Green Software Products

Erik Jagroep

Green Software Products
ISBN: 978-90-393-6840-4

Green Software Products

Duurzame Software Producten
(met een samenvatting in het Nederlands)

Proefschrift

ter verkrijging van de graad van doctor aan de Universiteit Utrecht op gezag van de rector magnificus, prof. dr. G.J. van der Zwaan, ingevolge het besluit van het college voor promoties in het openbaar te verdedigen op maandag 18 september 2017 des middags te 2.30 uur door

Erik Arijender Jagroep

geboren op 7 maart 1987
te Leidschendam

Promotor: Prof. dr. Sjaak Brinkkemper

Co-promotor: Dr. ir. Jan Martijn E. M. van der Werf

This research was financially supported by Centric Netherlands B.V.

Acknowledgements

At the time of writing it has been approximately six years since I've started my PhD, and little did I know back then what this endeavour would bring me. In a rapidly changing domain that drives the digitalization of society and is focused on delivering high performance ICT solutions, being green often makes you the odd one out. Seeing the domain develop through the years and thoroughly enjoying my small role in this development, convinced me to stay on this path and keep tackling any challenges I would come across. After a rough start I got up to pace and the publications confirmed that I was on the right track. Looking back, I have no regrets of starting this endeavour and am proud to present the results in this dissertation.

Of course this research would not have been possible without a number of people that played a key role throughout my research. First of all, I would like to thank Centric for providing me with the unique opportunity of conducting my PhD research combined with a position that also allowed me to gain experience in industry. Specifically, I would like to thank Rob van Vliet and Leen Blom for their continuous support throughout the years. I've learned a lot from our discussions, both academically and as a person, and am very grateful for all your efforts to make this research possible. Also I explicitly want to mention Hadewijch, Quirijn, Tom, John, Bart, Chris, Alexander, Frank and Ben. Thank you for your continued support in these past years.

Second, I would like to thank my promotor, Sjaak Brinkkemper, for this opportunity, the insightful discussions we had throughout the years and keeping faith in a successful outcome of the research. The same holds for my co-promotor, Jan Martijn van der Werf, your move to Utrecht University was the official start sign for my publication stream. Who knew that a model train fanatic, vocally skilled computer scientist was exactly what my research needed. Your guidance and support has been invaluable.

My gratitude also goes to all other Centric and Utrecht University colleagues that were in one way or another involved with this research, but thanking everyone individually would require a sizable appendix on its own.

However, I will make an exception for Garm, Vincent, Jaap, Kevin, Willem, Wienand, Ravi, Amir, Michiel, Leo, and all other PhDs I've come across during the years. Thank you for providing me with a healthy dose of distraction to stay sharp while writing papers.

Finally, my friends and family. Thank you for your encouragement, believing in me and supporting me every step of the way. No more paper deadlines, experiment planning around social occasions or data collecting at the diner table. From now on I will have a lot more time to spend with you all. To Rahat, Imran, Friso, Priyish and Melle: now you are free to kidnap me and drive off to any city you like.

Last but not least, Maaike; this research has often been as demanding for you as it has been for me. Thank you for your love and support, and for taking care of me when I forgot to take care of myself. You inspired me to succeed.

Erik Jagroep,
September 2017

Contents

1 Introduction

In the last decades sustainability has become a booming topic of interest. Since 1987, with the report of the world commission on environment and development by the Brundtland Commission [18], a global strive is being pursued towards sustainable development. With the recent Paris Agreement [37], the strive has been reinforced by building a bridge between current policies and climate-neutrality.

The strive to become more sustainable, which is to meet the needs of the present without compromising the ability of future generations to satisfy their own needs [18], has increased the awareness on the role Information and Communication Technology (ICT) can play in fulfilling this strive. Greening by ICT positions ICT as an enabler for other industries to become more sustainable, whereas greening of ICT aims at improving the sustainability of the ICT solutions themselves [46, 50]. The importance of the latter is stressed by a doubling of the energy consumption of the ICT industry between 2000 and 2006, equaling approximately 1.5% of the energy consumption of the United States [17], and growth to at least 10% of the worlds' energy consumption in 2010 [95].

Contributing to the growing energy consumption of the ICT industry are trends like cloud computing, software-as-a-service (SaaS) and increased smartphone usage. The changes could provide a more sustainable alternative compared to a previous setting, e.g. scale advantages with cloud computing, but also present new challenges to the industry [12, 13]. In addition, following the Jevons paradox [1], increased energy efficiency could lead to more energy consumption in absolute terms. The expected growth and increasing growth rate raise concerns with respect to the greenhous gas emissions induced by the ICT industry [29].

1.1 Green ICT

Multiple research fields relate ICT with sustainability [50]. With contributions ranging from monitoring the environment to understanding and using ICT as a transformational technology, it is clear that greening *by* ICT is already a prominent topic on the research agenda. However, of the identified research fields only 'Green ICT' explicitly focuses on the greening *of* ICT by reducing the environmental impact of ICT hardware and software itself [50].

According to Murugesan [99], a holistic approach is required to comprehensively and effectively address the environmental impact of ICT. The approach identifies four paths, i.e design, manufacture, use and dispose (see Fig. 1.1), that cover the life cycle of electronic equipment. While valuable, the approach exemplifies a concern in the field of green IT by solely considering a hardware perspective on sustainability [81, 130]. Software provides instructions to the hardware and as such can be considered as the true consumer of energy [140]. Specifically in the design and use paths, the measures provided with the approach seem to neglect the potential contribution of software towards sustainability.

The focus on hardware can be explained, as hardware improvements are the low-hanging fruit in greening of ICT: the energy consumption of hardware is directly measurable and hardware is relatively tangible. Following the developments of the industry, every new generation of hardware increases its energy efficiency and delivers more 'power per Watt'. Moreover, hardware renewal is a standard business process for most organizations in industry. However, although optimizing and renewing hardware is essential, a single software component can thwart all power management benefits built into the hardware [139]. Consequently, hardware and software should be considered as two sides of the same coin that require optimization in synergy to effectively influence the energy consumption of ICT solutions.

A second motivation to focus on software stems from a more societal consideration and corresponding market demands. Corporate Social Responsibility (CSR) drives an organization to make more sustainable choices and reduce their overall environmental impact. In relation to ICT, we observe a change in customer demands in the Netherlands. Predominantly in the public sector, likely driven by enforced regulations and policies, municipalities and other governmental agencies increasingly realize that hardware is only part of the solution and formulate sustainability requirements for the software. In tenders, specific clauses could prevent a hardware-based approach to continuously reduce the environmental impact of ICT solutions. Consequently, green ICT

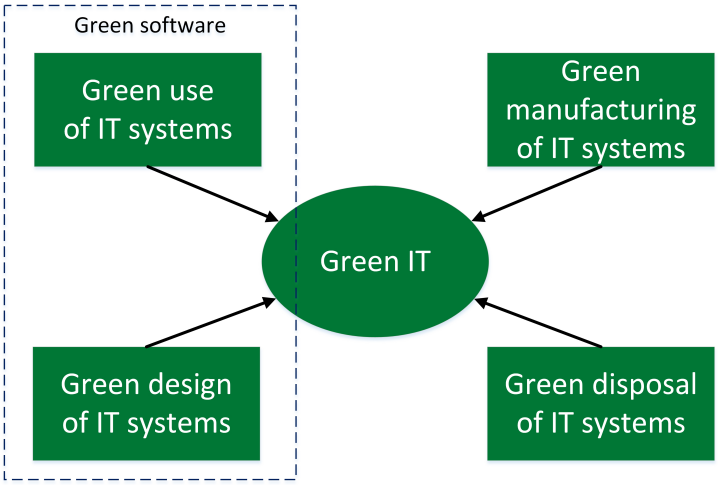


Figure 1.1: Holistic approach to green ICT, after [99].

becomes a concern that should be addressed by software as well.

To concretize the software effect and explain the relation between power and energy, consider the server tests performed by Beckett and Bradfield [9]. In their active idle states the Dell servers drew 1647 Watt of power, compared to 4797 Watt when stressed to their maximum capacity. The 3150 Watt difference is the power consumption range driven by software activity. Put otherwise, the software can account for 66% of the power consumed by the servers and thereby have a significant impact on the energy costs of the hardware.

In relation to energy, which is the power consumed over time, the software can also influence how a specific workload is handled. For example, a calculation can be designed to stress the servers to 100% for a short period of time or 40% for a longer period of time. In the first design, 10 seconds of calculating consumes 47.970 Joules (10×4797). With the second design, assuming linear scaling of the power consumption, 30 seconds of calculation consumes 87.210 Joules ($30 \times (1647 + (0,4 \times 3150))$). Hence, a low power consumption does not always imply the most sustainable, i.e. energy efficient, solution.

The topic of this dissertation is to facilitate the contribution of software towards sustainability. Facilitating implies that industry is in control of the energy consumption of software and can realize software-based sustainability contributions on a structural basis. Since energy efficient software encompasses non-functional requirement [6], software architecture provides a good starting

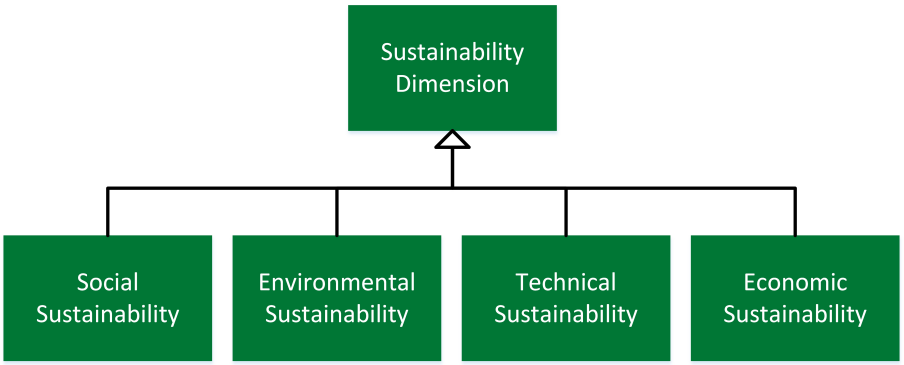


Figure 1.2: The four sustainability dimensions for software-intensive systems, after [83].

point for this purpose. We aim to shift the focus from solely hardware towards a synergy between hardware and software and enter new terrain with respect to software production.

1.2 Green Software

Green and sustainable software is defined as software whose development, deployment, and usage results in minimal direct and indirect negative impacts or even positive impacts on the economy, society, human being and the environment [102]. To achieve sustainability, green software engineering specifies the strive to optimize software solutions with regard to their energy consumption [104]. In this case optimization refers to minimizing the energy consumption while the software performs its tasks.

Both definitions acknowledge the impact of software, but overlook the ability of a system to preserve its functionality over a period of time [50]. Preservation over time implies that the software facilitates change. For example, to positively impact human being the software is required to satisfy changing needs. According to Lago et al. [83] four related dimensions have to be satisfied to achieve sustainability with software-intensive systems: social, environmental, technical and economic (Fig. 1.2).

Consider a Software Producing Organization (SPO), e.g. independent software vendor or open-source foundation, that reduces the energy consumption of software by replacing existing functionality with functionality provided in

a specific software framework. A reduced energy consumption positively contributes towards the environmental dimension, whereas a potentially lowered total cost of ownership positively contributes to the economic dimension. On the other hand, limited control over or discontinued support of the framework could harm the long-term use and evolution of the software (i.e. the technological dimension). This limits the systems' capacity to endure. Also, the social dimension could be negatively affected in terms of employee engineering competencies.

The simplified example shows that a focus on the environmental dimension, like with green software engineering, also affects the other dimensions characterizing sustainability. To truly make software sustainable, SPOs are required to make informed decisions on each dimension and potentially even trade-offs between dimensions. In the example, the decision to include a framework could be an intended trade-off between the technical and economic dimensions. Deciding to rewrite the existing code could have been a more sustainable solution, showing that the balance between dimensions potentially affects system design.

In the remainder of this dissertation, following the definitions, we use the terms 'green software' and 'sustainable software' interchangeably to indicate software that has the capacity to endure over time and has a minimal negative impact on the environment by minimizing energy consumption.

1.2.1 Software Products

According to Xu and Brinkkemper [150], software can be classified in four categories according to *what* is sold and the *number of copies* that are sold (Fig. 1.3). Sustainability is important in all categories, but the impact of having green software significantly varies per category. Micro-programs and tailor-made software are characterized by a one-of a-kind application, which, despite the potential scale of tailor-made software, limits the impact of green software to a small number of instances. With embedded software, considerable savings could be realized by optimizing embedded ICT. However, as the savings are realized by the application of smart technologies and devices, e.g. smart meters and sensors, the contribution of the software itself is limited [29].

In contrast, with product software the software itself is the main product and thus object for optimization. A change in a software product finds its way to each deployment, thereby multiplying the potential impact on energy efficiency with each installation. In [51] a decrease of 0.25 W with four million installations, e.g. with a browser, is presented to save the energy equivalent to that of an American household per month. Showing that even the smallest

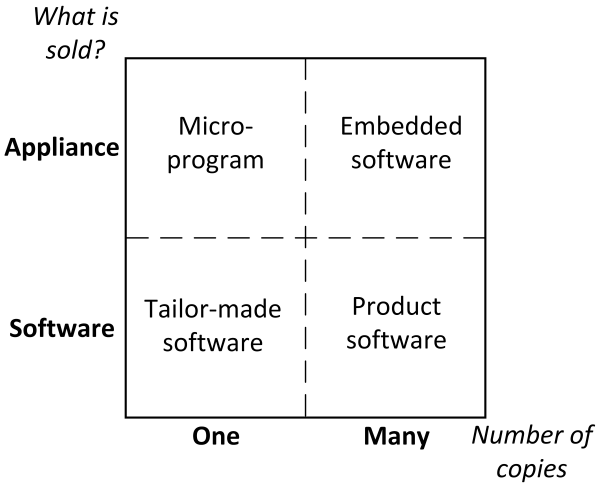


Figure 1.3: The classification of software, after [150].

change with software products could have a major impact.

The classification indicates that software products have the greatest potential in reducing the environmental impact of ICT solutions. This impact is strengthened by the movement towards cloud based solutions, where on-premise installations are increasingly moved to a more energy efficient infrastructure. Software that is optimized to fully utilize the scale advantages and possibilities of a cloud infrastructure is able to significantly reduce the resource and energy consumption on its behalf [12]. An SPO needs to be in control of the energy consumption of their software products to fully realize this potential.

1.2.2 Sustainability and Software Architecture

To influence the energy consumption of software, an SPO needs to know what software elements invoke specific energy consuming behavior [47]. However, due to limitations with respect to energy consumption measurements, software is often treated as a single, complex entity instead of the interrelated entities it actually consists of [45]. While more detailed measurements can be performed, both hardware-based [39] and software-based [108], only few are able and willing to invest in the equipment or efforts required for such ap-

proaches. Consequently, it often remains unclear for an SPO where to direct sustainability efforts concerning the software.

Software architecture has the potential to play a key role in solving the measurement problem. Software architecture is defined as the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them [6]. Thus, an architectural description of a software product enables a stakeholder to identify the software elements that are actively stressed while performing an activity with the software. Enriched with energy consumption measurements, an architecural description can be used to narrow the scope in relation to specific concerns [126] and identify the energy hot spots [124].

In relation to software development, an architectural approach enables SPOs to address energy concerns on a structural basis through different phases of product maturity. Boehm [14] describes four major activities to comprising the development process (Fig. 1.4): (1) determine objectives, alternatives and constraints, (2) evaluate alternatives, and identify and resolve risks, (3)

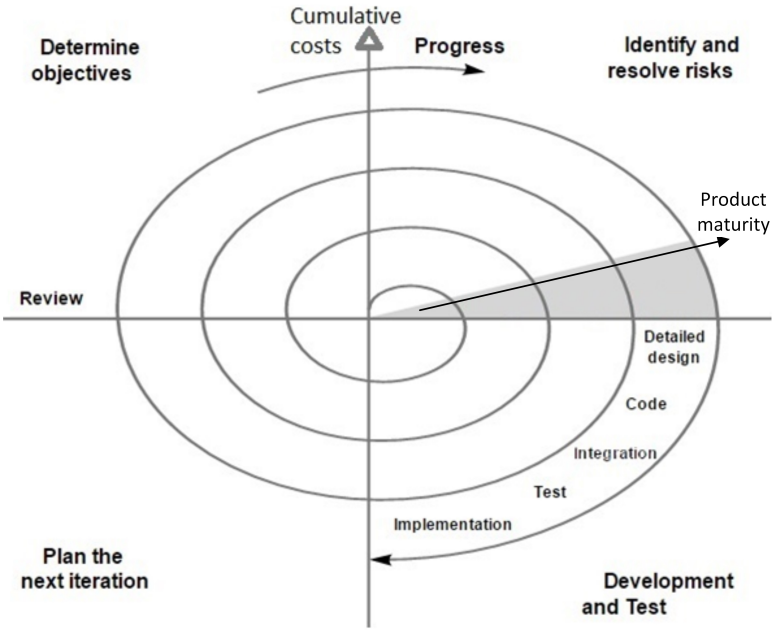


Figure 1.4: The spiral development process, after [14].

develop and verify, (4) plan next phase. One of the main tasks of a software architect is to consider all concerns that a software product should address (1), both functional and non-functional [6]. As not all concerns can always be satisfied, the architect performs trade-off analysis to find a balance (2). After development, verification takes place to ascertain concerns are addressed (3) and the next cycle is planned (4). Identifying concerns in an early stage, allows an SPO to satisfy concerns when the costs are lowest [103].

Finding the balance, often implies trade-offs are made that affect system design. However, performing trade-off analysis is not always straight-forward. For example, it is a common misunderstanding that performance and energy consumption are always positively correlated [20,144]. In addition, techniques like multi-objective Pareto analysis could still provide multiple optimal alternatives that require manual prioritization between alternatives [15].

On a higher level, trade-offs also concern exchanging modules or services for more energy efficient variants, e.g. self-reconfiguration [116] or cloud federation [122]. To support the software architect, sustainability can be positioned as a qualitative aspect for the software. This provides access to the rich body of knowledge on software architecture, including attribute-driven design [7] and the architecture trade-off analysis method (ATAM) [69].

1.2.3 Energy Awareness in Software Engineering

Despite the potential savings and benefits, the adoption of green software practices differs significantly depending on the context in which software products are intended to operate. For example, driven by the limited battery life, mobile app developers are keen to find the optimal balance between the performance and energy efficiency of their software [26, 96]. Sustainability gains can be concretely measured through the battery drain and are relatively tangible for the software engineer [84, 113]. If a balance is not found, the energy inefficiency negatively impacts the end user experience and thereby potentially the expected lifespan (i.e. economic dimension) for an app.

On the other side of the spectrum we find enterprise software that operates on-premise or in data centers [150]. Sustainability concerns in this context relate to the environmental impact and facility costs of hardware infrastructures [128]. However, energy consumption measurements are more complicated to perform given the diversity of deployments and levels of abstraction introduced by, among others, virtualization [45]. Additionally, as this concern is further away from the software engineers, there is no sense of urgency like in the mobile domain. In this case, an SPO potentially faces a principal-agent problem [34]: the SPO (principal) strives for achieving sustainability goals,

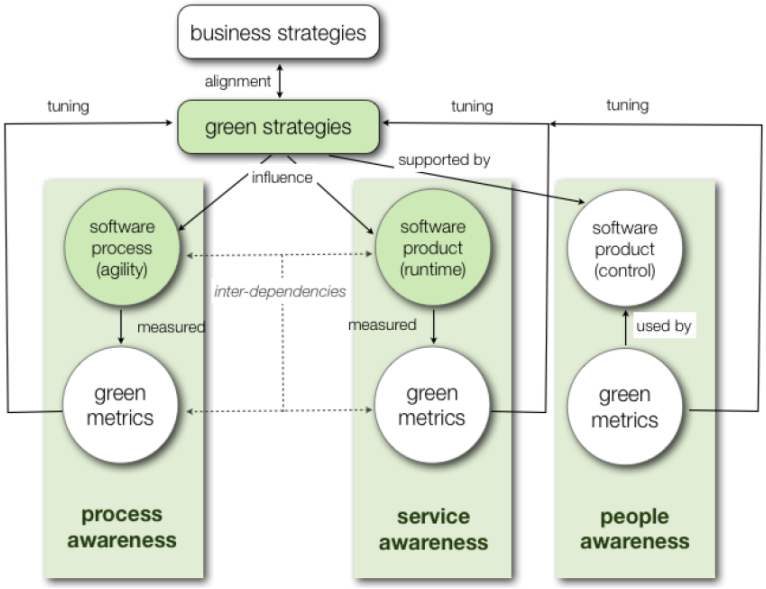


Figure 1.5: Awareness in service oriented software engineering, after [81].

where a software engineer (agent) might choose for a 'quick and dirty' instead of energy efficient implementation.

With enterprise software, the stakeholders that are able to solve sustainability concerns, i.e. software architects and engineers, should become aware of their role in the greening of ICT [130]. According to Lago and Jansen [81], awareness in SPOs should be created in three green problem areas Fig. 1.5):

- *Process awareness*: Considers the sustainability of the software development process. Green strategies can redefine business processes and potentially impact policies on corporate level [141].

- *Service awareness*: Considers delivering energy efficient software services. Green software engineering fits this area, where green strategies influence the energy efficiency of software at run-time.

- *People awareness*: Considers the awareness among actors like software engineers and end-users. Increased awareness among actors and control over ICT, support green strategies.

The green metrics, e.g. the proportionality gap [67] and computational energy cost [16], enable an SPO to tune green strategies and ensure strategic alignment with business strategies [115].

To create awareness among stakeholders of a software product, i.e. people awareness, a product manager should acknowledge sustainability by defining a green strategy in line with business strategy [33]. In turn, green metrics should provide feedback, i.e. eco-feedback [50], to the stakeholders, which enables them to realize the green strategies. For example, simulating the impact of adjustments on a test infrastructure allows an engineer to consider the quantity and proportionality of resource utilization given performance [130]. Increased awareness could affect beliefs, which are bound to change the software engineering practice [30] and potentially affect process and service awareness.

1.3 Research Approach

The research approach of this dissertation can be best characterized as design-science research [49]. Our results aim at solving multiple problems in relation to the energy consumption of software products, and a solution is sought through improving existing methods and instruments or proposing new arti-

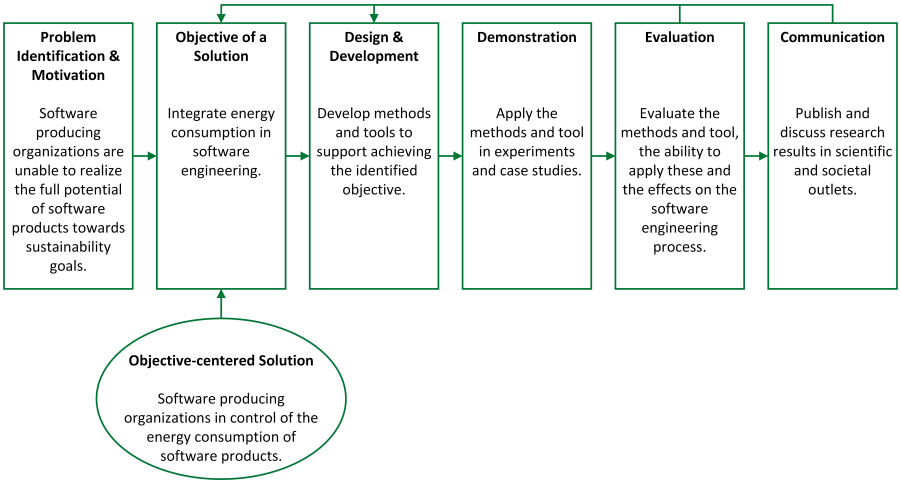


Figure 1.6: The Design Science Research Methodology applied to our research, after [114].

facts. To structure our approach, we followed the activities comprising the Design Science Research Methodology (Fig. 1.6) [114].

The problem identification and motivation, i.e. the first activity, has been performed by introducing the research field and discussing the issues that are currently present. With regard to the objective, we aim to integrate energy consumption in software engineering and thereby bring SPOs in control of this qualitative aspect of their software products. The remaining activities are performed individually in the chapters comprising this dissertation. Each chapter presents a different study and addresses one or more research questions to achieve the objective. In the remainder of this section we present the research questions and the research methods that are applied.

1.3.1 Research Questions

The role of software as the true consumer of power and its potential contribution to reach sustainability goals has increasingly been acknowledged [82], yet it has proven to be a complex issue to address. This research focuses on enabling SPO to deliver green software products whilst still meeting other (quality) requirements for their software. Therefore, the main research question (MRQ) in this PhD dissertation is:

MRQ - ***How can software producing organizations be in control of the energy consumption of their software products?***

In the MRQ, being in control implies SPOs can make informed decisions (including trade-offs) with respect to the energy consumption of their software products and influence this qualitative aspect accordingly.

To provide an answer on the main question of this dissertation, three Research Questions (RQs) are formulated. First we focus on how to measure the energy consumption of software products. Second, we relate energy consumption to the software architecture domain and investigate how to structurally address energy concerns. Finally, we investigate how to embed energy consumption in software engineering in an industrial context.

RQ1: What are effective measures to quantify the energy consumption of software products?

Given the novelty of the research domain, the first step is to determine how the energy consumption of software can be measured. Various measurement approaches are available, both hardware- and software-based, each with their own strengths and limitations. Additionally, given the relation between

hardware and energy consumption, hardware performance is also included as an important aspect that requires measurement in this context. We scope the research for RQ1 to the following sub-questions:

RQ1.1: Can energy profilers be used to accurately estimate the energy consumption of software?
An energy profiler is a software tool that estimates the energy consumption of applications based on the computational resource usage. However, although based on the same set of variables, the estimations per profiler can differ significantly. Hence, study is required into the reliability and accuracy of these tools before they can be safely used for their intended purpose.

RQ1.2: How can we effectively express the resource utilization for executing a software product in relation to the software energy consumption?
Performing measurements in relation to software, often provides multiple, complex metrics to characterize relevant aspects. In some cases deep knowledge on the matter is required to understand specific metrics and identify the key findings. To make the results more accessible for stakeholders, the raw measurements should be expressed and presented in a more effective format. This helps to quickly identify concerns for a software product and evaluate the results of sustainable undertakings.

RQ2: How can energy consumption concerns be addressed in software architecture?

Energy consumption in relation to software remains a complex aspect to address. A software developer, for example, could make multiple adjustments to the software on different locations, without successfully reducing its energy consumption. Hence, an approach is required to guide such efforts and incorporate sustainability in the software design. The research for RQ2 is scoped to the following sub-questions:

RQ2.1: How can we position the energy consumption of software products within the scope of software architecture?
At its core, the creation of green software products start with the design of the software, i.e. its architecture. Making energy consumption explicit, allows this aspect to be structurally included in trade-off analysis and extends the control a stakeholder has over the desired quality properties. To optimize the efforts, a method is required to systematize the tasks of an architect in relation to green software products.

RQ2.2: How can we create an in-depth energy profile of a software product?
Unraveling a software product through its architecture enables stakeholders to identify, measure and analyze the actual drivers behind the energy consumption. In support of RQ2.1, a method is required to obtain a detailed view into the software elements (e.g. functional modules or lines of code) that are the main drivers behind the energy consumption. To increase the effectiveness, the method should be applicable during development and with minimal investment.

RQ3: How can energy awareness be embedded in the software engineering process?

Corporate social responsibility requires SPOs to be aware of the contribution their software might have towards achieving sustainability goals. However, awareness on the topic is only effective when there is awareness throughout the organization. Ranging from strategy to operation, different stakeholders need to be in line to fully capitalize on the green software potential. The research for RQ3 is scoped to the following sub-questions:

RQ3.1: How can we reliably compare the energy consumption of large-scale software products across different releases?
A first step to introduce energy consumption in the software engineering process, is to provide insight in the differences between releases. Comparing releases shows the effects of newly introduced requirements, or even design decisions, and allow stakeholders to determine whether software behavior is in line with expectations. If not, corrective adjustments can be planned for the next cycle.

RQ3.2: What is the added value for a software producing organization to perform energy consumption measurements on software products?
Detailed information on energy consumption allows a SPO to plan the evolution of a software product. However, the efforts of performing energy consumption measurements need justification to satisfy the economic dimension of sustainability. SPOs, or more specifically product stakeholders, should identify business opportunities for green software products.

