Computing (2018) 100:1155–1173
https://doi.org/10.1007/s00607-018-0632-7

Finding correlations of features affecting energy consumption and performance of web servers using the HADAS eco-assistant

Daniel-Jesus Munoz[1] · **Mónica Pinto**[1] · **Lidia Fuentes**[1]

Received: 29 January 2018 / Accepted: 1 June 2018 / Published online: 18 June 2018

Abstract The impact of energy consumption on the environment and the economy is raising awareness of "green" software engineering. HADAS is an eco-assistant that makes developers aware of the influence of their designs and implementations on the energy consumption and performance of the final product. In this paper, we extend HADAS to better support the requirements of users: *researchers*, automatically dumping the energy-consumption of different software solutions; and *developers*, who want to perform a sustainability analysis of different software solutions. This analysis has been extended by adding Pearson's chi-squared differentials and Bootstrapping statistics, to automatically check the significance of correlations of the energy consumption, or the execution time, with any other variable (e.g., the number of users) that can influence the selection of a particular eco-efficient configuration. We have evaluated our approach by performing a sustainability analysis of the most common web servers (i.e. PHP servers) using the time and energy data measured with the *Watts Up? Pro* tool previously dumped in HADAS. We show how HADAS helps web server providers to make a trade-off between energy consumption and execution time, allowing them to sell different server configurations with different costs without modifying the hardware.

✉ Daniel-Jesus Munoz
danimg@lcc.uma.es

Mónica Pinto
pinto@lcc.uma.es

Lidia Fuentes
lff@lcc.uma.es

[1] Andalucía Tech, Universidad de Málaga, Málaga, Spain

Keywords Energy efficiency · Performance · Web servers · Linux

Mathematics Subject Classification 68U35 · 68N30 · 68M20 · 97K80

1 Introduction

Nowadays, most of the actions in a smartphone involve an interaction with *cloud services*. Each service that companies like *Google* provide is connected to web servers (e.g., data centres), which consume certain amounts of electricity to power and cool them, affecting the increasing rate of carbon dioxide emissions. Right now, about 7% of the world's electricity consumption is due to the web services ecosystem, but this is forecast to rise to 12% by 2020, and it is expected to grow annually at a rate of approximately 7% through to 2030 [5]. Electricity costs money; data centres and hosting services therefore need proper insights into reducing costs. Although software systems do not directly consume energy, they affect the energy consumption of the hardware. (e.g., micro-processor, memory). As a result, a new discipline (Green Computing) has recently emerged, which aims to promote energy-aware designs and implementations to ensure that the resulting software makes efficient use of the hardware resources. Therefore, the developer should not only consider the modularization, performance or maintainability of the software, but must also take into account energy savings [12,33]

The majority of the studies on software energy efficiency are empirical studies that focus on those parts of the code that consume most energy ('energy-consumers' or hotspots [8,22,23]), or on the energy consumption of several alternatives [2,7,15,20, 32]. We have found studies that compare energy consumption measured at runtime; networking of Android devices [21], of data centres [3], or of Java Collections [15]; other studies simulate rather than measure the energy consumption of applications, such as Android applications [14] with eCalc, Java Enterprise Edition (EE) applications [4], or audio compression [20,33] with Palladio PCA. A common conclusion is that *It is worthwhile addressing the optimisation of energy consumption at the code level*, reaching a 70% reduction. However, the specific results are not easily transferable to industrial development processes. For example, in [15] it is demonstrated that the most popular data collections (e.g.: ArrayList, HashSet, Map, etc.), implemented using different *frameworks*, differ in their energy consumption. The authors posted comma-separated values (CSV) files with the experimental results (metrics and graphs). It is hard, for developers, to understand and make use of the proposed insights simply by observing the raw results. A solution to this shortcoming would be to store the results of all these experimental studies in a common energy-related repository with sustainability analysis support, so that the information could be consulted and reused by software developers. An example of this kind of repository is our work, the HADAS eco-assistant [26,27].

The HADAS eco-assistant[1] has been developed using a Software Product Line (SPL) approach [19,30]. An SPL is a software engineering method that focuses on identifying and modelling the commonalities and variabilities of a software product,

[1] https://hadas.caosd.lcc.uma.es/.

generating a variability model. Using the SPL approach, HADAS formally models the variability of those software concerns that most influence energy consumption (i.e. the *energy consumers*, e.g., security), in conjunction with its design alternatives (e.g., using the AES encryption algorithm) and implementation alternatives (e.g., Apache Commons Crypto library). Then, the variability model is used to generate partial or total configurations satisfying the constraints imposed by the model and the HADAS users. Thus, the objective of HADAS is to help developers to analyse different alternatives and aid them in selecting the most suitable configuration. The usefulness of the assistant is twofold. On the one hand, energy researchers can store their experimental results in a repository that is globally available as a web service. On the other hand, developers can use it to reason about different energy consumer configurations, selecting the solution that satisfies response times and energy demands.

The HADAS reasoning capability is based on exploiting the variability model [16]. The objective of the assistant is not to give concrete values of energy consumption, since these heavily rely on the hardware, but rather to provide a visualisation of how the energy consumption of different solutions varies (e.g., graphs). Therefore, it is intended that developers identify consumer trends and correlations, instead of numbers (e.g., Joules). In fact, none of the energy measurements performed with both, software (e.g., PowertTop, IPPET) and hardware tools (e.g., Watts Up? Pro, Multimeter), can be taken as absolute values since the actual consumption of a running application depends on a multitude of factors that cannot be controlled (e.g., devices temperature, load, compiler, garbage collector). Other repositories exist like the one provided by the mining tool GreenMiner [17], which simply aim to store energy consumption experimental results obtained with the miner. Concretely, GreenMiner just deals with the execution and storage of concrete tests and metrics, and it is beyond its scope to advise developers or analyse data.

In this paper, we illustrate how *researchers* can automatically dump energy-consumption data, and *developers* can perform a sustainability analysis, of different software solutions. We have enriched the HADAS sustainability analysis adding Pearson's $\tilde{\chi}^2$ differentials [31] and Bootstrapping statistics [11], allowing the significance of the correlation of the energy consumption or the execution time to be checked with any variable feature of any complete configuration (i.e., a case study). Then, HADAS can mathematically identify the variables that have higher time and energy consumption effects, considering the constraints of the case studies being analysed. Additionally, developers will be able to know which elements can be changed without affecting the end consumption or runtime of the system (e.g., programming languages). In parallel, we extend previous experiments studying the consumption of common web servers including Linux Operating Systems (OS) and other PHP/HTML benchmarks, as they are the most widely used configurations for web services and data centres. The new measurements are done physically with the tool *Watts Up? Pro*, the data is dumped to the HADAS repository and then analysed with the service (web interface) provided by the HADAS eco-assistant.

