Noname manuscript No.
(will be inserted by the editor)

Extending Software Architecture Views with an Energy Consumption Perspective

A case study on resource consumption of enterprise software

Erik Jagroep · Jan Martijn E.M. van der Werf · Sjaak Brinkkemper · Leen Blom · Rob van Vliet

Received: date / Accepted: date

Abstract The rising energy consumption of the ICT industry has triggered a quest for more sustainable, i.e. energy efficient, ICT solutions. Software plays an essential role in finding these solutions, as software is identified as the true consumer of power. However, in this context, software is often treated as a single, complex entity instead of the interrelated elements that it actually consists of. Although useful results can be gained, this approach fails to provide detailed insight in the elements that invoke specific energy consumption behavior. As a result, software vendors are not able to address energy consumption on software level.
In this paper, we propose an energy consumption perspective on software architecture as a means to provide this insight and enable analysis on the architectural elements that are the actual drivers behind the energy consumption. In support of this perspective, we also position sustainability as a potential quality attribute thereby provide a means to quantify energy consumption aspects related to software. In a case study using a commercial software product the perspective and quality attribute are applied, demonstrating the potential by achieving an energy consumption saving of 67.1%.

Keywords Software Architecture · Energy consumption perspective · Sustainability · Quality attributes

Mathematics Subject Classification (2000) 68N99

Erik Jagroep, Jan Martijn E.M. van der Werf, Sjaak Brinkkemper
Utrecht University
Department of Information and Computing Sciences
P.O. Box 80.089, 3508 TB Utrecht, The Netherlands
E-mail: {e.a.jagroep,j.m.e.m.vanderwerf,s.brinkkemper}@uu.nl,

Erik Jagroep, Leen Blom, Rob van Vliet
Centric Netherlands B.V.
P.O. Box 338, 2800 AH Gouda, The Netherlands
E-mail: {erik.jagroep, leen.blom, rob.van.vliet}@centric.eu

1 Introduction

The Energy Consumption (EC) of the Information and Communication Technology (ICT) sector is a booming topic of interest. Recent figures indicate that at least a tenth of the world's electricity use is on behalf of ICT [23]; a figure that has kept growing over the years. As a result of the increased awareness on the subject, the term 'sustainability' has emerged which is to "meet the needs of the present without compromising the ability of future generations to satisfy their own needs" [25]. Within the research community this resulted in much attention going towards increasing the energy efficiency of ICT.

Until recently the focus has mostly been on hardware related aspects as improvements on hardware level are relatively tangible and easy to apply, e.g. renewal of hardware. However, in [21] the role of software is also stressed in finding sustainable ICT solutions. While energy is directly consumed by hardware, the operations are directed by software and can eliminate any sustainable features built into the hardware [34]. Thus software is argued to be the true consumer of power [35].

In research software is often treated as a single, complex entity (i.e. considered on application level) instead of the inter-related elements it actually consists of (cf. [12, 17]). A breakdown into hardware components and 'units of work' is made, but allocating EC to individual software modules has proven to be a difficult task [29]. Consequently, a stakeholder does not know which modules and functions invoke specific energy consuming behavior making it difficult to direct sustainability efforts concerning software to where they are needed.

We argue Software Architecture (SA) is able to fill this gap and in this paper we investigate how EC can be positioned within the scope of SA in the context of product software, i.e. 'a packaged configuration of software components or a software-based service, with auxiliary materials, which is released for and traded in a specific market' [36]. An Architecture Description (AD) of product software complemented with EC measurements, could help to direct green computing efforts (i.e. Energy Efficient (EE) algorithms [14]) and determine appropriate adjustments on the right locations. More specifically we focus on on-premise software products, i.e. products of which multiple instances exist on different locations (e.g. due to regulations). The potential of on-premise product software, in contrast to tailor-made software, is in the fact that a change finds its way to each deployment, thereby multiplying the potential impact with each installation. In [13] a decrease in EC of 0.25 Watt with four million installations is presented to save the EC equivalent to that of an American household per month, showing that even the smallest change could have a major impact.

With our research we contribute to the research domain in multiple ways. First and foremost, we provide an EC perspective on SA following the detailed format and viewpoint catalog described by Rozanski and Woods [32] including: the applicability to views through key questions, concerns that can be addressed, activities for application and architectural tactics. Problems and pitfalls and checklists, also included in the format [32], come with experience and are deemed future research. Second, since a perspective addresses quality properties [2,32], we position sustainability as a Quality Attribute (QA) following the ISO 25010 standard format. Quality properties, measures and measure elements provide a means to quantitatively describe software

aspects related to EC. Through an experiment, that builds on a recently published case study [16], the perspective is validated. The potential of our research is demonstrated by realizing a reduction in energy consumption of 67.1% in a case study.

The remainder of this paper is structured as follows. We first present the related work on energy consumption and SA Sect. 2 and position sustainability as a QA (Sect. 3). With this knowledge an energy consumption perspective (Sect. 4) is constructed and applied in practice (Sect. 5). Finally, we provide a conclusion and identify directions for future research (Sect. 6).

1.1 Document Generator as a Real-Life Example

Before continuing with the related work we introduce a case in the form of Document Generator (DG); a commercial software product used to generate over 30 million documents per year. DG is used by approximately 300 customers with 1000 end-users as a complementary product with other commercial software products.

The basic workflow for DG (right side of Fig. 1) is initiated by a trigger from an external application. After validation of the received input, DG collects the data and document definitions, managed in separate systems, and merges the data into a preview for approval. After approval, the actual generation is performed and the documents are archived (optionally in a DMS) and communicated to a required outlet.

Figure 1 also includes a mapping of the workflow onto the functional architecture of DG. The ‘Connector’ element contains four sub-elements and is responsible for four of the six activities in the workflow, namely receiving input, collecting data, archiving and communicating. Together with the ‘Composer’ element, responsible for merging the document definitions with data, the ‘Connector’ element handles the required activities before and after document generation. ‘Utilities’ and ‘Interface’ respectively provide configuration options and an interface for DG and do not map

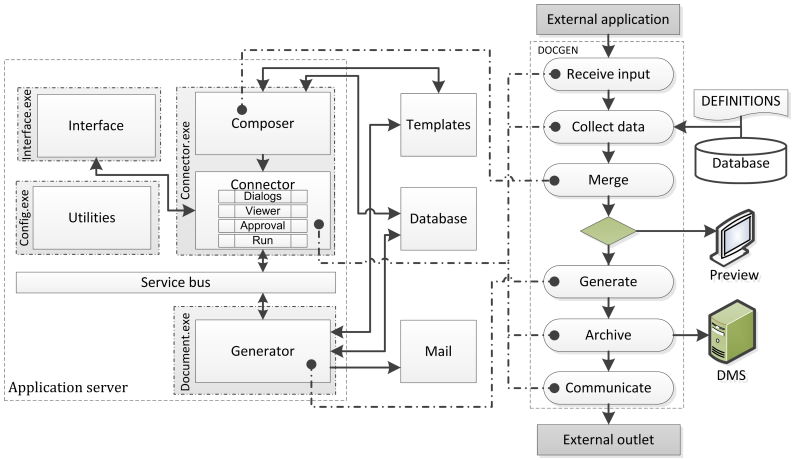

Fig. 1 The Document Generator (DG) workflow mapped on the functional architecture.

onto an activity in the workflow. Finally the ‘Generator’ is responsible for the actual generation of the documents and corresponds to the generate activity.