RQ3.3: How to create and maintain awareness of the energy consumption perspective in software product development?
Software engineers, architects and other product stakeholders are primarily

functionality driven, and there is often little awareness of the energy consumption aspect of the software. Hence, awareness should be created to enable alignment between green strategies and business strategies. Creating awareness requires an understanding of the constructs that affect awareness among the stakeholders. Additionally, an appropriate stimulus is required to trigger awareness during software engineering.

1.3.2 Research Methods

To answer the research questions, different research methods are applied both qualitative and quantitative in nature. In this section we elaborate on the research methods and relate them to the individual research questions to which they apply (Tbl. 1.1). Apart from the methods discussed below, literature study is performed for each research question to reflect on the objective and results of our work.

Case Study
The case studies in our research can be characterized as exploratory case studies [152] aimed at evaluating a research artifact and gaining insight in specific aspects affecting software development. Following the typology of case study designs, specifically single-case holistic (RQ2.1) and multiple-embedded (RQ3.3) case studies are performed in an industrial setting using multiple commercial software products. The cases are selected based on predetermined criteria to ensure access to and availability of the required resources. Additionally, for RQ3.3, specific criteria were formulated in relation to the software development process. While case selection potentially limits the generalizability of the results, appropriate cases studied following a protocol provide valuable results in software engineering research [127].

Table 1.1: Overview of the research methods applied per research question.

	RQ1		RQ2		RQ3		
	1.1	1.2	2.1	2.2	3.1	3.2	3.3
Case study			X				X
Controlled experiment	X	X		X	X		
Interviews						X	
Survey							X

Controlled Experiment
A controlled experiment allows us to test the relation between cause and effect by studying treatment and outcome (Fig. 1.7) [148]. In the envisioned experiments, the treatments encompass software modifications (RQ2.2, RQ3.1), variations with respect to applications (RQ1.1, RQ1.2) and different software configurations (RQ1.2). The observed outcome is the difference caused by the treatment, i.e. the delta (Δ), and is quantified in terms of the resource utilization (e.g. CPU utilization) and energy consumption. Following [66], the experiments can be qualified as laboratory experiments.

Interviews
In interview-based data collection, a dialog is held between researcher and subject(s) guided by a set of interview questions [148, 152]. For our research, i.e. RQ3.2, we conduct semi-structured interviews with multiple stakeholders of a software product to gain insight in the added value of energy consumption measurements. Semi-structured interviews allow us to obtain the desired insights while still providing room for the interviewees to discuss their interests in relation to the energy consumption of software products. To ensure consistency and reliability, the interviews are to be performed following a protocol including a verification of the results by each interviewee.

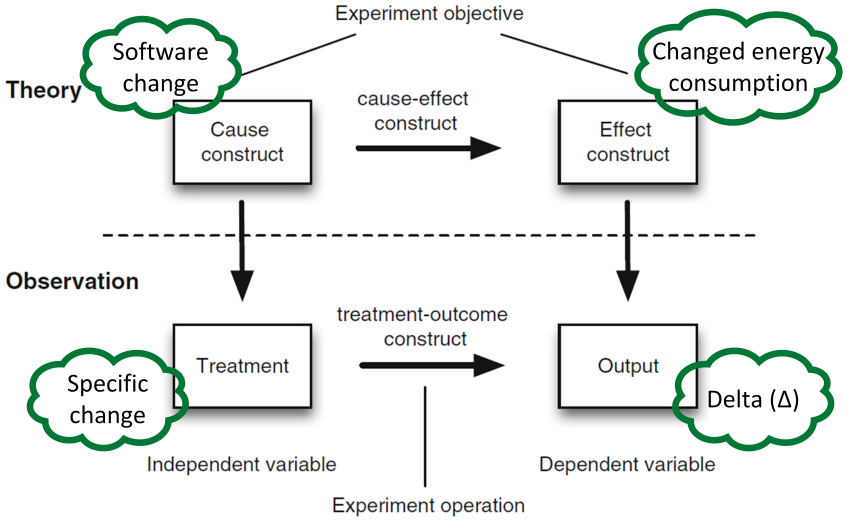


Figure 1.7: The experimentation principle applied to our research, after [148].

Survey
The survey method is applied as part of a case study, i.e. a mixed method approach for RQ3.3 [152], to collect data from product stakeholders on their awareness of energy consumption and acceptance of an energy dashboard. To keep track of the constructs over time, the survey is presented multiple times in the duration of the case study. The survey questions themselves are based on existing literature and, depending on the case, presented as either pencil-and-paper or online survey. Data analysis is performed following a method inspired by sentiment analysis [111].

1.4 Dissertation Outline

The dissertation is structured into four separate parts that each address one research question and the underlying sub-research questions. Chapters 2 to 7, comprising Parts I, II and III, are written as individual papers for publication in scientific conference proceedings or journals. Chapter 8, Part IV, concludes the dissertation and provides an answer to the main research question.

Chapter 1: Introduction

Part I: Quantifying Software Energy Consumption

Part I describes two studies in relation to quantifying the energy consumption of software, as required for RQ1.

Chapter 2: Profiling Energy Profilers

This chapter answers RQ1.1 and forms the basis for performing energy consumption measurements with software products. A selection of energy profilers, tools that estimate energy consumption based on resource utilization, is evaluated in terms of its functionality and accuracy of the reported measurements. While there is still work to be done for these tools to be safely used for its intended purpose, basic insight can be obtained into the matter.

Published as: E. Jagroep, J. M. E. M. van der Werf, S. Jansen, M. Ferreira, and J. Visser. Profiling energy profilers. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, pages 2198–2203. ACM, 2015

Chapter 3: A Resource Utilization Score for Software Energy Consumption

In this chapter we answer RQ1.2 and propose a metric to express the resource utilization, such as power consumption, of a software product. The metric is designed as a single score and allows for objective comparison of application configurations and versions. In addition, the visualization of resource consumption measurements is investigated to enhance communication and highlight key findings.

Published as: E. Jagroep, J. M. E. M. van der Werf, J. Broekman, S. Brinkkemper, L. Blom, and R. van Vliet. A resource utilization score for software energy consumption. In *Proceedings of the 4th International Conference on ICT for Sustainability*, pages 19–28. Atlantis Press, 2016

Part II: Energy Consumption in Software Architecture

In Part II we explicitly relate software energy consumption to the software architecture domain and provide an answer to RQ2.

Chapter 4: Extending software architecture views with an energy consumption perspective; A case study on resource consumption of enterprise software

In this chapter we answer RQ2.1 and propose an energy consumption perspective on software as a means to provide detailed insight in the software elements that invoke specific energy consumption behavior. The perspective is applied in a case study using a commercial software product, demonstrating its potential by reducing the energy consumption with 67.1%.

Published as: E. Jagroep, J. M. van der Werf, S. Brinkkemper, L. Blom, and R. van Vliet. Extending software architecture views with an energy consumption perspective. *Computing*, 99(6):553–573, 2017

A short version of this chapter was published as: E. Jagroep, J. M. E. M. van der Werf, R. Spauwen, L. Blom, R. van Vliet, and S. Brinkkemper. An energy consumption perspective on software architecture. In D. Weyns, R. Mirandola, and I. Crnkovic, editors, *Software Architecture: 9th European Conference, ECSA 2015, Proceedings*, pages 239–247. Springer International Publishing, 2015

Chapter 5: Hunt the Energy Guzzler in my Software: An Architectural Approach

In this chapter we answer RQ2.2 and extend the energy consumption perspective, as presented in Chapter 4, through a method for creating a detailed energy profile of the software. We apply the method in an experiment using a commercial software product and unraveled the energy consumption related to its core functionality.

This chapter was submitted for publication as: E. Jagroep, A. van der Ent, J. M. E. M. van der Werf, S. Brinkkemper, L. Blom, and R. van Vliet. Hunt the energy guzzler in my software: An architectural approach. *Submitted for publication*

Part III: Energy Awareness in Software Engineering

In Part III we embed software energy consumption in the context of software engineering and provide an answer to RQ3.

Chapter 6: Energy efficiency on the product roadmap: an empirical study across releases of a software product

In Chapter 6, we provide an answer to RQ3.1 and 3.2 through a method to reliably compare the energy consumption of consecutive releases of a software product. We demonstrate the method using a commercial software product and continue to investigate the added value of performing energy consumption measurements for product stakeholders.

Published as: E. Jagroep, G. Procaccianti, J. M. van der Werf, S. Brinkkemper, L. Blom, and R. van Vliet. Energy efficiency on the product roadmap: An empirical study across releases of a software product. *Journal of Software: Evolution and Process*, 29(2), 2017

A short version of this chapter was published as: E. Jagroep, J. M. E. M. van der Werf, S. Brinkkemper, G. Procaccianti, P. Lago, L. Blom, and R. van Vliet. Software energy profiling: Comparing releases of a software product. In *Proceedings of the 38th International Conference on Software Engineering Companion*, pages 523–532. ACM, 2016

Chapter 7: Awakening Awareness on Energy Consumption in Software Engineering

To answer RQ3.3, Chapter 7 describes a case study performed with two commercial software products. We followed the respective development processes for these products and provided direct feedback to the stakeholders on the effects of their development efforts. Awareness measurement and dashboard

Published as: E. Jagroep, J. Broekman, J. M. E. M. van der Werf, S. Brinkkemper, P. Lago, L. Blom, and R. van Vliet. Awakening awareness on energy consumption in software engineering. In *Proceedings of the 39th International Conference on Software Engineering: Software Engineering in Society Track*, pages 76–85. IEEE Press, 2017

Part IV: Concluding the Research

Part IV concludes the research and provides an answer to the main research question.

Chapter 8: Conclusions

In this final chapter of the dissertation we answer our research questions based on the results presented in the previous chapters. We provide an overview of the relevant findings and contributions, and discuss the implications and limitations thereof. Furthermore we provide directions for future work in this research area.

Part I

Quantifying Software Energy Consumption

2
Profiling Energy Profilers

While energy is directly consumed by hardware, it is the software that provides the instructions to do so. Energy profilers provide a means to measure the energy consumption of software, enabling the user to take measures in making software more sustainable. Although each energy profiler has access to roughly the same data, the reported measurements can differ significantly between energy profilers. In this research, energy profilers are evaluated through a series of experiments on their functionality and the accuracy of the reported measurements. The results show that there is still work to be done before these software tools can be safely used for their intended purpose. As a start, a correction factor is suggested for the energy profilers.

This work was originally published as:

E. Jagroep, J. M. E. M. van der Werf, S. Jansen, M. Ferreira, and J. Visser. Profiling energy profilers. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, pages 2198–2203. ACM, 2015

2.1 Introduction

The search for more environmental friendly ICT has already had an impact on the energy awareness and energy consumption of the sector [24]. A distinction is often made between 'greening by IT', i.e., using ICT to make other industries more sustainable, and 'greening of IT', the process of making ICT itself more sustainable. This research focuses on the latter, i.e., on sustainable software. Sustainable software is "software whose direct and indirect negative impacts on economy, society, human beings, and the environment resulting from development, deployment, and usage of the software is minimal and/or has a positive effect on sustainable development" [102].

Typically, sustainability research focuses on hardware aspects, i.e., making the hardware more energy efficient [81]. However, it is the software that determines the use of the hardware, and can therefore be seen as the true consumer of power [140]. In addition, "a single ill-behaving power-unfriendly software component on the system can thwart all of the power management benefits built into the hardware" [139].

An energy profiler (EP) is a software tool that estimates the energy consumption of a system based on the computational resources used by applications, and by monitoring the hardware resources [2, 24, 131]. The use of EPs enables practitioners to investigate the energy consumption behavior of their software without having to invest in specialized hardware. However, many different EPs exist and each has its own internal model to estimate the energy consumption. Although all base themselves on the same set of variables, their estimations differ. Consequently, reliability, and hence accuracy, of the estimations is unclear. For example, these tools interpret variables such as multi-core processors, data access and memory consumption differently [107].

In this research, we focus on the question: *Can energy profilers be used to accurately estimate the energy consumption of software?* In a laboratory environment at the Software Energy Footprint Laboratory (SEFLab) [39], we performed experiments with the objective to evaluate the reliabilty of EPs. In the evaluation we focus on (1) the maturity of the EPs in terms of operationalization, and (2) on the accuracy of the reported energy consumption estimations compared to the actual power consumption.

This paper is structured as follows. In Section 2.2 we introduce the experiment that was performed with the EPs. The results of the experiments are presented in Section 2.3 and in Section 2.4 the threats to validity are discussed. Finally Section 2.5 concludes the paper and provides suggestions for future work.

2.2 Experiment setup

In this research we evaluate the accuracy of EPs by comparing the actual energy consumption of the hardware with the reported outcomes of the EPs under study. To this end, idle, varying load and full load scenario's were simulated. In the idle scenario, the system is in rest. For the variable load scenario we chose to simulate a representative real usage scenario of the system by inducing a variable load on the CPU at random intervals using 'SEFLab-Experiments' [1]. Last, in the full load scenario, a continuous full load is placed on the CPU using 'HeavyLoad' [2] (Microsoft Windows) and the 'Stress' package [3] (Linux), maxing out its capacity.

Per EP 35 runs, of approximately one minute, were simulated per scenario, during which three resources were monitored:

- the output of the EP under study;
- the performance measurements of the system; and
- the real energy consumption of the system, monitored by the SEFLab.

Although we acknowledge the importance of the other hardware components [73], we only induce CPU load in this experiment, as it has been identified as the main driver for energy consumption [13, 39, 139].

2.2.1 Energy profilers

An exploratory search was conducted to identify relevant EPs for this research. For this search, we formulated the following requirements to include an EP in our experiment:

- there is at least a 'beta' version available online;
- intervals between two measurements are at most one second;
- it should run on Ubuntu Linux or Microsoft Windows; and
- it should be compatible with the hardware available in the SEFLab.

[1] `https://github.com/SEFLab/SEFLab-Experiments`
[2] `http://www.jam-software.com/heavyload/`
[3] `http://packages.ubuntu.com/search?keywords=stress`

For the exploratory search the Google, Scholar Google and Bing search engines were used for querying the terms 'energy profiler(software)', 'software energy profiler', 'energy consumption software'. The term energy was also exchanged for 'power'.

Table 2.1 presents the selected EPs and their main properties in terms of measurement level and detail. The search showed a clear difference in capabilities between the found EPs. EPs that could not be properly installed in the SEFLab were excluded beforehand from further research.

2.2.2 Equipment

The experiments were performed on a Dell PowerEdge SC1425 server, referred to as the "test-server". To measure the real energy consumption, we used a WattsUp? Pro (WUP) power consumption meter, which is a physical device used to measure the total power consumption of the test-server directly from the power socket. As some EPs run on different operating systems, we swapped between identical hard disks on which Microsoft Windows 7 and Ubuntu version 12.04 were installed. Further details on the available hardware can be found in [39].

Ideally, considering the nature of electrical power, we would have liked for EPs to measure more frequent than once per second since the equipment in the SEFLab allowed us to measure more than ten thousand times per second. However, no EP found in the exploratory search possessed this ability. Hence the choice was made to only measure with the WUP, which is accurate to 1.5%.

2.2.3 Experiment protocol

The experiment consists of three scenarios per EP. For each scenario and EP, we performed the following activities.

Preparation Setup of the test-server, including synchronization of the system clocks using the 'network time protocol' (NTP) and the installation and operationalization of the EP under study. Finally, we ensured that unnecessary applications were closed for clean measurements.

Perform run Execution of the different scenarios using a specialized tool called 'SEFLab-Experiments'. The tool automatically records time pulses to indicate the beginning and end of a run, providing consistency in the duration, and collecting the data from the different sources.

The tools report the energy consumption in joules per time unit. As the typical time interval is one second, this equals to power (W). To calculate the

Table 2.1: Specification of the EPs and the extent in which they are included in this research.

	Level					Detail										
Profiler	**IT Environment**	**System**	**Application**	**Process**	**Line of code**	**Hardw. dependent**	**System**	**Process**	**CPU**	**Memory**	**HDD**	**Network**	**Base**	**Calibration required**	**Installable**	**Operationalizable**
Active Energy Manager (W, L)	✓					✓								?	-	-
Computer Power Log (W)		✓					✓							?	-	-
EC Tools (L)			✓				✓	✓						✓	✓	✓
Energy-aware profiler (W)		✓							✓	✓	✓		✓	?	-	-
Eprof (OS unknown)		✓	✓		✓				✓	✓	✓		✓	✓	-	-
ESSaver (W)	✓		✓	✓				✓	✓	✓	✓	✓		✓	✓	-
Hardware Sensors Monitor * (W)		✓							✓	✓	✓	✓		?	-	-
Joulemeter (W)		✓	✓						✓	✓	✓		✓	✓	✓	✓
PowerAPI (W, L)		✓		✓					✓	✓	✓			-	-	-
PowerTOP (L)		✓		✓				✓						✓	✓	-
powometer (W)		✓				Unknown								?	-	-
pTop (L)			✓						✓		✓	✓		✓	✓	-
pTopW (W)			✓						✓		✓	✓		✓	✓	-
Sensorsview * (W)		✓							✓	✓	✓	✓		?	-	-

* = voltages only, W = Windows, L = Linux

total energy consumed during a run, we aggregate these measurements and report in Watthour (Wh). Throughout this paper the measurements provided by the EPs are referred to as the *reported measurements*, whereas the *actual measurements* are obtained from the SEFLab, i.e. WUP.

2.2.4 Reboot vs no reboot

To determine the influence of rebooting the test-server between runs, a small experiment was performed comparing five full load runs with reboot to five full load runs without reboot. Although the runs with reboot reported a higher average (Mdn = 298.78 W) and energy consumption (Mdn = 403 Wh) than without reboot (Mdn = 298.28 W and Mdn = 402 Wh), the difference was not significant (U = 9.00, z = -0.731, p >.05, r = -.23; for both tests). Measurements for the scenario without reboot were more stable, i.e., showing less outliers than the scenario with reboots. A possible explanation is that the initial start-up processes cause these outliers, resulting in the higher measurements. Therefore, we decided not to reboot the test-server after each run.

2.3 Results

The evaluation of an EP consists of (1) operationalization of the tool, and (2) its accuracy in terms of energy consumption and timeliness. To evaluate the energy consumption, the averages and total energy consumption per run are compared between actual and reported measurements. Timeliness considers whether measurements are reported on the right moment in time, and is done by comparing the energy consumption graphs of each run (cf. Fig. 2.1).

2.3.1 Operation

We first consider the operationalization of the EPs.

Joulemeter is straightforward to install and, after calibration using the WUP meter, provides a proper dataset to compare against the actual measurements. Calibration requires the WUP to be connected to the test-server and pressing the 'calibrate' button on the interface of Joulemeter. The interval between measurements is one second; the results are stored in a comma separated values (CSV) file.

Energy Consumption Tools (EC Tools) is a collection of tools, including a core library with sensors and power estimators that can be re-utilized in other programs, a data acquisition tool, a monitoring tool on the level of

processes, and an application power profiler. It provides real-time resource usage and power estimations for the running processes and is built to allow calibration based on the power consumption reported by the machine's Advanced Configuration and Power Interface (ACPI) subsystem or by external power meters.

Although the installation is straightforward, calibration is required before EC Tools can be used. Unfortunately, EC Tools had to be adjusted [4] to work with the WUP device. Once fully operational EC Tools provides a CSV file containing the power consumption measurements. The interval between two measurements is one second.

pTop [31] relies on a MySQL database to store data it collects and produces. Despite a successful installation and calibration, we did not manage to get pTop to produce any energy estimations. After code inspection we found that two functions responsible for inserting data in the "*process_energy*" and "*device_energy*" tables, named "*insert_ process_energy*" and "*insert_device_energy*", were never called at in the pTop code base. We have been in contact with the developer of pTop, but contact was broken and the decision was made to exclude pTop from further research.

pTopW does not share the same code-base of its Linux counterpart and is calibrated by providing power characteristics of the hardware (e.g. thermal design power of the CPU) in a calibration file. Although most hardware characteristics could be obtained via the respective vendors, there were parameters in the file that we did not understand properly. Documentation on this matter was lacking and contact with the developer did not provide the required clar-

[4] `https://github.com/cupertino/ectools/pull/7`

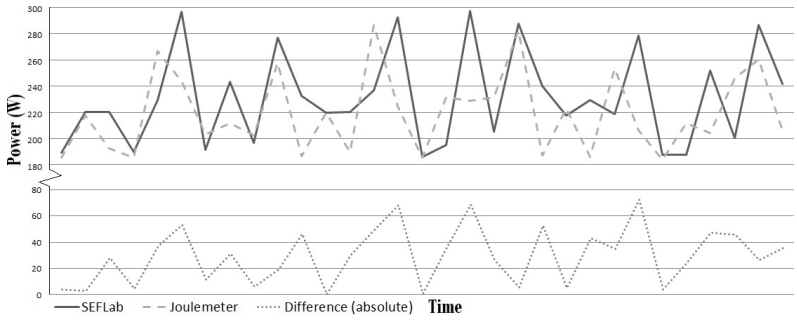


Figure 2.1: Actual (SEFLab), reported (Joulemeter) and difference in energy consumption over time.

ity. Using a configuration file containing all information that we could provide at that time, pTopW produced estimations of 50+ kWh for the CPU alone. The inability to calibrate led to the exclusion of pTopW from further research.

PowerTop is fundamentally different from the other EPs as it uses an external measurement of the power being drawn and breaks this down per process using the computational resource consumption. Since the powermeter that PowerTop is designed to work with was not available during the experiment, the source code was adapted to read the power consumption from the WUP meter [5]. Then, conform described functionality, PowerTop ran for over half an hour to collect enough data points to produce estimations. Unfortunately we were not able to get PowerTop to produce energy consumption estimations and thus excluded the tool from further research.

One other potential problem we noticed in the source code of PowerTop, was that the external power sensor was only called before measurement starts. In the varying load scenario this could mean that that PowerTop will not be able to properly keep track of variations in power consumption.

ESSaver is composed of a data collection agent, that runs as a windows service, and a reporting tool, that produces energy consumption reports based on the collected data. With the help of the developer, we were able to configure the EP to work with the SEFLab hardware and perform the experiments. Unfortunately, we were not able to transform the data collected by ESSaver into valid power estimations.

We can only speculate about the reasons behind this problem, but one observation is that the agent installed on the test-server is more mature than the reporting tool. The agent is installed with just a few clicks of the mouse, whereas much more configuration is involved for the reporting tool which was, at that time, not shipped to the users of ESSaver. As a result, ESSaver was excluded from further research.

2.3.2 Accuracy

After first inspection, the accuracy was evaluated of the EPs that could be made operational (Tbl. 2.1). The Joulemeter and EC Tools data were tested for normality [40] to determine the correct tests to apply. For the normally distributed data, the independent samples t-test was applied, whereas the non-parametric Mann-Whitney U test has been applied for non-normally distributed data [40].

[5] `https://github.com/pyrovski/watts-up`

Joulemeter

Joulemeter provides the richest dataset, containing valid 35 runs for all three scenarios.

Idle: the Joulemeter averages (Mdn = 193 W) are *significantly* higher than the SEFLab averages (Mdn = 184 W), U = 306, z =-3.60, p <.05, r = -0.43. This in contrast to the energy consumption figures where Joulemeter (Mdn = 181 Wh) reports *significantly* lower figures than the SEFLab (Mdn = 182 Wh), U = 393, z = -2.548, p <.05, r = -.30.

Varying load: the averages for Joulemeter (M = 218 W, SE = .6) are lower than the SEFLab (M = 230 W, SE = .56). This difference is *significant* t(68) = 14.27, p <.05 and represents a large effect size r = .87. Concerning the energy consumption again Joulemeter (M = 224 Wh, SE = .0012) reports lower figures than the SEFLab (M = 236 Wh, SE = .0013). This difference is *significant* as well t(68) = 6.67,p <.05 with a medium effect size r = .40.

Full load: For the full load scenario, opposite to the idle scenario, Joulemeter (Mdn = 316 W) reports *significantly* higher averages than the SEFLab (Mdn = 308 W), U = 98, z =-6.043, p <.05, r = -.72. This also holds for the energy consumption, Joulemeter (Mdn = 330 Wh) compared to SEFLab (Mdn = 320 Wh), however this difference is *not significant*, U = 454, z =-1.862, p >.05, r = -.22.

The results are summarized in Tbl. 2.2, which shows the significant differences from Joulemeter compared to the SEFLab. Fig. 2.2 visualizes the values of the different scenarios as boxplots. Surprisingly, the plots show relatively large overlaps in the idle and varying load. Given the non significant difference this was only expected for the full load scenario. Looking at the average differences in absolute terms we find 2 Wh for the idle, 12 Wh for the varying load and 8 Wh full load scenario.

With regard to the timeliness of Joulemeter, we observe no significant difference compared to the SEFLab. The figures (varying load example in Fig. 2.1) show spikes and 'lows' at approximately the same moments in time

Table 2.2: EP results compared to the SEFLab.

	Joulemeter		EC Tools	
	Avg.	Consumption	Avg.	Consumption
Idle	↑	↓	↑	↑
Varying	↓	↓	↓	↓
High	↑	-	↓	↓

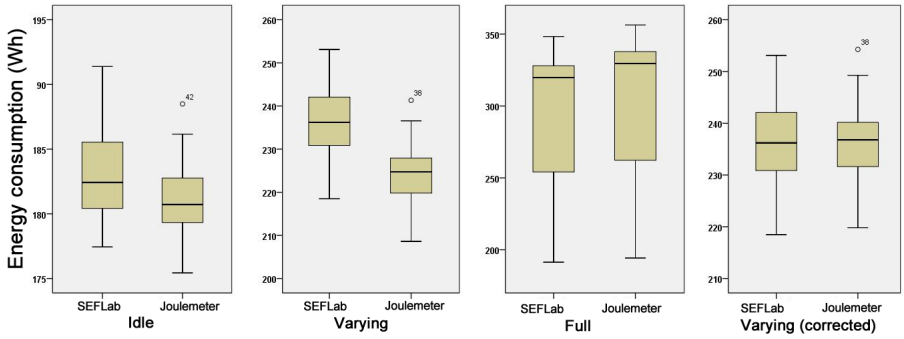


Figure 2.2: Joulemeter and SEFLab energy consumption averages per scenario.

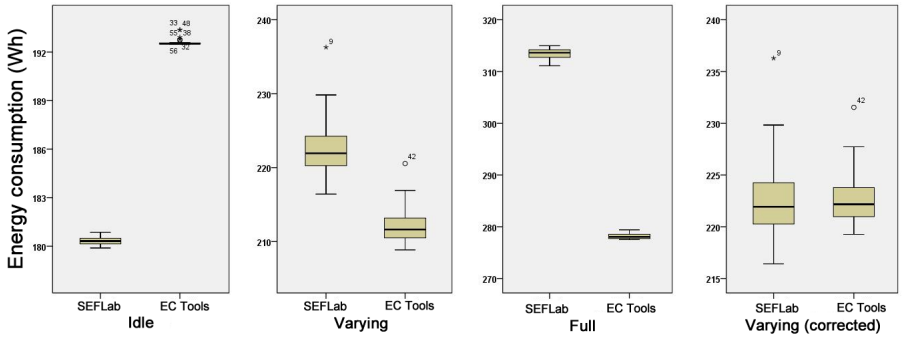


Figure 2.3: EC Tools and SEFLab energy consumption averages per scenario.

during a run, enabling users to assign power consumption patterns to specific activities that are being performed.

Energy Consumption Tools

Unfortunately the EC Tools dataset is less stable compared by Joulemeter, as the tool turned out to not always produced reliable measurements. For this reason, we obtained 31 for the idle, 33 for the varying load and 34 valid runs for the full load scenario.

Idle: The figures reported for the idle scenario indicate that EC Tools (Mdn = 196 W) reports *significantly* higher figures than the SEFLab (Mdn = 183 W), U = .00, z = -6.765, p <.05, r = -.86. This line continues with the energy consumption where EC Tools (Mdn = 193 Wh) reports *significantly*

higher figures than the SEFLab (Mdn = 180 Wh), U = .00, z = -6.765, p <.05, r = -.86.

Varying load: In the varying load scenario, EC Tools (Mdn = 215 W) reports *significantly* lower averages than the SEFLab (Mdn = 226 W), U = 10, z =-6.855, p <.05, r = -.84. For the energy consumption the statistics also indicate that EC Tools (M = 212 Wh, SE = .0004) on average reports lower figures than the SEFLab (M = 225 Wh, SE = .001) and again a *significant* difference is found t(52) = 12.939, p <.05 that represents a large-sized effect = .76.