The paper is organised as follows. Section 2 discusses the related work. Section 3 describes de main contributions and provides an overview of the HADAS eco-assistant. Section 4 details the usability and an extension of HADAS for mathematical analysis. In Sect. 5 common web servers are measured and analysed with the HADAS eco-

assistant, discussing the results and consequences. Section 6 highlights conclusions and future work.

2 Related work

There are articles that highlight the importance of having a repository with software energy metrics such as Selflab [10], the future work of which contains the deployment of a repository that offers *greener* configurations to developers. The most cited is GreenMiner [17], nowadays called GreenOracle [6]. GreenMiner is a complete solution, as it provides power metering tools, as well as a repository that stores the collected data, where data mining techniques could be applied. GreenMiner research discusses the difficulty of energy-data processing due to the lack of normalisation of the extracted data. There is also no basic format in which matching the results of all the experiments can be done. In addition, the repository is local (deployed individually) to the measurement system. Therefore, we propose a collaborative approach for HADAS, where the results of the work done by different researchers can be added.

A similar approach is followed in [8], highlighting the usefulness of a hypothetical cloud repository that helps identify software design, architecture, and components that are responsible for energy leakage. The authors discuss the difficulty of performing energy-data analysis using a specific case study, for which, they define an Energy Modeller that allows them to restrict the search results of the repository. In [22] an Internet of Things (IoT) components repository is proposed instead of a generic repository. It specifies a repository to search for components that consume less, providing a required and supplied interface as specified in the architecture of an IoT service. This approach is not generic and does not include measurements of specific implementations as proposed by HADAS, and the reasoning is done just at the architectural level, so the usefulness is notably limited.

Regarding sustainability analyses, there are several publications. In the case study of Java collections [15], measurements are taken (GreenMiner), and different implementations with different frameworks are compared, concluding that the ideal configuration depends on the energy context (e.g., type and size of the data, and the operation). In [25] the authors study how different Android cryptographic providers, algorithms and operations behave from an energy consumption point of view. In [19] the authors apply an SPL approach to generate energy and time-efficient product configurations of reusable implementations. A MySQL database study is conducted in [24], concluding that globally available metrics are needed to assess products' greenness and performance, as the general knowledge does not always fit the reality. In [2] the energy consumption of different technologies used in web services (e.g., Apache, PostgreSQL and PHP) is compared, based mainly on measuring the length of time these web service components are in use. It is verified that, of all network protocols, HTTP/2 is the most client-side energy efficient, given the reduction of requests to the web server, regardless of the web client. However, it is also interesting to evaluate the server-side protocol as done in [32]. The paper compares several protocols and web servers, concluding that HTTP/2 consumes the least, especially if used in conjunction with the H2O web server. Due to the commonality of the PHP language

in web servers, in [29] the different consumption rates between single and double quotes is evaluated, with single quotes being found to be more efficient on almost all scenarios. As mathematical procedures are important in sustainability analyses, a few web server combinations are measured in [28], with the aim of predicting CPU consumption with a machine learning approach with linear models. The results lack some accuracy, but they do find the need for an SPL service to store and analyse data, as local measurements of every possible configuration is an almost impossible task. A similar study is carried out in [34], with the measurements taken physically with *Watts Up? Pro*, and estimated with PowerTop and virtual machines. The study's conclusion is the same. Although all these studies comparing different implementations extract interesting conclusions, this knowledge is not stored anywhere. If the developers want to apply new insights, they have to search for, find, read and comprehend all these papers. HADAS plans to bridge the gap between researchers, who produce energy-related data, and developers, who want to reuse the knowledge and have an analysis tool of experimental data. The HADAS eco-assistant is based on: (i) the creation of a global and collaborative repository of energy consumers; (ii) the metrics, graphs and mathematical systems that supports the analysis of energy consumption of assorted alternatives.

3 The HADAS eco-assistant overview

In this section we present an overview of the features of our HADAS approach.

3.1 HADAS main contributions

The main contributions of the HADAS eco-assistant are:

- C1. *Identification of energy consuming concerns (ECC)* HADAS helps identify the elements that have a greater impact on energy consumption (i.e., the energy consumers). HADAS focuses on the most common and recurrent ones (e.g., security, communication).
- C2. *Formal representation of the variability of energy models* HADAS formally models the variability of different design and technical alternatives, alongside the identified constraints (e.g., the selection of a Nginx web server implies using FastCGI), and dependencies between concerns.
- C3. *Store of records of energy consumption* HADAS stores energy consumption data of each complete configuration in a globally accessible service, considering the parameters that affect these configurations (e.g., file size affects the communication framework used for data transfer). It is not viable if the developers of HADAS have to be responsible for measuring and storing all possible configurations of every consumer that exists in an application domain. It has to follow a collaborative and incremental approach, where the repository is ready to include the design/implementation technologies together with the energy measurements provided by professionals, researchers, and even extracted from already published studies.

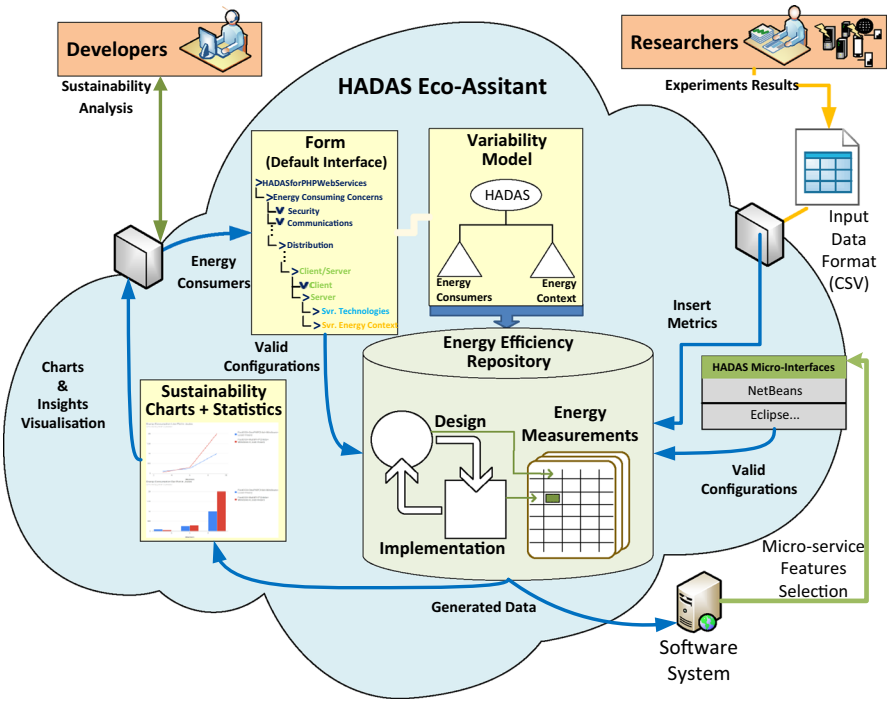

Fig. 1 HADAS eco-assistant overview

– C4. *Support for eco-efficiency analyses* Developers can select an energy consumer and, then, select/unselect the alternatives that implement it (model constraints). HADAS provides comparative graphs with energy and time trends of all the possible solutions that conform to the constraints. Then, developers can visualise energy and time consumption trends, and opt for the solution that suits their specific trade-off.