For this research we focus on the application server including the connector.exe, config.exe, interface.exe and document.exe processes (i.e. concurrency units). Representative for an actual production installation, these processes run on a single server (i.e. deployment) which is labeled the ‘Application server’. The database server is considered out of scope for this research. In the remainder of this paper, the DG case is used to provide concrete examples for the perspective.

2 Related Work

In [8], the term sustainability is used for analyzing ecological, economical and social dimensions (of ICT), without compromising the ability of future stakeholders to meet their needs. Our research fits the area of green software, a niche of sustainability where the software is the object of optimization [21], and is mainly focused on the ecological aspect. Before we can continue to construct the EC perspective we need to discuss the matters of measuring the EC of software and its relation to SA.

2.1 Energy Consumption Measurements

One of the main issues with respect to green software is to perform detailed EC measurements. Specialized environments, e.g. [11], enable measurements on each individual hardware component and provide detailed insight into how software affects these components. In these setups, the EC of software is measured by relating the EC of hardware to computational resource usage on behalf of the software and, consequently, energy efficiency refers to the efficient use of computational resources [12]. However, these environments are rare, difficult to expand to more complex environments (e.g. data canter) and only few are able (and willing) to invest in the equipment required for such a solution. At the cost of details, external power measurement equipment can also be used.

Compared to hardware measurements, software approaches can specifically focus on the software under investigation and measure on more detailed levels. The ‘E-Surgeon’ solution [28] for example, enables its user to monitor the EC of software during runtime down to the classes and methods. Although promising results are obtained, applying the ‘E-Surgeon’ solution requires expert knowledge on the subject which potentially inhibits its adoption. The same holds for instrumenting the software [31]. A more simple software-only approach is to use energy profilers [27]; software tools including a power model with the ability to estimate the software EC on different levels of granularity. Unfortunately recent study has found energy profilers to not always provide the desired results [15].

EC measurements become even more complex when the entire computing stack is in play, as each layer between software and hardware level (e.g. operating system, virtual machine) is said to amplify the EC induced by the software [7]. In the paradigm of ‘programmable web’ [6], end users can easily create applications tailored to their

own needs. Without proper control of the layers at play, the explosion in number of applications could have disastrous effects on the EC of the underlying infrastructure. Since the impact of each layer is a research topic on its own, we consider this as future research and maintain focus on the software product and its architecture.

A final aspect with regard to EC measurements is the deployment of the software product. Nowadays, software is often distributed across multiple servers, or even across federated data centers, and resources are shared with other applications. In [10] 'Green Performance Indicators' (GPIs) are proposed for these environments that, apart from EC measurements, require detailed performance monitoring to assign (portions of) the EC to specific software elements. Obtaining the required data requires appropriate performance measurements and either a software or hardware approach for EC measurements, depending on the level of detail required.

2.2 Relating Green Software to Software Architecture

While the hardware and software approaches for measuring have up- and down-sides to them, both serve the same purpose; identify 'software energy hotspots' [31] which are the measurable elements or properties that have a significant impact on the EC. The proposition of green software is to have software that requires the least amount of resources as possible while performing the required task(s).

From a software vendors' perspective, investment in new hardware or optimizing the current hardware is considered less costly than having a slower development cycle. Currently, performance is optimized on hardware level and software engineers are instructed to write software at a high pace with the risk of delivering sub-optimal code and algorithms. It is the experience of the authors in industry that still too little is known with regard to the potential benefits of green software to create a valid business case. There is light on the horizon, however, with green software examples and guidelines becoming increasingly more available [3, 24, 26, 39] and concrete [1]. Even without changing the current practice of software engineers [26].

At its core the creation of EE software starts with the design of the software [3], i.e. with its architecture. Using the SA to determine meaningful units, e.g. architectural elements, can make the software and its context for development easier to understand, control and influence. A similar approach is applied in a wider context for the green performance indicators model [10, 19] containing separate controllable elements on different layers. The relation between these elements shows how 'green goals' shine through from top (organizational) level down to the hardware level with software in between. Through software architecture, we propose to unravel the application layer of the model and take a more detailed look into the software itself, i.e. turn software into a 'white box'.

Different views that map the EC on software artifacts already exist. The node map presented in [12] for example, closely resembles what could be labeled as a deployment view showing the installation of software elements across the available

[1] `https://wiki.cs.vu.nl/green_software/index.php/Best_practices_for_energy_efficient_software` accessed 19-08-2015

hardware. Analyzing the node map on EC provides a so-called 'heat map' of the system. Following this same line, [17] presents the 'ME^3SA' model in which again the deployment and functional components of the software are investigated. In relation to green software, a limitation of both approaches is that most recommendations relate to hardware aspects and only provide 'strong clues' on software level.

For embedded software, software that accompanies an appliance [36], research on EC also adopts an architectural approach. In [37] a method is presented to determine the minimum EC path through Petri Nets and reachable state graphs. Others propose to create modular software and collect utilization data of functional elements which is used by a resource utilization model [5] functioning as 'energy broker'. An approach that has recently also been applied to software in general [28]. Although embedded software is often less complex compared to product software, due to its specific, limited functionality for the appliance, a modular approach, which resembles an AD, helps to better understand its energy consuming behavior. Similar to product software, a multiplying effect could also be achieved with embedded software.

An advantage of using the SA is to address concerns related to EC in an early stage of the software life cycle, namely during its design. Through the architecture, a product manager, that determines the strategic direction (including green goals) for a product [9], has a means to address his concerns in the product design [32] and determine whether the desired quality of service is achieved [20]. From an organizational perspective such an approach emphasizes the role that green software can play in reaching sustainability goals. On this level, green software has the potential to reduce the operational costs related to a software product, enabling sustainability on the economical dimension [8].