Full load: The statistics for the full load scenario show that EC Tools (Mdn = 282 W) reports *significantly* lower averages than the SEFLab (Mdn = 318 W), U = .00, z = -7.09, p <.05, r = -.86. The same holds for the energy consumption where EC Tools (M = 278 Wh, SE = .00007) reports lower figures than the SEFLab (M = 313 Wh, SE = .0002). This difference is *significant* t(43.9) = 176.322, p <.05 and represents a large-sized effect r = 1.0.

Statistics indicate that the EC Tools measurements in all cases *significantly* differ from the SEFLab. Looking at the boxplots presented in Fig. 2.3, this finding is supported through the minimal overlap between measurements. In absolute terms we found an average difference of 11 Wh, 10 Wh and 34 Wh for respectively the idle, varying and full load scenario. An interesting observation is the fact that the idle and full load measurements are within a dense range of around 3 Wh, which might be caused by the type of operating system.

On the aspect of timeliness, using similar graphs as Fig. 2.1, less overlap and in general a lower responsiveness was perceived, though still acceptable. Overall, a doubt remains on the operationalizability of EC Tools, considering the stability issues and large differences found in the experiment.

2.3.3 Correction factor

Looking at the Joulemeter measurements for the varying load scenario Tbl. 2.3, the error between the actual and reported measurements seems to be relatively constant at 5%. The error percentages presented in this table are calculated by dividing the difference in energy consumption by respectively the SEFLab and Joulemeter measurements. The corrected figures for Joulemeter are calculated using the average error percentage for Joulemeter, i.e., 5.37%.

Table 2.3 shows that after correcting the Joulemeter varying load measurements with this factor, the average difference between Joulemeter and the SEFLab becomes 0.48%. Repeating these calculations for the varying load scenario of EC Tools, we find that a correction of 4.98% can be applied to

Table 2.3: Joulemeter example data and correction factor for the varying load scenario.

Run	SL (Wh)	JM (Wh)	Diff. (Wh)	SL error (%)	JM error (%)	JM corr. (Wh)	Corr. diff.	New error (%)
1	229.925	220.840	9.085	3.95	4.11	232.695	2.770	1.20
2	230.544	217.763	12.780	5.54	5.87	229.453	1.091	0.47
3	253.106	241.310	11.796	4.66	4.89	254.264	1.158	0.46
4	232.446	222.420	10.027	4.31	4.51	234.359	1.913	0.82
5	242.479	227.599	14.881	6.14	6.54	239.816	2.663	1.10
...	...	...	...	...	...	...	...	...
Averages	236.223	224.186	12.037	5.09	5.37	236.221	1.141	0.48

reduce the average difference to 0.78%. Looking at the difference in absolute terms, supported by the utmost right boxplots in Fig. 2.2 and Fig. 2.3, we hypothesize that the corrections bring the measurements within acceptable bounds.

After correction Joulemeter (M = 236 Wh, SE = 1.29) reports the same mean as the SEFLab (M = 236 Wh, SE = 1.33), and a slightly higher median (see Fig. 2.2, which is *not significant* t(68) = -.001, p ¿ .05 and has no effect size r = .00. EC Tools (Mdn = 222.2 Wh) also reports a slightly higher median than the SEFLab (Mdn = 221.8 Wh), and again this difference is *not significant* (U = 514, z = -.391, p¿.05, r = .05). Although the initial results are promising, further research to investigate and determine the correction factors is required.

2.4 Threats to validity

The threats to validity are identified according to the four major classes of validity aspects [148, 152]. Construct validity covers identifying the correct operational measures for the concepts being studied. Considering the nature of the experiments and the measurements that were obtained, we ensured that there is no room for ambiguity in interpreting the results.

In the light of the internal validity, it cannot be 100% certain that the only load generated on the test-server was caused by the tools used for experimentation. Although all unnecessary applications were closed, the behavior of services can not be completely controlled. To exclude the influence of (start-up) services as much as possible, the choice was made not to reboot the server between runs. In order to control environmental conditions that could influence the experiment, e.g. room temperature, the test-server was situated in a former server room.

For the external validity we argue that the installation and configuration difficulties will be experienced by anyone wanting to use a specific EP. However, although the measurements themselves are not questioned, the availability or lack of specific hardware might have enabled or inhibited us to get an EP operational. Different hardware setups could yield different results in getting an EP operational.

Concerning the reliability of the experiment, we argue that the described experiment protocol should yield similar results. One aspect that might differ is the choice to take the measurements as given and not to transform the data when not normally distributed.

2.5 Conclusions

In this paper we question EPs in terms of their ability to accurately estimate the energy consumption of software. Through experimentation we evaluate the ability to operationalize EPs and whether the reported energy consumption estimations are accurate compared to the actual energy consumption figures.

For our experiment we were only able to install six out of fourteen EPs found and only two out of those six fully operational for experimentation. Although installation could be performed successfully, configuration turned out to be problematic. Even with the help of the respective developers we could not solve the issues that we came across. Although hardware independence was claimed, the main flaw remained the ability to cope with different hardware configurations.

The actual experiments were performed with Joulemeter and EC Tools, on respectively the Windows 7 (x64) and Ubuntu 12.04 operating systems, where we considered the power consumption averages and the total energy consumption during idle, varying load and full load scenarios. Except for the energy consumption in the full load scenario with Joulemeter, we found significant differences compared to the actual usage. Concerning the timeliness of the measurements, we found that both EPs are acceptable.

Hence we argue that EP in general can not be used to estimate the energy consumption of software yet. Although the tested EPs can be used to get a sense of the energy consumption habits of software, both EPs showed a significant differences in the one minute runs. We expect this difference to become unacceptable when longer periods of measurement are performed. The proposed correction factors of 5.37% for Joulemeter and 4.98% for EC Tools, although further research is required, are a first step in bringing the measurements within acceptable bounds.

Based on our experiment, we identify several directions for future research. A first direction would be to evaluate multiple EPs on different platforms and hardware setups and also evaluate cross-platform EPs. Apart from clarifying the generalizability of our results, more insight could be gained on differences between platforms. A second direction is to investigate how our results, and EPs in general can be used by different stakeholders, e.g. software engineers. Third, is to investigate the results when the experiment consists of lengthier runs. It is our belief that in this case all measurements will significantly differ compared to the actual usage and could provide a test for the proposed correction factor. A final direction is to repeat the experiment at a later moment in time to determine whether progress is made with regard to the development of EPs. Apart from determining their accuracy, EPs might also allow for more detailed measurements (e.g. individual hardware components).

3

A Resource Utilization Score for Software Energy Consumption

Software as the true consumer of power and its potential contribution to reach sustainability goals is increasingly being acknowledged. Studies so far have presented successful results and methods to address the energy consumption of the software, indicating that different stakeholders striving for green software have different information needs with respect to their goals. However, currently there is no uniform manner to communicate measurements to the different stakeholders such that key findings are clearly identifiable and easy to understand, which is likely to hamper green software practices.

In this paper we propose a metric that expresses a score for the resource utilization, such as power consumption, of a software product. The metric is designed to be a single score and is flexible to encompass those aspects that a stakeholder considers relevant in the context of software energy consumption. The metric was applied on two applications and allowed for objective comparison of application configurations and versions. Also the behavior of these applications across different hardware configurations could be analyzed. In addition to the metric we investigate means to visualize measurements which enhances communication and helped with highlighting the key findings.

This work was originally published as:

E. Jagroep, J. M. E. M. van der Werf, J. Broekman, S. Brinkkemper, L. Blom, and R. van Vliet. A resource utilization score for software energy consumption. In *Proceedings of the 4th International Conference on ICT for Sustainability*, pages 19–28. Atlantis Press, 2016

3.1 Introduction

The recent focus on the Energy Consumption (EC) of software has had a positive impact on the spectrum of sustainable, i.e. energy efficient [99], solutions in the ICT sector. Although hardware consumes energy, software directs the hardware on using the available resources [140] and numerous studies become available that report improvements on energy related aspects with the software itself as the central topic [45, 109]. In a recent study, Hindle [51] presents a method to analyze EC across releases of software products, which provides a basis for sustainable endeavors a software producing organization [65] might undertake. Despite this, organizations still struggle with addressing the software products in terms of their EC [112].

Key in this struggle is that addressing the EC of software confronts a software producing organization, specifically software developers, with a multifaceted issue. Depending on its deployment, measuring the EC of software can be done using relatively cheap hardware devices. However, apart from EC measurements, the software is also characterized using performance measurements which allows for analysis of the software's energy consuming behavior and resource usage. Performance measurements in our case refer to hardware resource performance and provide insight in how the hardware components are stressed when processing instructions. Developers should be able to use this information to address this relatively unknown, non-functional aspect [6] of the software.

Based on previous work (i.e. [61, 64]), however, we found that these measurements are not easy to communicate to stakeholders. Our experience is that in some cases deep knowledge is required to understand the measurements, i.e. how should a specific (performance) measurement or metric be interpreted, and that the key findings that require further investigation are difficult to identify. Issues that are strengthened by developer knowledge that is lacking in this area [112]. As a result, we had developers and software architects searching for the right information and identified a potential inhibiting factor to start addressing the EC of the software.

In this paper we investigate a means to effectively communicate Software Energy Consumption (SEC) related measurements to stakeholders wanting to address the sustainability of their software. Effectively in our case means that the information is easy to understand, is reported uniformly to enhance recognition, and clearly communicates any key findings. Ideally we are able to express the results in a metric that allows to objectively compare the software across different contexts (e.g. releases, installations).

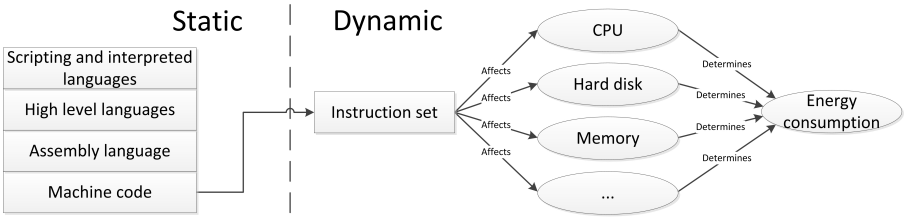


Figure 3.1: A translation from the language abstraction stack (static) to the energy consumers (dynamic).

Based on the above we formulate our main research question as follows:

RQ: *How can we effectively express the resource utilization for executing a software product in relation to the SEC?*

In the RQ, we refer to various resources as it is clear that energy is not the only involved resource. However, as this differs per study, a possible metric should be flexible to encompass those resources that are considered important in a given context. Translating the focus on resource usage to a metric, we contribute by providing a Resource Utilization Score (RUS) for the SEC. The RUS helps in the analysis of SEC related measurements and their visualization.

The remainder of this paper is structured as follows. We first present the related work (Sect. 3.2) and continue with the creation of the RUS (Sect. 3.3). After constructing the score we apply the RUS in an experiment (Sect. 3.4) and evaluate the results (Sect. 3.5). Finally, we discuss our findings (Sect. 3.6) and provide a conclusion including directions for future research (Sect. 3.7).

3.2 Related Work

The term sustainability in the ICT domain is aimed at controlling ecological, economical and social dimensions (of ICT) to the extent that future stakeholders are not compromised in the ability to meet their needs [25]. Sustainable ICT, also coined 'Green IT' [99], helps to improve energy efficiency, lower greenhouse gas emissions and promotes reuse and recycling. Recently a technical dimension has been added for software intensive systems [83], addressing the aspects of the constantly changing environment in which software is executed. Our focus on optimizing the EC of software, i.e. 'green software' [82], is mainly concerned with the ecological dimension, however economic, social and technical goals could also be addressed using a RUS.

3.2.1 Addressing Software Sustainability

A popular approach towards creating green software is to consider sustainability [82], or energy efficiency [67, 123], as a quality aspect for the software. Doing so allows software producing organizations to consider sustainability aspects during the design of the software, i.e. with its software architecture, and make trade-offs with other quality aspects (e.g [129]). However, analogous to the other quality attributes [54], working on software quality could require significant investments in terms of time, specialized knowledge to address specific issues and analysis across architectural views [126]. An EC perspective [64] could help guide any efforts in this regard, but the complexity of the matter could still pose difficulties.

If we look at the language abstraction stack (Fig. 3.1), we can explain the complexity. After a blueprint for the software is made in the form of its software architecture [6], the actual software development can take place. Through several layers of abstraction an instruction set is acquired that is executed by the hardware. This brings us from the static to the dynamic aspect of software. Performing the sequences of instructions affects the hardware components and available (virtualized) resources, which in turn determines the EC induced by the software. Hence, as developers have limited control over the instruction set, they can only await what effect changes in the source code might have. Some tools are available though, e.g. Big-O notation [5], but again complexity issues rise due to the large software systems that organizations produce.

To exert control, apart from EC measurements, studies in the area of green software report a variety of metrics depending on the context in which a study was performed and the stakeholders that are involved. For example, performance [64, 68] and software [51] metrics are used to characterize a software product, which is typically input for developers and architects. These two stakeholders could also benefit from knowing the EC on process level [109]. As the developer is responsible for writing the code, these insights could stimulate to, for example, minimize the number of invocations for a specific, high energy consuming method.

On the other hand we find metrics on infrastructure and organizational level, that are useful for higher level sustainability goals (e.g. by product management [33]). Green performance indicators [73] can, among others, be used to monitor infrastructure facilities and are of interest when EC needs to be considered on datacenter level. In their work, Lundfall et al. [88] present a tool to make the economic impact of green practices explicit with the purpose of justifying green practices on management level. The relation with

green software is apparent though as the figures often still stem from low level computing and application measurements.

3.2.2 Resource Utilization

Attributing the EC to the software itself requires monitoring the usage of the available hardware resource; i.e. performance measurements. Performance measurements provide insight in how the hardware components are stressed when processing instructions and specific performance metrics can be identified for each individual component [64]. For example, [89] monitors the overall system throughput, CPU, memory and hard disk through the 'number of instructions', 'CPU utilization', 'memory utilization', and 'disk transactions per second' performance metrics. A different approach is to assume theoretical EC figures, e.g. based on the specifications provided by the manufacturer [120], however this approach fails to account for the dynamic behavior of the software.

Important in this field of research is to select the relevant hardware components to monitor and the right instructions to process. Traditionally the CPU has been identified as the most decisive component for the EC by a system [12,136]. However, CPU based energy models do not capture all the power drawn by a system [76]. In [138] the contribution of each laptop component to the energy consumed is identified, e.g. the optical drive and LCD-backlight, and shows only 20% can be attributed to the CPU. On the other hand, in large-scale infrastructures the EC of network equipment is argued not to fluctuate heavily with increased traffic [23]. With regard to the instructions to process, a workload model should be made to reflect realistic conditions [89].

Resource monitoring is relevant on different levels related to green software. While investigating the EC of a server, a static and dynamic component can be identified [76]; static is the EC while the system is idle, i.e. minimum resources are used, and the dynamic EC fluctuates with the usage of the resources. As the static component is a large part of the EC, minimizing the absolute number of physical servers could significantly contribute to achieving the desired EC savings. In a cloud architecture the load of resources can be monitored (CPU, disk storage and network interface) and nodes switched on or off to minimize the overall power consumption [12]. This potential to scale up or down, based on performance monitoring, could help in achieving cost-effective scalability [36]. The dynamic part is relevant for green software practices and is determined by the resource utilization of the software.

3.2.3 Labeling Software Products

In [70] work has been done towards creating eco-labels for software in terms of a definition, criteria, form of representation, target groups and stakeholders. Sustainability is considered in the broadest sense of the word and, for example, also includes the sustainability aspects of the development process for the software. Following the main criteria that are identified, Kern et al. [70] continue with selecting those criteria that should be considered based on the life cycle phase of the software. This selection appears to be in line with the metrics and information needs as discussed above.

We deviate from [70] with respect to the form of representation. The authors build on international examples of eco-labels, although valuable in their own rights, which often do not allow for many details (i.e. low-level metrics) and are solely focused on specific aspects (e.g. CO_2 emissions). Consequently, apart from providing a starting point, the suggested eco-labels would be of limited practical value for those wanting to address the sustainability of their software.

Having said this, we consider the RUS and the eco-labels complementary in the area of green software. Eco-labels could help in selecting the tools, frameworks and services that positively impact the EC of a software product. For example as a criterion for the service-adaptation tactic [122] in a cloud context. The RUS, on the other hand, could serve as a more hands-on tool for software developers and architects.

3.3 Resource Utilization Score

In the search to determine a RUS for software EC, we investigate a means to combine performance metrics into a single score. However, we also acknowledge the importance of clearly communicating any key findings and the importance of lowering the threshold to interpreting the measurements through its presentation. To this end, we include a means to visualize measurements in our investigation.

3.3.1 Visualizing Measurements

In general visualizing a measurement, e.g. per software element [109], simplifies its communication and interpretation compared to raw measurements. However, visualizing measurements individually limits the user in combining metrics and neglects any relation between them. In the case of SEC, the

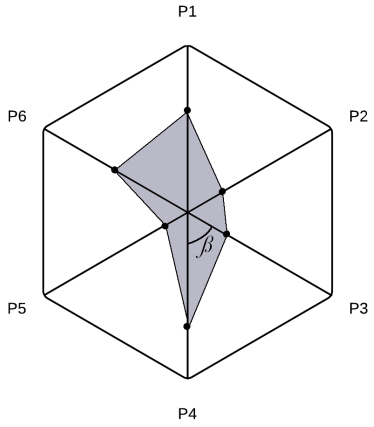


Figure 3.2: An example of a radar chart with example profile.

hardware components receive instructions from some instruction set translated from the software. As such there is bound to be a relation between the instructions that the components have to process. Consequently, we aim for a visualization method that is able to encompass all measurements in one figure and can serve as a basis for determining a score.

For our purposes we found a solution in the radar chart. According to Schutz, Speckesser and Schmid [132] the radar chart serves four goals:

1. Visualize interrelated performance measures through standardized scales.
2. Produce an effective description of selected performance dimensions in one synthetic indicator.
3. Analyze the change in overall performance between two points in time by comparing the surface of the same object.
4. Compare different objects through the shape of the surface for these objects.

Translating these goals to our context we can use a radar chart to visualize the performance dimensions related to the SEC, combine the dimensions into one single indicator (i.e. a score), analyze changes on different points in time (e.g. across releases [61]) and compare software products to one another. Under the condition that the same metrics are used for the chart. An example of

the radar chart is provided in Fig. 3.2, showing the dimensions (P1 through P6) for an unspecified object and the forthcoming surface (grey area) resulting from the scores on these dimensions.

In [98] the idea of the radar chart is applied to benchmark the performance of national labor markets. Although the results look promising, limitations of using the surface of the radar chart are also identified:

- The right dimensions should be selected for benchmarking a specific aspect.
- The right metrics should be selected to characterize these dimensions.
- The dimensions could contribute differently to an indicator (score) and as such could require a weighted inclusion.

The first two limitations concern selecting the right dimensions, i.e. axes, and the right performance metrics to characterize these axes. We address these limitations for our research in Sect. 3.3.2. The third limitation affects the calculation of the RUS and is addressed in Sect. 3.3.4.

3.3.2 Determining the Axes

It should be clear that the aspect under investigation in our research is the SEC. The first step is to determine the dimensions, i.e. the axes, that will form the radar chart for this aspect. From the related work we are able to distill the following dimensions that are directly or indirectly affected by the instruction set:

- CPU
- Memory
- Hard disk
- Network
- Power consumption
- Execution time

Each dimension has its own performance metrics (e.g. % usage versus bytes total per second [61]) and each metric is expressed in its own unit and scale (e.g. utilization percentage versus number of (M)Bytes). Selecting the dimensions and corresponding metrics should be done for each individual case as this

depends on the product under study. Note that the dimensions are not orthogonal, e.g. higher resource utilization results in increased power consumption.

Following the first goal presented for the radar chart, we should aim for a standardized means to present the measurements. Looking at the diversity of the list, one of the few options to determine a standardized score on each dimension is to use ranges. With regard to resource usage, a minimum resource usage can be determined in the situation where the hardware is idle and a maximum where the hardware is stressed to its maximum capacity [10]. The available resource, i.e. the margin between the minimum and maximum resource usage, forms the range. Note that the range should be determined individually for each performance metric. When an activity is performed using the software, the required resources can be divided by the range. This transforms measurements to a value between 'zero' and 'one' for that specific metric.

There is however a downside to working with ranges, as not every aspect can be expressed using a range. The execution time, for example, could have an infinite maximum, i.e. run as long as required without limitations. As such, we suggest to exclude these aspects from the chart itself and instead report these separately. For example, the units of work [67] to create a workload model are described separately to correctly interpret the measurements and the context in which they were found. We continue our work using the range method.

A final aspect is the order in which the axes are included in the radar chart. Using the same data in a different order can result in a difference of up to 300% [98], posing a threat to the third and fourth goal identified with the radar chart. We take this issue into account in the next section where we calculate a score for SEC.

3.3.3 Calculating the RUS

Continuing on the path of the radar chart, following goal three, we are able to obtain an objective performance measure by calculating the surface of the chart. This calculation is described by Mosley and Mayer [98] as the Surface Measure of Overall Performance (SMOP) and is calculated using Eq. 3.1.

$$\text{SMOP} = ((P_1 \cdot P_2) + \ldots + (P_n \cdot P_1)) \cdot \sin(\frac{\pi}{n}) \tag{3.1}$$

In the equation, the P-values represent the axes of the radar chart. The resulting number represents the surface of the figure created by all of the connected dots on the chart, i.e. the grey surface in Fig. 3.2.

The problem with Eq. 3.1 is that the axes are ordered implicitly. As there is no clear order between the different measures the axes represented, ordering them differently results in different values for the surface. As we do not have an explicit, clear order for the measures, a solution is sought by calculating the average surface based on all possible surfaces. Rephrased, we consider all possible relations between the axes. Instead of calculating for each possible order the corresponding surface, we observe that each possible triangle of axes is taken into account an equal number of times. Hence, calculating the surface for all different triangle suffices. Translating this to an equation results in Eq. 3.2:

$$\text{SMOP} = \sin(\frac{\pi}{n})(\sum_{i=1}^{n}\sum_{j=1}^{n}(P_i \cdot P_j)) \tag{3.2}$$

Observe that each score is multiplied by a constant factor. As we want to use the score for comparing different solutions, this constant can be left out. However, in its current form we see that axes could be paired with themselves, and that all pairs are counted twice (i.e. the symmetrical pairs $P_i \cdot P_j$ and $P_j \cdot P_i$). Taking these elements into account, the equation can be simplified as follows:

$$\text{RUS} = \sum_{i=1}^{n}\sum_{i<j}^{n}(P_i \cdot P_j) \tag{3.3}$$

A side effect of excluding the constant factor is that we do not longer calculate the surface of the chart, but rather a *score* based on the relation between the axes. As a result we are able to include non-standardized dimensions (e.g. execution time) in the equation that are not included in the radar chart. We labeled the score as the Resource Utilization Score.

3.3.4 Weight Factor

With the axes for benchmarking SEC identified and the ability to calculate a score, one important limitation remains that should be addressed: weighting the contribution of the different indicators. In our case this limitation counts for the performance metrics, but extends to the level of dimensions (i.e. axes). Concerning the first, we can only argue that the right performance metrics should be used to determine the scores on the respective axes. A hard disk, for example, could be characterized using the 'Disk I/O per second' and the '# Mb per second' metrics [64].

With regard to the axes we acknowledge that some combinations could be considered more important than others and have a bigger influence on EC [146] depending on the context. In large scale infrastructures, for example, memory plays an important role. As such, specific combinations that include memory could be argued have a greater contribution to the RUS. As these combinations are context dependent, we introduce a weight to the different ax-combinations, which results in the following score:

$$RUS = \sum_{i=1}^{n} \sum_{i<j}^{n} W_{i,j}(P_i \cdot P_j) \tag{3.4}$$

This score (Eq. 3.4) provides us with the RUS to characterize the resource utilization in relation to SEC. The weight factor can be used to reduce or amplify the effect of certain combinations of measures. In-depth analysis of the performance data, for example through a regression model [61], can help in determining the weight factors.

3.4 Experiment Design

To evaluate the RUS, an experiment was performed to select the most resource efficient algorithm to calculate the first N decimals of π from an application that provides many algorithms to calculate π. Additionally, as a more practical evaluation, we apply the RUS to an already available dataset [64]. In this section we describe the setup of the experiment for which we followed the guidelines provided in [66, 127, 148].

3.4.1 Experiment environment

For our experiment a test environment was prepared consisting of an application system, a logging system, and a measurement device (Fig. 3.3). The application system is the test hardware on which the software product is to be installed and as such the system to monitor. The logging system collects data from multiple sources and provides task instructions to the application system. Finally, the measurement device, a WUP[1], is used to measure the power drawn by the application system and calculate the SEC. Since the WUP is a separate device, the energy usage of the application system was not influenced by its measurements.

[1] https://www.wattsupmeters.com/secure/products.php?pn=0&wai=0

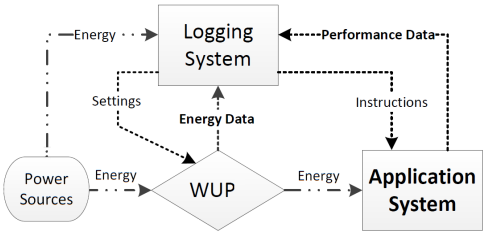


Figure 3.3: Experiment setup

In total three different application systems were included in the experiment, a laptop, desktop and a server, each representing a different computer class. For a system to be included in the experiment the device had to run the Microsoft Windows 7 *Professional* (or higher) operating system and be equipped with a multi-core Intel processor. These requirements provided us with systems that are capable of remote performance monitoring and are representative for modern systems in terms of computational capabilities. Details of the selected systems are shown in Table 3.1.

Performance measurements were collected using the Windows Performance Monitor[2] (Perfmon). Perfmon enables remote performance monitoring of systems with a one second interval in between measurements and is freely available with the Windows operating system.

[2]https://technet.microsoft.com/en-us/library/cc749154.aspx

Table 3.1: Specifications of the application systems.

	System		
Property	Laptop	Desktop	Server
Model	ASUS F3JA	*Custom PC*	HP DL380 G5
Processor	Core2Duo T7200	Core2Duo E6750	Intel Xeon E5335
FSB / TDP	667 / 34	1333 / 65	1333 / 80
Chipset	Intel i945PM	Intel P35	Intel 5000P
Memory	2GB DDR2	2GB DDR2	4GB EDO
Operating System	Windows 7 Professional (32 bit) Servicepack 2	Windows 7 Professional (32 bit) Servicepack 2	Windows server 2008 (64 bit) Servicepack 1

3.4.2 Test Application

To simulate activity, we used *Systester* (version 1.5.1)[3]; an application that calculates Pi decimals using the *The Quadratic Convergence of Borwein* and *Gauss-Legendre* algorithms. For the first algorithm both a single- and multi-core variant was available This enabled us to not only compare the difference between the two algorithms, but also between a single- and multi-core configuration. In order to have controllable runs the choice was made to calculate $8 \cdot 10^6$ Pi decimals per run.

For the actual experiment the remotely executable command line version of Systester was used, i.e. from a batch script using the logging system. In addition, a modified version of Systester was compiled where the application waits five seconds after initiating and before ending the process. The presence of this five second interval allowed Perfmon to collect all data related to a task and made it easier to identify the specific runs during processing, thereby directly contributing to the quality of the data and forthcoming analysis.

3.4.3 Metrics and Utilization Ranges

Visualizing measurements on a radar chart requires the metrics to be expressed on a standardized scale. To do so, we require the minimum (zero) and maximum (one) resource utilization figures which allows us to express measurements as a value on this continuum. The idle measurements (minimum) were performed by leaving the application systems idle (without going to a sleep state) for at least 30 hours and monitoring the resource utilization and power consumption during this period. To determine the maximum resource utilization figures the application systems were stressed to their maximum capacities using HeavyLoad[4].

Based on the characteristics of Systester, the following metrics were selected to create a radar chart:

- **CPU**: '% CPU time'. The maximum utilization is 100% per core adding up to a percentage above 100% for multi-core systems. The range was determined with the '% CPU time' while idle and using HeavyLoad.
- **Memory**: 'Available bytes'. The number of bytes that is available of which the value decreases as processes require memory. The maximum is the available bytes while idle which also indicates the range for this metric.

[3] http://systester.sourceforge.net/
[4] http://www.jam-software.com/heavyload/

- **Disk**: '% disk idle time'. The time that the disk was idle. The maximum utilization for the hard disk is 100% and its range is found by subtracting the '% idle time' of an idle system from this 100%. While the '% disk time' metric can also be used, this metric exaggerates[5] disk utilization.

- **Power consumption**: The 'power consumption' (in Watt) by the system while performing a run. The range was determined per system by measuring the 'power consumption' while idle and while using HeavyLoad.

