sheer difference in magnitude between the Ocs of both operations in the same step, which is evidenced by the observed distance in Fig. 3.

Additionally, we used the data in Table 10 and found that a positive correlation exists between the Gross CPU efficiency $\frac{TDP}{Ca}$ of the CPU in a hardware host and the Step Ocs as we intended, with $r(13) = 0.96, p < .05$, hence why almost all the laptops involved (84.6 % of them) were labeled G, and the remaining laptops with a D. The complete sample setup, hardware specifications, results and labellings for every sample in TC1 through TC4 are available in Larracoechea et al. (2025). To conclude, we experimented with the thresholds that label the CS of each sample with two additional distributions: Pareto and a Gaussian bell curve. This change in the distribution of the values assigned to the thresholds changed the rigor of the categorization, as can be seen in Fig. 4.

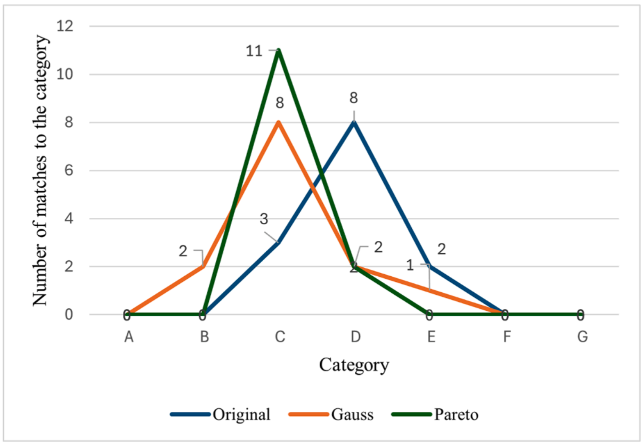

Fig. 4. Recategorization of the results obtained from TC4S4 with a rebalancing of the thresholds using Pareto and a Gaussian bell curve.

Table 10
Data summary for laptops.

Laptop Model	CPU efficiency	Step OCS	Label
HP 14 (HP 14-ep1001ns)	25	179.854	G
Apple MacBook Air 13 (2022)	6.198	99.101	G
Apple MacBook Air 15 (2024)	7.273	105.840	G
Acer Aspire 3 (A315-24P)	5.357	92.502	G
Lenovo V15 G2 IJL	5.455	93.183	G
Lenovo IdeaPad Slim 3 Chromebook	7.143	105.05	G
HP 15-7H3D6UA	5.357	92.49	G
HP 17-cn0000	5.957	96.60	G
HP 15-fd0166ns	1.765	61.32	F
Lenovo Legion Go (5W)	1.515	57.25	E
Lenovo Legion Go (15W)	4.545	86.56	G
Lenovo Legion Go (30W)	9.091	115.29	G
Alienware 13 R3 2013	16.071	147.19	G

Table 11
Recorded system-wide power consumption during each test case executed in the source machine @ 15W of TDP unless contradicted and a sample of the recorded power usage in the guest machine, all using a smart meter.

Test Case	Average power recorded (W)	CS obtained
TC1S1	17	13.57
TC1S2	17	14.428
TC2S1	15	18.310
TC2S2	15	19.196
TC2S3	15	23.301
TC2S4	15	23.071
TC3S1	19	57.32
TC3S2	19	59.020
TC3S3	16	49.234
TC3S4	17	50.191
TC4S1	17	39.708
TC4S2	17	39.017
TC4S3	17	39.150
TC4S4	17	39.686
Blender - Lenovo 5W (source)	16	19.224
Blender - Lenovo 15W (source)	35	78.058
Blender - Lenovo 30W (source)	52	126.092
Blender - Alienware 45W (guest)	68	172.654

5.3. *Labelling a 3D modelling and rendering application*

In addition to Discord, we also performed an experiment to categorize the hardware consumption of Blender (Foundation, 2024), an open-source 3D creation suite, while rendering a scene and generating a still image from a 3D scene. We selected Blender due to its open-source nature and repository of free test scenes. These previous and last experiments were created to demonstrate: (1) the categorization of a more

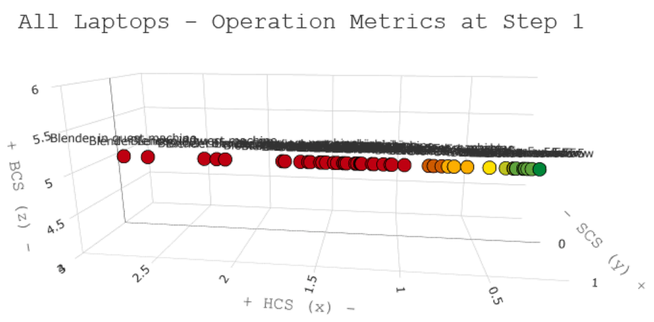

Fig. 5. Scatter plot of the results of each laptop sampled in the Blender test case.

Table 12
Specifications of the portable computer (guest machine) used for the rendering experiment with Blender.