Figure 1 shows the main elements of the HADAS approach. Developers can access the ECCs contained in the assistant to identify those that may affect the energy consumption of their applications (C1). HADAS models these *Energy Consumers* and their *Energy Context* using a *Variability Model*. Additionally, it provides a user-friendly visualisation of the same information (the *Form*). The form-based default interface represents the details of the variability model in a tree-view that facilitates user navigation and selection; while the *Variability Model* is a formal representation (C2) of the consumers and their *Energy Context*. This representation allows reasoning and automatic generation of a customised partial or complete *configuration*, based on the user-form selections (user constraints). These models are stored in the *Energy Efficiency Repository*. Once a *configuration* has been constrained by the *Form* and the solver procedure, we have complete configurations, which are linked to their measurements in the repository. The energy measurements are previously stored in the repository by the researchers that have performed the experimental studies (C3). Then, the *Sustainability Charts* (energy and time graphs) of the generated configurations are

plotted (C4). Although the basics of HADAS are explained here, a deeper analysis of the design, structure and applications of its first prototype can be found in [27].

3.2 HADAS variability model and repository

The structure of the *Energy Efficiency Repository* is given by the variability model, which, in our case, is a representation of the variability tree shown on the right-hand side of Fig. 2 in a manageable format for the eco-assistant, which is Clafer [1]. With Clafer, variability models can be easily transformed into a constraint satisfaction problem [1] to automatically generate and reason about valid configurations, fulfilling model and user constraints. As seen on the left-hand side of Fig. 2, the HADAS variability tree has three layers: (1) a first layer with energy consumers identified not only by us, but in different experimental work (e.g., security, communication, distribution); (2) each concern contains a sub-tree with the available design options (e.g., for the distribution concern we have the client/server architecture). In the terminology of variability models, these are optional features and only one of them can be selected in a complete configuration (see the `1..n` legend in Fig. 2). Depending on the complexity of the energy consumer there will be one or more sub-layers; (3) for each leaf of the design sub-tree (e.g., Design 2.1 in Fig. 2) two sub-trees are defined (*Technologies* and *Energy Context*). These are mandatory features in our variability model and will always be present in any configuration. *Technologies* defines different implementation alternatives for each design. Here we can add a leaf or a sub-tree, modelling different frameworks, APIs, etc. Thus, HADAS can provide developers with the energy consumption of a concrete design implemented with different technologies. The *Energy Context* contains three optional child features: (1) *Parameters*: the energy consumption of an implementation varies in accordance with some parameters, for example the compression of a file will depend on the size, therefore, the developer could indicate a maximum size; (2) *Data type*: energy consumers typically manipulate data of different types (e.g., plain text, XML); (3) *Operations*: each design offers a set of operations that can be used within the application (e.g., for data collections: insert, delete, etc.). As different parameters are possible for the same energy-consuming concern, the variability model provides the possibility of selecting/defining `1..*` optional features/numeric variables. Neither, the operations nor the data type, are always implemented in each technology, and this is specified in the variability model with cross-tree constraints like `Technology i implies NOT Op j`. HADAS provides a web representation of the variability model, with a depth-first tree-folding-transversal scroll. The form applies the dependencies dynamically, namely the branches automatically consider the constraints if the constraints are activated. However, the variability tree does not contain the metrics, which are stored in a database. Therefore, we link the tree to a database which, together, conform the repository. The structure of the database also corresponds to the variability model structure, where each layer has its referent in a specific table (tables Reference, Design, Technology, DataType, Operation, Variable). In addition, the Metric table will relate these tables with energy consumption and execution time. HADAS allows the inclusion of energy data collected from various sources for the same configuration to ensure a richer analysis. This is achieved by

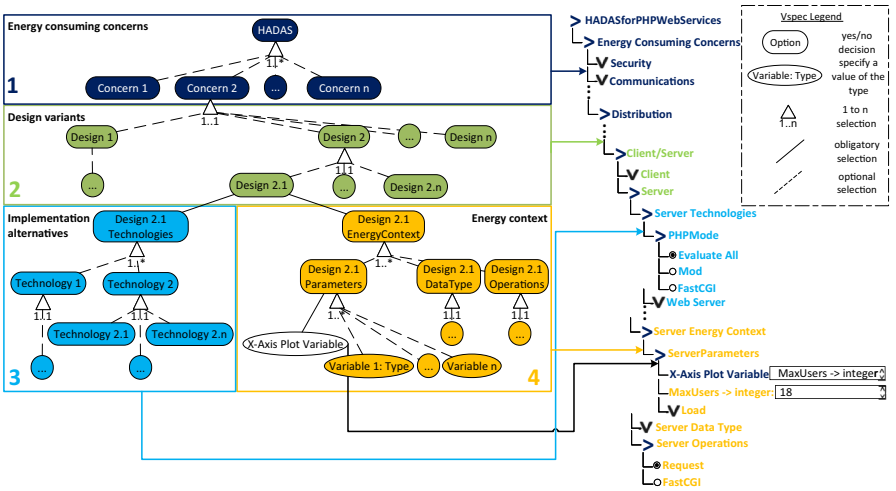

Fig. 2 HADAS tree variability structure layers and its form (real printed screen)

including the Meta-data table that contains additional information such as the measurement system and the hardware characteristics. The different measurements for the same configuration are properly labelled and distinguished in the graphs, so the inherent bias error due to the measurement system and context is not mixed.

4 HADAS usage

In this paper we extend HADAS to improve its usability. Concretely: (1) reducing the effort required to integrate and consult the data in the repository, and (2) improving the sustainability analysis. Two different roles of HADAS users are clearly distinguishable: *Developers* and *Researchers*.