Given the nature of the application we decided to exclude network metrics. Note that the actual SEC is calculated using the WUP measurements and stems from a different source than the performance measurements.

To calculate the RUS, the standardized metrics of the radar chart will be combined with the non-standardized 'execution time' metric. The 'execution time' is defined as the time required to perform a specific task and could be a determining aspect for SEC [64]. In our experiment the execution time is the time for Systester to calculate $8 \cdot 10^6$ Pi decimals.

3.4.4 Experiment protocol

To actually perform the experiment a protocol was followed containing every activity required to perform a series of runs. A run is one time for the application to calculate $8 \cdot 10^6$ Pi decimals plus the five seconds before and after performing this task. A series can be configured to include multiple runs. For each series a script was used, *PiBatch*, which automates monitoring with PerfMon, optionally includes rebooting the system, and performs a specified number of runs. The script minimizes human interference, provided that the following preparations are made:

- Install PsTools[6] for executing commands remotely.

- Install software to remotely manage the WUP.

- Remove the battery from the laptop to eliminate battery charging and discharging effects.

- Configure Windows power settings and disable unnecessary services (e.g. Windows Search and Update).

[5] https://technet.microsoft.com/en-us/library/cc938959.aspx
[6] https://technet.microsoft.com/en-us/sysinternals/bb896649.aspx/

- Configure Perfmon data collector set.

Additionally, the effect of rebooting the application systems was investigated. In a small experiment we found that a system was 'unpredictable', i.e. random active processes, in the first 15 minutes after rebooting. The PiBatch script takes this into account by waiting at least 15 minutes before starting the first run of a series. This resulted in the following protocol:

- Clear WUP meter data and test connections.
- Configure and initiate PiBatch script .
- Collect data from PerfMon and WUP.

At the time of the experiment, rebooting was made optional in the script as rebooting the server appeared not possible with a virtual machine running. The virtual machine was running on one single, dedicated server and was isolated from other infrastructural facilities ensuring that the only hardware that is affected is the hardware being measured. To get a representative data set, we decided to continue until at least thirty clean measurements per combination were obtained.

3.4.5 Post-processing the Measurements

After performing a series of runs, post-processing was required before analyzing the data.

Determine run execution time; The runs appeared of variable length and hence we needed to determine the exact execution time for each run using the '%CPU Time' of the application process (provided by PerfMon). The execution interval started when the '%CPU Time' was more than zero and ended when it went back to zero again.

Synchronize WUP and Perfmon timestamps; Since the WUP and PerfMon data stemmed from separate sources, the timestamps of the measurements required synchronization. The start of an interval in the WUP measurements, i.e. an increase in power drawn, was matched to the moment of initiation of the associated processes in the PerfMon data. Cross-checking on corresponding end points ensured that the synchronization was correct.

Assess quality of runs; Despite our efforts to control any effects that could influence the experiment, we observed activity unrelated to Systester during the experiment. First, we used the '% CPU time' on process level to check whether a system was solely processing tasks related to Systester during a run. In addition we monitored the EC for discrepancies as the performance

Table 3.2: The RUS for each combination (lower is better).

	Laptop	Desktop	Server
Borwein, single-core	393.44	421.15	354.57
Borwein, multi-core	358.87	405.73	377.52
Gauss-Legendre	158.53	194.56	147.74

Table 3.3: EC consumption figures for each combination in Joule.

	Laptop	Desktop	Server
Borwein, single-core	17,989	26,092	103,165
Borwein, multi-core	14,724	20,555	82,204
Gauss-Legendre	7,442	10,891	44,870

measurements could not always explain increases in power consumption. Runs showing odd patterns were excluded from further processing.

3.5 Evaluating the RUS

The results of the experiment are summarized in Fig. 3.4 which shows the radar charts for each combination of the three systems and Systester options. The charts show the average score on each metric for the specific configuration, e.g. the laptop on average used 69% of the available CPU resources with the multi-core quadratic convergence of Borwein. In total 32 runs were performed for each algorithm on the laptop, which were are all clean. On the desktop 34 runs were performed with the Gauss-Legendre algorithm and 32 runs with both Borwein algorithms, providing 31 clean runs per algorithm. The server appeared the most problematic system where we performed 80, 72 and 55 runs for the Gauss-Legendre, single-core and multi-core Borwein algorithms to obtain 42, 42 and 55 clean runs respectively. Given the instability, we decided to obtain at least 40 clean runs for the server.

Since the execution time was not suitable to express on an axis, we provided this information alongside the corresponding radar chart. At a glance we can conclude that the Gauss-Legendre algorithm is the fastest option of the three, and that the multi-core variant of the Borwein algorithm is faster than its single-core variant. Surprisingly we find that the server, despite its computational capacity, on average is at least 30 seconds slower compared to the other systems with the Gauss-Legendre algorithm. A trend that also shows

with the other algorithms. We were not able to find the cause of this discrepancy, but we argue these figures could be typical for the server and associated hardware possibly in combination with the multi-core capabilities of Systester itself.

The impact of the execution time can be made clear using the actual SEC figures (Tbl. 3.3) on the account of Systester. Although the metrics indicate a fairly similar resource utilization pattern across machines, in terms of absolute SEC we find that the server consumes more energy. The same holds for the desktop compared to the laptop; similar scores, higher SEC by the desktop in absolute terms. If we consider the SEC findings in light of the radar charts we solidify the argument on adding the execution time to the visualization.

Looking at the dimensions themselves we find that in this particular case there seems to be a relation between the scores on the CPU and power dimensions, i.e. an increase on the CPU dimension pairs with an increase on the power dimension. Also, as expected, the CPU scores are highest with the multi-core Borwein algorithm, but do not double in comparison to the single-core version. The difference between the Borwein measurements could be an indication of the potential for multi-core (i.e. multi-threaded) applications.

With regard to the memory and disk metrics, the laptop and server charts indicate minimal impact on the memory and disk dimensions. However, the radar charts clearly indicate a different situation for the disk utilization by the desktop. While we find the high disk utilization for the desktop peculiar, we cannot attribute this utilization to Systester as the other systems do no exhibit this behavior. Based on the information a further analysis can be performed on the desktop.

3.5.1 RUS Scores

The corresponding RUS for each combination is provided in Tbl. 3.2. The RUS was calculated using the standardized scores of the performance metrics and the execution time in seconds (non-standardized). Hence, the scores are larger than 'one'. As there were no indications to prefer a specific dimension over the others we set the weight factor for all pairs of dimensions to 'one'. Important to notice is that a lower score means that a specific combination scores better; i.e. requires less resources.

Comparing the RUS with the EC figures (Tbl. 3.3) we find that in general lower RUS scores are accompanied by lower EC figures. The only exception is with the single- and multi-core Borwein algorithms on the server. Looking at the radar charts we find that the multi-core variant shows higher utilization scores on the power and CPU dimension, an increase that is also visible on the

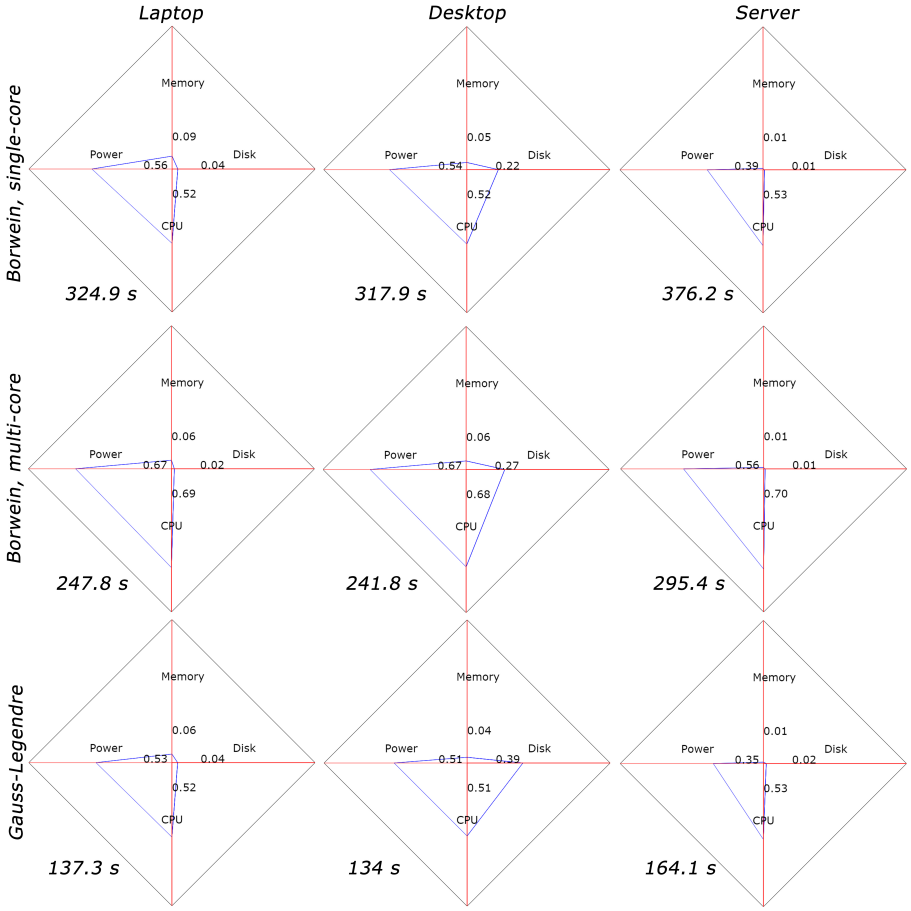


Figure 3.4: The collection of radar charts showing the resource utilization on each dimension and execution time for the nine investigated combinations.

other systems. In relation to the other systems we can only conclude that the difference in execution time is enough to offset the EC figures but not the RUS. From a practical perspective, the single-core variant could be preferred above the multi-core variant when trade-offs should be made (e.g. when resources are shared).

If we solely use the RUS scores to choose an algorithm and platform, the decision would be to run the Gauss-Legendre algorithm on the server. How-

ever, based on the EC we should actually prefer this algorithm on the laptop or, if we favor speed, on the desktop. This observation learns that the RUS should be considered complementary to the EC and that we cannot use the RUS to compare across classes of systems.

3.5.2 RUS for a Commercial Software Product

As an additional evaluation we applied the theory to a commercial software product (Document Generator) with the instructions to generate 5000 documents [64]. The metrics for the radar chart were '% CPU time' (CPU), 'available MBytes' (memory), '% disk idle time' (hard disk), 'power consumption' (in Watt) and the 'total Bytes per second' (network). Compared to Systester the network metric was included and and its range was determined using Lan Speed Test[7].

In this case an architectural change was applied to make Document Generator multi-threaded. The resulting decrease in CPU Utilization, i.e. from 49% to 19.2%, lowered the EC per document with 67.1% This finding is visible in the radar charts (Fig. 3.5) where a decrease in the the CPU and power utilization can be observed. A minimal increase in utilization of the disk and memory was found, whereas the network utilization remains unchanged. Especially the CPU utilization, or more specifically the division of the workload between CPU cores [64], seems decisive for the power dimension.

The RUS scores (Tbl. 3.4) were calculated using the execution time and appear to be in line with the EC measurements; i.e. a lower score means less SEC. This finding possibly suggests that the utilization patterns after adjusting the software are more 'natural' to the system.

[7] http://totusoft.com/lanspeed/

Table 3.4: The EC (in Joule) and RUS for the Document Generator software product before and after changing the software.

	EC	RUS
Single-threaded	17,560	2,313.09
Multi-threaded	5,782	1,644.49

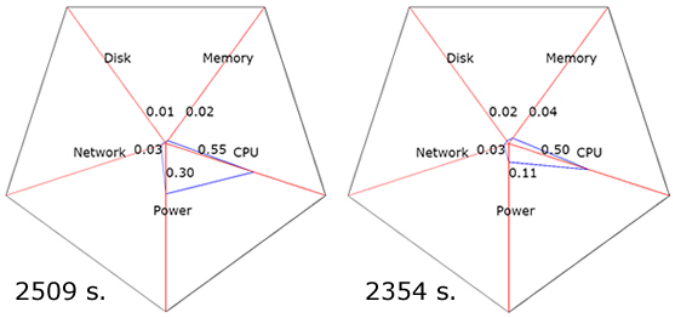


Figure 3.5: The radar charts for the Document Generator software product before (left) and after (right) making the software multi-threaded [64].

3.6 Discussion

In this research we investigated the possibility for a RUS to express resource utilization in relation to the SEC. The constructed score is based on those dimensions that are deemed relevant for the software and is flexible to be adjusted depending on the product and the environment in which it will be executed. Initial evaluation showed promising results to engage in green software practices. There are, however, several limitation to our work which we discuss below.

Hardware dependency; Like EC, the RUS is dependent on the hardware that the software is executed on. Although the measurements are standardized, the ranges themselves showed considerable differences across systems. Comparing the RUS and EC figures led to the insight that the RUS cannot be used to compare a software product across classes of systems. Additionally, it can be impossible to determine the ranges in environments with 'unlimited' resources (e.g. cloud data centers). Different benchmarks should be found when this is the case, for example benchmark release of software to one another to visualize the effect of software changes.

Visualization; The RUS is based on the theory related to the radar chart. Even though we investigated different theories, other options could exist that better fit the purpose. For example, theories that better consider the relation between dimensions or express a score based on the dimensions.

Robustness; The RUS was successfully applied to both a synthetic (Systester) and commercial (Document Generator) application, which shows the generic ability of the theory to be applied. Although we are confident in the

validity of our results, more applications of RUS are required to prove or disprove the robustness of the RUS.

Weight factor; The weight factor for calculating the RUS is a topic that still requires further investigation. In our experiment we could not motivate a higher score on one combination of dimensions over the other and decided to set the weight factor to '1' for each combination. However, other situations might require a thorough investigation to determine the correct weight factors.

3.6.1 Experiment Limitations

Despite our best efforts, there are limitations to the experiment as described:

Windows processes; Thirty minutes after rebooting we observed an increase in activity for an unspecified period of time. The cause is unknown, but we assume that Windows-related processes are triggered by a timed mechanism which we cannot control. However, we did not find significant differences between runs executed after twenty or 200 minutes and between Windows 7 and Windows Server 2008.

Measurement interval; WUP and Perfmon perform measurements with a one second interval, while computers process millions of instructions per second. Although we argue our measurements are sufficient for our purposes, we acknowledge the fact that data is lost with the instruments at hand.

Room temperature; Of the three systems the server was the only one situated in a climate-controlled data center and as a consequence we can only guarantee identical conditions for this system. Although we tried to maintain consistency, we acknowledge the fact that, among others, room temperature could have influenced our measurements. We consider the insignificant differences found between measurements as a confirmation that the influence in our experiment was limited.

3.7 Conclusion

In this paper we propose a metric to effectively communicate resource utilization measurements for a software product in relation to EC. The metric should be easy to understand, reported uniformly and clearly communicate key findings. We consider the viewpoints of multiple stakeholders wanting to address the sustainability of their software product through green software practices, and posed the following **research question**: 'How can we effectively express the resource utilization for executing a software product in relation to the SEC?'. We provide an answer by constructing the RUS.

Following the goals of the of radar chart, the RUS delivers a single score based on selected dimensions and performance metrics. To calculate the RUS, the equation to calculate the surface of a radar chart was transformed into one that considers the relation between dimensions. A weight factor is added that enables stakeholders to determine the importance of each pair of dimensions. As the measurements can be expressed on a standardized scale they can be interpreted more easily, do not require knowledge on the individual metrics and can be compared between software applications. Additionally, the radar chart provides a means to visualize the measurements which helps to identify key findings.

Evaluating the RUS with two different datasets, showed that the RUS should be considered complementary to the EC and the execution time related to a software product. In general a lower RUS corresponds to a lower EC consumption figure, but with the server a case was also found where a lower RUS was accompanied by a higher EC. In these situations a trade-off should be made, like with quality attributes, favoring the aspect that is considered more important in a specific context. A limitation of the RUS found in its inability to be compared across systems of different classes.

Based on the work presented in this paper, we identify several direction for future research. First is to investigate the RUS more thoroughly. For example, the RUS could be used to compare between systems within the same class. Also a (standardized) means to determine the weight factor could aid in the RUS' acceptance. A second direction is to investigate the positioning of the RUS in relation to more the generic eco-labels for the ICT domain. A final direction is to use RUS to create awareness on green software during the development process. By showing the impact of software development activities, software developers are enabled to address sustainability issues that might arise.

Part II

Energy Consumption in Software Architecture

4

Extending Software Architecture Views with an Energy Consumption Perspective

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%.

This work was originally published as:

E. Jagroep, J. M. van der Werf, S. Brinkkemper, L. Blom, and R. van Vliet. Extending software architecture views with an energy consumption perspective. *Computing*, 99(6):553–573, 2017

4.1 Introduction

The EC of the 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 [95]; 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" [99]. 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 [82] 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 [139]. Thus software is argued to be the true consumer of power [140].

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. [45,67]). 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 [113]. 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' [150]. An Architecture Description (AD) of product software complemented with EC measurements, could help to direct green computing efforts (i.e. energy efficient algorithms [53]) 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 [51] 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 [126] 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 [126], come with experience and are deemed future research. Second, since a perspective addresses quality properties [6, 126], 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 [64], 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. 4.2 and position sustainability as a QA (Sect. 4.3). With this knowledge an energy consumption perspective (Sect. 4.4) is constructed and applied in practice (Sect. 4.5). Finally, we provide a conclusion and identify directions for future research (Sect. 4.6).

4.1.1 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. 4.1) 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.

Fig. 4.1 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 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.

4.2 Related Work

In [25], 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,

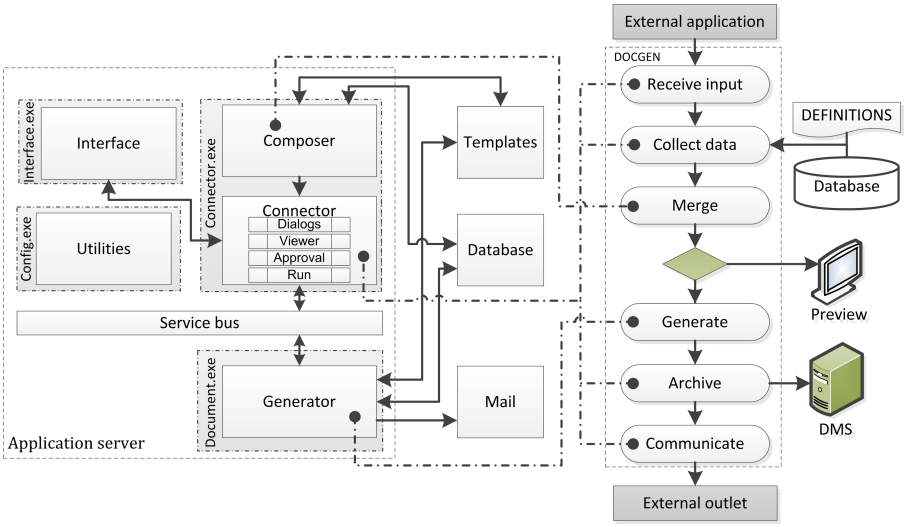


Figure 4.1: The Document Generator (DG) workflow mapped on the functional architecture.

a niche of sustainability where the software is the object of optimization [82], 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.

4.2.1 Energy Consumption Measurements

One of the main issues with respect to green software is to perform detailed EC measurements. Specialized environments, e.g. [39], 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 [45]. 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 [109] 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 [124]. A more simple software-only approach is to use energy profilers [107]; 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 [63].

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 [22]. In the paradigm of 'programmable web' [21], 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 [38] '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.

4.2.2 Relating Green Software to Software Architecture

While the hardware and software approaches for measuring have up- and downsides to them, both serve the same purpose; identify 'software energy hotspots' [124] 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 [8, 97, 106, 156] and concrete [1]. Even without changing the current practice of software engineers [106].

At its core the creation of energy efficient software starts with the design of the software [8], 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 [38, 73] 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'.

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

Different views that map the EC on software artifacts already exist. The node map presented in [45] for example, closely resembles what could be labeled as a deployment view showing the installation of software elements across the available hardware. Analyzing the node map on EC provides a so-called 'heat map' of the system. Following this same line, [67] 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 [150], research on EC also adopts an architectural approach. In [153] 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 [142] functioning as 'energy broker'. An approach that has recently also been applied to software in general [109]. 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 [33], has a means to address his concerns in the product design [126] and determine whether the desired quality of service is achieved [79]. 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 [25].

4.3 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 [82, 83]. 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. [45, 67, 71]. In [67], for example, 'performance efficiency' (ISO 25010) is transformed into 'energy efficiency' with three

high-level issues ; energy behavior, capacity and resource utilization. Also 'energy efficiency' itself has been positioned as a QA [122].

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' [83]. The (sub)characteristics are not mutually exclusive though, as the associated measures could be similar.

Since a SA allows or precludes nearly all QAs [6], 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 [16,45,67,71,73], *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

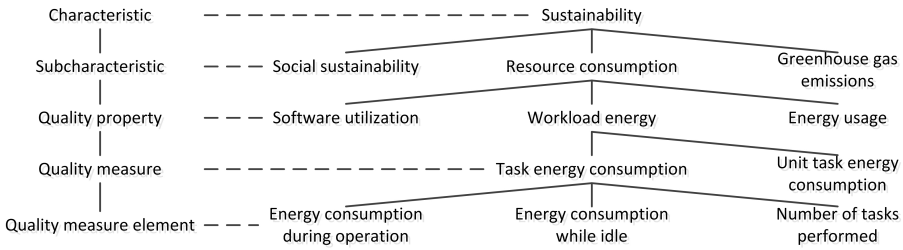


Figure 4.2: A partial breakdown of the sustainability characteristic linked to the ISO 25010 standard.

discussion between stakeholders [45]. 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 [16], cover the resource consumption subcharacteristic.

4.3.1 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. 4.2, 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 4.1 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)' [67], 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

Table 4.1: 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. $\frac{UEC}{\#\text{ of tasks performed}}$

instance and can be used to quickly identify outliers in EC.

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*)' [67].

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.

4.3.2 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 [38]) 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 [156]. 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 [4].

4.4 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' [126]. 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 [126]. As it is not possible to provide an exhausting set of guidelines, we recommend similar research (e.g. [45, 67, 109]) as guidance.

4.4.1 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. 4.3) [126] 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. 4.3), 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

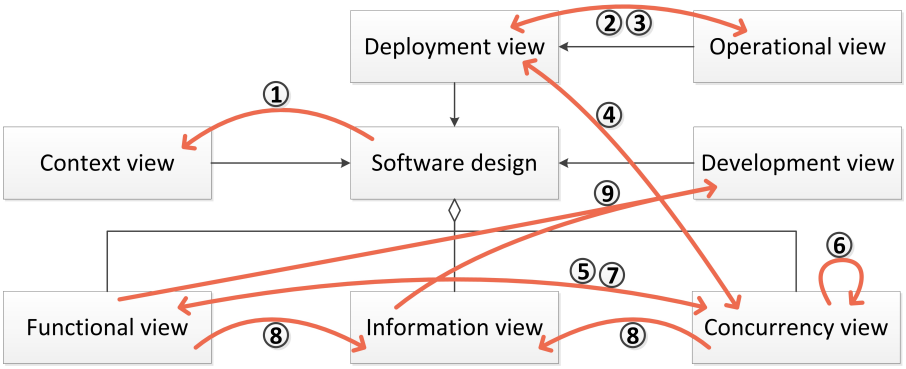


Figure 4.3: Viewpoint catalog after [126], expanded with the flow of key questions to address EC concerns.

total cost of ownership for a software product, i.e. it influences the economic aspects (cf. [16]).

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. 4.1), 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. 4.1) 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 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. 4.1) 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, energy efficient 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 Energy Efficiency (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.

4.4.2 Perspective Activities

Following [126], we provide a set of activities (Fig. 4.4) 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 [6] 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 EC and performance measurements and can be visualized by creating an overlay for the AD, e.g. Fig. 4.5 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. 4.4.3), patterns and

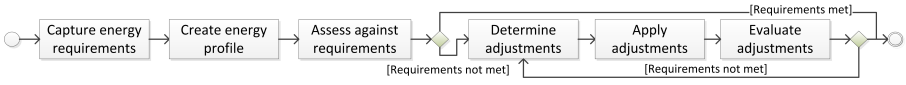


Figure 4.4: The activities to apply the perspective to software architecture.

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.

4.4.3 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 [6] 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 [122] 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 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 [142], 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 [45]; 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 [155]; 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.

4.5 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. 4.5).

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. 4.6). DG was installed on the test server[2] which consequently was the system to perform measurements on. The measurement equipment, a 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 [63]. 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

[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

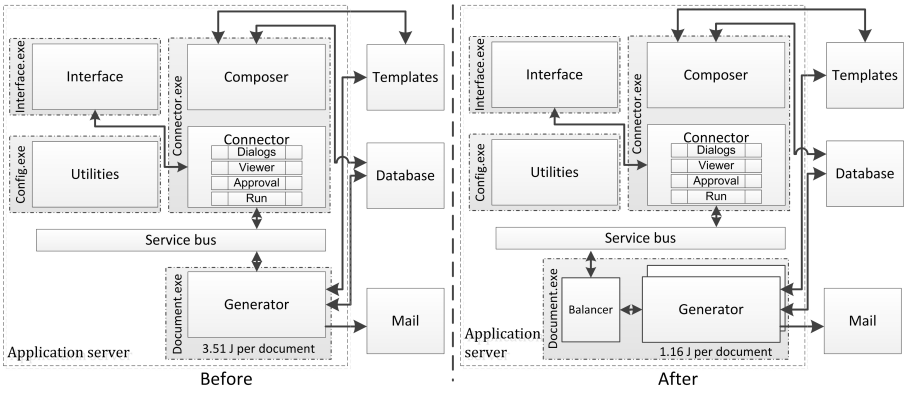


Figure 4.5: The functional architecture of DG for subsequent releases including a concurrency view (executables) and deployment view (installed on a single application server).

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. 4.1.1) that the 'Generator' element is responsible for the actual document generation. Mapping the measurements on the AD (Fig. 4.5), 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 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.

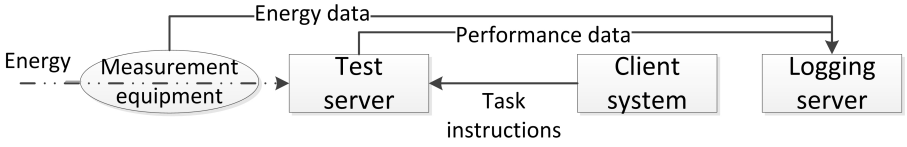


Figure 4.6: Setup of the test environment used to perform the case study.

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. 4.5). 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. 4.7). 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.

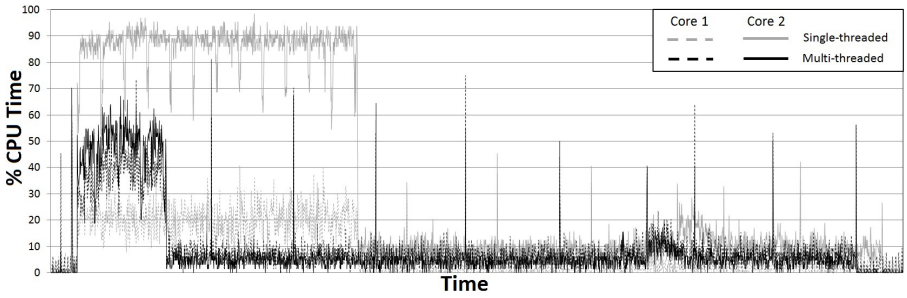


Figure 4.7: Comparison of CPU activity (2 cores) of the test server during a measurement before (single-threaded) and after (multi-threaded) applying the architectural change.

4.5.1 Threats to validity

With regard to the validity of the case study, an evaluation is performed following the threats as identified in [127].

The *construct validity* considers whether the correct measures were identified for the object under study. In Section 4.3 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.

4.6 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 identified 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.

5

Hunt the Energy Guzzler in my Software: An Architectural Approach

Software producing organizations nowadays have the ability to address the energy impact of their software products through their source code and software architecture. In spite of that, the focus mostly remains on hardware aspects, which limits the contribution of software towards obtaining more energy efficient ICT solutions. A major contributor to this imbalance is the inability to measure the energy consumption of specific software elements. Consequently, software producing organizations are unable to fully control this aspect of the software. In this paper we propose a method to determine the energy consumption of individual architectural elements. We apply the method in an industrial experiment, using a commercial software product, and unravel the energy consumption related to its core functionality. This method enabled the organization to identify the major energy consuming elements and thereby the code that allows this qualitative aspect of the software to be addressed. Additionally, the results provided valuable insights for future design and trade-off decisions with respect to the sustainability of the product.