3 Sustainability as a Quality Attribute

Within the research community sustainability is proposed as a QA with *resource consumption*, *greenhouse gas emissions*, *social sustainability*, and *recycling* as subcharacteristics [21, 22]. However, while the importance of relating EC to software products is acknowledged, there is still dividedness as a solution is also sought with existing QAs (cf. [12, 17, 18]. In [17], for example, 'performance efficiency' (ISO 25010) is transformed into 'energy efficiency' with three high-level issues ; energy

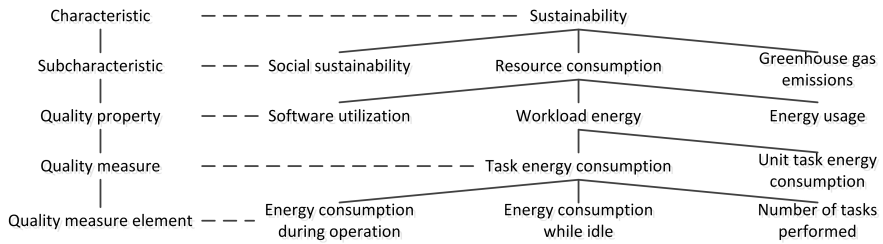

Fig. 2 A partial breakdown of the sustainability characteristic linked to the ISO 25010 standard.

behavior, capacity and resource utilization. Also 'energy efficiency' itself has been positioned as a QA [30].

Following the format of the existing ISO 25010 standard we continue by proposing sustainability as a QA and direct our focus specifically on the *resource consumption* subcharacteristic. While *resource consumption* closely resembles the existing 'resource utilization' subcharacteristic, i.e. a specific resource (energy) is utilized, there is a significant difference in focus between sole computational resources and what we described as 'sustainability' [22]. The (sub)characteristics are not mutually exclusive though, as the associated measures could be similar.

Since a SA allows or precludes nearly all QAs [2], the relation between sustainability as a QA and our research is explained. Following conventions of the ISO 25010 standard, resource consumption should decomposed into quality properties complemented by a measurement method to make the attribute measurable. From literature study [4, 12, 17–19], *software utilization*, *energy usage* and *workload energy* were distilled as potential quality properties:

- **Software utilization** is the degree to which resources specifically utilized on the account of a software product meet requirements.
- **Energy usage** is the degree to which the amount of energy used by a software product meets requirements.
- **Workload energy** is the degree to which the EC related to performing a specific task using a software product meets requirements.

The first two properties represent the low-level measurements, whereas the latter is used to characterize a software product in such a way that it facilitates discussion between stakeholders [12]. Although further research into this matter is required, for now we assume that these quality properties, representing four out of six metric types identified in [4], cover the resource consumption subcharacteristic.

3.1 Quality Measures for Energy Consumption

Unless (sub)characteristics can be directly measured, the measurable properties of a system (quality properties) can be quantified using quality measures and measure elements. Following the framework of the ISO 25010 standard consider the example in Fig. 2, where the 'sustainability' characteristic is broken down to the level of quality measure elements for the 'workload energy' property. To quantify 'workload energy', the *task energy consumption* quality measure was identified along with three quality measure elements for the measurement function.

In Table 1 a list is proposed of the quality properties, measures and measure elements identified for the 'resource consumption' subcharacteristic, including a definition of the quality measure and a measurement function containing quality measure elements. For example, the *task energy consumption* quality measure (measure for the energy consumed while performing a task) is calculated by subtracting the *idle EC* from the *EC while operating* and divide by the *number of tasks performed*. As the list is a starting point and by no means definitive, it could be changed or extended based on new insights.

Starting with software utilization we argue that the knowledge gained from performance research can be re-utilized, which is reflected through the fact that the presented measures (*CPUU*, *MU*, *NT* and *DT*) are common performance metrics providing insight into the behavior of hardware components. Compared to pure performance metrics however, the presented measures are explicitly related to software. Since our focus is on adjusting the software to become more sustainable, it is essential to know how the hardware is stressed under conditions dictated by the software.

Quality measures directly related to EC are described under the energy usage and workload energy properties. *SEC* is the most basic means of relating EC to software, i.e. measuring the total EC and subtracting the EC while idle. The measure closely resembles 'annual component consumption (ACC)' [17], without the inclusion of a specific time frame. The remaining properties, *UEC* and *RUEC*, can be derived using *SEC* and relate EC to a specific unit, e.g. separate elements or combinations thereof. *UEC* is a measure to allocate a portion of *SEC* to a defined unit and requires detailed performance data. *RUEC* puts the EC of a defined unit in perspective of the entire software instance and can be used to quickly identify outliers in EC.

Table 1 Quality properties, measures and measure elements for the resource consumption sub-characteristic.

Resource consumption		
Software utilization		
	CPU Utilization (CPUU)	Measure of the CPU load related to running the software. *current CPU load − idle CPU load*
	Memory Utilization (MU)	Measure of the memory usage related to running the software. $\frac{allocated\ memory}{total\ memory} \times 100\%$, *working memory*, *Private bytes*, *Virtual bytes*
	Network Throughput (NT)	Measure of the network load related to running the software. *Packages per second*, *sent bytes per second*, *received bytes per second*
	Disk Throughput (DT)	Measure of the disk usage induced by running the software. *Disk I/O per second*
Energy usage		
	Software Energy Consumption (SEC)	Measure for the total energy consumed by the software. *EC while operating − idle EC*
	Unit Energy Consumption (UEC)	Measure for the energy consumed by a specific unit of the software. $(\frac{Unit\ CPUU}{CPUU} \times \frac{Unit\ MU}{MU} \times \frac{Unit\ NT}{NT} \times \frac{Unit\ DT}{DT}) \times SEC$
	Relative Unit Energy Consumption (RUEC)	Measure for the energy consumed by a specific unit compared to the entire software instance. $\frac{UEC}{SEC} \times 100\%$
Workload energy		
	Task energy consumption (TEC)	Measure for the energy consumed when a task is performed. $\frac{SEC}{\text{\# of tasks performed}}$
	Unit task energy consumption (UTEC)	Measure for the energy consumed when a task is performed by a specific unit of the software. $\frac{UEC}{\text{\# of tasks performed}}$

Using *SEC* and *UEC*, also *TEC* and *UTEC* can be calculated provided that the stakeholder is able to define a task and knows the number of times this task is performed during a measurement. *TEC* provides insight in the EC to perform a specific task across units (e.g. functional elements) whereas *UTEC* considers the EC within the limits of a defined unit. If *UTEC* is related to a specific software component, the measure corresponds to the 'component consumption per unit of work (*CCUW*)' [17].