4.1 HADAS for software researchers

How can researchers use HADAS? The researcher produces experimental data for each variant of an energy consumer and wants to integrate it into HADAS. Basically, as shown in Fig. 1, the researcher represents the obtained data in the variability model (if the model does not already contemplate those configurations) and dumps the metrics to the repository. As an extension to HADAS in this paper, this process has been semi-automatised for certain, defined formats, as can be seen in Fig. 3. Energy researchers who wish to insert the results of their experiments into HADAS need to provide both the ECC energy model and the experiment data. The format of the Energy Model must fit with the variability structure and layers of HADAS (Fig. 2) and the experiment data must fit with the HADAS input data format. Thus, the first step is to use a variability language to model the variability of the energy model, including the design and implementation solutions previously tested by the researcher (step 1 in Fig. 3).

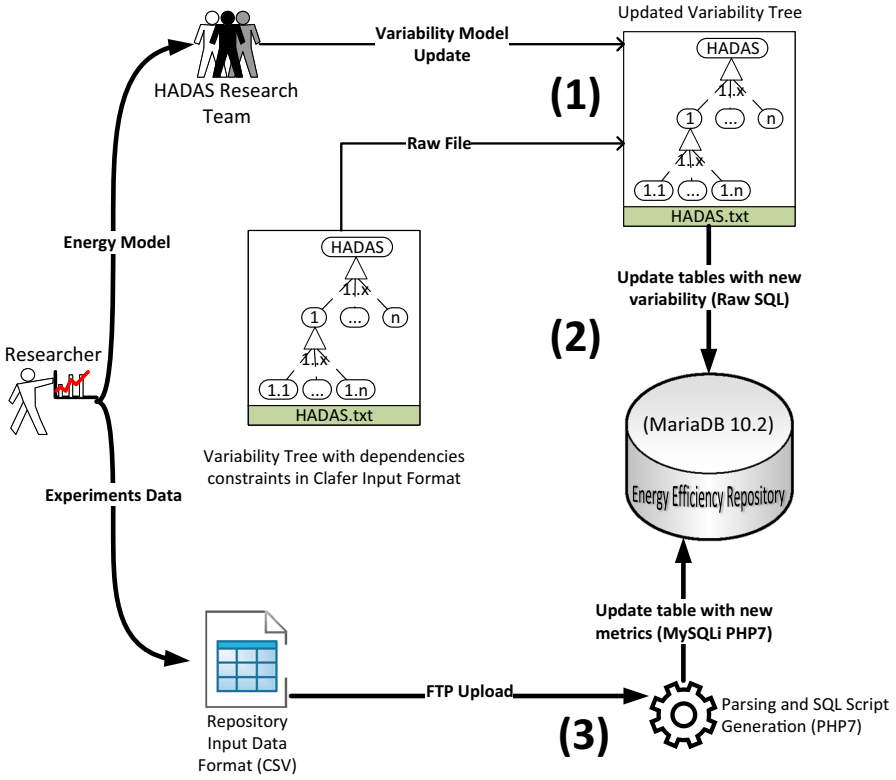

Fig. 3 Models and automatised procedure to insert energy and time data into HADAS

HADAS internally uses Clafer, so researchers should preferably provide the variability model in the Clafer text format. Other formats such as CVL can be easily mapped to Clafer, as shown in [35]. HADAS will automatically check the validity of the input model. However, where to insert the new model in the HADAS variability model must be approved in advance by the HADAS research team (HRT). The main reason for this is that the energy data a researcher wants to add could not only be related to a new EEC, but also to a new design or implementation of an existing ECC. It could also entail adding or modifying the energy context of an existing ECC with a new energy-related parameter, data type or function. Consequently, this step can be semi-automatised, with the HRT ultimately being responsible for manually update (technically called Evolve) the variability model of HADAS. Once the variability model has been updated, the new nodes are added to the database (just the leaves of each layer), preparing it for the additional energy data (step 2 in Fig. 3). Right now, it is also done manually by the HRT with simple INSERT queries, storing each name in its corresponding table (layer). Additionally, the researcher has to provide the experimental data by means of one or several CSV file/s. These files will be automatically processed to generate the necessary database queries (*INSERT* operations) (step 3 in Fig. 3). Our CSV format conforms to the variability model and database structure.

4.2 HADAS for software developers

4.2.1 How can developers use HADAS?

A developer wants to know the potential leakage points of the different alternatives of a specific use case. As shown in Fig. 4, using HADAS he will access the HADAS tree view form (step 1), a web view representation that is automatically generated from the ECC variability model (step 2), and will navigate through the form, selecting the energy consumers that he needs to analyse (variants) in an intuitive way (step 3). Then, HADAS generates a partial configuration, which is the input for the Clafer constraint-satisfaction solver, which automatically generates the different configurations to be analysed (step 4). Once a configuration has been defined, HADAS automatically generates the database queries to retrieve the metrics of that configuration (step 5). Finally, the graphs that allow sustainability analyses are generated (step 6). Moreover, HADAS has been extended with a 'Significant Correlation Analysis' [9] to improve the accuracy and richness of the information provided to the developers. In order to generate the web user interface, we have developed a PHP 7 parser (step 2) that loads and parses the variability model, which is in Clafer text format, and stores the generated information in a multidimensional array. In this array each layer in the variability model is a sub-array, and each node is the key of a new sub-array. The array is then processed to generate the proper HTML, CSS and JavaScript code to build its web view representation, which is shown to the user through the HADAS web user interface (step 3). Then, the file with the partial configuration selected by the user is the input to the Choco Solver 4.0, which automatically generates all the possible complete configurations, following Clafer (step 4). The database queries are run with the MySQLi connector (step 5) and we use the information returned by these queries to first generate the Google Charts header, and then generate the Google Charts footer (step 6). The web view is sent to the developer's web explorer, and thereby he can, transparently for the whole process, visualise the graphs and statistical results, performing the desired sustainability analysis.

4.2.2 Significant correlation analysis extension

As the literature highlights the importance of statistical analysis in the ambit of green software and systems [28,34], we propose to add the next step to the HADAS sustainability analysis support: analyses of linear correlations of the energy and time (the considered free variables) with the rest of the features. As can be seen in Fig. 4, we have extended HADAS with *Step 7* adding a significant correlation analysis procedure.

The procedure has two sequential steps: (1) calculate if a linear correlation exists between two features using the least squares regression method; (2) analyse if the established correlation is significant. It is worth mentioning that, if the data-sets have outliers,[2] they are removed from the analysis in order to ensure higher numeric results and more accurate insights. Firstly, in the least squares adjustment we have two indexes

[2] An outlier is a value point that is very distant from regular values.