This work has been submitted for publication as:

E. Jagroep, A. van der Ent, J. M. E. M. van der Werf, S. Brinkkemper, L. Blom, and R. van Vliet. Hunt the energy guzzler in my software: An architectural approach. *Submitted for publication*

5.1 Introduction

Software is increasingly being acknowledged as the driver behind the energy consumption of ICT solutions [82].A striking example is the impact a single app can have on a smart-phone. Poorly written software can drain the battery, despite a large variation of possible hardware components. This effect on energy consumption also holds for larger applications, e.g. software products [150], where software can eliminate any energy efficient features built into the hardware [139]. Unfortunately, with the latter, hardware investments remain the preferred option, thereby limiting the contribution of the software towards the energy efficiency of a system.

On the other hand we find software qualities to increasingly become a decisive factor with respect to software products [6]. If a software product is not energy efficient, data center operating costs will increase the Total Cost of Ownership (TCO) thereby diminishing the economic viability of the product. Equally, if a software product does not perform as required, a functionally equivalent and better performing product is often quickly found. Implying trade-offs have to be made between qualities. In light of these examples, we consider sustainability as a property of software quality with a social, environmental, technical and economic dimension [83]. Energy related concerns for the software, as discussed in this paper, can be attributed to the environmental dimension of sustainability.

Recent studies, e.g. [51, 56, 121], show the significant impact green software can have on energy consumption. For a SPO, e.g. independent software vendors, applying such changes to their software products requires them to measure the energy consumption of their software products: a relatively unfamiliar terrain. In practice, performance is often used as a proxy despite the fact that this aspect does not always have a direct relation with energy consumption [144]. As a result, SPOs are unable to control the contribution of software to the energy consumption of their ICT solutions. One way to obtain insights in energy consumption would be by performing measurements on the software in production. However, applying adjustments to the software in this stage of the lifecycle are often most expensive [102]. Consequently, energy efficiency of software should be studied in the earlier stages of software development.

In this paper, we study energy consumption of software in more detail. Based on the Energy Consumption Perspective (ECP) [60], we propose the Stubbed Energy Profiling Method that enables SPOs to create an in-depth energy profile of their software products. The methods builds upon substitut-

ing functional elements by stubs, to study the change in energy consumption. The delta between the initial and stubbed version explains the energy consumption of the stubbed functional element. In this way, a SPO can not only investigate the software product in total, but also of individual important elements in the software like functional modules and individual services.

To validate the Stubbed Energy Profiling Method (SEPM), we applied the method in an experiment using a commercial software product. Together with the architect and product owner, we determined the important architectural elements in relation to the core functionality, and created an energy profile that identified the most prominent energy consumers in the software. Stakeholders of the product now know what software should be targeted to maximize the impact of sustainability efforts. Additionally, the energy profile of the software helped the architect to make better informed trade-off analysis with respect to the maintainability and sustainability of the software.

The remainder of this paper is organized as follows. Sect. 5.2 provides background on the ECP and introduces our energy profiling method. In Sect. 5.3, we describe the design of our experiment where we apply the proposed method, followed by the results (Sect. 5.4) and discussion thereof (Sect. 5.5). In Sect. 5.6 we discuss the limitations and threats to validity of our work, followed by the related work in Sect. 5.7. Last, Sect. 5.8 concludes the paper.

5.2 Energy Profiling Method

An essential step in enabling the ICT industry to reduce the energy consumption of their software, is to address this qualitative aspect in the SA. For this purpose, we introduced a quality perspective on energy consumption, i.e. the ECP [60]. The ECP provides a collection of activities, tactics and guidelines stakeholders can use to ensure that a system exhibits a particular set of related quality properties [126]. However, although the introduced perspective includes a method to measure and relate energy consumption to the SA, we found that creating a detailed energy profile of a software product remains challenging.

An energy profile portrays the energy consumption of software elements related to an energy requirement; a condition or capability needed to solve a problem or achieve an objective [6]. Stakeholders of a software product, e.g. software architects and developers, use the energy profile to determine what changes can be applied to meet the requirement and where they can be applied. Hence, an energy profile is an essential instrument in addressing the energy consumption of software.

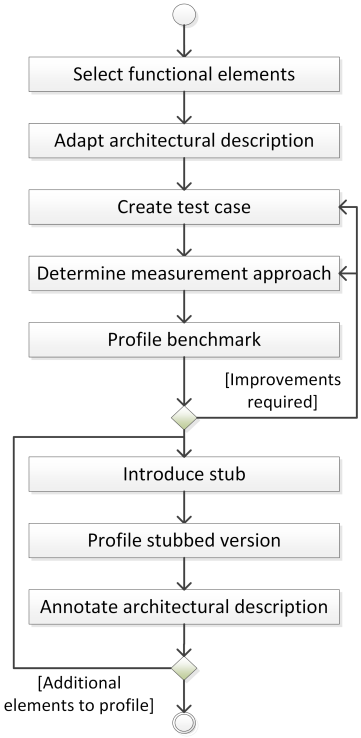


Figure 5.1: The Stubbed Energy Profiling Method extending the 'Create energy profile' activity in the energy consumption perspective [60]

In relation to software products, we found that energy consumption is often investigated through differences between releases [51, 61] or at runtime [68, 133, 134]. Such approaches are appropriate to determine energy consumption changes in hindsight, but fails to provide insight when there is still the possibility to address any remaining concerns, e.g. during development. From a practical point of view, addressing concerns at a later stage in the development cycle drives the total cost of ownership for a software product. This negatively affects the economic dimension of sustainability [83].

To address these issues, in this paper we focus on analyzing a single version of a software product to be able to pinpoint the energy consuming elements within a single release. In this way, the proposed method can be seen as part of the 'create energy profile' activity in the ECP [60]. Our method, the SEPM,

builds upon creating stubs for individual functional elements [100]. The stubs are configured such that they return a standard response instead of executing the intended code, while ensuring the runtime capabilities of the release.

The main idea of the SEPM is depicted in Fig. 5.2: for each architecturally important element, in this example A, B, and C, we create stubs A', B' and C', and perform tests on the resulting versions $S1$, $S2$, and $S3$. Comparing these versions with the Benchmark, the non-stubbed original, provides insights in the energy consumption of the stubbed element. For example, the difference between the Benchmark and $S1$ provides insight in the energy consumption of element A. Depending on the scope of the requirement and the test to be performed, more detailed stubs can be introduced, e.g. by individually stubbing sub-elements B1 and B2.

The Stubbed Energy Profiling Method is depicted in Fig. 5.1. In the remainder of this section, we present the activities of the method.

Select functional elements. The first activity is to identify the target functionality in relation to an energy requirement and the related functional software elements. As the functional viewpoint is the most commonly created architectural description, this activity should be easy to perform for many software products [126].

Adapt architectural description. After identifying the functional elements, the architectural description needs to be adapted to provide a sufficient level of detail on the functions comprising the target functionality. These details are used to identify where stubs are required and what stubs can be created for the different functions. We acknowledge that not all software el-

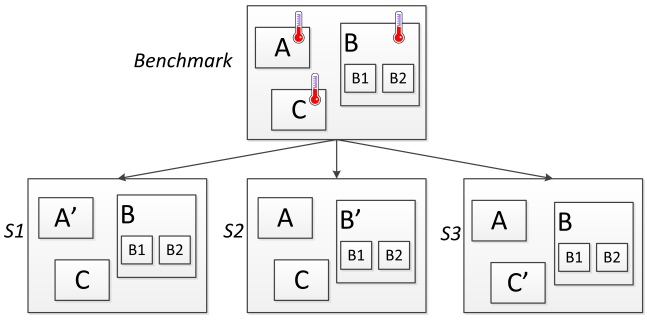


Figure 5.2: By stubbing elements A, B and C the energy effect of each individual element can be determined.

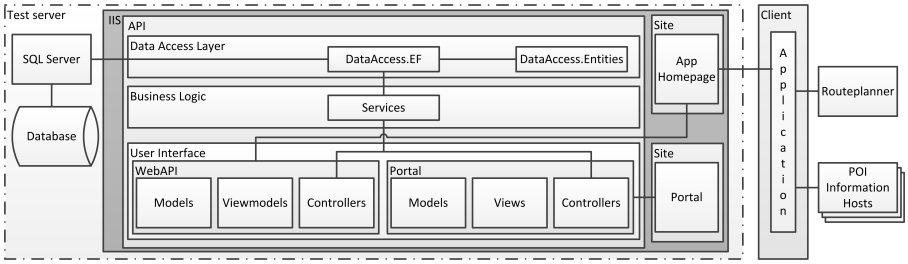


Figure 5.3: The software architecture of the benchmark CITEX version.

ements can be stubbed individually, e.g. due to the inability to isolate the component or module under test. In such a case, the adapted architectural description should still identify the meaningful software units in relation to the requirement.

Create test case. Based on the identified functional elements, a test case should be designed that executes these functional elements. While a stakeholder is free to design the test, realistic usage scenarios are found to best fit practitioners information needs when discussing energy usage [91]. Additionally, the stakeholder should ensure the software is stressed such that differences can be observed. For example, a test case could be designed to simulate a significant amount of concurrent transactions, thereby stressing the software to its maximum capacity. It is key to design a test case that reliably excludes incidental findings and can be consistently applied for all relevant software elements.

Determine measurement approach. Based on the test case, the appropriate metrics should be identified that quantify the target functionality and allow for evaluating whether requirements are met. In turn, the metrics, of which a solid list is available in literature [16], determine the measurement approach that a stakeholder should apply. Multiple measurements approaches exist depending on the equipment and resources available, e.g. [68, 109, 133]. In Sect. 5.3 we provide a detailed description on how energy measurements were performed in the experiment comprising our research.

Profile benchmark. The next activity is creating the benchmark to evaluate the stubbed versions of the software against. Hence, the test case is performed on the software without stubs. Since this is the first time that the test case is performed, the results from this activity could require improvements of the test case and measurements approach.

Introduce stub. If the benchmark is profiled and the test case and measurement approach are found suitable, the first functional element is stubbed. Inserting the stub in the original software results in a new software version, S'.
Profile stubbed version. Once version S' with the stub is obtained, the test case can be performed on it, resulting in a new set of measurements.
Annotate architectural description The final activity is to annotate the architectural description with the energy consumption effects caused by the stub. By comparing the newly obtained measurements with the benchmark, the difference in energy consumption van be assigned to the stubbed functional element.

The final three activities of the SEPM are repeated until all functional elements identified in the adapted architectural description are profiled.

5.3 Research Design

To demonstrate the SEPM, we performed an experiment on a commercial software product. In this product, we selected the main functional element, and compared the energy and performance aspects to determine the impact of (i) an energy requirement and (ii) of refactoring the software. In relation to the energy requirement, the architect also wanted to study different design alternatives, resulting in a total of four versions that we compared to the benchmark.

In this section we present the design of the experiment, which follows the guidelines presented in [66, 148, 152], followed by the results (Sect. 5.4) and a discussion thereof (Sect. 5.5).

5.3.1 Product Under Study: City Explorer

City Explorer (CITEX) is a commercial software product that enables visitors of a city to explore Point Of Interests (POIs), such as historical buildings, through predefined routes. The routes are created by CITEX customers, called route managers, and made available to visitors via a smartphone app. With each POI, additional information is provided through the app. Additionally, the app provides an augmented reality experience using, e.g., historical pictures. The product is used by 7 route managers, mostly municipalities and tourist information offices, and serves over 8000 visitors on an annual basis with peaks of 500 daily visitors during special events.

Table 5.1: The CITEX releases included in the experiment.

Version	Version label	Description
1	Benchmark	Production version of CITEX without any stubbed elements.
2	NoInfo	Version based on the Benchmark where the 'Info' element is stubbed. The code to obtain POI information is not executed.
3	NoPOI_NoInfo	Version based on the Benchmark where the 'POI' and 'Info' elements are stubbed. The code to obtain POIs and associated information is not executed.
4	RouteFilter	Version based on the Benchmark where a filter is applied by the client to only obtain the relevant routes based on the geographical location of the user. No specific stubs are applied in this version.
5	Monolith	First production release of CITEX that the team describes as a monolithic application. This version is the predecessor of the Benchmark version, in which no specific stubs are applied.

The SA of the CITEX is depicted in Fig. 5.3. The application consists of three layers: Data Access, Business Logic and User Interface. The User Interface Layer contains two major elements: the WebAPI and Portal, that provide an interface for the app and a management portal for route managers, respectively. A SQL Server instance is used to store route information. The use of third party components is limited to (i) a Routeplanner, that provides the routes to follow based on GPS locations, and (ii) multiple information hosts providing the information concerning the POIs. All third party components are accessed directly by the app running on the client's smartphone.

5.3.2 Target Functionality and Architectural Description

Together with the architect of CITEX, we formulated the requirement for CITEX to increase the energy efficiency while executing its core functionality: providing routes to visitors. Following the SEPM, a second architectural description was created to provide detailed insight into the target functionality

(Fig. 5.4). This description allowed us to break down the core functionality into three separate activities:

A obtaining a route including route information,

B obtaining the POIs,

C obtaining the information related to POIs.

Each activity was mapped on the corresponding functional element in Fig. 5.4, which provided guidance to the developer to apply the required stubs.

Ideally, individual stubs would have been applied for the 'Route', 'POI' and 'INFO' elements to determine their energy impact. However, investigating the code, learned that there is a dependency between the elements that would not allow such an approach. For example, it appeared that the POIs and corresponding information are linked to a route in the database. Hence, stubbing the route prevented CITEX from validly obtaining POIs and the corresponding information. In this same line, the dependency between POIs and POI information prevented us from stubbing the POI element individually without breaking the associated functionality of the 'INFO' element. Consequently, we were able to apply two stubs to CITEX involving the 'INFO' and 'POI' element: (1) stub the POI information (NoInfo) and (2) stub both POIs and POI information (NoPOI_NoInfo).

As the 'RouteController', 'GetRoutes', 'Entities' and 'Context' elements do not contribute to our energy requirement, these were not stubbed. To get

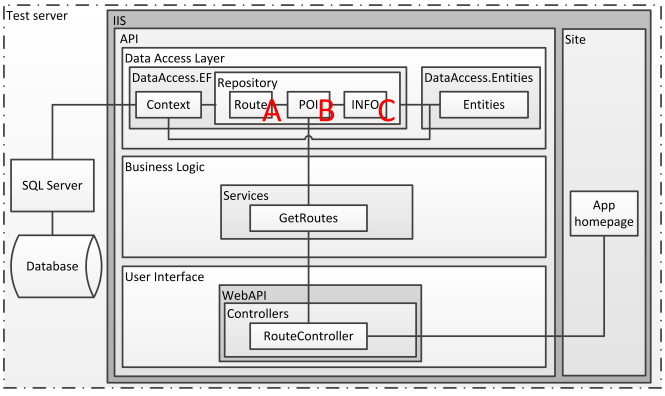


Figure 5.4: The architectural description of CITEX used to guide the stubbing activities for the experiment.

an impression of the database impact of the stubs, the SQL Server database will be monitored separately.

Route Filter and Monolith Versions

In relation to the energy efficiency of the target functionality, an additional version (RouteFilter) was investigated where the routes are filtered based on the location of the visitor. In the Benchmark version, the client retrieves all routes that are created by the route managers including routes that are not relevant to the visitor at that time, i.e. routes from different cities. With the RouteFilter version we do not stub the application, but rather investigate a design decision for CITEX that potentially affects energy demand. As the design decision affects the entire application, the AD in Fig. 5.3 is sufficient.

Additionally, we studied the very first release of CITEX, (version 5), a monolithic system with exactly the same functionality. The Monolith version was refactored by the CITEX team to improve the maintainability of the application. The result of refactoring was the Benchmark version of our experiment. Among others, the software was restructured to have a clear separation of the three layers (Fig. 5.3) and the database was normalized including a modification of the corresponding business entities. With the Monolith version we investigate the effects of refactoring software on its SEC, which does not require a separate architectural description.

A summary of all CITEX versions included in the experiment is provided in Tbl. 5.1. To prevent having to look up the differences between versions, each version has a label that summarizes the adjustment made to that specific version. In the remainder of this paper we will refer to each version through their respective labels.

5.3.3 Creating a Test Case

For the experiment, we stressed CITEX with its core functionality, i.e. providing routes to visitors. To ensure the software is stressed to the intensity that differences can be observed, we chose to simulate 10000 visitors that each follow three different routes for three times. The number of simultaneously active users was steadily increased at a rate of three additional users per second, which resulted in a load test that would last for approximately one hour. This one hour test comprises a single execution and was scripted to ensure consistency.

To obtain valid measurements, we set out to obtain at least 30 valid executions per CITEX version. During an execution, we logged all instructions

CITEX that had to process which allowed us to monitor for errors.

5.3.4 Determining the Measurement Approach

For the experiment, a setup was created (Fig. 5.5) comparable with a commercial setting. The software was deployed on a test server [1] that included the 'API', 'App Homepage', 'SQL server' and 'Database' elements as portrayed in Fig. 5.3. To exclude logging from the energy and performance measurements, a second server, the logging server, was included in the setup. The logging server was configured to remotely collect measurement data, e.g. performance data, and perform the load test (see Sect. 5.3.3 on the test server. As such, the logging server also functioned as the Client device for CITEX. The role of the power meter is explained in Sect. 5.3.4.

Our focus is on the back-end of the app, i.e. all elements on the 'Test server' (Fig. 5.3), as this is under the control of the SPO. Hence, the 'POI Information Hosts' and 'Routeplanner' elements are considered out of scope. Additionally, considering our load test, the 'Portal' is also considered out of scope. To ensure consistency with regard to external factors, e.g. room temperature, the server was installed in an operational data center. All CITEX versions used the same data set for the experiment, which was a copy of the production database.

Power Consumption and Performance Measurements

Following the methods described in [60] and [61], we applied a software- and hardware-based approach to measure the energy consumed by the software.

[1] HP Proliant DL380 G5, Intel Xeon E5335 CPU (4 cores @ 2GHz), 8GB PC2-5300, 300GB hard disk (15.000 rpm), 64-bit MS Windows Server 2008R2, Service Pack 1, .Net Framework 4.5, IIS7, SQL Server 2016

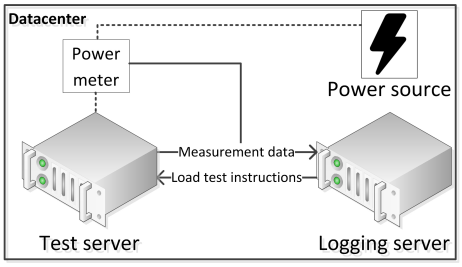


Figure 5.5: Experiment environment.

For the software-based approach Microsoft Joulemeter (JM) was used: a software tool that estimates the total power (in Watt) consumed by a system and individual processes based on the required computational resources [68]. After calibration using the WUP, JM allows us to monitor energy consumption changes on both system and process level. Process level measurements were performed on the SQL Server, Homepage and API processes respectively. The first processes represent the 'SQL Server' and 'Database', whereas the latter two are related to Internet Information Services (IIS) and provide insight in the 'App Homepage' and 'API' (Fig. 5.3).

The hardware-based approach was applied using a WUP device [2], i.e. the power meter in Fig. 5.5, to more accurately determine the power consumption on system level [63]. By determining the average %-difference between JM and WUP, we are able to correct the JM estimations [63], which positively contributes to the quality of our data set. As the WUP does not provide 'exact' measurements on process level, the estimations on process level are reported as provided by JM. Both JM and the WUP operate with a one second interval between measurements.

The performance of the test server was measured using Windows Performance monitor, a standard tool with the Windows operating system. Following [62], the relevant performance metrics were determined with the product owner:

- CPU: % processor time
- Memory: private working set
- hard disk: % disk time, % disk idle time
- Network: bytes sent/sec, bytes received/sec

The performance measurements were collected remotely on the logging server to minimize the overhead of the data collection process. Similar to the power measurement tools, the performance measurements are logged with measurement interval of one second. To ensure accuracy across measurement tools, i.e. be able to link events across sources on specific times, we synchronized the clocks across systems and devices using the Network Time Protocol (NTP).

Software Energy Consumption

To determine the energy consumed by the product under study, i.e the SEC [60], we subtract the idle power consumption from the power consumption as

[2] `https://www.wattsupmeters.com/`, last visited 20^{th} April 2017.

measured during an execution. The difference is considered to be the power consumed by CITEX, provided that no other applications were active during an execution. The idle power consumption, i.e. the baseline, is determined by performing power consumption measurements while the test server is idle, i.e., running without any active software. To obtain the SEC, power measurements need to be converted into energy consumption. Hence, we multiply the average power between two consecutive measurements with the time between measurements, i.e. one second, and sum up the results for the duration of each separate execution. We report our findings in Watt (W) or Joule (J) where applicable.

The SEC also forms the basis to determine the energy consumed by the individual architectural elements using the stubs. We are able to determine the SEC for each element as follows:

- 'Route': $SEC_{Route} = SEC_{NoPOI\_NoInfo}$

  By stubbing the 'POI' element, and thereby disabling the 'Info' element, the total SEC for the NoPOI_NoInfo version effectively measures the SEC of the element that is not stubbed, being 'Route'.

- 'INFO': $SEC_{INFO} = SEC_{Benchmark} - SEC_{NoInfo}$

  Deducting the SEC of the NoInfo version from the Benchmark results in the SEC for the 'INFO' element.

- 'POI': $SEC_{POI} = (SEC_{Benchmark} - SEC_{Route}) - SEC_{INFO}$

  Deducting the energy consumption associated with the Route and INFO elements, i.e. SEC_{Route}) and SEC_{INFO}, provides the SEC of the POI element.

In our approach, we assume that any increase in power consumption is caused by the activities performed by the software. To minimize overhead, we stopped services and processes not related to or required for CITEX, such as the Windows update service. Additionally we determined the cooldown time [61] for our test server; the time after which uncontrolled services and processes become inactive after a reboot. In our case we could start an execution eight minutes after rebooting.

5.3.5 Measurement Protocol

To actually perform the measurements, we combined seven executions (see Sect. 5.3.3) into one series with eight minutes in between two executions. More

executions in a series were not possible due to the limited memory available in the WUP device. To ensure the validity of the EC measurements, a protocol was followed to perform each series:

1. Restart the test server.
2. Close unnecessary applications, services and processes.
3. Remain idle for the duration of the cooldown time.
4. Start WUP, performance and JM measurements.
5. Start load test and wait for test to finish.
6. Collect data.

Each series resulted in a data set containing a performance log, three JM logs for the SQL Server, Homepage and API processes, including measurements of the entire system, a WUP log, and seven log files for each execution in the series. To increase the reliability of the measurements we aimed to have at least 30 valid executions, i.e. without errors, per CITEX version.

5.4 Results

In this section we report on the execution and the results of our experiment for all five CITEX versions.

5.4.1 Experiment Execution

By following the protocol as described in the previous section, we managed to obtain the required amount of valid executions for each CITEX version. On average, an execution lasted for 57 minutes and 40 seconds (SD = 11 seconds). To ensure validity of the executions according to our test script, we deliberately polluted one execution with the Benchmark version by performing a mid-execution check for unexpected issues. In this specific execution, which was excluded from further processing, no issues occurred, leading us to conclude that the executions were performed as designed.

Additionally, despite consistency in performing the series, eleven executions across different versions were found to contain errors and thus excluded from further processing. Further investigation into these executions was performed, to assure that the impact of the errors was limited to the single executions. We found the number of errors to be small, i.e. one to 20 errors on a total of

Table 5.2: The measurement averages across executions for each CITEX version.

Version	SEC (J)	SQL Server (J)	API (J)	CPU (%)	Memory (MB)	HDD (%)	Network received/sent (MB)
Benchmark	19221.72	2428.28	4596.39	28.43	1090.55	3.85	0.05 / 0.55
NoInfo	8499.36	653.25	1979.10	11.71	1114.08	3.34	0.04 / 0.27
NoPOI_NoInfo	5836.20	297.12	1348.18	7.85	1096.14	2.85	0.04 / 0.24
RouteFilter	14757.01	1077.93	3598.39	19.78	1101.31	3.72	0.04 / 0.43
Monolith	17684.83	1157.50	5693.91	26.30	1168.18	4.38	0.05 / 0.38

90000 activities per execution, which were caused by incidental time-outs of modules in the application or third party components. Hence, we can assume our results (Tbl. 5.2) to provide valid insights into the differences between CITEX versions.

5.4.2 Process Level measurements

On process level, we noticed a lack of energy consumption by the Homepage process. As our load test focused on the API process, our expectation was to find activity of this process. Investigating the issue more closely, learned that this specific process only shows activity when the process itself is starting up. Additionally, it is very likely that the webserver creates separate processes for calls, which could not be measured by JM. Hence, the Homepage process was excluded from further processing. The remaining energy consumption figures show clear differences between CITEX versions, allowing us to create

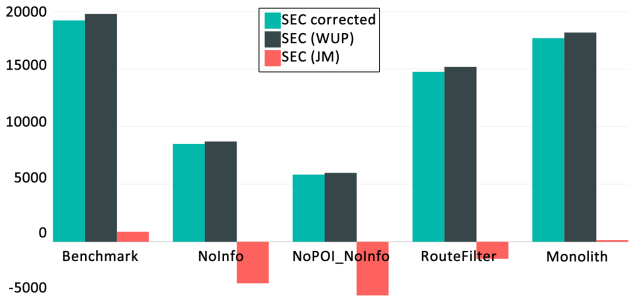


Figure 5.6: A comparison of SEC (in Joule) across versions, according to different measurement sources.

the desired energy profile for our analysis.

5.4.3 Energy Consumption Corrections

In line with previous experiences, i.e. [63], we found that JM overestimates the idle energy consumption compared to the WUP and underestimates the energy consumed while processing a varying load, i.e. during an execution. The baseline for the test server was determined to be 136.3 W using the WUP and 136.95 W using JM. After correction with the average difference, i.e. -0.48%, a corrected JM baseline of 136.3 W was found and used in the further SEC calculations. Similarly, a correction of 2.67% was applied to the total energy consumption figures during an execution.

With these corrections, the SEC for an execution was obtained as follows:

$$SEC = EC_{Total} - EC_{corrected\_baseline} + (d \cdot EC_{Total})$$

First, the correction is applied to the compensate for underestimating the energy consumed while processing a varying load. This correction is found by multiplying d, i.e. the load correction, with the total energy consumption for an execution. Second, the baseline energy consumption is subtracted from the energy consumed during an execution, which results in the energy consumption on behalf of the software being active. Adding the correction provides the corrected SEC for an execution.

The necessity to correct energy consumption measurements appears from the results in Fig. 5.6. The figure shows the differences between the SEC calculated according to the WUP (SEC (WUP)), the corrected JM estimations (SEC corrected) and the uncorrected JM estimations (SEC(JM)). It appears that the uncorrected Joulemeter estimations cannot cope with subtracting the baseline energy consumption, even resulting in a negative SEC for the NoInfo, NoPOI_NoInfo and RouteFilter versions. The corrected estimations on the other hand approach the WUP measurements, justifying our efforts to triangulate the JM data.

5.4.4 Performance

The results of the performance measurements are, in general, in line with our expectations: decreasing CPU utilization, hard disk utilization and network throughput with decreased software activity. Notice that we report one metric in relation to hard disk utilization, whereas two metrics were monitored. While the '% disk time' metric seems a logical metric to indicate hard disk

activity, this metric was found to exaggerate the disk utilization [3]. Instead, we subtracted the '% idle time' from the maximum 100% load, and proceeded to calculate the average hard disk utilization.

5.4.5 Energy Profiles

To create an energy profile, we annotate the adapted architectural description, e.g. Fig. 5.4, with the SEC for the elements under study. In our experiment, the different CITEX versions require different variants of an energy profile. Specifically, the energy profiles for the RouteFilter and Monolith versions, where we compared the entire application, would encompass annotating the architectural descriptions with the total SEC found with each version. As this does not provide additional insights over the measurements presented in Tbl. 5.2, we decided not to annotate the architectural descriptions for these versions. Instead we are able to calculate that adding the filter for routes saves 4464.71 J compared to the Benchmark, whereas the Monolith version consumes less 1536.89 J compared to the Benchmark.