Component	Specification
CPU	2.8 GHz (I7 7700HQ)
TDP	45W (OEM setting)
Storage	500 GB
RAM	32 GB
Network	867 Mbps

computationally expensive software operation and (2) the contrast in labels generated using two different hardware platforms. Table 7 summarizes the circumstances of the experiment surrounding each sample.

The same Algorithm 1 was used to sample each test case, with slight variations depending on the sample's demands. The test case consisted of the following:

1. The source machine was left rendering a control scene as-is (available in: https://www.blender.org/download/demo/bundles/bundles-3.0/asset-demo-bundle-3.0-cube-diorama.zip/), with no parameter modifications to the render settings.
2. The resource manager recorded the process from the beginning to the end of the render.
3. We created a BBCP and characterized it in a similar fashion to the previous samples to obtain a final label.
4. We repeated the steps with another machine we call the "guest machine", whose hardware specifications can be found in Table 12. Three samples were recorded with the test machine at a TDP of 5W, 15W, and 30W respectively.

We hypothesized, for the last experiment, that the energy label would be close to the worst, a G, as the rendering process employs "ray tracing": a technique where a collection of vectors representing a "ray of light" are scattered from a pre-defined source point in the 3d scene. The consequential bounces of the rays that go back into the source are taken as input for generating reflections (such as a reflection of a mirror) and shadows. This rendering technique is popular due to the quality of its results. Furthermore, it is the main feature in many contemporary video games which use this technology to create impressive surroundings for the players to play in.

5.3.1. Results

The data of the results for each of the samples in this test case can be found in Section 1.6 of the supplementary material (Larracoechea et al., 2025). They depict a similar scenario to Dusk without additional software labeled simultaneously (a step rating). The similarities are evident by the spread data points per laptop in the scatter plot of Fig. 5, which we attribute to a positive correlation in efficiency to Ocs.

5.4. Discussion

As justified by the results, and the significant amount of additional data we present in our supplementary online material (Larracoechea et al., 2025), it is manageable and informative to consider the peculiarities of a software product being executed in multiple tentative hardware hosts from a black box perspective and extrapolate the results from a single recorded sample (the guest machine) to generate labels that represent both the peculiarities of software and the appropriateness of a hardware host for execution. While the positive correlation between the CPU's gross power efficiency ratio may initially appear modest, we believe it provides meaningful support for our proposal. It highlights a valuable mechanism for promoting efficiency and greener software practices by punishing raw processing power or capacity. Moreover, we believe that delving deeper into the peculiarities of hardware would misguide our objective of labelling software (Comission, 2024b). While our study could be extended with even more experiments, the consideration of more variables, and fine-tuning of the weights for each component in our proposal, the results obtained and the recorded system-wide power consumption data available in Table 11 are already correlated by $r(18) = 0.937$, $p < .05$, and so they can be considered reliable.

However, we noticed that there are several research directions that could be approached. Firstly, as the hardware resource usage metrics are an average generated per second by the performance monitor, a study on the intensity of software usage (how long software is used over time) could be done to re-adjust or re-purpose the algorithm to consider the intensity of the usage, considering more than a single instance of a profile. We are currently exploring this direction, as the experiment with Blender demonstrated that, even though the proportional usage of the CPU for both machines remained more or less the same (around 94 % with the TDP set to 15W in the source machine), the disparity between the amount of time it took to render the scene in each machine is an important metric that matters to long-term consumption of resources.

Secondly, as the hardware resource usage we measured produced a low ratio of consumption to availability, more tests could be performed under flexible hardware availability to catalog software under restrictive hardware circumstances. Especially when software depends on others, such as how Discord depends on a game to perform live game streaming. This could be done by prioritizing the operations in the profiles with the highest computational criticality over other profiles, reducing the hardware resources available for the subsequent parallel execution of profiles.

Thirdly, the weights' values we assigned to the CS (0.7 for the HCS and 0.3 for the SCS) were selected before testing, with the initial hypothesis that the change in hardware consumption would be drastic among Discord's features. This proved to be true, and we found a positive relation, which is why we maintained these values. A deeper sensitivity analysis could be performed to better understand and assign weights that better reflect the intent of the software against its instant consumption, to produce a label based on projections, for example, an *indefinite* consumption rate could increase the weight of HCS (HCS_w).

We must also highlight that the best hypothetical hardware host for every experiment was the Lenovo Legion Go @ 5W of TDP (as seen in Tables 5, 9, 13, 17, and 21 of the supplementary material (Larracoechea et al., 2025)). This came partially as a surprise to us because we expected it to under-perform with such a harsh limit applied. The maximum clock speed we observed during the Blender sample @ 5W of TDP for instance was 0.55 GHz, from which it only consumed an average of 0.13 % during the experiment. This sparked in us enthusiasm for devices that let knowledgeable users tailor the performance to their needs against conforming to software demands. This could lead our readers to conclude that our results are skewed by the manual setup of our source machine, but we would like to remind them that the tests were conducted with the device set at 15W of TDP and that the consumption of resources recorded with it were used as the consumption baseline and, therefore, the baseline for cross comparison with other laptops. We are eager to see more manufacturers adopt features that cede the control over energy-hungry hardware components to the user in such a simple manner.