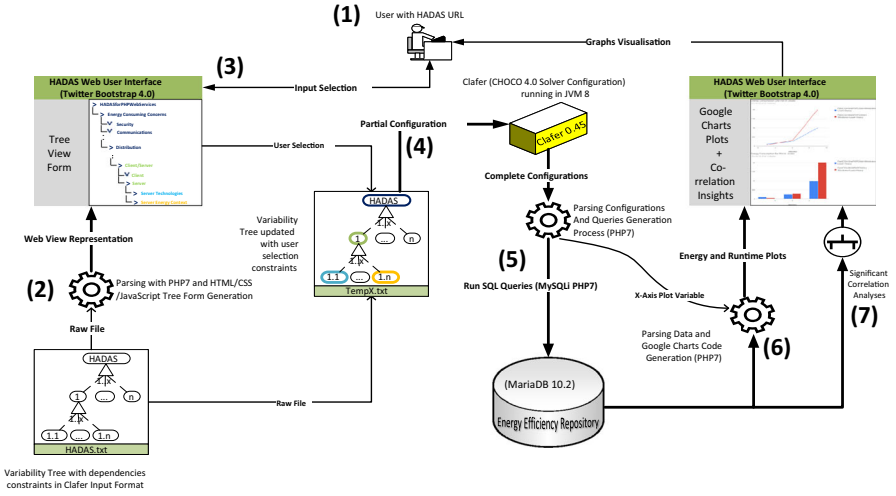

Fig. 4 Detailed process of HADAS functionality under developer usage

x_i and y_i. y_i could be time or energy consumption, leaving x_i for any other feature or variable of a complete configuration. In order to check that a linear correlation exists between them we adjust a linear equation ($y_i = a + bx_i$), where a and b are independent parameters (intercept and slope). In the case of non-numeric features, a compound $y_i = f(x_i)$ would be necessary, as well as some extra computation, however this article concentrates on numerical variable features for a clearer understanding of the process. Assuming a simple null model, like the mean of the data-set, we set $b = 0$ and $a = \overline{y}$. To analyse the significance of the correlation of two indexes we have used the following procedure [9]. This test consists in determining whether it can be concluded that a linear adjustment reflects the data correlation better than a null model. For this task, we have used the Pearson's $\tilde{\chi}^2$ differentials [31] defined thus:

$$\Delta\tilde{\chi}^2 = \sum_i \left(\frac{y_i - \overline{y}}{\sigma_{yi}}\right)^2 - \sum_i \left(\frac{y_i - (y_i = a + bx_i)}{\sigma_{yi}}\right)^2 \tag{1}$$

Then, we can determine if the improvement is caused randomly by the inherent noise of the data. Instead of using the analytical expression of $\tilde{\chi}^2$ distribution for noisy data, we obtain the empirical distribution of a Monte Carlo approach. We carry out a Bootstrapping statistics [11] process, linking each x_i to the duo (y_i,σ_{yi}), mixing the data randomly. For example, to test the correlation between response time as a function of the number of concurrent users:

- x_i = users. Clarifying, x_i is each one of the values of the concurrent-users data-set.
- y_i = response time. Clarifying, y_i is each one of the values of the response time data-set.
- σ_{yi} = Error of the response time (contemplated error due to the measurement/simulation system, room temperature, etc.).

Then, to run the statistical process, the bootstrapping is repeated N times (with N equal to the number of experiments), obtaining N $Bootstrapped\,\Delta\tilde{\chi}^2$ values. From the N $Bootstrapped\,\Delta\tilde{\chi}^2$ values, M are those with a higher value than the $\Delta\tilde{\chi}^2$ obtained in the hypothesis, that is to say, they are the experiments that give better results despite a randomly mixed data. If M is too high, it means that the $\Delta\tilde{\chi}^2$ of the hypothesis can be rendered with a combination of noise, so the adjustment is invalid. However, if the value $\Delta\tilde{\chi}^2$ of the hypothesis is quite far from the N experiment values, the results are solid, implying a confirmed significance. The Probability of False Alarm (FAP) mathematically formalises the concept. The FAP is defined as the number of bootstrapping experiments that result in a higher $Bootstrapped\,\Delta\tilde{\chi}^2$ than the initial hypothesis (M) divided by the number of experiments (N). Thus, the probability that the statistic conclusion is a false alarm can be defined as: $P(\Delta\tilde{\chi}^2) = M/N$ and $\text{FAP} = 100 * (M)/N$ (as a percentage). The error of this probability is $\sqrt{M}/N$ with $\text{errFAP} = 100 * (\sqrt{M})/N)$. This probability follows a Poisson Distribution [13], and therefore converge on a value while increasing the number of experiments. As a result, if not enough experiments are executed (low N), the result has a high error rate, but, if lots of experiments are executed, a lot of computing power is necessary. A good balance is to establish N accomplishing $2 * errFAP/FAP < 1$. This way, we ensure that the value is over two times the error rate.

The extra analysis, alongside the HADAS approach, this extension provides us with, insights into what feature is most affecting the energy and/or time consumption (does not need to be the same culprit) quite easily. For instance, it is obvious that the number of concurrent users substantially linearly affects the energy consumption and response times of a data-centre, but it is not so clear if we talk about data-size; if computation is always low, for a powerful data-centre there is almost no difference in server consumption when the data-size varies. Another example is to note that the Operating System or the Programming Language is affecting the energy the most, so, if the developer really needs to reduce energy, he must migrate the system to the lowest consuming alternative, Operating System and Programming Language, shown in the graphs.

5 Evaluation

In this section, we apply the whole process to the PHP web services development case study. Taking the researcher role, the first step is to evolve (update the variability model) adding new features to the *Distribution* energy consuming concern and its design variant *Client/Server->Server*. We need to, also, add the corresponding trending implementation alternatives and energy context variability. The result is shown in Fig. 5, where the model follows the three layers structure defined in Fig. 2. Since the consumption of PHP frameworks depends on the consumption of Plain PHP (as programming language frameworks primarily depend on the languages in which are written), this time we focus on the metrics for Plain PHP consumption, leaving out framework variability. For Plain PHP, on the one hand, there are different web servers and restrictions regarding the operating system compatibility (*Web Server* feature in Fig. 5), where the most common alternative technologies are currently: Apache,