For the architectural elements 'Route', 'POI' and 'INFO' (cf. Fig. 5.4) we did annotate the architectural description, of which an excerpt is depicted in Fig. 5.7. The SEC for the 'Route' element equals the SEC found with the NoPOI_noInfo version of CITEX, i.e. 5836.20 J. For the SEC of the 'INFO' element we subtract the SEC of the NoInfo version from the Benchmark, resulting in an energy consumption of 10722.35 J. Finally, we deduce the SEC for the 'POI' element, which is 2663.17 J.

[3] https://technet.microsoft.com/en-us/library/cc938959.aspx

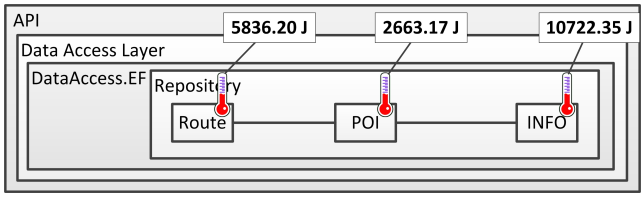


Figure 5.7: The energy for CITEX obtained by testing stubbed versions of the software.

Table 5.3: The SEC percentages for each element measured on different levels.

Element	Total SEC	SQL Server	API
INFO	55.75%	73.10%	56.94%
POI	13.85%	14.67%	13.73%
Route	30.36%	12.24%	29.33%

5.5 Discussion

In this section, we discuss the main results and findings of our experiment in which we applied the proposed SEPM.

5.5.1 Energy Consumption of CITEX

Recall that our load test was designed to simulate 10000 users that obtain three routes, i.e. the core functionality, for three times. The energy profile shows that obtaining the POI information is the most energy consuming activity related to this functionality. On average, this specific activity consumes 10722.35 J, compared to 5836.20 J and 2663.17 J for respectively obtaining the route and POIs. The measurements with respect to the SQL Server and API processes show a similar pattern, i.e. the highest energy consumption with obtaining the POI information, with a slight variation for the database (Tbl. 5.3). While the SEC divided according to the API measurements are consistent with the total SEC, the SQL Server shows a relatively higher energy consumption for obtaining POI information.

The results suggest that if an SPO wants to reduce the SEC of CITEX, the most prominent code to address is related to obtaining POI information. Put otherwise: the source code comprising the 'INFO' element is accountable for 55.75% of the SEC, and even 73.10% for the energy consumed by the database, while the software performs its core activity. Performance measurements, i.e. Tbl. 5.2, support this finding through the significant decrease of CPU utilization in the NoInfo version.

A different approach is to consider the energy consumption related to the actual activities that are performed, i.e. the Task Energy Consumption (TEC) [60]. The TEC provides an energy consumption for each individual time a task has been performed. For example, in total CITEX obtained 540000 POI information items, 570000 POIs and 90000 routes per execution. Dividing the SEC of each element, i.e. Fig. 5.7, by these numbers, learns that CITEX consumes 0.019 J per POI information item, 0.005 J per POI and 0.065 J per route. Hence, the TEC suggests the 'Route' element could be the first object

for optimization. In the calculation of the TEC, note that the discrepancy between POI information items and POIs is caused by a so-called waypoint POI: a POI that does not contain information and is only used to ensure a user follows the prescribed route.

Route Filter Design Decisions

With respect to the RouteFilter version, the client sent specific requests to only obtain the information related to the three routes included in the load test. Hence, in our implementation we assumed the client to know all routes upfront and apply a filter based on the GPS location. In this design, we found the back-end to consume 4464.71 J less energy than the benchmark and an overall reduced hardware utilization. Implementing this filter into the application would thus reduce the SEC and thereby TCO of CITEX from the perspective of the SPO. An additional effect of the reduced hardware utilization, is the potential to further reduce the TCO by down-scaling the hardware requirements for the back-end. A reduced TCO allows the SPO to maintain a higher profit margin or apply a more competitive pricing model.

On the other hand, as the filter is applied by the client, i.e. the app used by customers, there is bound to be an effect on the client device. For example, actively maintaining a list of routes on the client requires some sort of synchronization which could result in a deterioration of the battery life. Another variation is that the app collects the available routes upon launching and actively applies a filter in all request after this initial call. Based on the TEC calculation, the latter implies that a relatively energy inefficient activity, i.e. obtaining routes, is performed more often compared to the current implementation.

In the end, finding the right implementation of the filter requires an SPO to evaluate multiple design decisions with respect to the client-side software. These design decisions will likely involve multiple quality aspects for the software, e.g. usability, and surpass the focus of the SEPM. On the other hand, extending the energy profile with the software elements on the client device could steer the design decision. A GPS call, for example, is often associated with a high energy consumption. As this functionality is provided by the operating system, e.g. Android, a software developer has limited control over this call. The energy profile could provide insight into the battery-drain by this call, i.e. by stubbing the function to return a standard response, and lead to a design decision where the number of GPS calls is actively minimized.

Refactoring effects

For the Monolith version, we can identify several interesting findings caused by the refactoring. First, we found a lower total SEC for the Monolith version compared to the Benchmark. The data shows that the impact of normalizing the database is the primary cause of the increased energy consumption after refactoring. On the other hand, the API process consumes less energy in the Benchmark version.

With respect to the performance measurements, we find the refactored version to more intensely utilize the CPU and encompass increased network activity. Memory and hard disk utilization, on the other hand, decrease. Alongside the SEC, this finding suggests that CPU utilization and network activity are the main drivers of energy consumption for CITEX. Additionally, the architect could not explain the increased network activity measured with the Benchmark version: since no client-side changes were applied, the same amount of information should have been sent to the client. Hence, our results surfaced an unexpected change.

Looking at the SEC changes on process level, the increased CPU utilization is likely to be on the account of the database. The decrease in disk utilization suggests that a pattern was applied that favors CPU utilization over hard disk utilization.

The results quantify the impact of refactoring on the sustainability of the software, and we specifically discussed the environmental dimension. However, the trade-off made with maintainability also affects the social, economic and technical dimensions [83]:

- the costs of applying changes, i.e. maintaining the software, decreases (economical)
- the software is better prepared to evolve with changing needs (technical)
- the increased performance demand could require more hardware resources to execute the software (economical)
- improved competencies of employees in relation to refactoring (social)

The decision to apply, or accept, a change depends on the balance between sustainability dimensions. A lack of balance, e.g. favoring the economic over the technical dimension, could results in an unsustainable software product on the long term.

5.5.2 Stubbing Software for SEC Measurements

Applying a stub and thereby providing a standard response for a specific software element, is already commonly used to enable SPOs to test specific functionality early on in the development cycle [100]. The stub allows the software to process requests as intended, without requiring the (incomplete) code to be fully executed. Hence, while stubbing software is not new, its application in relation to energy consumption had not yet been made explicit.

In relating software stubs to energy consumption we were inspired by genome research, where disabling a gene to investigate its effects is a commonly applied method. With the SEPM, we translate this approach to the SA domain and focus on the effects caused by the disabled software elements. Without requiring deep knowledge on the code and internal structure comprising CITEX, the SEPM allowed us to quantify qualitative aspects of the product and provide input for future design decisions and trade-offs.

Based on our experiment, we identify two aspects that should be taken into consideration by an SPO in applying the SEPM: the stakeholders in relation to a requirement and the resources (e.g. time) available to apply the SEPM. These aspects determine the context in which the SEPM is to be applied and thereby the preparations required to successfully apply the method.

For example, a software architect is likely to investigate the target functionality on the level of major architectural elements. As such, an energy profile would be used to compare between releases, on an environment that resembles the production environment. Additionally, the test case can be set on a large scale to simulate longer periods of operational time of the software. In this context, the same measurement approach can be applied as described in this paper. The resulting energy profile, similar to the RouteFilter and Monolith versions, is not likely to include measurements on the level of individual software elements.

On the other hand, in line with the NoPOI and NoPOI_NoInfo versions, a developer is likely to investigate the target functionality on code level and wants to know the exact functions that are performed in relation to the target functionality. With developers, the energy profile is created during development using the local system. As using the system during a load test will pollute measurements, the test case should be quick to perform while still producing observable discrepancies between the stubbed versions and the benchmark. Concerning the measurement approach, several tools exist [2, 31, 107] that are able to replicate our profiling approach on a smaller scale. However, keep the accuracy of these tools in mind during selection.

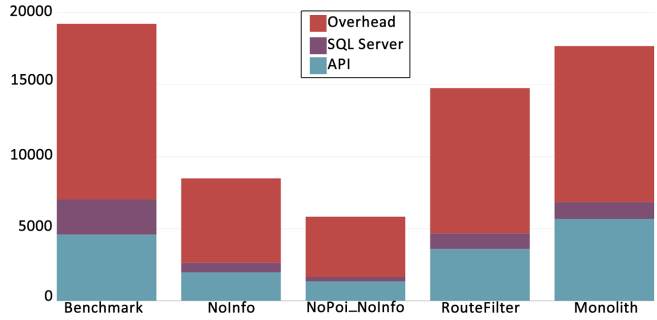


Figure 5.8: The energy consumption overhead (in Joule) not explained by the SEC of the individual processes comprising CITEX.

5.5.3 Complex Environments

The SEPM was applied in a relatively simple test environment, representative for a production deployment of CITEX. In our view, applying the method in a more complex, e.g. virtualized, environment does not impact the method, provided that reliable SEC measurements can be performed. As additional elements could significantly affect energy consumption, more effort will be required to control these elements to obtain valid measurements. For example, a stakeholder should know, or even influence, how the load is distributed over the available resources. Hence, with increased complexity we expect an increase with respect to the costs of applying the SEPM.

Our advice is to apply the method in relatively simple test environments and maintain focus on optimizing the software. Once optimized and deployed, e.g. on a cloud based platform, the focus of optimization shifts to the synergy between software and hardware. In this stage, power saving features in relation to networking activity [27] and the resource allocation algorithm [12] can be implemented to even further improve the sustainability of a software product.

5.6 Limitations and Threats to Validity

This section presents the limitations of our study and the threats to internal, external and construct validity, as required by [66, 127, 148].

5.6.1 Limitations

The energy profiling method builds on introducing stubs in software to determine the energy consumed by the stubbed elements. Although the experiment provided valuable insights for CITEX, there are limitations to our profiling method.

Stub effects.

Despite the quality of the stubs that are applied, modifying software could lead to unexpected changes in software behavior that potentially affect the measurements that are performed. Additionally, the change could occur outside of the scope of the test case and thus go by unnoticed. Detailed performance measurements help to identify such occurrences, allowing the stakeholder to adjust the stub accordingly.

Energy Overhead

A large part of the SEC, on average 66.77%, could not be attributed to the CITEX processes monitored with JM. However, as we were not able to monitor the effects on OS level, we can only summarize this portion as overhead. Similar to CPU intensity and energy consumption, the overhead changes with each version and is not considered to have a linear relation with software activity. In general though, increased activity from an application also implies increased activity from the OS to coordinate tasks, as shown in Fig. 5.8. Further research is required to fully understand and control this effect.

Visualizing the Energy Profile

To visualize the results of the energy profiling method, we annotated the ADs with the energy consumption figures for the respective elements. However, different visualization methods could be applied that better fit the needs of stakeholders and simplify the interpretation of the measurements. For example, when the energy profile is created for multiple small elements, an annotated AD could overwhelm stakeholders with too much information at once. In such a case, a heatmap (Fig. 5.9) could provide a means to minimize the information overload, while still communicating key findings to stakeholders.

5.6.2 Validity Threats

Internal Validity

The internal validity is concerned with uncontrolled factors that might affect the results of the experiment.

Observer effect. While our experiment was designed to minimize the impact of performing the experiment on our results, we could not completely rule out the effects of JM and Windows Performance Monitor. We know JM uses a small amount of memory, adding to the performance measurements, which does not translate to observable energy consumption [61]. For the latter, we were not able to determine a potential effect on the energy consumption. However, the performance measurements are consistent across executions and as such represent a consistent part of the monitored network activity.

OS effects. Despite our efforts we cannot be fully certain that no OS processes became active during an execution of the load test. Configuration tools do not provide full control over, e.g., services and internal timers that might induce activity without direct input from a user. We analyzed our performance measurements in detail, i.e. for each process separately, and did not find unusual activity during the executions.

External Validity

The external validity addresses the extent to which the results can be generalized beyond the experiment.

Experiment environment. Given the relation between power consumption and the hardware components, performing the experiment in a different environment could yield different measurement results. The generalization of the study could be improved by diversifying the hardware involved in the experiment. However, while SEC differences could be found in absolute terms,

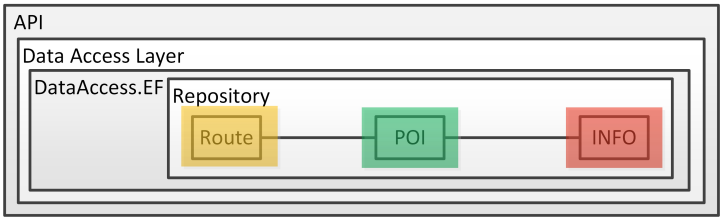


Figure 5.9: A heatmap overlay to visualize the energy profile for CITEX.

the relative SEC differences should show consistency across measurements.

Test case load. For our test case, we decided to stress CITEX by simulating a large number of concurrent users. However, a different load could change hardware behavior and thereby energy consumption. While the total SEC is expected to vary with the intensity of the test case, specifically the relative energy consumption between software components could differ between loads. A discrepancy that can also caused by using different hardware, e.g. hardware that is better capable of managing dynamic frequency scaling. To obtain the best possible energy profile, we designed our test case to represent a typical use case scenario for CITEX and ensured a consistent load across versions.

Measurement equipment. Given the nature of electrical power, the one-second sampling interval for both hardware- and software-based approaches might have resulted in an underestimation of the SEC. However, this interval is also applied in the state of the art [51]. Also, as shown in Fig. 5.6, large differences exist with respect to the quality of the measurements between the tools we utilized. Given the diversity of power meters and software tools, there is an unavoidable dependency with respect to the accuracy and detail of the obtained measurements.

Construct Validity

Construct validity addresses the degree to which the measures capture the concepts of interest in the experiment.

Sustainability and software quality. Central to performing measurements is to have a clear vision on the metrics that should be reported. In our research we quantify the environmental dimension of sustainability through the SEC, however, we acknowledge different metrics could have been applied. The measurements explained in the research design reflect the data required to calculate the SEC and were identified in collaboration with the CITEX architect.

Non-stubbed elements. In our experiment, we were not able to stub every element identified to the activities comprising the target functionality. Hence, our results assume the impact of the other elements to scale with the stubs applied in the NoInfo and NoPOI_NoInfo versions, and we deduced the SEC for the 'POI' element. A decision that was made in collaboration with the developer that applied the stubs. Despite this constraint, we were able to quantify SEC differences, supported by performance measurements, and demonstrate our Energy Profiling Method (EPM).

5.7 Related Work

Measurements on the the energy consumption of software products are difficult to perform accurately, and have proven to be time-consuming in nature [154]. Each change in the software, e.g. code obfuscation [129], requires its own test to be designed, executed multiple times and the results analyzed afterwards. Although specialized equipment can be utilized, e.g. [39], such an approach is both difficult and expensive to apply in an industrial context.

The complexity of performing energy consumption measurements increases in cloud computing environments due to, e.g., virtualization techniques and load balancers. The additional layers between the software and hardware, make it difficult to allocate energy consumption of a distributed system to specific software activity [134]. Although these layers can also assist with identifying the main sources of energy consumption, e.g. [13], performance-based power models should be used with caution [90].

In comparison, developers of mobile software are able to estimate the SEC of their apps on their workstation through emulation tools [96]. However, as the energy impact of a software product scales with each installation, the efforts towards this aspect are worthwhile. For example, if four million users changed to a different, functionally equivalent web browser, the monthly energy consumption of an American household could be saved each hour [51].

Our approach of using stubs to analyze software is not new. For example, in gray-box testing [100], stubs are used to study the effects of certain components in isolation. The software architecture can be used to define stubs, as for example has been done in [43] to test a flight software product line. However, in our approach, we invert the idea of gray boxing: comparing the overall system with the system where some functional element has been substituted by a stub, provides information about the isolated functional element, rather than on the effect on the complete system.

In this way, our approach may support feature slicing for hypothesis based software development [110]. This approach may assist architects in studying different design alternatives, also called design diversity [86, 137]. With our proposed method, different alternatives and tactics can be compared to obtain a catalogue of green tactics [122].

Additionally, considering energy consumption as a first class citizen allows for trade-off analyses, e.g. with respect to reliability [93]. Traditional software architecture evaluation methods, such as ATAM [69] mainly rely on expert opinions. Relating test-based measurements with the software architecture, allows to create better, objective insights for software architects, supporting

the longetivity of the architecture [78]. Compared to more recent methods, e.g. KLAPER [44], test-based measurements do not require the construction and maintenance of expensive performance models to evaluate alternative system designs [77].

5.8 Conclusions

In this paper we propose the Stubbed Energy Profiling Method to determine the energy consumption of software, i.e. SEC, on the level of architectural elements, and applied this method in an experiment performed using a commercial software product, i.e. CITEX. The method is intended as a detailed elaboration of how to create an energy profile, as required for applying the Energy Consumption Perspective [60]. The method is based on creating multiple versions of a software product, each including a stubbed software element, which are then compared to a non-stubbed benchmark. Quantifying the differences between versions provides the SEC of the stubbed elements, which creates the energy profile for the software.

In total, five different versions of CITEX, including the benchmark, were included in the experiment, which allowed us to thoroughly investigate the energy implications related to the core functionality of the product. First, two stubbed versions were created that enabled the SPO to create an energy profile of CITEX and identify the major energy consuming architectural elements while performing its core functionality. The level of detail, also helped to identify the source code developers should target to address energy related concerns. From the analysis, we were able to identify multiple energy hotspots in the software. We identified the element that consumed the most energy and the element that consumed relatively much energy in relation to the specific activity it performed.

Second, we investigated two versions where the SEPM was mostly focused on system level. We evaluated one implementation of added functionality, i.e. applying a content filter, and showed the value of the method in evaluating the impact this design decision. Furthermore, we quantified the effects of refactoring CITEX, which was performed by the SPO. We showed how the impact of refactoring on the environmental dimension and the implications of the results on trade-offs involving sustainability in general.

An SPO that applies the SEPM extends its control over the environmental sustainability aspect of their software products. Specifically, the SPO gains the ability to target the most energy consuming elements of its software products. Additionally, as the method can be applied during development, energy related

concerns can be addresses when the costs of doing so are lowest.

For future work we look into refining the SEPM and the SEC measurements. For example, the approach relies on creating stubs. However, it is not always possible to create stubs for a functional element. By recording sufficient software operation data for each functional element, we envision the automatic creation of stubs that are more realistic, and thus provide better approximations in the analysis. Furthermore, in our experiment the overhead on average accounted for 66.77% of the SEC. We look to performing an in-depth analysis of the overhead as this is a significant contributor to the SEC of a software product.

Part III

Energy Awareness in Software Engineering

6

Energy Efficiency on the Product Roadmap: an empirical study across releases of a software product

In the quest for energy efficient ICT, research has mostly focused on the role of hardware. However, the impact of software on energy consumption has been acknowledged as significant by researchers in software engineering. In spite of that, due to cost and time constraints, many software producing organizations are unable to effectively measure software energy consumption preventing them to include energy efficiency in the product roadmap.

In this paper, we apply a software energy profiling method to reliably compare the energy consumed by a commercial software product across two consecutive releases. We demonstrate how the method can be applied and provide an in-depth analysis of energy consumption of software components. Additionally, we investigate the added value of these measurement for multiple stakeholders in a software producing organization, by means of semi-structured interviews.

Our results show how the introduction of an encryption module caused a noticeable increase in the energy consumption of the product. Such results were deemed valuable by the stakeholders and provided insights on how specific software changes might affect energy consumption. In addition, our interviews show that such a quantification of software energy consumption helps to create awareness and eventually consider energy efficiency aspects when planning software releases.

This work was originally published as:

E. Jagroep, G. Procaccianti, J. M. van der Werf, S. Brinkkemper, L. Blom, and R. van Vliet. Energy efficiency on the product roadmap: An empirical study across releases of a software product. *Journal of Software: Evolution and Process*, 29(2), 2017

6.1 Introduction

In order to make the ICT sector more environmentally sustainable, research has mostly focused on hardware improvements. Indeed, every new generation of hardware improves its EE by either increased performance (i.e. more performance per Watt) or decreased EC in absolute terms. Considering the growing number of hardware devices, the impact of these improvements can be significant. However, a crucial aspect that has been long overlooked is the role of software [82]. Although hardware ultimately consumes energy, software provides the instructions that guide the hardware behavior [140].

The sustainability of software is still in its infancy as a research topic. Previous work [60, 83] defines sustainability as a multi–dimensional concept that identifies requirements for multiple software QAs. In particular, *environmental* sustainability identifies EC requirements for energy efficiency. Sustainability requirements also impact other QAs: for example, in the mobile domain, the EC of mobile applications directly impacts usability, as it shortens battery life [26, 96, 113].

Despite this, we do not witness a significant increase in the energy efficiency of mobile applications over time. On the contrary, software updates require the user to buy a new mobile phone every few years, sometimes even without a clear benefit in terms of performance. Additionally, new phones are often equipped with higher capacity batteries, to prevent deterioration of the operation time. Looking at larger software products, e.g. business applications, a similar pattern can be observed. Depending on the deployment, increasingly more powerful hardware is required to run new releases of applications. In contrast to the mobile domain, though, EC measurements on business software products are more complicated to perform. The diversity of deployments and levels of abstraction (e.g. virtualization and cloud computing) require more sophisticated measurement approaches to properly analyze software EC [124]. Recently, several of such approaches have been proposed, both hardware [45] and software based [109], which were able to identify opportunities for considerable savings in EC.

However, these approaches have not been adopted in industrial contexts so far. While SPOs, e.g. independent software vendors and open-source foundations, have software development as their core activity [65], having accurate software EC measurements still requires significant investments in terms of resources and specialized knowledge [118]. As a consequence SPOs do not plan the evolution of their product, i.e. with a product roadmap [42], on its energy efficiency aspect, potentially leading to not meeting market requirements [11].

For example, in the Netherlands the government specifies EC related requirements in their tenders.

In practice, performance is often used as a proxy for energy efficiency. Software performance optimization is a more mature field of study, hence more people with such skills are available on the market. However, although much can also be derived from performance measurements, EC and performance are not always positively correlated [20,35,117,125,144]; contradicting goals could require a trade-off to be made [60]. Hence, a deeper understanding of the matter is required to properly address the EC of the software itself.

For this purpose, we formulate the following **main research questions**:

RQ1: *How can we reliably compare the energy consumption of large-scale software products across different releases?*

In RQ1, we explicitly refer to large-scale software products as multi-tenant, multi-user distributed software applications, as opposed to e.g. single-user mobile applications which are out of scope for this study. RQ1 is further divided in 2 sub-research question:

- **SQ1**: *How can we reliably measure the EC of a software product?* A prerequisite for comparing the EC of a software product is being able to measure the software EC.

- **SQ2:** *How can we attribute energy consumption to individual software elements?* For SPOs to actually optimize the EC of their products, it is necessary to identify how individual software elements affect EC. For a more precise definition of what we mean as a software element, see Section 6.3.

In Sect. 6.3 we describe the design of an experiment where we used software profilers to obtain fine-grained, software-level estimations and validated them with hardware measurements obtained via power meters. The results of this experiment allow us to answer RQ1.

Additionally we investigate the benefits of measuring the EC of a software product for stakeholders in SPOs responsible for a product. Comparing EC across releases of a software product will, most likely, only be done when there is added value from this effort. To put EC on the product roadmap [33] we formulated a second main research question:

RQ2: *What is the added value for a software producing organization to perform EC measurements on software products?*

In Sect. 6.3, we describe the design of a secondary empirical study encompassing interviews with the stakeholders from an SPO. The results of this study allow us to answer RQ2, from the perspectives of the different roles involved in software product development.

This paper extends our previously published work [61] in several ways. First of all, we performed a more in-depth analysis of the data, i.e. including software metrics in the analysis, propose a technique to visualize the results in the form of radar graphs and discuss the impact of energy consumption in software design. Moreover, this study poses an additional Research Question (RQ2), answered by means of a series of interviews with practitioners from the SPO which provided the product for our experimentation. During the interviews, we discussed our experimental results and their implications for their product related activities.

The remainder of this article is organized as follows: in Section 6.2 we review the related work. In Sect. 6.3, Sect. 6.4 and Sect. 6.5 we describe the design, execution and results of our empirical studies (experiment and interviews). We discuss the results in Section 6.6 and threats to validity in Sect. 6.7. Concluding remarks and an outline for future work are provided in Section 6.8.

6.2 Related Work

6.2.1 Product Roadmap

To identify the added value of EC measurements for product development, a basic understanding of the product dynamics is required. Changes in the product market have significantly shifted the focus of software development towards the goal of achieving competitive advantage [41]. Since EC could be considered as a non-functional, strategic aspect of software [60, 83], this topic fits the software product management competence model [11] in the area of product planning, or more specifically product roadmapping. The product, or software, roadmap translates strategy into short- and long-term plans and could be considered as planning the evolution of a product [42].

An important aspect for creating a roadmap is being aware of the lifecycle phase a product is in; beginning of life, middle of life or end of life [85]. Depending on the phase different drivers, economical and technical, direct investments for the product, taking into account the current position of the product in the market. SPOs are, for example, not eager to invest in technology that has become obsolete in a specific market segment. Depending on

the lifecycle phase the SPO could consider different investment strategies to minimize losses.

Parallel to the three phases, a different lifecycle representation is presented by Ebert and Brinkkemper [33] ranging from strategic management, product strategy, product planning, development, marketing, sales and distribution to service and support. The beginning of life phase is characterized by creating a product strategy and planning, which leads to the initial development of the product. Development continues in the middle of life phase where the marketing, sales and distribution, service and support activities are key to deliver a 'mature' product to the market and keep the product financially viable. During the end of life phase marketing, sales and distribution, service and support activities are key to minimize costs and stretch the financial viability of the product. If required, a substitute product is sought when a current product is considered end of life. Typically major investments are done in the first two stages of the lifecycle.

From an EC point of view the first two stages are where the product team forms and executes short- and long-term plans for a product and where measuring the EC could prove helpful to increase the product success. Sales, an internal stakeholder for a software product [11], could benefit from having low EC as a unique selling point for the software product. When nearing the end of life phase a product, its EC characteristics potentially contribute to extending the lifecycle by ,e.g., lowering the total cost of ownership.

Apart from creating the roadmap the product manager, the one responsible for the future of a product [33], also has to ensure development activities are in line with the roadmap. Among others, developers should obtain requirements based on the roadmap and the team has to plan their releases (or sprints) to fulfill these requirements. Not meeting the requirements, or not meeting them in time, could potentially negatively affect the success of the software product.

6.2.2 Software Energy Consumption Measurements

The techniques for measuring software energy consumption are rapidly advancing, however a distinction must be made based upon the software execution environment. EC measurements on *mobile* devices are commonly performed to prevent the software from having a deteriorating effect on the battery life of the device., e.g. by software tools performing measurements on the device itself (Joulemeter [47], eprof [113]), or by emulation tools that allow developers to estimate the EC of their application on their development environments [96]. Since battery drain can be monitored relatively easily and mobile devices have similar hardware architectures, some approaches were able to relate EC to

source code lines [84] with reasonable accuracy (within 10% of the ground truth), although only for Android applications. Additionally, as performance profilers are quite mature in mobile computing, EC profilers can build upon such tools [87].

In the area of *large-scale* software products, the execution environment is more complex and approaches for energy profiling are more elaborate. In such environments, hardware-based approaches (e.g. [39]) rely on physical power meters to be connected to hardware devices. These approaches do not provide fine-grained measurements at software level, i.e. they are not able to trace the energy consumption of single software elements such as processes or architectural components.

Software-based approaches can be roughly categorized in two sets: source code instrumentation [109] and energy profilers [63]. Source code instrumentation consists in injecting profiling code into the applications code (or byte code), to capture all the necessary events related to energy consumption. For example, JalenUnit [108] is a bytecode instrumentation method that can be used to detect energy bugs and understand energy distribution. JalenUnit infers the energy consumption model of software libraries from execution traces. However, source code instrumentation always results in a noticeable overhead in performance.

Energy profilers rely on fine-grained power models [107] to deliver more accurate measurements at software level. Typically, profilers use performance measurements to explain and characterize software and its EC characteristics [16, 67]. The power model is typically generated via linear regression from performance measurements or resource usage data. This technique could be potentially applied on multiple software products using public repositories and benchmarks, an approach known as *green mining* [52]. Unfortunately, due to lack of publicly available performance data, green mining is still an immature area. Despite the differences, these approaches all focus on identifying energy hotspots [124] i.e. elements or properties, at any level of abstraction of the system architecture, that have a measurable and significant impact on energy consumption.