Finally, we recognize the relevance of incorporating insights from related fields such as energy analytics in cloud computing, process scheduling, and energy-aware AI design. While our current work focuses on hypothetical consumer hardware environments from a design-stage labelling perspective, we believe that the core ideas, particularly the black-box profiling approach and consumption-based software categorization, could be extended to cloud environments, where virtualization and dynamic scheduling decisions significantly impact energy consumption. Similarly, integrating our methodology with process scheduling strategies or AI systems that optimize for energy efficiency could unlock promising avenues for future research. We believe that the advanced features of the BBCP (Larracoechea et al., 2024a), including its support for time-restricted and shifting stochastic behaviours, already provide a solid foundation for exploring such avenues. This opens the door for us and for others who wish to adapt our approach to extend the proposal toward domains such as cloud energy analytics, process scheduling, and energy-aware AI design.

6. Conclusions and future work

In this paper, we presented our expert algorithm for categorizing the hardware consumption, behavioural characteristics, and sensor usage of prospective (and existing) software applications and their features. In addition, we proposed a variant of the first algorithm that categorizes the intensity of the hardware consumption and behaviour that prospective software could exhibit. The novelty of our proposal, compared to existing approaches—such as measuring software power consumption through direct interaction with CPU-level APIs, power meters, or specialized testbeds—lies in the ability to generate a label at the design stage of the SDLC, before any software exists, without relying on mandatory tools or intricate processes, and tightly integrated with a DSL and our custom CASE tool, RADIANCE+. This tool offers a flexible framework that can be configured and updated according to the preferences of the user and emerging recommendations in this area. Our experimental tests of the algorithm and its variant show that the labels can be obtained by inferring the behaviour of a similar software and testing its consumption of hardware resources or obtaining test data from its hardware requirements, and cross-examine alternative hardware hosts to find the most applicable hardware hosts from an efficiency point of view that uses readily-available hardware specification metrics. We also found a positive correlation between the recorded system-wide consumption and final scores of the samples that we validated our proposal with. Finally, we do not want to understate that this proposal forms part of cohesive approach for software design. The intricacy of the algorithm proposed here and the data generated with it do not necessarily imply that the process to obtain it should be cumbersome. A guide for replicating the experiments in this paper with very simple steps is available in our supplementary material (Larracoechea et al., 2025), which includes the detail data of our experiments. Besides, a website that contains additional information and other resources related to our proposal of EnergyScore is available (University of Zaragoza & Pau, 2025).

As future work, we will explore how the algorithm can be adapted to consider the evolution of prospective hardware workloads, shifts between software behaviour and software features, and the change in usage patterns over time. In addition, we will explore how scheduling algorithms that take into account the labels produced with our proposal could produce "greener" software deployments. Finally, we will consolidate our proposal for a green software design method that fosters energy saving starting from, but not limited to, the design stage of software.

CRediT authorship contribution statement

Jorge Andrés Larracoechea: Conceptualization, methodology, software, validation, formal analysis, data curation, visualization, writing – original draft preparation.; **Philippe Roose:** Conceptualization, methodology, supervision, resources, funding acquisition, writing – review and editing.; **Sergio Ilarri:** Conceptualization, methodology, writing – review and editing, visualization, supervision, resources, project administration, funding acquisition..

Data availability

Our research data is available at the following Mendeley Data Page: 10.17632/bfxp7p422w.2

Declaration of competing interest

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

Acknowledgments

This publication is part of the project PID2020-113037RB-I00, funded by MICIU/AEI/10.13039/501100011033. Besides the NEAT-AMBIENCE project, we also thank the support of the Departamento de Ciencia, Universidad y Sociedad del Conocimiento del Gobierno de Aragón (Government of Aragon: Group Reference T64_23R, COSMOS research group).

Appendix A. Properties of the BBCP employed

Table A.1 presents the high-level properties considered by the algorithm when analyzing software behaviour and energy consumption (in the table, "Com. Cent." is a shorthand for "Computation Centric", "Dat. Cent." for "Data Centric", "Con. Cent." for "Conduct Centric", and "Comput." for "Computational"). These properties are categorized into three dimensions: computation-centric, data-centric, and conduct-centric, each highlighting different aspects of software operation. From distribution strategies and computational criticality to data handling and usage patterns, the table outlines key qualitative and quantitative traits that help determine the possible energy-impact and potential optimization strategies. The combination of tags, definitions, and value ranges offers a structured overview to guide profiling and decision-making.

Table A.1
High-level properties used by the algorithm.