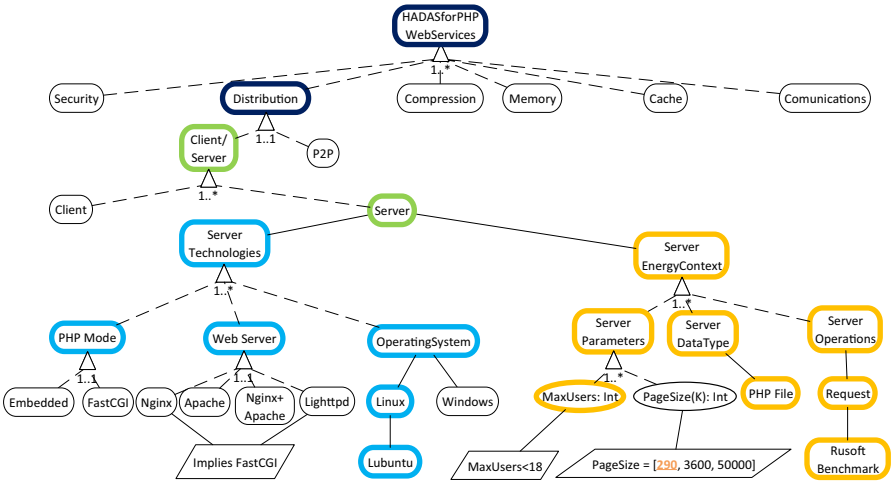

Fig. 5 Case study of under Linux

Nginx, and Lighttpd. On the other hand, we have the mode in which PHP is working (*PHP Mode* feature): a module of the web server as an embedded interpreter, or PHP FastCGI processes. The variability model specifies constraints such as Nginx implies FastCGI mode. As our experiments are conducted in Windows and Linux, it divides the research into two subsections, depending on the Operating System, as they have sufficient particularities and normally are not subject to change in real scenarios.

5.1 Linux

Ubuntu is the most common operating system for web servers (39%) as of January 2018.[3] We have chosen to measure and analyse an updated Lubuntu 17.10 x86_64 (last version for 64 bits capable CPUs). Lubuntu is a clean and lightweight Linux distribution, with the core of the system based on Debian and Ubuntu distributions. As it is biased to consume the fewest computer resources possible, Lubuntu uses the minimal desktop LXDE and just basic light applications. Lubuntu has very low hardware requirements, which makes it the most suitable for energy experiments, as it provides low noise and minimises other applications' interactions that could affect the results. The kernel version is *Linux 4.13.0-25-generic (x86_64)*, which is executed in an Intel (R) Core i7-4790 CPU @ 3.60 GHz with other high-end characteristics such as 16 Gigabytes of memory RAM and a Solid State Drive (SSD) disk. The server runs in high performance mode, with a couple of internal fans running at maximum capacity to decrease data noise caused by the temperature of the server. The most recent x64 software versions are used, and requests to the server are sent to the URL from an external computer. To isolate any effect that could be caused by a specific web browser, the JMeter tool is used to create up to 18 concurrent virtual users, activated

[3] https://w3techs.com/technologies/details/os-linux/all/all.

sequentially every 0.1 s. In a previous study, we found that up to 18 users is a sufficient number to expose trends [27]. To cover a large number of situations, we have chosen a synthetic benchmark.[4] The most complete and up-to-date (PHP 7.1) Plain PHP benchmark is developed by Rusoft Ltd.[5] and includes mathematical operations, string manipulations, regular expressions, loops and recursion, serialisation and encryption. Each JMeter request is a web page consisting in the Jumbotron Twitter Bootstrap 4.0 template page with the benchmark and several images embedded. Depending on the number of images, the *Page Size* is different (260, 3600 and 50,000 kB). This scenario is modelled in the energy context part of the Fig. 5. A data-centre system developer is interested in the entire server's consumption, as each Joule consumed costs money. In addition, due to the lack of applications to estimate measurements of each service in a CPU (in Linux exists PowerTop, but requires a battery fuelled device to provide energy consumption values) we have opted for *Watts Up? Pro*, a professional energy monitor.

Regarding PHP Modes, in the case of Apache plus **mod_php** the interpreter is embedded in each Apache process dynamically spawned with each server request. Apache workers can handle and execute PHP scripts by themselves removing the need of any external processes; unlike FastCGI. With the **FastCGI** mode, each request is passed from the web server to the FastCGI processes via a communication socket. This allows for much greater scalability as the web server and the PHP interpreter can be split into their own individual server environments if necessary. Summarising, *mod_php* is a multi-process mode (a process deals with just one request), and *FastCGI* is a multi-process and multi-thread mode (a process shares its resources with several requests). This means that *FastCGI* mode requires fewer resources (e.g., CPU, memory) to attend to concurrent requests. The default settings, which usually implies FastCGI mode, have been used except Apache *mod_php* and a mixture of Nginx and Apache *mod_php*. Nginx can be configured as a reverse proxy of Apache, with Nginx processing the front-end (HTML, CSS, JavaScript code and media files) and Apache the back-end (PHP code). This is a configuration that tries to exploit the best of the two technologies: the lightness of Nginx (designed to use fewer resources) and the compatibility and speed of Apache and *mod_php*. Lighttpd is a minimalistic and lightweight web-server, which makes it unsuitable for some purposes due to its lack of extra functionality. After measuring 25 runs of each complete configuration, we calculated the median, and dumped the measurements with HADAS automatic script into the database of the HADAS repository. A developer introduces a case study with the constraint of 290 Kilobytes (Fig. 5) and the number of users as the x-axis variable. Then, he analyses the graphs with each valid complete configuration generated that HADAS generates after running its service (Fig. 6). Even if the developer does not have any knowledge, he can easily visualise that, as the concurrent number of requests increase, Nginx has a clear advantage over the rest with respect to energy consumption. Looking at the runtime, Nginx with Apache *mod_php* is the fastest, closely followed by a standalone Apache *mod_php*. As the interpreter is initialised with Apache in *mod_php* mode, the requests

[4] Synthetic benchmark: specially created program that impose specific workloads on a system as a generalisation of real applications.

[5] https://github.com/rusoft/php-simple-benchmark-script.