We see two potential issues with applying source code instrumentation on large scale, e.g. 30000 lines of code, software products: the performance overhead and the required investment (in time and money) to instrument the code [151]. Hence, we do not see them as viable in an industrial setting. On the other hand, energy profilers do not require a high effort to be adopted, but are shown to be inaccurate in their measurements [63]. Hence, for the purpose of our study (see Sect. 6.3), we use software profilers to obtain fine-grained, software-level estimations and validate them with hardware measurements ob-

tained via power meters.

6.2.3 Software Architectural Aspects of EC

The EC can be significantly influenced by the way software is designed and architected. For example, recent study shows data locality plays an important role in the EC of multi-threaded programs [119]. An information viewpoint [126] could be used to structurally consider this aspect. Characterizing software using performance measurements on the other hand is more related to the deployment and functional viewpoint. Combining multiple viewpoints of a software product, i.e. creating a perspective [126], enables stakeholder to structurally address concerns on different aspects of the system design.

SA also allows a stakeholder to explore design trade-offs for the software [60]. Increased performance, a quality attribute for the software, does not always have a direct relation with EC [144]. A different design trade-off is to exchange modules or services for more energy efficient sustainable variants, e.g. cloud federation [122]. SA helps to identify adjustments on different levels in complex environments [38].

6.2.4 EC Comparison Between Releases

Comparing aspects across releases is often discussed in terms of software evolution [135]. However, only few papers were found that investigate the EC of software and include a comparison between different releases. In [51] a comparison is made between three releases of rTorrent by 'mining'EC and performance data. A direct relation is described between the granularity of the measurements and the ability to determine the cause of changes in EC. Another approach is to characterize software using Petri nets [153]. Assuming that a complex software product can be fitted into a Petri net, analysis could show the path of lowest EC to perform a specific task. If the changes in a new release can be included in the Petri net, the difference(s) between releases can be quantified.

6.2.5 Awareness

A different approach is to increase developer awareness in software energy efficiency. The 'Eco' programming model [156], for example, introduces energy and temperature awareness in relation to the software and challenges developers to find energy friendly solutions. Awareness of the software community about the impact of software on EC is increasing [8]. However, Pinto et al. [118]

point out that this is still far too little to make a difference. In spite of recent progress, the state-of-the-Art in software energy efficiency did not reach sufficient quality yet to deliver reliable, detailed measurements. Comparing the EC between releases can be used to create awareness at the right place for a SPO, and hence exert control over the EC of their software.

6.3 Study Design

To answer the research questions presented in the Section 6.1, we performed two empirical studies: an experiment to compare the EC of a commercial software product (DG) across different releases and an interview with stakeholders from an SPO.

6.3.1 Experiment design

Our experiment follows the guidelines provided in [66, 75, 127, 148] and the "green mining" method [51] consisting of seven prescribed activities; (1) choose a product and context, (2) decide on measurement and instrumentation, (3) choose a set of versions, (4) develop a test case, (5) configure the testbed, (6) run the test for per each version and configuration, and (7) compile and analyze the results. In this Section we describe our experimental design, in terms of Product Under Study, setup, metrics and protocol used for the experimentation. We report on compiling and analyzing the results in Sect. 6.4 and Sect. 6.5 respectively.

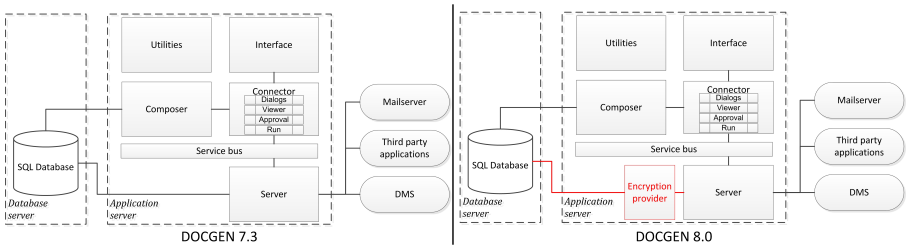


Figure 6.1: The functional architectures for Document Generator (DG) releases 7.3 (left) and 8.0 (right) portrayed on a commercial deployment. The changes are in red.

Table 6.1: Specifications of the hardware and software used for the experiment.

	Application server	Database server
Hardware	HP Proliant G5, 2 x Intel Xeon E5335 (8 cores @ 2GHz), 8 GB DDR2 memory, 300 GB hard disk @ 15.000 RPM	HP Proliant G5, 1 x Intel Xeon E5335 (4 cores @ 2GHz), 8 GB DDR2 memory, 300 GB hard disk @ 15.000 RPM
Operating system	Windows Server 2008 R2 Standard (64-bit), Service Pack 1	Windows Server 2008 R2 Standard (64-bit), Service Pack 1
Software	DOCGEN 7.3 and 8.0	Oracle 11.0.2.0.4.0

Product Under Study:

Document Generator (DG) is a commercial software product that is used to generate a variety of documents ranging from simple mailings to complex documents concerning financial decisions. The product is used by over 300 organizations in the Netherlands, counting more than 900 end-users, and annually generates more than 30 million documents. This experiment focuses on two releases of DG, 7.3 and 8.0, allowing us to compare the effects of a major release change [150].

In Fig. 6.1 the SA is shown for the DG releases included in the experiment. Starting with the *Connector* element, we have a central hub in the SA responsible for receiving user input through the *Interface*, collecting data from the *Composer* and handling communication with the *Service bus*. Together with the *Composer* element, responsible for merging document templates and definitions with database data, the *Connector* element handles all activities before documents are generated. *Utilities* and *Interface* respectively provide configuration options and an interface for DG. The final element on the application server is the *Server* element responsible for the actual generation of the documents and delivering the documents to where they are required. The database server hosts an Oracle SQL Database. Specifications of the hardware used in our experiment, i.e the application and database server, are provided in Tbl. 6.1.

Differences between Releases:

Looking at the SA, the major difference between the two selected releases is the encryption provider introduced on the application server in release 8.0. Data encryption was introduced in release 8.0 in order for DG to comply with the upcoming General Data Protection Regulation (GDPR) set up for

the European Union. In the case of DG 'Microsoft Enhanced Cryptographic Provider' is used: a module that software developers can dynamically link when cryptographic support is required. Encryption is applied in relation to the 'Server' element to remain independent from the database that is used, i.e. encrypted data is sent to the database

Another difference, which is not visible in the SA, can be found in the data model for the database. As release 8.0 is compliant with a new document management system, the datastructure is more complicated compared to release 7.3. We cross-checked our findings with the DG architect, to ensure completeness of our list of relevant changes between releases.

Test Case:

For the experiment we selected the core functionality of DG, the generation of documents, as test case. DG was instructed to erase existing documents of a certain type and consecutively regenerate these documents. The selected document type contains both textual information and financial calculations and a total number of 5014 documents was generated per each execution of the test case. During each execution, the 8 processes 'Interface', 'Run', 'Connector', 'Server', 'Oracle', 'TNSLSNR', 'omtsreco' and 'oravssw' processes were monitored on their respective servers. As the 'Microsoft Enhanced Cryptographic Provider' is not an executable but a dynamic library, it could not be monitored in isolation.

Metrics:

Comparing literature (cf. [51, 64, 67]) we find similarities in the measurement method that is applied, but a clear difference in the reported metrics. Although

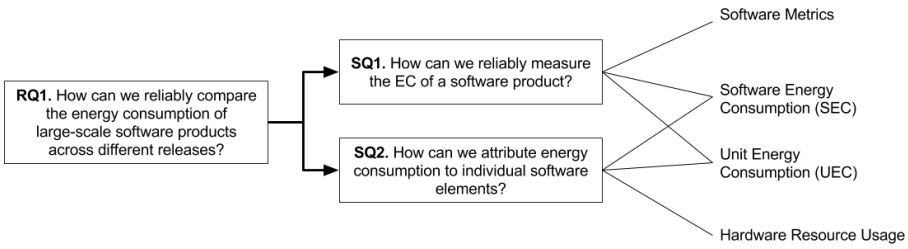


Figure 6.2: Overview of how the RQ and SQs of the experiment are linked to the reported metrics.

all report EC, the metrics target different stakeholders while still providing the details required to be in control of the software EC. During the design of an experiment, a choice should be made on what metrics are to be reported, as they should facilitate discussion between stakeholders, e.g. product managers and (potential) customers [33], especially in the case of a pioneering topic like the EC of software [45]. In Fig. 6.2 we show how the research questions driving our empirical experiment (RQ1, further divided into SQ1 and SQ2) are answered in terms of quantitative metrics. In the following, we further motivate our metric selection and rationale.

As regards the energy consumption of software, we measured the *Software Energy Consumption (SEC) and Unit Energy Consumption (UEC)* metrics [64]. The SEC is the total energy consumed by the software, whereas the UEC is the measure for the energy consumed by a specific unit of the software. In our experiment the units, i.e. *software elements* in our RQ, are the individual processes that comprise the product. This is not to be intended as a formal definition of what a software element is, but it is rather a choice determined by a practical aspect: our profiling method and tools are only able to attribute energy consumption at process level. Any finer granularity, although desirable, is not possible with current techniques.

In addition to the EC, we recorded *hardware resource usage*, as it can be used to accurately relate EC to individual software elements [16, 64, 67]. Profiling the performance requires the user to have a basic understanding of the hardware components that have to be monitored (e.g. hardware-specific details) and the context in which they are installed.

Following the definition of the 'Unit Energy Consumption' [64], in our experiment we monitored the following hardware resources:

- Hard disk: disk bytes/sec, disk read bytes/sec, disk write bytes/sec
- Processor: % processor usage
- Memory: private bytes, working set, private working set
- Network: bytes total/sec, bytes sent/sec, bytes received/sec
- IO: IO data (bytes/sec), IO read (bytes/sec), IO write (bytes/sec)

We also collected *software metrics* for both DG releases using CppDepend 6.0.0[1]. The tool provides several software size and complexity measures, such as 'cyclomatic complexity' and 'nesting depth' which allows us to more extensively identify differences between DG releases. These metrics are related to SQ1 as ideally they could provide an early indication of software EC at design time: by analyzing whether there are any correlations between specific

[1] http://www.cppdepend.com/

software metrics and the EC of the different releases, we could provide such an indication. Reporting software metrics is also useful to identify potential trade-offs between energy efficiency and other aspects of software quality (e.g. maintainability).

6.3.2 Interview Design

To follow up our experiment we investigated how the results were picked up by the DG team through interviews. More specifically, we looked into their views on the information provided by the EC measurements and the effects of having this information on their tasks. Additionally we explored the opinions and views on how EC measurements in relation to software can be promoted within their organization. To provide the most complete answer on RQ2, we aim to include different roles within the DG team in the study and provide insight on the operational aspects in relation to DG, e.g. its development, and strategic aspects like the product roadmap. For the interviews we followed the guidelines presented in [148, 152].

As the interviews took place after the case study, we could build on a common understanding of SEC between the interviewer and interviewees. However, given the relatively little experience of the team with SEC, we still decided to conduct semi-structured interviews. Structured interviews would have limited the interviewees to only think of those aspects that have a direct relation with the specific questions, instead of actively considering SEC in relation to their tasks and responsibilities. On the other hand, an unstructured interview could result in interviewees focusing on only those aspects they are more experienced in and might not be directly related to SEC.

The questions comprising the semi-structured interview were formulated during multiple brainstorming sessions between the authors, and tailored to help answer RQ2 in light of the experiment results. For each question (Tbl. 6.2) a goal was formulated in relation to determining the added value for an SPO. Note that the order of presentation corresponds with the order in which the questions will be posed to the interviewees. Given the novelty of the topic (i.e SEC) and the focus on the added value from the perspective of a product team and SPO, we were not able to validate our questions through a pilot interview with a person independent from the research.

For the interview itself, each interview was conducted following a protocol where the interviewees are first presented with a summary of the data, i.e. our previous work [61], followed by the interview questions. During the interviews notes were made on the important aspects mentioned by interviewees and, with consent of the interviewee, the interviews were recorded. Processing

the interviews was done directly after the interview, to prevent inaccuracies due to poor recall [152], and encompassed the identification of themes across interviews: aspects that are mentioned by multiple interviewees or are stressed from the perspective of a specific role. The notes made during interviews served as a guide to identify themes and were completed, e.g. by adding missed aspects, using the recordings. As such the notes served as a qualitative summary of the individual interviews and the main source to extract the final set of themes.

Table 6.2: The questions comprising the interview including the goal for each question.

Question	**Goal**
What do you think of measuring the energy consumption of software?	Elicit position of interviewee.
Does it seem useful to measure this aspect of the software?	Elicit position of interviewee.
What do you think of the changes that are measured across releases?	Determine opinion on measurements and differences.
Are you able to relate the measurement to your tasks as $< role >$?	Gain insight in $< role >$-perspective.
How would you apply the information that is provided?	Gain insight in value of measurements for $< role >$.
Looking at the data, did you miss aspects that would have been useful to include in the measurements?	Identify gaps in measurement information.
What do you think of software energy consumption in relation to quality attributes of the software?	Identify relations with SEC and determine opportunities for trade-off analysis.
What do you think of software energy consumption in relation to software metrics (e.g. lines of code, number of types, complexity measures)?	Identify relations with SEC and determine opportunities for further analysis.
In your opinion, what is required to put SEC on the agenda within the organization?	Identify strategic opportunities from SPO perspective.
What is required to have you consider this aspect as part of the job?	Identify opportunities from $< role >$-perspective.

6.4 Study Execution

6.4.1 Experiment Execution

Setup:

in line with the deployment portrayed in Fig. 6.1, two servers have been used: one for the application and one for the database. The setup is depicted in Fig. 6.3. The specifications of the application and database servers are provided in Table 6.1. To ensure consistency with regard to external factors (e.g. room temperature), the servers were installed in the same data center.

Both releases of DG were installed on the application server and Oracle was installed on the database server. The setup of the experiment, including the servers, is comparable with a commercial setting of the product. In the experiment, both releases use the same data set of an actual customer. To increase the consistency across measurements, a script is used to generate 5014 documents using DG.

Baseline Measurements:

to obtain a clean measurement of the EC related to solely DG, we determined the idle EC for the hardware that is used. This represents our *baseline*, and as such is subtracted from the total EC during a measurement, under the assumption that the increase in EC solely depends on running the software under test. As the idle EC heavily depends on the used hardware, this number should be determined separately for each hardware device in the experiment by performing measurements while the hardware is running without any active software.

However, using this method, the EC is not only related to DG, but also includes the effects of measurement software and Operating System (OS)-specific activities (e.g. background daemons), which we are not (yet) able to consider separately and thus considered to be part of the idle measurement. As we cannot completely control these aspects, we stopped any service and process known not to be required by the software product under test to minimize their effects (e.g. the automatic Windows update service). Additionally, we used a separate logging server to minimize the overhead caused by the data collection process.

Another aspect that we had to consider is the *cooldown time* a server needs after rebooting: after a reboot, several services related to the OS are active without direct instructions from a user. As these services require computa-

tional resources, they will most likely pollute measurements if the experiment starts while these services are running. Hence, measurements have to be taken in a "steady state" i.e. when the extra services become inactive.

As with the idle baseline, the cooldown time was determined for every hardware device included in the experiment. EC and performance measurements give an indication of when the steady state is reached. The cooldown time for our servers was determined to be 15 minutes.

Hardware– and Software–Based Measurements:

a measurement method concerning software EC should include both hardware and software approaches to obtain the right level of detail in the measurements. In terms of hardware measurements, we relied on power metering devices. As these meters are installed between a device and its power source, a meter was needed for each power supply unit of the devices under test. Although these meters are capable of achieving high levels of accuracy, their specifications was taken into account in the data analysis as even measurement errors of a fraction of a percentage point might prove significant at software level. Each of the servers in our setup is instrumented with a single WUP device[2] (see Fig. 6.3). WUP devices record the total energy consumption of the hardware once per second.

In order to profile individual software processes, we used software energy profilers (see Sect. 6.2). These tools estimate the EC of both the whole system and individual processes at run time, using power models based on computational and hardware resource usage. Unfortunately, most energy profilers record measurements with a 1-second interval, although a higher frequency is desirable [52]. While the usability and accuracy of energy profilers still have

[2]http://www.wattsupmeters.com/secure/products.php?pn=0, last visited on August 13, 2017

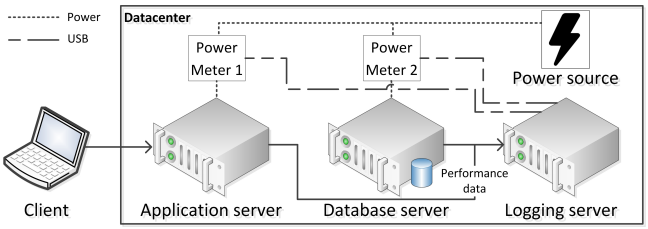


Figure 6.3: Experiment environment.

margins for improvement [63, 107], the reported measurements could still be used to detect differences in EC. In other words, although measurements in absolute terms may not be fully accurate, the relative differences between EC of the two releases we analyzed still provided useful insights.

In our experiment, we used the tool JM of Microsoft, that allows to estimate the power consumption of a system down to the process level. JM estimates EC on a model that first needs to be calibrated for the hardware it runs on. Previous experience with JM [63] shows that although JM provides a general idea of EC, it differs significantly from the actual EC. Since only one process can be measured per instance of JM, a separate instance for each of the concurrent DG processes is instantiated (see Sect. 6.3.1). Although relatively coarse, measurements on process level (i.e. the concurrency views on the system [126]) can be translated to more fine-grained aspects using an architectural perspective [64].

The hardware resource usage of the application and database servers were measured using the standard performance monitor (*perfmon*) provided with Microsoft Windows. Performance data is remotely collected using the logging server, thereby minimizing the overhead of measurement on the actual hardware.

Summarizing the data collected for each individual measurement we have:

- WUP measurements of the energy consumption at the level of the hardware;
- JM estimates for each of the processes together with an estimate of the total energy consumption;
- one *perfmon* file containing resource usage data for both the application and the database server;
- the start and end timestamp for each measurement;

After each measurement, both servers were been reverted to the initial state, restarted and were left untouched for the determined cooldown times.

Data Synchronization:

an important requirement for data analysis is to have synchronized measurements. As measurements are obtained from different sources, their timestamps have to be synchronized to avoid irregularities in the data. For example, if a specific activity is performed and the timestamps across sources are not in sync, there is a risk of missing the data related to this activity. To address this issue, in our experiment we continuously synchronized the clocks for all measurement sources using the Network Time Protocol (NTP).

Measurement Protocol:

while the "green mining" method [51] provides a solid basis for designing an experiment, no details are provided on how to actually perform reliable measurements within an experiment. To this end, we propose the following protocol applying the information presented in this section, which is an extension to the activities presented by [51]:

i Restart environment;
ii Check time synchronization;
iii Close unnecessary applications;
iv Start performance measurements;
v Remain idle for a sufficient amount of time;
vi Start EC measurements;
vii Run measurement and wait for run to finish;
viii Collect and check data;
ix Revert environment to initial state;

The protocol ensures consistency across measurements and improves the reliability of each measurement [148].

6.4.2 Interview Execution

The interviews were conducted with the architect, the product manager [33], a developer and a tester of the DG team, the latter also being the 'Scrum master', and took place between four to seven months after the results of the SEC measurements (i.e. [61]) became available. Given the nationality of the team the interviews were conducted in Dutch, which meant we had to translate the interview questions to Dutch and the interview results from Dutch to English. Also, as not the entire team was situated in the same office building, we had to conduct one interview remotely. On average an interview lasted approximately one and a half hour.

For the analysis, the notes made during the interviews appeared sufficient to identify all relevant themes and in practice the recordings were only played back once to confirm the themes. Unfortunately, even though all interviewees gave their consent for recording the interview, only three out of the four interviews were successfully recorded. In the case where we lacked the recording, we cross-checked the processed results with the interviewee for inaccuracies: none were identified.

The results of the interviews are reported in the results sections (Sect. 6.5), sorted by the themes that we identified. Combined with the other information at hand, these results are used to provide an answer to RQ2 (Sect. 6.6).

6.5 Results

6.5.1 Experiment Results

In this section we extensively report our experimental results. The complete dataset is openly available[3].

Both the WUP as well as the JM measurements report the EC as an average of the instantaneous power over the sampling interval. To calculate the total EC, we either multiply the average power with the time the system was running, or sum up the recorded energy measurements. We report our findings in Watt (W) or Watthour (Wh) where applicable.

6.5.2 Baseline Measurements

The results of the idle and JM overhead measurements are presented in Table 6.3 along with the measurement time to determine the averages. The measurements were collected over 5 runs per scenario, spanning more than 50 hours of measurement time. Starting with the idle EC we found an average power consumption of 274.54 W and 252.59 W for respectively the application and database server. Considering that the servers are almost identical, we can only allocate this difference of 21.95 Watt (W) to the extra processor available in the application server.

An interesting finding is the fact that there is minimal to no overhead on the account of JM. Further investigation showed a base memory usage by JM, which increased when JM was actually logging measurement data. While

[3]https://www.dropbox.com/sh/kk9kastzo2cypur/AABA3ZuWbSi-F4k8o8Af6KJJa?dl=0

Table 6.3: Comparison of server power consumption in different "idle" scenarios including measured time.

Server	**Idle**		**Idle (JM running)**		**Idle (JM measuring)**	
	Total time	Avg. Power (W)	Total time	Avg. Power (W)	Total time	Avg. Power (W)
Application	57:11:30	274.54	54:06:21	275.28	54:06:21	276.18
Database	57:11:30	252.59	54:06:21	252.79	54:06:21	253.39

logging, performance measurements show increases in the memory usage of the JM instances which are periodically 'reset' to a base memory usage. Our guess is that the pattern in memory usage corresponds to incrementally adding measurements to the CSV file. Despite this variability in memory usage we could not detect any change in EC.

As part of the baseline measurements, we also determined the power consumption interval of the servers. Based on 36 hours of running the servers at full capacity, we determined a maximum power consumption of 367.3 W for the application server and 291.2 W for the database server providing a range of 92.02 W and 38.41 W respectively. Again we can only explain the difference due to the impact of the additional processor, showing that, all other things equal, the power consumption range increases with a factor of 2.4. Using the range we are able to normalize the measured power consumption and better investigate the impact of the software on the hardware EC.

6.5.3 DG measurements

We performed 20 executions for each DG release (7.3 and 8.0). During each execution, we collected the data described in Sect. 6.4.1. Tables 6.4 and 6.5 summarize the results in terms of mean (μ) and standard deviation (σ) for the application and database server, respectively. Notice that the process-level results for the database server only include the JM results for the 'Oracle' process. The other processes were excluded from the table as their EC was reported as 0 by JM, despite them being active. The 'Interface' process on the application server, that runs the GUI of DG, was not active during the experiment as the DG execution was scripted.

Comparing the measurements between releases, two differences are clearly visible. First, the run length increases by 12 seconds on average in the 8.0 release. A second difference is the overall increase in energy consumption of DG 8.0 as compared to 7.3 of 4.14 Wh according to the WUP measurements: 2.97 Wh for the application server and 1.17 Wh for the database server. Such increase, to a lower extent, is also reflected in the JM data. This difference cannot be explained only by the increase in execution time: if we subtract the average amount of energy consumed by DG in 12 seconds from the 8.0 average, we still find a difference of 2.05 Wh and 0.32 Wh.

The SEC for both DG releases is calculated by subtracting the 'idle with JM' EC from the total EC as reported by the WUP for the length of the run. For release 7.3 we find a SEC of **2.57 Wh for the application server and 8.03 Wh for the database server.** Measurements for release 8.0 provide a SEC of **4.61 Wh and 8.34 Wh for the application and database**

server.

Placing the SEC in the perspective of the ranges calculated for each server, we find that only a relative low portion of the available resources is actually used by DG. Even when considering the total power consumed by the servers, the average power consumption figures for release 7.3 are 276 W and 255.66 W for the application and database server. For release 8.0, the averages are 277 W and 256.00 W respectively. Considering this in relation with the power interval we reported in our baseline measurements, at most 1.87% and 8.36% of the application and database server capacity is used, respectively. These figures in our opinion underline why virtualization, or resource sharing in general, could still be an important aspect to reduce the EC related to software.

6.5.4 Joulemeter Estimations

The SEC can also be calculated using the estimations provided by JM (Tbl. 6.6). Using this data we find a SEC of **1.45 Wh and 5.69 Wh** for the application and database server with release 7.3, and **1.57 Wh and 5.72 Wh** with release 8.0. There are evident differences between these SEC figures and the

Table 6.4: Summary of the experimental results on the Application server for both DG releases.

		Application server				
		7.3		**8.0**		**Diff**
		μ	σ	μ	σ	Δ
Run length (hh:mm:ss)		2:48:16	4 s	2:48:28	7 s	+12 s
Processed Documents		5014		5014		
WUP (Wh)		774.59	1.18	777.56	0.84	+2.97
Run	Total (Wh)	765.20	0.32	766.21	0.63	+1.01
	Process (Wh)	0.0002	0.00009	0.0003	0.0001	+0.0001
Server	Total (Wh)	765.18	0.33	766.21	0.63	+1.03
	Process (Wh)	0.744	0.00002	0.758	0.007	+0.014
Connector	Total (Wh)	765.19	0.34	766.22	0.63	+.03
	Process (Wh)	0.144	0.004	0.22	0.004	+0.76

Table 6.5: Summary of the experimental results on the Database server for both DG releases.

		Database server				
		7.3		**8.0**		**Diff**
		μ	σ	μ	σ	Δ
Run length (hh:mm:ss)		2:48:16	4 s	2:48:28	7 s	+12 s
Processed Documents		5014		5014		
WUP (Wh)		716.99	0.45	718.16	0.61	+1.17
Oracle	Total (Wh)	706.37	0.29	707.27	0.51	+0.9
	Process (Wh)	5.63	0.02	5.62	0.02	-0.01

ones obtained using WUP. In our data we observe that the WUP on average provides a higher SEC of 1.12 Wh and 2.34 Wh for the application and database servers. This difference is probably due to an underestimation given by the JM power model.

Apart from the total EC, the JM data allows us to calculate the SEC according to measurements on process level, i.e. the 'Run', 'Server' and 'Connector' processes on the application server and the 'Oracle' process on the database server. The measurements for release 7.3 provide a SEC of **0.89 Wh and 5.69 Wh** for the application and database server. With release 8.0 we find a SEC of **0.97 Wh and 5.62 Wh** respectively. The large differences in the SEC figures could be an indication that, despite our efforts, several processes are still active in the background alongside the DG processes.

6.5.5 Software Metrics

The results of the analysis on software metrics are shown in Tbl. 6.7. Our results show a size increase of DG 8.0 in terms of lines of code (LOC) (6.3%) and number of types (19.64%), projects (33.19%), namespaces (31.46%), methods (13.88%), fields (33.23%) and source files (27.80%). Since our case study was performed after the releases were commercially available, we were not able to determine all churn measures presented in [51]. Specifically, the added and removed lines and the file churn require a fine-grained tracking during development.

If we consider EC in relation to the metrics, we find that the EC per line of code is 0.047 Wh for release 7.3 and 0.044 Wh for release 8.0. suggesting an increased efficiency per line. This increased efficiency also holds for the other size-related metrics. However, any usage of LOCs for quantitative analysis of software products is under the strong assumption that every LOC is equivalent in terms of efficiency. Inefficiently written code (e.g. resulting in more LOC) could result in a lower and incorrect EC per line of code.

Table 6.6: The SEC according to Joulemeter calculated using the total power consumption and the power consumption per process.

	Application server		Database server	
Release	7.3	8.0	7.3	8.0
Total EC (Wh)	1.45	1.57	5.69	5.72
Process level EC (Wh)	0.89	0.97	5.69	5.62

Table 6.7: Software metrics obtained using CppDepend 6.0 for the DG releases.

Size metrics	DG 7.3	DG 8.0	
Lines of code	31770	33770	
Types	1389	1663	
Projects	74	103	
Namespaces	89	117	
Methods	9658	10999	
Fields	10757	14726	
Source files	1698	2170	
Complexity metrics			Units
Max cyclomatic complexity for methods	152	165	Paths
Max cyclomatic complexity for types	2723	3145	Paths
Average cyclomatic complexity for methods	2.48	2.53	Paths
Average cyclomatic complexity for types	37.35	40.78	Paths
Max nesting depth for methods	30	32	Scopes
Average nesting depth for methods	0.89	0.82	Scopes
Max # methods for types	535	614	Methods
Average # methods for types	7.63	7.2	Scopes

6.5.6 Interview results

The interview results on the stakeholders' views on EC measurements are presented below, arranged by the common topics that emerged across the interviews. For each topic we combined the results gained from each interviewee.