Tag and property	Com. Cent.	Dat. Cent.	Con. Cent.	Definition	**Possible values**
A: Task distribution	✓	×	×	This qualitative property defines if the computation that software performs has to be always executed in the same computational entity or it can be distributed among several others. It also helps us to understand where energy is going to be consumed.	Centralized, Distributed
B: Distribution strategy	✓	×	×	This qualitative property defines if the user has a distribution strategy planned for distributed tasks.	Yes, No
C: Comput. criticality	✓	×	×	This qualitative property is meant to define if the computational results are tied to a time constraint. Knowing there is a time constraint, requiring the computationally fastest and physically closest host.	Low, Medium, High
D: Comput. complexity	✓	×	×	This qualitative and quantitative property establishes an amount of computation required to accomplish the goal of a software or software component.	Low, Medium, High
E: Data flow behaviour	×	✓	×	This qualitative property refers to the consistency of the data flow where consumption of data can be, or not, interrupted. It helps us to identify if intermittent energy management strategies can be applied.	Regular, Irregular
F: Data flow direction	×	✓	×	Complementary to the previous property, this qualitative property defines whether data flows from or to the software or from and to the software. It helps us to identify a relationship of energy consumption between software and its components, as well as other software units.	Unidirectional, Bidirectional
G: Data handling	×	✓	×	This qualitative property establishes what is done to data after the software consumes it. This helps us to assess the storage consumption of software.	Keep, Destroy, Store and broadcast
H: Access frequency	×	×	✓	This qualitative property is meant to define how often the software is used. It helps us to understand the predictability of its usage.	Regular, Irregular
I: Consumption rate	×	×	✓	This qualitative property complements the previous one (access frequency). It defines if the usage of the software is well defined or undefined; it can also be understood as "how long" whereas access frequency is "how often".	Definite, Indefinite
J: Depth	×	×	✓	This property defines an execution level for the software fragment you are profiling within the system the application is executed in. It also allows us to know if energy management strategies can be applied to the service without detriment to the user. In other words, this property allows us to determine if we can apply energy management strategies to the software you are currently profiling without affecting the user.	Foreground, Background
K: Dependence	×	×	✓	This property establishes whether the software profile is subject to any dependency relationship with other software or profiles.	Dependee (other entities depend on it), Dependant, Independent

A.1. BCS-equivalent values per BBCP property

Table A.2 details the reference values assigned to each possible value of the high-level properties identified earlier. These reference values, ranging from 0 to 1, represent the relative impact or relevance of a property value in terms of energy consumption or management complexity. Higher values indicate a greater potential influence. The table also outlines any pre-requisites for a given value to apply and provides justifications based on empirical evidence or real-world scenarios. This mapping enables the algorithm to translate qualitative characteristics into a quantifiable framework for energy-aware profiling.

Table A.2
Property score value assigned to the possible value of each property.

Tag and Property	Property value	Score (0–1)	Pre-requisite	Justification
A0 Task distribution	Centralized	0.5	None	We decided to leave the score value as a 0.5 because we are unaware if a strategy for producing green software will be employed or not.
A1 Task distribution	Distributed	1	None	We assigned a score value of 1 due to the complexity of managing a distributed architecture, as software units (such as microservices) are volatile in behaviour, availability, cost, and quality of service (Caporuscio et al., 2020). The aforementioned reasons could produce a computational management and energy consumption overhead of attempting to use such an architecture, especially without previous experience. Recently, Amazon cited some of the previous challenges as motivators for switching from a distributed microservices architecture to a monolith application (Tech, 2024).
B0 Distribution strategy	Yes	0	None	The score value of this property was set to 0 as its purpose is to provide a combinational score; a score created from the combination of 1 or more values.
B1 Distribution strategy	No	1	None	We punish the lack of distribution strategy as it could harm the energy consumption of hardware in multiple locations with no holistic vision of how it will be consumed.
C0 Computational criticality	Low	0.3	None	An elevated priority superposes a profile over others, with the possibility of it demanding more resources and, therefore, consuming more energy or impacting the possible quality of service of other software.
C1 Comput. criticality	Medium	0.6	None	
C2 Comput. criticality	High	1	None	
D0 Comput. complexity	Low	0.3	None	The score value increases with the increase in the demand of computational resources.
D1 Comput. complexity	Medium	0.6	None	
D2 Comput. complexity	High	1	None	
E0 Data flow behaviour	Regular	0.8	None	As there is a constant exchange of data, fewer opportunities to apply energy-saving strategies are present, furthermore, a constant data exchange means that there is a constant consumption of energy involved in transmitting and processing it. As we previously stated, the energy consumption of network hardware is one of the biggest power hogs, the score value is meant to reflect this.
E1 Data flow behaviour	Irregular	0.4	None	As the value of this property represents the opposite of the previous value, we reduced its score value without bringing it down to 0, as data exchange is still a relevant variable for categorizing consumption.
F0 Data flow direction	Unidirectional	0.5	None	We decided that the score value was appropriate as data is sent, not exchanged, creating a weak relationship of energy consumption among software entities.
F1 Data flow direction	Bidirectional	1	None	In contradiction, to the property value above, this property value establishes a strong relationship of energy consumption to maintain the communication among software entities. Hence the score value. Cloud-based applications such as Google Meet or GeForce Now are good examples of this, as a constant recollection of input from the user is sent to the cloud and a stream of data is constantly sent from the cloud to the user, creating a feedback loop.
G0 Data handling	Keep	0.5	Task distribution is set to distributed	Keeping incoming data and generated data produces hardware consumption. The score value was chosen to reflect the increasing energy consumption of cloud storage (Pesce, 2021), as writing to a local storage device is far more efficient (Magazine, 2017) and the energy consumption of updated network transmission infrastructures, such as 5G is still unknown (Kamiya, 2021).
G1 Data handling	Destroy	0	None	Destroying data after its consumption prevents the usage of storage.
G2 Data handling	Store and broadcast	0.5	Task distribution is set to distributed and distribution strategy is set to yes	Storing data to become a provider of it, coupled with a distribution strategy, can benefit the energy consumption by becoming the closest source of data to other devices. A good example of this is CDNs (Content-Delivery Networks). Broadly speaking, CDNs allow frequently accessed data to be replicated in geographically distributed servers responsible for delivering it to the closest users. Even though a similar strategy could be replicated using devices as data nodes, a lot of variables involved in keeping the data up to date could negatively or positively impact energy consumption (Bianco et al., 2017).
H0 Access frequency	Regular	0	None	A regular task frequency aids in predicting when strategies and deployments could be employed.
H1 Access frequency	Irregular	0.5	None	An unpredictable frequency means that software could be consumed at any point in time, making usage chaotic.
I0 Consumption rate	Definite	0	None	We can predict exactly for how long software will be used when it is solicited.
I1 Consumption rate	Indefinite	0.1	None	We cannot predict exactly for how long software will be used when solicited.