To increase the applicability of the measures, we decided to provide quality measure elements that can be either directly measured or derived from the total EC. For the measurement method this implies that performance monitoring tooling is required, ideally on the level of individual hardware components and processes, as well as tooling to perform EC measurements. For the latter both software and hardware solutions exist, capable of measuring at least the total power or EC for a system. Combining measurements from these sources in the measurement functions, provides the required information to quantify the subcharacteristics. Applying the measurement method might require more effort as environments become more complex. Shared resources, for example, require detailed performance measurements to allocate EC to specific instances of software.

3.2 Trade-offs between Quality Attributes

In relation to the other QAs, the possibility exits that trade-offs have to be made when conflicting goals arise. For example, keeping a log to maintain non-repudiation (security) could negatively affect the *TEC* of a software product. Making trade-offs however, should not be considered as an inhibiting factor. By making EC explicit, this aspect can be structurally included in a trade-off analysis. Rather than sustainability as an optional goal, a stakeholder can make a shift towards structurally relating sustainability to software products, i.e. sustainability by design. Although our research is focused on the application level, adjustments on a different level (i.e. infrastructure and middleware [10]) could turn out to be more appropriate.

A concrete trade-off related to EC is on the balance between demand and supply in resource allocation [39]. If more computational resources are available than required for a task, this should not automatically mean that extra resource should be assigned to this task (i.e. sustainability versus performance). Making trade-offs becomes more dynamic when service level agreements are in place. To minimize the total cost of ownership, software vendors and hosting parties strive for the lowest possible EC while still meeting agreements with customers. Effective resource management is essential to manage these environments [1].

4 Energy Consumption Perspective on Software Architecture

A common way to address the consequences of design decisions on a QA is via an architectural perspective, which is 'a collection of activities, tactics, and guidelines that are used to ensure that a system exhibits a particular set of related quality properties that require consideration across a number of the systems architectural views' [32].

Perspectives are a means to systematize the tasks of an architect, e.g. identify, test, and select architectural tactics to address cases when the architecture is lacking, and provide a framework to guide and formalize the process.

In this section we present the EC perspective including the applicability to views (including concerns), activities and tactics, following the detailed format presented by Rozanski and Woods [32]. As it is not possible to provide an exhausting set of guidelines, we recommend similar research (e.g. [12, 17, 28]) as guidance.

4.1 Viewpoint Catalog

To characterize the different views within an AD, views are grouped into viewpoints that focus on similar aspects within the design. Together, these viewpoints define the viewpoint catalog consisting of seven viewpoints (Fig. 3) [32] that can be used to create an AD of the software product focusing on different aspects of the system. Each of the viewpoints defines concerns of a stakeholder, such as requirements, objectives, intentions and aspirations, that the views following that viewpoint should address.

The system design is reflected in the *functional* viewpoint, the *information* viewpoint, and the *concurrency* viewpoint which focus on respectively the product's functionality, data aspect and the runtime. The *deployment* viewpoint defines the runtime environment for the software, complemented by the *operational* viewpoint defining the operation of the software when deployed. Implementation constraints for the software are defined in the *development* viewpoint. Finally, the *context* viewpoint defines economic and social aspects of EC in relation to the software design.

To develop an architectural perspective on EC, we address each of the viewpoints and explain how EC affects the viewpoint. For each viewpoint a key question is formulated, that addresses the insight that a viewpoint should provide in relation to EC. As these views cannot be seen in isolation, their consistency and interdependencies, as portrayed in the flow of the key questions (Fig. 3), are crucial. Although we acknowledge that not all viewpoints are required for each concern, given the novelty of the perspective all viewpoints are explained.

Context viewpoint Views in the context viewpoint focus on the environment of the software product, such as business drivers. The experience of the authors in the Dutch software industry is that increasingly more organizations have sustainability in their mission statement. As a consequence, customers of the software industry add sustainability, among others EC, demands to their tenders. This demand is twofold. On the one hand, there is an increased call for software to be developed in a sustainable manner, on the other hand there is the focus on EC of the software product itself. Thus, the contextual view should focus on answering how the software product can help in achieving an organizational sustainability strategy.

Key Question 1 *How can the software product architecture assist in achieving an organization's sustainability strategy?*

One way to contextualize a sustainability strategy is to portray it as strategic goals that should be met. For example, energy efficient software, does not only contribute

to sustainability goals, it also provides a means to lower the total cost of ownership for a software product, i.e. it influences the economic aspects (cf. [4]).

In our case study (DG), a lower total costs of ownership enables the software vendor to be more competitive in terms of pricing, and helps in realizing sustainability targets like reducing carbon emissions. In terms of the quality measures (Tbl. 1), a reduction of *SEC* or *UEC* indicates a lower power consumption and thus reduced carbon emission (not to mention the energy costs).

Operational viewpoint The operational viewpoint focuses on how the software is executed, and is where the quality measure elements (Tbl. 1) are measured. Changes from this viewpoint are often system-wide strategies that address operational concerns. Thus a first step to improve the EC, from this viewpoint, is to fine-tune the hardware configuration.

Key Question 2 *How can run-time aspects be fine-tuned to reduce EC?*

However, as software products typically run on diverse platforms, the architect should specify in this viewpoint which elements are to be recorded and how these elements can be combined into the different measure elements. This leads to a second key question in the operational viewpoint:

Key Question 3 *How can we measure the EC of the different nodes the software is executed on?*

For DG, in our limited scope, the architect specified to focus on the CPUU of the 'Interface.exe', 'Connector.exe' and 'Document.exe' processes which were executed on the same server. If the database had been within scope, MU and DT would have been added to the list for the database processes.

Deployment viewpoint The deployment view portrays the actual hardware environment of the software in terms of the processing nodes, data storage nodes and the network topology of how these nodes are connected. Whereas in the operational viewpoint the focus is on the complete system, the deployment viewpoint assists

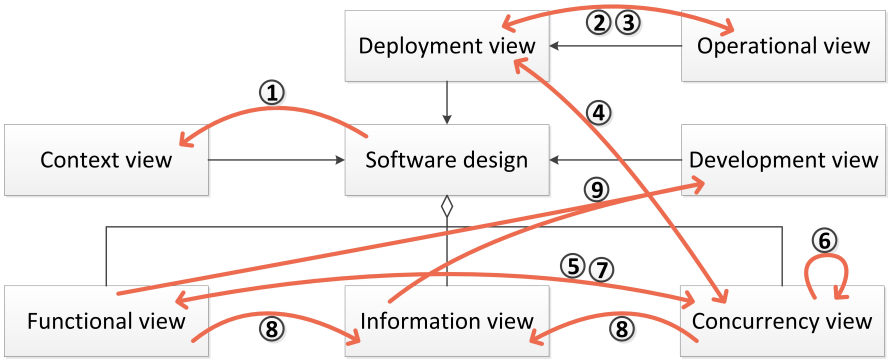

Fig. 3 Viewpoint catalog after [32], expanded with the flow of key questions to address EC concerns.

the architect in relating the measures to the individual processes. Knowing which processes run on what hardware provides the architect with valuable insights where (EC) measurements should be performed. This aspect directly gives the key question the deployment viewpoint should address:

Key Question 4 *Which processes run on what hardware?*

For example, DG can be deployed on multiple servers: one data storage server and one or more processing nodes that each run parts of the application. Once the architect creates a mapping between the individual processes (in this case, the executables) to the different hardware nodes, the measures can be assigned and related to the different components DG consists of.