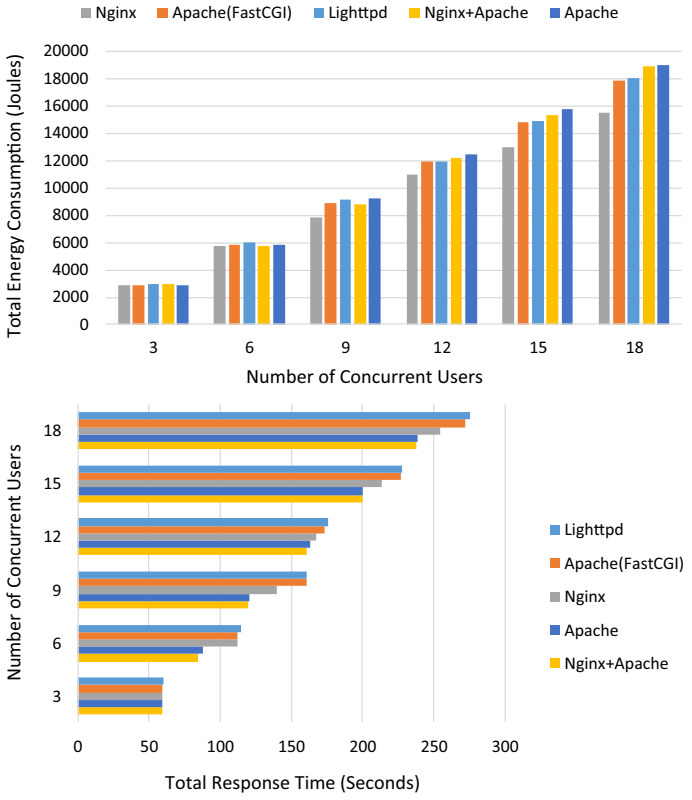

Fig. 6 HADAS: PHP web service graphs

become faster as certain information is cached, decreasing repetition of the same tasks each time a script is executed. The downside of *mod_php* is that the footprint for each Apache process is larger as it requires more resources with the PHP interpreter embedded, which is reflected in the energy consumption plot. Generalising, *mod_php* mode is faster (more scalable), but *FastCGI* is greener. If a trade-off is evaluated, Nginx (*FastCGI*) is the option.

5.2 Windows and statistical analysis

We conducted a similar case study under Windows Server 2016 [27] with certain differences due to the operating system. The measurements were obtained with the Intel Platform Power Estimation Tool (IPPET), which estimates the consumption of each PHP and web-server process. The processes' metrics were added together and dumped in HADAS. Apache *mod_php* under Windows is, in addition to being a multi-process, a multi-thread server. Nginx under Windows does not dynamically spawn processes; they need to be started with the initialisation of the web-server, so two options were evaluated, a mono-process and a maximum of 18 processes. Lastly, Lighttpd is not

available in Windows, but Windows has its own server, ISS. A detailed study can be found in [27], but the main insights were similar with regard to the PHP Modes, *mod_php* is faster and *FastCGI* is greener. However, as explained, the web-servers work differently under Windows, which results in Nginx with 18 processes being the greenest, Apache *mod_php* the fastest, and ISS the most balanced (trade-off).

Taking into account the Linux and Windows data, as HADAS was upgraded for this article, it can now perform a significant correlation analysis of energy and time with numerical variables alongside the plots. There are four extra results in this case study. With regard to margin error (necessary for Eq. 1), *Watts Up? Pro* is proven to have a 1.8% accuracy [18]. Energy correlated with number of users (result 1) gives a linear relation of $Y = -241.2 + 1022.3X$ and an False Alarm Probability (FAP) of almost absolute zero percent. It gives the same probability for time and users (result 2) with a linear relation of $Y = 20.7 + 12.6X$. In the case of energy and page size (result 3), and time, and page size (result 4), the FAPs are around 1% and the relations are $Y = 10865 + 0.00001X$ and $Y = 145 + 1.4X$. The four cases show a significant linear correlation, but much stronger in the case of the number of users. In other words, increments in the number of users (requests) affect the energy consumption and response time more than an increase in page size.

5.3 Summary

The developer must evaluate the trade-off between the cost of the energy consumption of his services and the response time that the stakeholders require. He can even sell more expensive services depending on how the trade-off is balanced, without without having to increase the hardware resources, making it worth the energy costs for his company. *How do developers benefit from these metrics?* Let us imagine that a PHP back-end developer already has a set of running services deployed in the servers of his facility, requested by a variable number of users. He is operating in a small developing economy, where the local government is increasing the electricity price. The developer does not want to invest time in changing and re-configuring the operating system, but to decrease his electricity costs, is willing to change the web server and the PHP Mode, as it is a fast procedure. The developer accesses the HADAS website, and chooses a partial configuration using the form, selecting his *Operating System*, and the *Request* of a *PHP File* with the *Rusoft Benchmark* as shown in Fig. 5. This is a partial configuration that is represented as a Clafer model and solved by Chocosolver, generating seven complete configurations. It also automatically generates the same number of SQL queries (database SELECT instructions), with the results sent to the Google Charts API, generating the graphs in Fig. 6. In addition, the meta-data is included in the results (software, and the hardware used for running, testing and measuring).

In summary, Nginx is the greenest alternative in both operating systems, and Apache the fastest (with Nginx as a front-end in Linux). If a trade-off is necessary, Nginx under Linux and ISS under Windows are the best options. There is a significant linear relationship of energy and time with the number of users and page size, however it is much stronger in the case of the number of users. Even if it is important to explain the

technical information so as to understand the results that have been obtained through experimentation, using HADAS, the developer can reach the same conclusions without any technical explanation or statistical background. Therefore, he only need focus on applying the knowledge previously stored in HADAS by the researchers, even if he cannot fully explain why. Also, both researcher and developer, will find an automatised service (the HADAS eco-assistant) for their distinct purposes.

6 Conclusions and future work

In this article, we have reviewed and automatised the HADAS eco-assistant. HADAS exploits the advantages of variability models to reason, and choose between assorted designs and implementation alternatives, considering energy consumption. From a researcher's point of view, HADAS allows his data to be almost automatically introduced into the repository, making it accessible through its services. For developers interested in exploring the energy consumption to develop eco-efficient applications, HADAS allows them to have a better understanding of all the variants that exist and to perform a richer software sustainability analysis, including significant correlation analyses between energy or time and every other feature. In addition, we have measured, and analysed with HADAS, the most common web servers and their intrinsic relationship with energy consumption and performance, where Nginx is always (Windows and Linux) the greenest alternative, and Apache the fastest (alongside Nginx as a front-end under Linux). If a trade-off is necessary, Nginx under Linux and ISS under Windows are the most balanced solutions. Additionally, there is a significant linear relationship of energy and time with the number of users and page size, stronger in the case of the number of users. This relationship is exposed by the new statistic module of HADAS based on Pearson's chi-squared differentials and Bootstrapping statistics, analysing the low False Probability Alarm of the results. As for future work, on the one hand we plan to develop better automatic processes for adding and analysing data, trying to ease the researcher and developer's roles as much as it is possible. On the other hand, we continue to improve the eco-assistant with stronger statistical analyses with non-numeric features, and with non-linear relationships, prior steps for machine learning and prediction of energy and time consumption.

Acknowledgements Work funded by the projects MAGIC P12-TIC1814 and HADAS TIN2015-64841-R, and by the University of Malaga.