Table A.2
Continued.

J0 Depth	Foreground	0.5	Task distribution must be set to centralized	The display is needed to use this software and it cannot be delegated.
J1 Depth	Background	0	Task distribution can be either centralized or distributed	The display is not employed.
K0 Dependence	Dependee	0.5	None	Other software depends on the software being profiled, establishing an energy consumption relationship elsewhere.
K1 Dependence	Dependant	0.5	None	
K2 Dependence	Independent	0	None	No energy relationship with other software is present.

Table A.3
Possible combinations of the properties used by the BCS and their combinational score values.

Tag	Combination	Score	Justification	Logical effects
C1	$A1 \wedge B1$	3	If the task distribution is distributed, we penalize not having a distribution strategy, as network consumption to distribute the tasks would produce the risk of overloading other devices.	Sets the score of A1 and B1 to 0
C2	$A1 \wedge B0$	−1	Applying a proven deployment or distribution strategy should allow for further energy savings. With this in mind we can possibly affect the energy score (reduce the score) to support the user's intentions of producing green software.	Sets the score of A1 to 0
C3	$C2 \wedge E0$	2	Contrary to the previous combination, a regular flow of data means that, now that software components are distributed in a network more network usage (among devices) is required. Therefore, energy consumption is unpredictable.	The scores of A1 and E0 are set to 0 C2 is invalidated (not applicable)
C4	$C2 \wedge E1 \wedge G2$	−2	A decentralized task distribution in addition to a strategy for distributing tasks and a strategy for broadcasting data could benefit (lower) the energy consumption by obtaining the most frugal execution and creating an easier to reach data node.	The scores of A1, E1 and G2 are set to 0 C2 is invalidated (not applicable)
C5	$C2 \wedge E1 \wedge G1$	−1	A decentralized task distribution in addition to a strategy for distributing tasks could benefit (lower) the energy consumption by obtaining the most frugal execution. A lack of data distribution strategy is not as beneficial.	The scores A1, E1 and G1 are set to 0 C2 is invalidated (not applicable)

A.2. BCS-equivalent values of combined BBCP properties

Table A.3 presents selected combinations of high-level property values used in the BCS (behavioural Consumption Score) and their associated combinational reference values. These combinations capture interactions between properties that, when occurring together, significantly influence the overall energy profile–either positively or negatively. Each combination is assigned a reference value and accompanied by a justification that explains its impact on energy consumption, along with the logical effects on individual property scores. This mechanism allows the scoring model to account for interdependencies between properties and supports more nuancedy energy-aware assessments.

Supplementary material

Supplementary material associated with this article can be found in the online version at 10.1016/j.eswa.2025.130046

References

Angel, B. (2023). Blue angel – good for me. good for the environment. https://web.archive.org/web/20240712124923/https://www.blauer-engel.de/en/blue-angel/our-label-environment. Archived on 12-07-2024.