Concurrency viewpoint The concurrency viewpoint shows how functional elements map onto concurrency units, e.g. processes and threads, and forms the bridge between functional elements and their deployment. Hence, the first key question is directly related to this insight:

Key Question 5 *How do the functional elements map onto processes?*

Additionally, in this viewpoint, the parts of the system are identified that can be executed concurrently. Concurrent processes potentially add to a reduced EC by means of performance efficiency, depending on the coordination and control mechanism required to do so. Again, like the operational viewpoint, a second key question can be formulated:

Key Question 6 *What processes can be executed concurrently without increasing the resource consumption related to their coordination and control?*

To calculate *UEC* for the 'connector' element of DG, the concurrency view shows the processes that comprise this element and thus should be considered for their resource consumption. By relating this mapping to the deployment viewpoint (key question 4) we can distill the hardware that should be monitored.

Functional viewpoint The functional viewpoint is part of the system design itself. It defines the elements of which the software is composed and the functions and features these elements offer. This viewpoint is essential to define the boundaries of a software product and identify the separate elements (and functions) of which the energy consumption is of interest. Ideally, after measurements have been performed, the view would include an indication of the EC per element:

Key Question 7 *How much energy does each function consume?*

The DG workflow (Fig. 1) is executed by three functional elements; 'composer', 'connector' and 'generator'. If an EC figure can be assigned to these elements, despite their deployment, a stakeholder can direct efforts to reduce the EC to where they are needed most. In the case of DG, regardless of deployment, one question of interest is "How much energy does the generation of one document cost?". As multiple elements are involved in this task, EE solutions can be directed to those functional elements that stand out in terms of EC,

Information viewpoint In the information viewpoint, the information is identified that is used by and communicated between functional elements. From an EC perspective, and efficiency in general, it is essential to have the right information on the right place at the right time:

Key Question 8 *How can the information flow be optimized to increase EE?*

For DG, critical data sections can be identified that might affect the processes in terms of efficiency. If, for example, the data is locked to show a preview, replicating the data could prevent the process from coming to a complete standstill. Of course, in this case, other measures should be introduced to solve any conflicts that might arise.

Development viewpoint The development viewpoint is a starting point to support the development process and contains aspects of interest to those stakeholders involved in building the system. An overview of elements, for example, simplifies the context for developers and prevents them from having to cope with the complexity of the entire application. A developer is facilitated to focus on the code that drives the EC.

Key Question 9 *What green algorithms can be applied to the software and where should they be applied?*

In the case of DG, the EC of the 'connector' element could be found disproportional. A proper AD could simplify the context for developers tackling the issue.

4.2 Perspective Activities

Following [32], we provide a set of activities (Fig. 4) to apply the EC perspective. By applying the perspective a stakeholder has a means to analyze and validate the qualities of an architecture and drive further architectural decision making. The activities follow a 'Plan-Do-Check-Act' cycle where the iterations are focused on whether the software meets certain requirements.

1. **Capture energy requirements;** Requirements form the basis for change in relation to SA [2] and should be considered when strategical, economical or customer motives are present. Energy requirements can be formulated like other requirements, however it might prove difficult to translate the requirements into quantitative goals. Cross-checking the goals with stakeholders is essential to ensure the software will fulfill the requirements.
2. **Create energy profile;** An energy profile of the software provides the stakeholder with an objective starting point and benchmark to identify ''hot spots' and determine whether the desired results have been achieved. Creating the profile requires

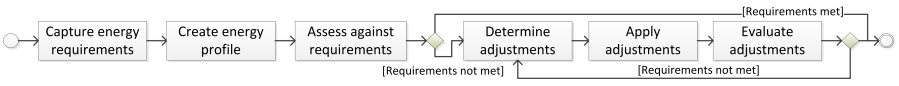

Fig. 4 The activities to apply the perspective to software architecture.

EC and performance measurements and can be visualized by creating an overlay for the AD, e.g. Fig. 7 is annotated with EC figures for the generator element. Mind that profiling an application could be time-consuming where a profile for the elements within the scope of current requirements could be sufficient.

3. **Assess against requirements;** Using the energy profile an assessment should be performed on whether the software meets the requirements. i.e. whether the quantified goals are met. If requirements are not met, the assessment should show what quantitative goals are not met and the (software) aspects that are directly related to these goals. Ideally this activity should be performed periodically or, more specifically, when the application has changed.
4. **Determine adjustments;** If required, adjustments should be determined to meet the requirements. Tactics (see Sect. 4.3), patterns and other known solutions should be considered that affect specific 'hot spots' signaled by the quality measures. To guide the selection process for selecting the right adjustments, a business case should be created projecting the expected costs (e.g. development time and costs) and benefits (i.e. EC savings over time). Since the area of green tactics is still relatively immature, the perspective can also be used to investigate the consequences of tactics. Using the AD, and relevant viewpoints, stakeholders can identify and control possible (unwanted) effects on related architectural elements.
5. **Apply adjustments;** Applying the adjustments depends on the nature of the adjustments. Adjustments related to the infrastructure, for example, can be applied on-the-fly by an administrator with little effort. Redesigning, or even adjusting, the software on the other hand, often requires development resources and capacity planning upfront. The resources required to apply an adjustments should be included accordingly in the business case for the adjustment.
6. **Evaluate adjustments;** Last is determining whether requirements are met and assuring no unwanted effects are brought about. Evaluation requires the stakeholder to perform measurements in the newly created situation and compare the figures against the benchmark (i.e. the energy profile). A stakeholder should give a statement on whether the adjustments are satisfactory.

4.3 Green Architectural Tactics

To address concerns for a software product on the level of the SA, tactics are applied. A tactic is a decision that influences the control of a QA [2] and is a design option that helps the architect in realizing a desired property for a system. In relation to EC, there is still work to be done to find a set of tactics that are able to satisfy the concerns. Consequently, the presented tactics are by no means a definitive list but should be considered as a source of inspiration for green software efforts.