References

1. Antkiewicz M, Bąk K, Murashkin A, Olaechea R, Liang JHJ, Czarnecki K (2013) Clafer tools for product line engineering. In: Proceedings of the 17th international software product line conference co-located workshops, pp 130–135. ACM
2. Bertini L, Leite JC, Mosse D (2007) Statistical QoS guarantee and energy-efficiency in web server clusters. In: 19th Euromicro conference on real-time systems, 2007. ECRTS'07, pp 83–92. IEEE
3. Bianchini R, Rajamony R (2004) Power and energy management for server systems. Computer 37(11):68–76
4. Brunnert A, Vögele C, Krcmar H (2013) Automatic performance model generation for java enterprise edition (ee) applications. In: European workshop on performance engineering, pp 74–88. Springer

5. Cabeza LF, Palacios A, Serrano S, Ürge-Vorsatz D, Barreneche C (2018) Comparison of past projections of global and regional primary and final energy consumption with historical data. Renew Sustain Energy Rev 82:681–688
6. Chowdhury SA, Hindle A (2016) Greenoracle: Estimating software energy consumption with energy measurement corpora. In: Proceedings of the 13th international conference on mining software repositories, pp 49–60. ACM
7. Chowdhury SA, Sapra V, Hindle A (2016) Client-side energy efficiency of HTTP/2 for web and mobile app developers. In: 2016 IEEE 23rd international conference on software analysis, evolution, and reengineering (SANER), vol 1, pp 529–540. IEEE
8. Djemame K, Armstrong D, Kavanagh R, Juan Ferrer A, Garcia Perez D, Antona D, Deprez JC, Ponsard C, Ortiz D, Macías Lloret M, et al. (2014) Energy efficiency embedded service lifecycle: towards an energy efficient cloud computing architecture. In: Joint workshop proceedings of the 2nd international conference on ICT for sustainability 2014, pp 1–6. CEUR-WS. org
9. Feng F, Tuomi M, Jones HR (2017) Agatha: disentangling periodic signals from correlated noise in a periodogram framework. Mon Not R Astron Soc 470:4794–4814
10. Ferreira MA, Hoekstra E, Merkus B, Visser B, Visser J (2013) Seflab: a lab for measuring software energy footprints. In: 2013 2nd International workshop on green and sustainable software (GREENS), pp 30–37. IEEE
11. Freedman DA et al (1981) Bootstrapping regression models. Ann Stat 9(6):1218–1228
12. Grosskop K, Visser J (2013) Identification of application-level energy optimizations. In: Proceeding of ICT for sustainability (ICT4S), pp 101–107
13. Haight FA (1967) Handbook of the Poisson distribution
14. Hao S, Li D, Halfond WG, Govindan R (2012) Estimating android applications' CPU energy usage via bytecode profiling. In: 2012 First international workshop on green and sustainable software (GREENS), pp 1–7. IEEE
15. Hasan S, King Z, Hafiz M, Sayagh M, Adams B, Hindle A (2016) Energy profiles of java collections classes. In: 2016 IEEE/ACM 38th international conference on software engineering (ICSE), pp 225–236. IEEE
16. Haugen Ø (2012) Common variability language (CVL) OMG revised submission. OMG Doc ad
17. Hindle A, Wilson A, Rasmussen K, Barlow EJ, Campbell JC, Romansky S (2014) Greenminer: a hardware based mining software repositories software energy consumption framework. In: Proceedings of the 11th working conference on mining software repositories, pp 12–21. ACM
18. Hirst JM, Miller JR, Kaplan BA., Reed DD (2013) Watts up? Pro AC power meter for automated energy recording
19. Horcas JM, Pinto M, Fuentes L (2017) Variability models for generating efficient configurations of functional quality attributes. Inf Softw Technol 95:147–164
20. Horcas JM, Pinto M, Fuentes L, Gámez N (2017) Self-adaptive energy-efficent applications: the hadas developing approach. In: 2017 IEEE 15th International conference on dependable, autonomic and secure computing, 15th International conference on pervasive intelligence and computing, 3rd International conference on big data intelligence and computing and cyber science and technology congress (DASC/PiCom/DataCom/CyberSciTech), pp 828–835. https://doi.org/10.1109/DASC-PICom-DataCom-CyberSciTec.2017.140
21. Kalic G, Bojic I, Kusek M (2012) Energy consumption in android phones when using wireless communication technologies. In: MIPRO, 2012 Proceedings of the 35th international convention, pp 754–759. IEEE
22. Kim D, Choi JY, Hong JE (2017) Evaluating energy efficiency of internet of things software architecture based on reusable software components. Int J Distrib Sens Netw 13(1):1550147716682738
23. Manotas I, Pollock L, Clause J (2014) Seeds: a software engineer's energy-optimization decision support framework. In: Proceedings of the 36th international conference on software engineering, pp 503–514. ACM
24. Miranskyy AV, Al-zanbouri Z, Godwin D, Bener AB (2017) Database engines: evolution of greenness. arXiv preprint arXiv:1701.02344
25. Montenegro JA, Pinto M, Fuentes L (2017) What do software developers need to know to build secure energy-efficient android applications? IEEE Access PP(99):1–1. https://doi.org/10.1109/ACCESS.2017.2779131
26. Munoz DJ (2017) Achieving energy efficiency using a software product line approach. In: Proceedings of the 21st international systems and software product line conference, vol B, pp 131–138. ACM

27. Munoz DJ, Pinto M, Fuentes L (2017) Green software development and research with the HADAS toolkit. In: Proceedings of the 11th European conference on software architecture: companion proceedings, pp 205–211. ACM
28. Murwantara I, Bordbar B, Minku LL (2014) Measuring energy consumption for web service product configuration. In: Proceedings of the 16th international conference on information integration and web-based applications and services, pp 224–228. ACM
29. Pattinson C, Olaoluwa PO, Kor AL (2015) A comparative study on the energy consumption of PHP single and double quotes. In: 2015 IEEE international conference on data science and data intensive systems (DSDIS), pp 232–239. IEEE
30. Pohl K, Böckle G, van Der Linden FJ (2005) Software product line engineering: foundations, principles and techniques. Springer, Berlin
31. Rao JN, Scott AJ (1981) The analysis of categorical data from complex sample surveys: chi-squared tests for goodness of fit and independence in two-way tables. J Am Stat Assoc 76(374):221–230
32. Sapra V, Hindle A (2016) Web servers energy efficiency under HTTP/2. PeerJ Prepr 4:e2027v1
33. Stier C, Koziolek A, Groenda H, Reussner R (2015) Model-based energy efficiency analysis of software architectures. In: European conference on software architecture, pp 221–238. Springer
34. Westin J (2017) Evaluation of energy consumption in virtualization environments: proof of concept using containers
35. Wijsman T (2015) Introduction to a CVL to Clafer transformation project