Behrouz, R. J., Sadeghi, A., Garcia, J., Malek, S., & Ammann, P. (2015). EcoDroid: An approach for energy-based ranking of android apps. In *2015 IEEE/ACM 4th international workshop on green and sustainable software* (pp. 8–14). https://doi.org/10.1109/GREENS.2015.9

Bianco, A., Mashayekhi, R., & Meo, M. (2017). On the energy consumption computation in content delivery networks. *Sustainable Computing: Informatics and Systems*, *16*. https://doi.org/10.1016/j.suscom.2017.08.008

Bordage, F. (2024). EcoIndex | how it works. https://web.archive.org/web/20240705110042/https://www.ecoindex.fr/en/how-it-works/. Archived on 05-07-2024.

buildcomputers.net (2023). Typical power consumption of PC components - power draw in watts. https://web.archive.org/web/20240626225144/https://www.buildcomputers.net/power-consumption-of-pc-components.html. Archived on 12-07-2024.

Calero, C., & Piattini, M. (Eds.) (2015). Green in software engineering. Springer International Publishing. https://doi.org/10.1007/978-3-319-08581-4

Caporuscio, M., D'Angelo, M., Grassi, V., & Mirandola, R. (2020). Decentralized architecture for energy-aware service assembly. In A. Jansen, I. Malavolta, H. Muccini, I. Ozkaya, & O. Zimmermann (Eds.), *Software architecture* Lecture Notes in Computer Science (pp. 57–72). Springer International Publishing. https://doi.org/10.1007/978-3-030-58923-3_4

Carroll, A., Heiser, G. et al. (2010). An analysis of power consumption in a smartphone. In *Usenix annual technical conference* (pp. 21). Boston, MA (*vol. 14*). https://doi.org/10.5555/1855840.1855861

Ceci, L. (2023). U.S. users who have a discord account by age 2023. https://web.archive.org/web/20240712124442/https://www.statista.com/statistics/1338865/us-users-having-discord-account-by-age/. Archived on 12-07-2024.

Chen, X., Chen, Y., Ma, Z., & Fernandes, F. C. A. (2013). How is energy consumed in smartphone display applications? In *Proceedings of the 14th workshop on mobile computing systems and applications* HotMobile '13 (pp. 1–6). Association for Computing Machinery. https://doi.org/10.1145/2444776.2444781

Chiaravalloti, S., Idzikowski, F., & Budzisz, Ł., et al. (2011). Power consumption of WLAN network elements. https://doi.org/10.13140/2.1.4424.8005

Code, R. (2025). Rosetta code. https://rosettacode.org/wiki/Rosetta_Code.

Comission, E. (2013). Commission regulation (EU) No 617/2013 of 26 June 2013 implementing directive 2009/125/EC of the European parliament and of the council with regard to ecodesign requirements for computers and computer servers text with EEA relevance. http://data.europa.eu/eli/reg/2013/617/oj/eng. Legislative Body: COM.

Comission, E. (2017). Methodology for ecodesign of energy-related products. https://web.archive.org/web/20240712130827/https://ec.europa.eu/docsroom/documents/26525. Archived on 12-07-2024.

Comission, E. (2021a). Commission regulation (EU) 2021/341 of 23 February 2021 amending regulations (EU) 2019/424, (EU) 2019/1781, (EU) 2019/2019, (EU) 2019/2020, (EU) 2019/2021, (EU) 2019/2022, (EU) 2019/2023 and (EU) 2019/2024 with regard to ecodesign requirements for servers and data storage products, electric motors and variable speed drives, refrigerating appliances, light sources and separate control gears, electronic displays, household dishwashers, household washing machines and household washer-dryers and refrigerating appliances with a direct sales function (text with EEA relevance). http://data.europa.eu/eli/reg/2021/341/oj/eng. Legislative Body: ENER, COM.

Comission, E. (2021b). Growth without economic growth – European environment agency. https://web.archive.org/web/20240712130328/https://www.eea.europa.eu/publications/growth-without-economic-growth. Archived on 12-07-2024.

Comission, E. (2021c). In focus: The improved EU energy label – paving way for more innovative and energy efficient products. https://web.archive.org/web/20240712131202/https://commission.europa.eu/news/focus-improved-eu-energy-label-paving-way-more-innovative-and-energy-efficient-products-2021-02-16_en. Archived on 12-07-2024.

Comission, E. (2024a). About the energy label and ecodesign. https://web.archive.org/web/20240712131025/https://energy-efficient-products.ec.europa.eu/ecodesign-and-energy-label_en. Archived on 12-07-2024.

Comission, E. (2024b). Energy labelling requirements for computers and computers servers - feedback. https://ec.europa.eu/info/law/better-regulation/have-your-say/initiatives/1580-Energy-labelling-requirements-for-computers-and-computer-servers/feedback_en?p_id=170283. accessed 12-07-2024.

Comission, E. (2024c). Energy labelling requirements for computers and computers servers - initiative. https://ec.europa.eu/info/law/better-regulation/have-your-say/initiatives/1580-Energy-labelling-requirements-for-computers_en. accessed on 12-07-2024.

Deneckère, R., & Rubio, G. (2020). EcoSoft: Proposition of an eco-label for software sustainability. In *Advanced information systems engineering workshops* (pp. 121–132). Springer. https://doi.org/10.1007/978-3-030-49165-9_11

Design, S. W. (2024). Estimating digital emissions. https://sustainablewebdesign.org/estimating-digital-emissions/. accessed on 24-07-2024.