In [30] a catalog is presented consisting of the *energy monitoring*, *self-adaptation* and *cloud federation* categories. The categories are aimed at respectively collecting power consumption information on infrastructure and software component level, optimizing during run-time and finding the most energy efficient services to perform a task, and include several tactics that address energy efficiency in the cloud. Even

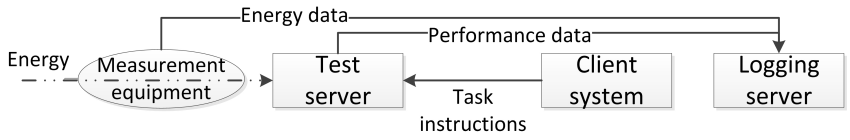

Fig. 5 Setup of the test environment used to perform the case study.

though the tactics are explained specifically in a cloud computing context, they could prove valuable for software in general.

Increase modularity; In terms of database calls, software consisting of fewer modules could require less calls while significantly more data is transferred per call. When software consists of more modules, an increase in database calls could be observed with the potential that less data is transferred per call, i.e. the calls are more fitted to the process at hand. In this case less CPU capacity is required for processing the call, lowering the EC per call. This tactic holds under the assumption that the increased disk usage has a marginal impact on the EC figures.

Network load optimization; Although modularity can positively affect the EC of software [5], more modules also implies a higher communication load. When the number of modules increases, depending on the deployment, the communication load that is induced on the infrastructure also increases. Although difficult to quantify in terms of EC, a positive effect is expected when the communication load is reduced.

Increase hardware utilization [12]; Ineffective use of hardware is a common source for energy inefficiency and is one of the triggers to consolidate the number of active servers within a data center. From an EC point of view there is less hardware in absolute terms reducing the idle energy consumption and the available hardware is used more effectively. Variations in deployment imply that the software is able to cope with variation and thereby could impose redesigning the software.

Concurrency architecture variation [38]; In this specific case the Half Synchronous / Half Asynchronous and the Leader / Followers concurrency architectures are compared and a significant difference was found in the advantage of the first. Further investigation is required to test the generalizability of this finding, but the tactic could prove useful for individual software instances.

5 Case Study: Applying the Perspective in Practice

To assess the perspective's applicability, a case study was performed in which the activities were applied to DG. Again following the red line throughout this paper, the main concern during the case study was to reduce the EC of DG. To address this concern we want to divide DG into separate, related elements and monitor the energy consumption of these elements accordingly. Consequently, we consider the functional, concurrency and deployment viewpoint in our case study (Fig. 7).

1. **Capture energy requirements;** We chose to focus on the main functionality of DG and investigated an activity to generate 5000 documents on house rental, where the generation of each single document is considered as a separate task. In

relation to EC we formulate the (non-functional) requirement for DG to consume less energy while performing the specified task.

2. **Create energy profile;** In our case study, DG was installed in a test environment consisting of a test server, logging server, client system and measurement equipment (Fig. 5). DG was installed on the test server[2] which consequently was the system to perform measurements on. The measurement equipment, a WattsUp? Pro (WUP) capable of measuring the total power drawn by an entire system with a one second interval, measured the power drawn by the test server and performance data was collected using Perfmon, a standard performance monitoring tool with Microsoft Windows. A different deployment could require a more software intensive approach using, e.g., energy profilers [15]. Finally, the client system was used to trigger the required activity and data was collected remotely (i.e. without human interference) using the logging server. To ensure consistency across measurements (e.g. constant room temperature), the test server was located in a data center.

   After configuring DG a protocol was followed to perform measurements:
   - Clear internal WUP memory.
   - Close unnecessary applications and services on the test server.
   - Start WUP and Perfmon measurements.
   - Perform specified task using client.
   - Collect and check Perfmon and WUP data from logging server.

   In total 22 measurements were performed divided over six series, of which 19 measurements were considered valid. On average DG required 41 minutes and 49 seconds to generate the documents, with a *SEC* of 17560 Joule (J) (standard deviation 3577 J) and an average *TEC* of 3.51 J per document.
3. **Assess against requirements;** The assessment consisted of creating a heat map to discover 'hot spots'. Recall (Sect. 1.1) that the 'Generator' element is responsible for the actual document generation. Mapping the measurements on the AD (Fig. 7), performance data shows a 49% *CPUU* of Document.exe, with an average utilization rate of 50.7% and 7.4% for the two available cores. The other

[2] HP Proliant DL380 G5, Intel Xeon E5335 CPU, 800GB local storage (10.000 rpm), 64GB PC2-5300, 64 bit MS Windows Server 2008R2 (Restricted to 2 cores), VMware vSphere 5.1

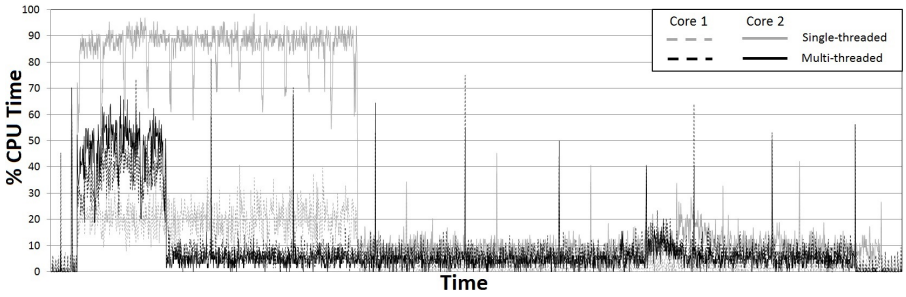

Fig. 6 Comparison of CPU activity (2 cores) of the test server during a measurement before (single-threaded) and after (multi-threaded) applying the architectural change.

processes (Configuration.exe and Connector.exe) did not appear active. Since no quantitative goal was formulated for the requirement, e.g. consume at most '*X*' Joule per document, the EC profile was labeled as benchmark.

4. **Determine adjustments;** With the 'Generator' element identified as a EC 'hot spot' and the *CPUU* imbalance as the possible driver, the tactic to increase hardware utilization could potentially resolve our issue. Discussing the options with stakeholders (i.e. architect and developer) brought to light DG's inability for multi-threading as possible cause.
   The business case for making DG multi-threaded encompassed an estimation of the development time and costs and the projected benefits. Estimations of the benefits were made using the EC range of the application server (i.e. EC with full CPU load - EC with idle CPU) and included a multiplication for each of the separate DG installations that would benefit from these changes.
5. **Apply adjustments;** As the payback period for the business case turned out relatively short (weeks), in collaboration with the developer DG was made multi-threaded, changing the SA to evenly divide the load over the available cores (shown on the right hand side of Fig. 7). The 'balancer' in the AD operates according to the broker pattern.
6. **Evaluate adjustments;** To evaluate the adjustment, 33 (out of 36) valid measurements were obtained (divided over seven series) following the described protocol. On average DG required 39 minutes and 14 seconds to generate the documents with a *SEC* of 5782 J (std. dev. 1647 J). Consequently *TEC* was reduced with 67.1% to an average 1.16 J per generated document and a significant decrease in CPU activity was perceived (Fig. 6). The *CPUU* for Document.exe decreased to an average 19.2%, divided 12.6% and 15.1% respectively, yielding higher gains than projected in the business case. A critical note though; as the database server was considered out of scope we did not include any effects on this hardware.