Discord (2024). Discord - group chat that's all fun & games. https://discord.com. accessed on 23-07-2024.

Discord Blog (2023). Why discord is sticking with react native. https://web.archive.org/web/20230611090340/https://discord.com/blog/why-discord-is-sticking-with-react-native.

Fernández-Montes, A., Gonzalez-Abril, L., Ortega, J. A., & Lefèvre, L., et al. (2012). Smart scheduling for saving energy in grid computing. *Expert Systems with Applications*, *39* (10), 9443–9450. https://doi.org/10.1016/j.eswa.2012.02.115

Forward, A., & Lethbridge, T. C. (2008). A taxonomy of software types to facilitate search and evidence-based software engineering. In *Proceedings of the 2008 conference of the center for advanced studies on collaborative research: Meeting of minds* CASCON '08 (pp. 179–191). Association for Computing Machinery. https://doi.org/10.1145/1463788.1463807

Foundation, B. (2024). Blender.org - home of the blender project - free and open 3D creation software. https://www.blender.org/. accessed on 23-07-2024.

Globemallow (2024). Homepage. https://web.archive.org/web/20240705125601/https://globemallow.io/. Archived on 05-07-2024.

Guamán, D., Pérez, J., & Valdiviezo-Diaz, P., et al. (2023). Estimating the energy consumption of model-view-controller applications. *The Journal of Supercomputing*, *79* (12), 13766–13793. https://doi.org/10.1007/s11227-023-05202-6

Guldner, A., Bender, R., Calero, C., Fernando, G. S., Funke, M., Gröger, J., Hilty, L. M., Hörnschemeyer, J., Hoffmann, G.-D., Junger, D., Kennes, T., Kreten, S., Lago, P., Mai, F., Malavolta, I., Murach, J., Obergöker, K., Schmidt, B., Tarara, A., De Veaugh-Geiss, J. P., Weber, S., Westing, M., Wohlgemuth, V., & Naumann, S. (2024). Development and evaluation of a reference measurement model for assessing the resource and energy efficiency of software products and components–green software measurement model (GSMM). *Future Generation Computer Systems*, *155*, 402–418. https://doi.org/10.1016/j.future.2024.01.033

Henderson, P., Hu, J., Romoff, J., Brunskill, E., Jurafsky, D., & Pineau, J. (2022). Towards the systematic reporting of the energy and carbon footprints of machine learning. https://doi.org/10.48550/arXiv.2002.05651

Hindle, A. (2016). Green software engineering: The curse of methodology. In *2016 IEEE 23rd international conference on software analysis, evolution, and reengineering (SANER)* (pp. 46–55). (*vol. 5*). https://doi.org/10.1109/SANER.2016.60

Junger, D., Westing, M., Freitag, C. P., Guldner, A., Mittelbach, K., Obergöker, K., Weber, S., Naumann, S., & Wohlgemuth, V. (2024). Potentials of green coding – findings and recommendations for industry, education and science – extended paper. https://arxiv.org/abs/2402.18227.

Kamiya, G. (2021). Data centres and data transmission networks – analysis. https://web.archive.org/web/20240712133554/https://www.iea.org/reports/data-centres-and-data-transmission-networks. Archived on 12-07-2024.

Kantar (2019). Europeans attitudes on EU energy policy - September 2019 – Eurobarometer survey. https://europa.eu/eurobarometer/surveys/detail/2238. accessed on 12-07-2024.

Khan, I., Khusro, S., Ali, S., & Ahmad, J., et al. (2016). Sensors are power hungry: An investigation of smartphone sensors impact on battery power from lifelogging perspective. *Bahria University Journal of ICT*, *9*, 8–19.

Laptop Media (2025). [Q4 2024] global ranking of top 10 best-selling laptops on Amazon - HP dominates with the #1 spot, Apple leads in revenue | LaptopMedia.com. https://laptopmedia.com/highlights/q4-2024-global-ranking-of-top-10-best-selling-laptops-on-amazon-hp-dominates-with-the-1-spot-apple-leads-in-revenue/.

Larracoechea, J., Roose, P., Ilarri, S., Cardinale, Y., Laborie, S., & González, M. J. (2021). Towards services profiling for energy management in service-oriented architectures. In *17th international conference on web information systems and technologies (WEBIST)* (pp. 209–216). Cluj Napoca, Romania. https://doi.org/10.5220/0010718600003058

Larracoechea, J., Roose, P., Ilarri, S., Cardinale, Y., Laborie, S., & Vara, O. (2022). Behavior-based consumption profiles for the approximation of the energy consumption of services. *International Conference on Information Systems Development (ISD)*, . https://doi.org/10.62036/ISD.2022.6

Larracoechea, J. A., Ilarri, S., & Roose, P., et al. (2024a). A proposal of behavior-based consumption profiles for green software design. *Applied Sciences*, *14* (17), 7456. Number: 17 Publisher: Multidisciplinary Digital Publishing Institute. https://doi.org/10.3390/app14177456

Larracoechea, J. A., Ilarri, S., & Roose, P. (2024b). RADIANCE: A case tool for green software design. In *2024 32nd International conference on enabling technologies: Infrastructure for collaborative enterprises (WETICE)* (pp. 86–91). https://doi.org/10.1109/WETICE64632.2024.00024

Larracoechea, J.-A., Roose, P., & Ilarri, S. (2025). Energy-aware software design: Experimental setup, data, and results (v2). https://doi.org/10.17632/bfxp7p422w.2.