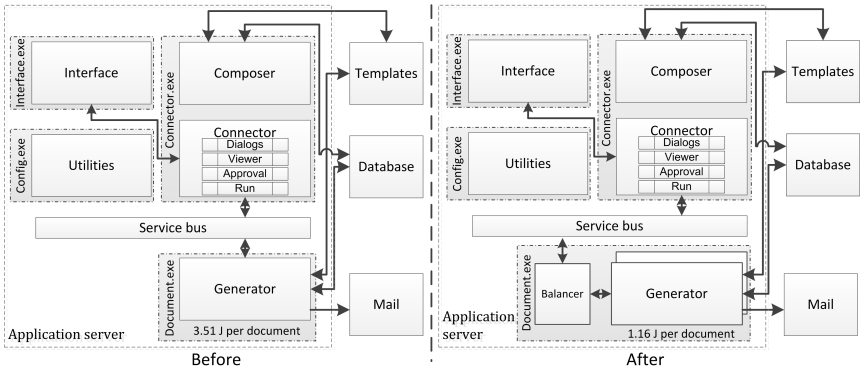

Fig. 7 The functional architecture of DG for subsequent releases including a concurrency view (executables) and deployment view (installed on a single application server).

5.1 Threats to validity

With regard to the validity of the case study, an evaluation is performed following the threats as identified in [33].

The *construct validity* considers whether the correct measures were identified for the object under study. In Section 3 we introduced sustainability as a QA and unraveled this attribute to low level quality measures related to EC. During this process we followed existing literature on both EC measurements and performance research to operationalize the quality measures in terms of being able to perform measurements. The resulting breakdown is similar to others in this field of research, where performance measurements are used to relate EC to the software or elements thereof.

In light of the *internal validity*, despite careful preparations, due to the behavior of services we can not be 100% certain that DG was solely responsible for the load on the test server. Therefore each individual measurement was checked for such processes using performance data. Also, due to time constraints, we could not balance the number of measurements between creating the energy profile and evaluating the redesign. At the start of the case study we lacked experience with configuring DG, e.g. we experienced firewall issues, resulting in a lower number of measurements. During evaluation we were more familiar with the case and relatively more valid measurements were obtained.

A threat to the *external validity* is the fact that the case study was performed in a separate test environment containing specific hardware. Given the relation between hardware and EC, different hardware could provide different findings. Although the EC figures could differ in absolute terms, since an actual commercial software product was used and installed according to production standards, we argue that the results are not specific to our environment.

Finally, *reliability* is concerned with the results, data and data analysis being dependent on specific researchers; i.e. replicating the case study under similar conditions should yield similar results. To this end the measurements within the case study were performed by following a strict, openly described protocol. A difference could occur with the statistical analysis of the data since, given the nature of the data, we decided to process the data 'as is' where others might prefer to normalize the data.

6 Conclusion

In our quest to reduce the EC of the ICT industry through the software, we set out to investigate how EC can be positioned within the scope of SA. We started out by positioning sustainability as a QA and identified measurable, low-level elements for the 'resource consumption' subcharacteristic. The presented quality properties, measures and measure elements provide a means to quantitatively evaluate the EC of a software product by combining performance and EC measurements. Also, by considering sustainability as a QA, a stakeholder has a practical means to structurally consider the sustainability of a software product; i.e. sustainability by design.

To actually relate EC to SA, an EC perspective was constructed that enables stakeholders to identify, measure and analyze the EC of architectural elements. We identi-

fied key questions for the viewpoints related to software design, described concerns that can be addressed, provided architectural tactics and the activities to apply the perspective in practice. Using the perspective, EC can be considered during the design phase of the software product extending the control a stakeholder has over the desired quality properties. From hardware, through architecture down to code level.

As an initial validation of the perspective, a case study was performed in which the perspective was applied to a commercial software product. The energy profile that was created directed us to the architectural element that was the main driver behind the EC and through an architectural change we managed to reduce the energy consumption of DG with 67.1% per generated document. Considering the frequency at which this task is performed and the number of DG deployments, the savings could add up significantly from an organizational dimension.

However, we do acknowledge that the presented perspective is by no means as mature as other perspectives related to QAs. Based on the results presented in this paper, several directions for future research can be identified. First is to further complete, i.e. by providing problems, pitfalls design patterns, tactics and checklists, and improve the perspective through practical experience. Second is to investigate how the work presented in this context can be translated to cloud environments. A final direction is to investigate, in depth, how insights gained from the architectural perspective can be translated to guidelines for software development.

Acknowledgements We would like to thank the DG team for providing us with an interesting case and help to improve the case study. Also we thank the PhD group on software architecture at Utrecht University for their feedback and input while executing this research. Furthermore, as this work is an extended version of a published paper [16], we would like to thank the reviewers for their valuable comments.

References

1. Ardagna, D., Panicucci, B., Trubian, M., Zhang, L.: Energy-aware autonomic resource allocation in multitier virtualized environments. Services Computing, IEEE Transactions on **5**(1), 2–19 (2012)
2. Bass, L., Clements, P., Kazman, R.: Software Architecture in Practice. SEI Series in Software Engineering. Pearson Education (2012)
3. Becker, C., Penzenstadler, B., Chitchyan, R., Seyff, N., Duboc, L., Venters, C.C., Easterbrook, S.: Sustainability design and software: The karlskrona manifesto (2015)
4. Bozzelli, P., Gu, Q., Lago, P.: A systematic literature review on green software metrics. Tech. rep., Technical Report: VU University Amsterdam (2013)
5. te Brinke, S., Malakuti, S., Bockisch, C., Bergmans, L., Akşit, M.: A design method for modular energy-aware software. In: Proceedings of the 28th Annual ACM Symposium on Applied Computing, pp. 1180–1182. ACM (2013)
6. Cappiello, C., Matera, M., Picozzi, M.: A ui-centric approach for the end-user development of multi-device mashups. ACM Trans. Web **9**(3), 11:1–11:40 (2015). DOI 10.1145/2735632
7. Capra, E., Francalanci, C., Slaughter, S.A.: Is software green? application development environments and energy efficiency in open source applications. inform software tech **54**(1), 60 – 71 (2012)
8. Chasin, F.: Sustainability: Are we all talking about the same thing state-of-the-art and proposals for an integrative definition of sustainability in information systems. In: ICT for Sustainability 2014 (ICT4S-14). Atlantis Press (2014)
9. Ebert, C., Brinkkemper, S.: Software product management an industry evaluation. Journal of Systems and Software **95**(0), 10 – 18 (2014). DOI http://dx.doi.org/10.1016/j.jss.2013.12.042
10. Ferreira, A.M., Pernici, B.: Managing the complex data center environment: an integrated energy-aware framework. Computing pp. 1–41 (2014)