Le Goaer, O., & Hertout, J. (2022). EcoCode: A SonarQube plugin to remove energy smells from android projects. In *Proceedings of the 37th IEEE/ACM International conference on automated software engineering* (pp. 1–4). ACM. https://doi.org/10.1145/3551349.3559518

Li10 (2024). Li10 – CO2 monitor. https://web.archive.org/web/20240705131753/https://www.li10.com/. Archived on 05-07-2024.

Magazine, S. (2017). Carbon and the cloud. https://web.archive.org/web/20240712133009/https://stanfordmag.org/contents/carbon-and-the-cloud. Archived on 12-07-2024.

Mightybytes (2024). Ecograder | how it works. https://web.archive.org/web/20240705110525/https://ecograder.com/how-it-works. Archived on 05-07-2024.

Mills, E., Bourassa, N., Rainer, L., Mai, J., Shehabi, A., & Mills, N., et al. (2019). Toward greener gaming: estimating national energy use and energy efficiency potential. *The Computer Games Journal*, *8*. https://doi.org/10.1007/s40869-019-00084-2

Naumann, S., Dick, M., Kern, E., & Johann, T. (2011). The GREENSOFT model: A reference model for green and sustainable software and its engineering. *Sustainable Computing: Informatics and Systems*, *1* (4), 294–304. https://doi.org/10.1016/j.suscom.2011.06.004

Naumann, S., Guldner, A., & Kern, E. (2021). The eco-label blue angel for software—development and components. In A. Kamilaris, V. Wohlgemuth, K. Karatzas, & I. N. Athanasiadis (Eds.), *Advances and new trends in environmental informatics* Progress in IS (pp. 79–89). Springer International Publishing. https://doi.org/10.1007/978-3-030-61969-5_6

Oliner, A. J., Iyer, A. P., Stoica, I., Lagerspetz, E., & Tarkoma, S. (2013). Carat: Collaborative energy diagnosis for mobile devices. In *Proceedings of the 11th ACM conference on embedded networked sensor systems, (SenSys)* SenSys '13 (pp. 1–14). Roma, Italy: Association for Computing Machinery. https://doi.org/10.1145/2517351.2517354

Pereira, R., Couto, M., Ribeiro, F., Rua, R., Cunha, J., Fernandes, J. P., & Saraiva, J. (2021). Ranking programming languages by energy efficiency. *Science of Computer Programming*, *205*, 102609. https://doi.org/10.1016/j.scico.2021.102609

Pesce, M. (2021). Cloud computing's coming energy crisis - IEEE spectrum. https://web.archive.org/web/20240712133340/https://spectrum.ieee.org/cloud-computings-coming-energy-crisis. Archived on 12-07-2024.

Spécinov (2024). Kastor | learn more. https://web.archive.org/web/20240705131113/https://kastor.green/learn-more. Archived on 05-07-2024.

Star, E. (2024). How to go green with your code | energy star. https://www.energystar.gov/products/ask-the-experts/how-go-green-your-code. accessed on 26-07-2024.

Sun, Z., Huang, H., Li, Z., Gu, C., Xie, R., & Qian, B. (2023). Efficient, economical and energy-saving multi-workflow scheduling in hybrid cloud. *Expert Systems with Applications*, *228*, 120401. https://doi.org/10.1016/j.eswa.2023.120401

Tech, P. V. (2024). Scaling up the Prime Video audio/video monitoring service and reducing costs by 90%. https://web.archive.org/web/20240712132516/https://www.primevideotech.com/video-streaming/scaling-up-the-prime-video-audio-video-monitoring-service-and-reducing-costs-by-90. Archived on 12-07-2024.

Universidad de Zaragoza, University of Pau, (2025). Energyscore website. Part of the NEAT-AMBIENCE project http://webdiis.unizar.es/~silarri/NEAT-AMBIENCE,PID2020-113037RB-I00project, led by Sergio Ilarri and funded by MICIU/AEI/10.13039/501100011033. Last access: October 3, 2025.

van Kempen, N., Kwon, H.-J., Nguyen, D. T., & Berger, E. D. (2025). It's not easy being green: On the energy efficiency of programming languages. https://arxiv.org/abs/2410.05460.

Wilke, C., Richly, S., Püschel, G., Piechnick, C., Götz, S., & Aßmann, U. (2012). Energy labels for mobile applications. In *Informatik 2012* (pp. 412–426). Gesellschaft für Informatik e.V.