11. Ferreira, M.A., Hoekstra, E., Merkus, B., Visser, B., Visser, J.: Seflab: A lab for measuring software energy footprints. In: GREENS, pp. 30–37. IEEE (2013)
12. Grosskop, K., Visser, J.: Identification of application-level energy optimizations. Proceeding of ICT for Sustainability (ICT4S) pp. 101–107 (2013)
13. Hindle, A.: Green mining: a methodology of relating software change and configuration to power consumption. Empirical Software Engineering pp. 1–36 (2013). DOI 10.1007/s10664-013-9276-6
14. Ihm, S.Y., Nasridinov, A., Lee, J.H., Park, Y.H.: Efficient duality-based subsequent matching on time-series data in green computing. The Journal of Supercomputing **69**(3), 1039–1053 (2014)
15. Jagroep, E., van der Werf, J.M.E., Jansen, S., Ferreira, M., Visser, J.: Profiling energy profilers. In: Proc. of the 30th Annual ACM Symposium on Applied Computing, pp. 2198–2203. ACM (2015)
16. Jagroep, E., van der Werf Jan Martijn E. M. Spauwen, R., Blom, L., van Vliet, R., Brinkkemper, S.: Profiling energy profilers. In: 9th European Conference on Software Architecture (2015)
17. Kalaitzoglou, G., Bruntink, M., Visser, J.: A practical model for evaluating the energy efficiency of software applications. In: ICT for Sust. 2014 (ICT4S-14). Atlantis Press (2014)
18. Kern, E., Dick, M., Naumann, S., Guldner, A., Johann, T.: Green software and green software engineering–definitions, measurements, and quality aspects. J. on Inf. and Comm. Tech. p. 87 (2013)
19. Kipp, A., Jiang, T., Fugini, M., Salomie, I.: Layered green performance indicators. Future Generation Computer Systems **28**(2), 478 – 489 (2012). DOI http://dx.doi.org/10.1016/j.future.2011.05.005
20. Kritikos, K., Pernici, B., Plebani, P., Cappiello, C., Comuzzi, M., Benrernou, S., Brandic, I., Kertész, A., Parkin, M., Carro, M.: A survey on service quality description. ACM Comput. Surv. **46**(1), 1:1–1:58 (2013). DOI 10.1145/2522968.2522969
21. Lago, P., Kazman, R., Meyer, N., Morisio, M., Müller, H.A., Paulisch, F., Scanniello, G., Penzenstadler, B., Zimmermann, O.: Exploring initial challenges for green software engineering: summary of the first greens workshop, at icse 2012. ACM SIGSOFT Softw. Engin. Notes **38**(1), 31–33 (2013)
22. Lago, P., Kocak, S.A., Crnkovic, I., Penzenstadler, B.: Framing sustainability as a software quality property. Commun. ACM p. 9 (2015)
23. Mills, M.P.: The cloud begins with coal: an overview of the electricity used by the global digital ecosystem. Tech. rep., Digital Power Group (2013)
24. Morales Ruiz, A., Daniels, W., Hughes, D., Grothoff, C.: Cryogenic: Enabling power-aware applications on linux. In: ICT for Sustainability 2014 (ICT4S-14). Atlantis Press (2014)
25. Murugesan, S.: Harnessing green it: Principles and practices. IT Prof. **10**(1), 24–33 (2008)
26. Noureddine, A., Rajan, A.: Optimising energy consumption of design patterns. In: International Conference on Software Engineering (2015)
27. Noureddine, A., Rouvoy, R., Seinturier, L.: A review of energy measurement approaches. ACM SIGOPS Operating Systems Review **47**(3), 42–49 (2013)
28. Noureddine, A., Rouvoy, R., Seinturier, L.: Monitoring energy hotspots in software. Automated Software Engineering pp. 1–42 (2015)
29. Pathak, A., Hu, Y.C., Zhang, M.: Where is the energy spent inside my app?: fine grained energy accounting on smartphones with eprof. In: Proc. of the 7th ACM european conf. on Computer Systems, EuroSys '12, pp. 29–42. ACM, New York, NY, USA (2012). DOI 10.1145/2168836.2168841
30. Procaccianti, G., Lago, P., Lewis, G.A.: A catalogue of green architectural tactics for the cloud. In: Maint. and Evol. of Service-Oriented and Cloud-Based Systems (MESOCA), 2014 IEEE 8th Int'l Symp. on the, pp. 29–36 (2014). DOI 10.1109/MESOCA.2014.12
31. Procaccianti, G., Lago, P., Vetro, A., Fernández, D.M., Wieringa, R.: The green lab: Experimentation in software energy efficiency. In: Proc. of the 37th Int. Conf. on Software Engineering (ICSE)
32. Rozanski, N., Woods, E.: Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives. Addison-Wesley (2011)
33. Runeson, P., Höst, M.: Guidelines for conducting and reporting case study research in software engineering. Empirical software engineering **14**(2), 131–164 (2009)
34. Steigerwald, B., Agrawal, A.: Green software. Harnessing Green IT: Principles and Practices p. 39 (2012)
35. Sun, Y., Zhao, Y., Song, Y., Yang, Y., Fang, H., Zang, H., Li, Y., Gao, Y.: Green challenges to system software in data centers. Frontiers of Comp. Sc. in China **5**(3), 353–368 (2011)
36. Xu, L., Brinkkemper, S.: Concepts of product software. eur j inform syst **16**(5), 531–541 (2007)
37. Zhang, G., Zhang, K., Zhu, X., Chen, M., Xu, C., Shao, Y.: Modeling and analyzing method for cps software architecture energy consumption. Journal of Software **8**(11) (2013)
38. Zhong, B., Feng, M., Lung, C.H.: A green computing based architecture comparison and analysis. In: Proc. of the 2010 IEEE/ACM Int'l Conf. on Green Computing and Communications & Int'l Conf. on Cyber, Physical and Social Computing, pp. 386–391. IEEE Computer Society (2010)
39. Zhu, H.S., Lin, C., Liu, Y.D.: A programming model for sustainable software. ICSE (2015)