Sustainability: in general, sustainability, including EC, is perceived as an important topic in the Dutch software industry and this importance has increased with the recent climate deals[4]. Dutch municipalities, which comprise a large part of the DG customer base, are compelled to consider sustainability in their processes and are becoming aware of the role IT can play. However, given the novelty of this area there are no hard requirements from the customers (yet).

The tester, from his product owner perspective, and product manager were enthusiastic about measuring the EC of DG as a way to gain competitive advantage. Striving for a reduced EC and thereby environmental impact is seen as desirable for the product and the company as a whole.

However, the team was convinced that the impact of renewing hardware on

[4] http://ec.europa.eu/clima/policies/international/negotiations/paris/index_en.htm

the EC is higher than changing software. According to the developer 'hardware changes are easily made and are still the low-hanging fruits when it comes to EC.' Although it is important to monitor the resource utilization, e.g. CPU utilization, to control and improve software, renewing hardware is expected to grant higher economic savings.

Experimentation: overall, the interviewees were satisfied with the results of the experiment and found no reasons for concern. The functionality, i.e. encryption, was added to comply with regulation and the differences were not significant. On a strategic level, the product manager was pleased by the fact that this aspect of the software could be measured and made tangible. Until now the aspect of EC was relatively abstract, especially in relation to software.

The results did raise the interest of the architect and developer: specifically, they were surprised by the elements and processes that were shown to be affected. However, further investigation into the matter would (among others) require isolating the encryption provider and testing this aspect separately (e.g. encrypt and decrypt a number of rows) to determine its impact. Given that, analysis of the code would still be required to find the actual cause(s) of the unforeseen change.

Consequently, a business case was made to further investigate the impact of the encryption provider on the software including the costs for investigating and potentially redesigning. Weighing the costs against the projected benefits (i.e. lower energy consumption and potential increased performance) it did not appear beneficial to invest in this matter at this point in time. Still, this aspect will be monitored as it could become more important in the future.

Software energy efficiency might become more important when the scale of the transactions increases. In the experiment DG was instructed to generate 5014 documents, which is only a small fraction of the 30 million annually generated documents. If the software is made more efficient, this is bound to have a significant effect on the resources, and as a consequence this will also affect the forthcoming EC.

In this sense, performing experimentation on a larger scale would be useful. For example, the tester and the developer suggested increasing the duration of the experiments. Simulating DG usage for an entire working day could help the tester detect EC patterns over time and possibly provide input for a smarter scheduling schema. For the developer, a longer experiment duration could help detecting errors and bugs that only show after a longer period in time. The effect of small loops or try-catch recursions, for example, can keep piling up over time until their presence is noticed. On the long run they could significantly affect the resource usage by the software. Even though errors like these are often not critical and can be resolved by rebooting the system,

as a developer they are valuable to ensure system stability over time. Also testing in different environments, e.g. SaaS, with multiple servers, layers of virtualization and different hardware resources in general was considered an interesting increase in scale.

The team also felt strong towards the idea of presenting results in relation to a benchmark instead of 'raw' measurement data. Comparison between releases directly identifies differences and can be used to pinpoint those aspects that have evolved dis-proportionally. Presenting 'raw' measurement data, e.g. CPU utilization, would require more effort to understand the measurements, correctly interpret the results and translate results to concrete actions.

With respect to EC the interviewees did not see a clear relation with software metrics. Software metrics are mostly used as an indication for the maintainability attribute of the software and as a means to monitor the evolution of DG in general. Higher technical debt, for example, could be an indication of poor maintainability of the software. Especially the comparison with other products is important, which is an internal indicator for the quality of the development activities.

Functional vs. non-functional aspects: in general the software development practice of the team can be characterized as functionality-driven [6]. The architect stated: 'writing code concerns functional aspects, not performance or energy. Developers do not consider non-functional aspects while writing code'. It was made clear that the only way a developer would work on, for example, performance is to make the requirement very concrete and functional; e.g. a window should open in one second.

With respect to EC and the quality attributes of the software product, the product manager admitted that this aspect was, due to the functionality driven development, not high on the priority list. As there are currently no customer requirements on this aspect, the product manager could not justify a trade-off in favor of sustainability against, e.g., performance. It would be valuable to consider EC on this level, but from a strategic perspective that would require more awareness on the customer side to justify why certain decisions are made. For DG the risk was considered too high to make these trade-offs themselves.

On the other side the developer pointed out the necessity of certain design decisions that have been made. Although not related to DG, the developer mentioned the usage of the HTTPS protocol which according to him requires twice as much resources compared to simple HTTP. On large scale systems the decision to apply HTTPS is bound to have a significant impact on the EC sometimes without having a clear benefit in certain cases. Any design decision should include trade-off considerations between the relevant quality attributes.

The team agreed that if EC is labeled as a key factor by the organization,

then decisions on adding or changing functionality should be made in the design phase and EC should be a prominent factor in the decision-making process. In a sense EC should be considered the same as the other quality attributes for the software and trade-offs should be made depending on the organizational policies. The tester admitted that testing on non-functional aspects, which EC is considered to be, is in general not done extensively. Given the current stage of the product life cycle where the product is transformed to a SaaS solution, there is no high priority to do so.

Relating measurements to roles: with respect to the measurements in relation to his tasks, the developer argued that the measurements are foremost an indication of whether he has done his job right. If large, unexpected discrepancies are observed, it could be an indication that a mistake has been made in the code itself. As such the measurements are used as a check. The same holds for software metrics, which essentially are used as a means to check whether any changes are in line with the adjustments that have been made.

As a software producing organization the product manager saw added value in having a unique selling point and also saw potential to strengthen the organizations' image with respect to sustainability. Compared to competing products, simply providing insight in the EC of the software could help in winning over customers. Performing EC measurements not only potentially helps the customer become more sustainable, but also visibly lets the organization take responsibility for their contribution.

The tester noted that a focus on non-functional aspects requires different tests performed in different environments. The added value for him as tester specifically was marginal in the form of the knowledge gained by performing these tests. Finally, the architect noted the strategic advantage of performing these measurements and added the potential increase in software quality. An aspect like EC requires trade-offs to be made and stimulates to rethink design decisions.

Putting EC on the agenda: To put EC of software products on the agenda would require a change in mindset within the organization. Progress with the software itself, i.e. functional improvements, is most important, but there are other developments that require a broader perspective on the software. For example, the scale of software products is changing, e.g. on-premise to cloud, which also affects the resources used by the software. In the end, to structurally consider EC, all interviewees agreed that the costs and financial gains should be made visible.

Also making EC tangible, like in this study, is essential. Presentations on being sustainable have been given in the past and often left the team with more questions than answers. From the perspective of the product manager

this topic can not be forced upon products due to other factors weighing in, but making EC concrete helps to include this aspect in the decisions that need to be made. The tester however disagreed and noted that a top-down approach would help to have an organization-wide focus on this aspect of software and will hopefully stimulate attention from the bottom up.

The future of DG: Currently release 9.0 for DG is being developed where the system is redesigned to a software as a service (SaaS) product. The bulk of the work for the architect is to redesign the system in terms of (functional) aspects that were originally not designed to be, for example, multi-tenant. Again the architect stressed that a new development viewpoint would only guide development activities (e.g. by providing insight in changed dependencies) while still a lot of work has to be done on code level.

Apart from the development activities, there is awareness on the 'different dynamics' with a SaaS product; shared resources, multi-tenancy, a changing pricing model, continuous delivery and the total cost of ownership in general. Deploying DG as a SaaS product will most likely emphasize non-functional requirements for the system, thus requiring a better comprehension on these aspects. In line with the insights provided earlier, the team expected EC to be more relevant in SaaS deployments where a lower energy consumption can directly affect the total cost of ownership and thereby the strategy for a product.

6.6 Discussion

In this section we discuss the results presented in the previous Section, answer the research sub-questions for RQ1 and provide an answer to RQ2.

6.6.1 SQ1: Measuring the EC

The protocol that was applied in the experiment ensures that the relevant variables (that can be influenced) are under the control of the researcher. It also provides guidelines for the data collection and processing. By following the measurement protocol we obtained consistent and comparable data across measurements, confirmed by the small standard deviations found with each item, and were able to compare different releases of DG from an EC perspective (Fig. 6.4). This allows us to conclude that **the measurement method we adopted can be used to reliably measure the EC of a software product**.

In terms of software metrics, due to our limited dataset we could not perform a statistical correlation analysis. Although the size metrics show an increased efficiency per line, we cannot claim a causation link between such increase and the increase in energy consumption. However, we argue more valuable insights can be gained from the complexity metrics. The cyclomatic complexity for types and methods is expressed in the number of independent paths through program source code where an independent path is a path that has at least one edge that has not been traversed before in any other paths. A high cyclomatic complexity over time increases the probability of errors while maintaining the software (i.e. decreases maintainability). The nesting depth represents the depth of a nested scope in a method body where a lower nesting depth is better for the readability and testablity of the software.

As per the size metrics, we cannot claim a direct causation link between the increase in complexity and the EC. However, given their relation to QAs (see the ISO 25010 standard), the complexity metrics could indicate a potential impact on the design of a system architecture in terms of allowing or precluding other QAs.

This allows trade-offs between different sustainability requirements, thereby enabling decision-making with respect to the EC of a software product. For example, one could consider EC in relation to its maintainability and specif-

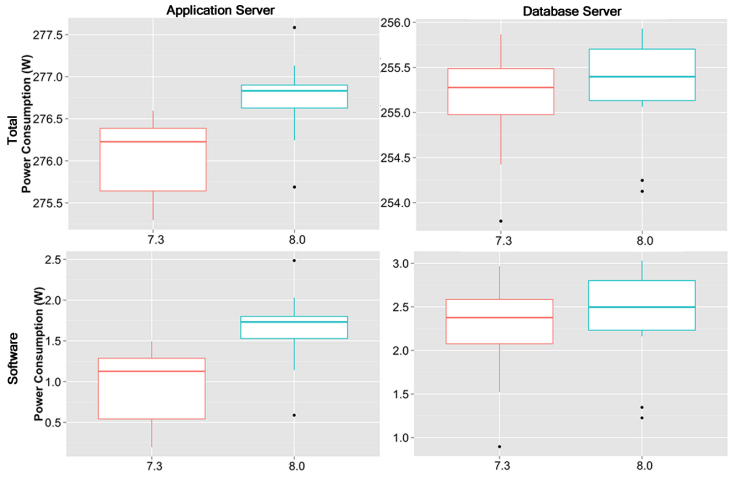


Figure 6.4: Boxplots summarizing the total and software power consumption measurements for the application and database servers.

ically its technical debt (i.e. results of past decisions that negatively affect its future [80]). Maintainability is a QA that is normally associated with the technical sustainability dimension.

Looking at the reported complexity metrics we find an increase in the average cyclomatic complexity values, whereas the average nesting depth for methods decreases with release 8.0. Most notable is the increase perceived in the average cyclomatic complexity for types from 37.35 to 40.78 paths. A lower cyclomatic complexity in general indicates an improved maintainability and testability of the software, and an improved readability of the code itself. While this might seem a deterioration of the software quality, this finding might indicate a (deliberate) trade-off has been made between the maintainability and security QAs.

6.6.2 SQ2: Relating EC to Software Elements

In order to answer SQ2, we used Joulemeter to estimate the energy consumption of individual software processes composing our application. We also validated the accuracy of Joulemeter, building a separate model from our performance dataset using linear regression. The model outperforms JM at machine-level prediction i.e. trying to predict the total system EC, see Fig. 6.5. More details on this model are provided in the Appendix.

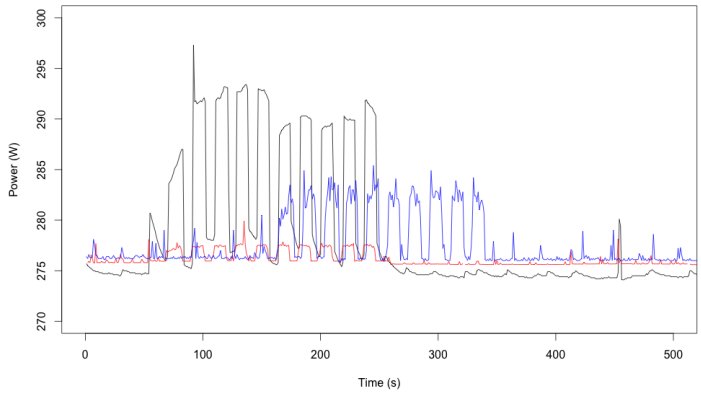


Figure 6.5: Performance of the penalized regression model (in red) vs. Joulemeter (in blue). Measured values by WUP are in black.

However, if we aggregate the estimation obtained using our model for all the processes running in the application, we obtain very similar percentages to those computed via JM. Given this validation, we must conclude that JM provides a fairly accurate estimation of the energy consumption of specific processes.

Although both Joulemeter and our regression model are unable to attribute the total SEC to specific processes, **our profiling method allows us to observe relevant changes between the different processes composing our software product** which allows us to make informed hypotheses about the impact of each elements on our software product. For example, the 'Oracle' process in the database server is by far the most energy–consuming. This indicates that the database is a potential *hotspot* [124] and, as such, a candidate for optimization.

The most apparent difference between the DG releases is the introduction of the encryption provider element on the application server. Unfortunately, as this element is a library, we were not able to perform measurements specifically on it and isolate its energy impact. We are, however, able to analyze the effects that are caused by the addition of this elements and infer possible explanations for EC differences.

According to the architect, the introduction of the encryption provider was accompanied by minor changes in the 'Server' element. Interestingly though, while an increase in EC is found in the 'Server' element, the main EC difference was found in the 'Connector' element going from 0.144 Wh to 0.215 Wh. This difference could not be explained based on the adjustments applied in release 8.0. This *unforeseen change* in EC was reason for the architect to further investigate the matter in the near future.

With regard to the difference in run length an explanation is sought in the encryption that is applied, possibly extending the time required to set up a connection and communicate data. Apart from increased duration of the run, we also found that the net number of seconds reported by JM increases with release 8.0 for the 'Server', 'Connector' and 'Oracle' processes. Considering that JM uses a linear model to estimate EC, a higher execution time should result in a higher EC for these processes. However, this only holds for the processes running on the application server.

Overall we can conclude that the changes applied in release 8.0 increased the SEC by 4.14 Wh for the generation of 5014 documents. Although small, these differences could add up significantly with each installation and generated document: in literature [51], a savings of 0.25 W is shown to potentially equal the power use of an American household for a month.

With these results stakeholders of DG are now able to quantify and justify

changes in EC. Considering the cause of this increase, i.e. being compliant with a new document management system and ready for the General Data Protection Regulation, the stakeholders deemed the increase in EC as acceptable. To increase efficiency, however, the software architect will still look into the 'Connector' element.

6.6.3 Visualizing Software Energy Consumption

Besides measuring on process level, the resource usage data was measured at server level (see Sect. 6.3.1). This data allows us to characterize the hardware aspects that affect the EC range (Sect. 6.5.2) and *visualize* the impact of release DG 8.0 on the servers. Specifically we use the disk bytes/sec, % processor usage and working set to respectively calculate the hard disk, CPU and memory dimensions. Note that the network dimension is not included since these metrics were also excluded for measurement on server level.

In order to create the visualization, we follow the approach described in [98] to create a radar graph. For each dimension, we calculate the normalized delta using the averages across measurements. Since the values are normalized, a delta larger than '1' shows a deterioration compared to release 7.3 and vice versa. Given the focus on EC, we included this dimension in our calculation.

The results for each server are shown in Fig. 6.6. The line surrounding the green area is the benchmark for the normalized delta and represents release 7.3. The line surrounding the red area, representing release 8.0, indicates that the impact of the encryption provider on the available resources is mainly through the memory and hard disk usage, i.e. delta > 1. Note that the radar

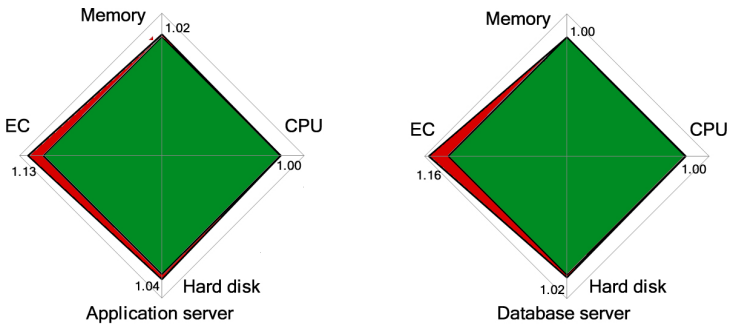


Figure 6.6: Radar graph showing the impact of DG release 8.0 on EC and the relevant hardware aspects.

graph is zoomed in, i.e. the maximum value of the graph is set to 1.2, to better portray the results. This finding is contrast with the expectations of the DG development team: the encryption was regarded as increasing the load on the CPU, however the normalized delta for the CPU usage is 1.00 and shows no difference across releases. Most prominent is the deterioration of the EC, with deltas of 1.13 and 1.16 respectively, which clearly skews into the red area of the graph.

6.6.4 RQ2: the added value of EC measurements for the SPO

Through interviews we obtained insights into how EC information for a software product affects different roles in the DG product team. With respect to the differences found between the DG releases, there was no reason for concern as the EC increase was considered marginal. However, a clear lack of reference material also prevented the interviewees to put the increase in perspective with other products. As such, a first aspect of added value for each role was to have EC measurements in the first place.

The lack of reference material implies that EC measurements could potentially provide a strategic advantage with respect to competitors. Even though all roles acknowledged the potential, only the product manager and architect roles are actually able to extract value from this advantage. The product manager can promote improvements on the energy consuming behavior of the product and stress the (temporary) uniqueness of these efforts, potentially leading to increased sales and an improved market position. On top of that, the product manager is able to include EC requirements on the roadmap and steer development towards strengthening this aspect of the software. The architect on the other hand, can plan technical adjustments that help to reduce the total cost of ownership for a product.

One added value that holds for all interviewees is the **increased awareness on the topic of sustainability and the relation with software products**. Before the experiment was performed, sustainability was considered as a topic that should be addressed with other aspects in the organization, i.e. renewing hardware. The relation with software would not have come to mind, which would be a missed opportunity according to the interviewees.

A final added value is related to non-functional aspect in general and not specifically to the EC measurements. By positioning EC in relation to quality attributes of the software, the measurements helped to re-introduce non-functional aspects on the agenda. The focus on functionality made that non-functional aspects were often only considered when issues were experienced by

the customer. Keeping non-functional aspects in mind allows to have more control over the software and the quality thereof.

6.6.5 Energy Consumption on the Product Roadmap

Although the provided delta analysis provides practitioners valuable insights in the EC of their software product, the creation of energy efficient software starts with the design of the software [8], i.e. with its architecture. Changes on this level often require thorough preparation and development, and should be planned ahead on the roadmap. Especially when a balance has to be found with planning and realizing customer requirements [41].

As the performed case study shows, the development team tends to mainly focus on the functional aspects, and postpone trade-off analysis of the relevant QAs. With the presented analysis, EC becomes tangible for the developers, and as such, allows them to start reasoning about the consequences of implementation choices. In this way, EC can be introduced as one of the QAs to be taken into account and the measurements themselves serve to **create awareness** on the topic. Ideally increased awareness would result in the inclusion of EC related requirements on the roadmap.

With EC related requirements on the planning EC becomes an aspect of the system design. Different studies have evaluated EC with respect to system design. For example, a recent study shows data locality plays an important role in the EC of multi-threaded programs [119]. Similarly, Trefethen and Thiyagalingam [144] have shown that increased performance does not always have a direct relation with EC. These studies show the need for a separate EC perspective on software architecture. We envision a EC perspective [60] for software architects to analyze and evaluate the EC of a software product. Applying a separate EC perspective enables practitioners to structurally consider aspects that concern the design of a system. On top of that, the knowledge gained from applying the perspective helps in making informed decisions on trade-offs with other design decisions.

For example, relating the energy consumption to the functional views of a software architecture, the architect can analyze the EC per functionality, and decide, based on performance indicators, such as execution time or frequency, whether the functionality requires separate attention. With the presented delta-analysis, the consequences of the proposed changes can be analyzed. By applying the EC perspective different trade-offs are possible. For example, to exchange modules or services for more energy efficient sustainable variants, such as cloud federation [122].

Since the roadmap encompasses the future direction of a product, the

roadmap could be considered a 'to be' system design. As such, a product roadmap allows for the **controlled evolution of the product**.

From an economic perspective, (re-)designing software should be done with the lifecycle in mind. For each investment in the product a **business case** should be created to ensure scarce resources are directed to where they add most to the product, and organizational, strategy. The lifecycle stage for DG, for example, is two-fold as there is a current product and a planned new product. Release 8.0 is considered a mature product, i.e. middle of life, but is going towards end of life on the technological aspect. However, strategic management has decided that DG is to be renewed and the product is redesigned to a Software as a Service solution. While the new version will replace the current one, the decision was also made to label this new version as release 9.0 to maintain the marketing position for the product. Release 9.0 is currently in its beginning of life phase and the decision of the architect to look into the 'Connector' element is should be considered with release 9.0 in mind. Investigating this element for release 8.0 would not be considered as economically viable.

6.7 Threats to Validity

This section discusses the threats to internal, external and construct validity [66, 127, 148] of our research.

6.7.1 Internal Validity

The internal validity is concerned with the uncontrolled factors that might affect the results of the experiment.

Energy measurement reliability. Although we were able to clearly identify differences between the estimated energy consumption of the selected processes, the estimations only accounted for percentages of the variation in EC. A brief cross–validation conducted by means of a self-obtained regression model based on resource consumption information (see Appendix) was still unable to fully explain the total EC. Hence, additional work is needed to have a clear and reliable attribution of the energy impact of single processes.

Sampling Interval. Both hardware and software measurement approaches have a sampling interval of one second. Given the nature of electrical power, this low sampling frequency might result in an underestimation of EC due to high-frequency energy components. However, this interval is also commonly used in the state of the art [51].

OS Effects. In the experiment the EC of the OS was included in the reported SEC for DG as we could not measure the OS separately. Ideally, the OS would be considered as a separate layer with its own, distinguishable EC. Also, we cannot fully exclude the possibility of OS processes and services becoming active in the background during a measurement. A deeper analysis of the performance measurements could detect such background activities, however this was deemed out of scope for our study. Instead, we mitigated this threat by performing a large number of trials (20 per each release) that should average out these effects as much as possible.

Interaction among multiple applications. The EC of software not related to DG was measured and taken into account (as overhead) while calculating the SEC. These measurements were performed separately to obtain clean overhead figures. However, by doing so we assume a negligible impact of the interaction among DG and other applications running in the system.

6.7.2 External validity

The external validity addresses the extent to which the results can be generalized beyond the experiment.

Experiment Setting. Our experiment is limited to a single application and tested on a single testbed. Hence, we cannot generalize the effect size of changes in the EC on our target population of commercial software products. Nevertheless, we argue that our work can be useful to generate awareness in software developers and architects about the knowledge gap in software energy efficiency.

Hardware Specificity. One of the main factors that could influence the EC measurements is the specific hardware; new generations of hardware often yield improved performance and EE. We mitigated this thread by performing extensive baseline measurements to determine the idle EC. We argue that differences might be found when comparing the absolute numbers, but that the relative proportions should be consistent across different hardware setups.

Measurement equipment. We applied both hardware and software measurement approaches to obtain our experimental data. Given the diversity of power meters and software tools available, each with their own advantages and limits, there is an unavoidable dependency on the equipment when it comes to the accuracy and detail of the measurements.

Team and organization dynamics. The added value identified for the different roles included in the interview could be specific for the team and the organization in which the team operates. An organization that does not have policies on sustainability or does not operate in a market that requires to do

so, will probably not experience the added value as described. The same holds for a team that does not have any affinity with the topic of sustainability.

Interview results. We acknowledge that the number of interviewees, four out of a six-person team, is too small to generalize the results. However, given the specific focus of the interview on the experiment results in the context of a software product team and SPO, we could not extend our population beyond our specific case. Still, we managed to include all roles represented in the DG team and our results should be considered as a first insight into SEC from the relevant perspectives for a software product.

6.7.3 Construct validity

Construct validity addresses the degree to which the measures capture the concepts of interest in the experiment.

Metrics vs. outcome. A central aspect in performing EC measurements is to have a clear view on the metrics that should be reported. To mitigate this threat, in our experiment design we extensively report on our metrics selection and rationale for the experiment and the stakeholders. With regard to the metrics themselves, a consolidated list is already available in the literature [16].

Definition of change. The goal of our study is to relate software changes with their effects on the EC. Although we can empirically assess the difference between the energy consumption of the two application releases, we do not aim to provide a general definition of what a 'change' represents in software. For that purpose, we simply use two different releases of the DG product. Then, we provide insight as to which specific changes could affect the observed difference in EC. Further work is needed to pinpoint (and predict) the exact energy consumption impact of a generic software change.

Interview questions. The interview questions allowed us to identify potential added value of EC measurements for an SPO, and the semi-structured nature provided room for the interviewees to express themselves beyond our predefined set of questions. However, as the questions themselves were not validated, we acknowledge a different approach, i.e. direct questions on the added value of EC measurements, could yield different results. While formulating the interview questions however, we carefully considered the trade-off with the generalizability of our results as such an approach could yield results that are too specific with respect to the software product, the target market and the SPO.

6.8 Conclusions

Software sustainability, and in particular software energy efficiency, is hardly addressed in industrial contexts. Previous work [118] shows that due to lack of tools and knowledge, energy efficiency is not on the software roadmap of most SPOs. To investigate this aspect, we posed two **main research questions**: 'How can we reliably compare the energy consumption of large-scale software products across different releases?' (**RQ1**), further divided in 2 sub-research questions: 'How can we reliably measure the EC of a software product?' (*SQ1*) and 'How can we attribute energy consumption to individual software elements?' (*SQ2*), and 'What is the added value for a software producing organization to perform EC measurements on software products?' (**RQ2**). We presented the design and results of two empirical studies: an experiment performed on the EC of a commercial software product and an interview with the stakeholders from the SPO of the product. In the previous section we provided an answer to both RQs.

To reliably measure the EC of a software product (SQ1), we followed a rigorous methodology and we extensively documented our energy profiling method. Our experimental results show that the total EC of DG increased with release 8.0 w.r.t. 7.3. While this increase was expected, actual EC data now verifies it quantitatively. The stakeholders deemed this increase as justifiable, and the SPO experts could use the results to establish a better causation link.

The second sub-question (SQ2) addresses how to attribute EC to individual software elements. By means of energy profilers our experiment includes the estimation of the EC at process level. Our analysis showed that energy profilers can only explain percentages of the total EC for the application server and an even lower percentage for the database server. We tried to find a regression model to fill this gap in the data (see Appendix), but were unable to create an accurate model at the process level. However, our method successfully identified changes in EC at process level. Any differences found in the measurements between releases are considered to be caused by at least one of these changes and as such should be further investigated in further experimentation. Ideally, aspects of the energy profile can be related to the individual software elements in order to find quantifiable possible explanations for any changes in the EC.

With respect to the second main research question (RQ2) we identified that the added value of EC measurements is on both operational and strategic level. On operational level, such measurements provide a new technique to

identify inefficiencies in software. On a more strategic level, the SPO is able to increase the success of a software product by planning its evolution also in terms of energy efficiency. In general, putting energy efficiency on the software roadmap is expected to produce an improvement in the overall quality of the product, also in relation to other quality attributes. However, in order to exploit the added value of software energy efficiency to its fullest, the team must be aware of its potential.

In future work, we will further focus on the development phase i.e. to provide software developers with direct EC feedback during development. The insights we gained from the follow-up interview allow us to understand what form of feedback is more suitable and helpful for the team. A related direction for future research is to further investigate the metrics that characterize a software product in terms of energy efficiency. Finally, an accurate attribution of EC to specific software elements requires further research. Current tools and techniques such as regression models, while promising, still suffer from many limitations. Ideally, such models should be extended to also include (e.g.) OS-level processes, or better, to accurately separate the EC of these processes from the SEC. More complex data analysis and machine learning techniques have to be investigated.

Our research contributes towards levering research on the EC of software products to a new level, instead of maintaining focused on the 'low-hanging fruits' as found with the interviews. The results show that software energy efficiency is a pioneering field, that still requires a large amount of empirical evidence before providing solid foundations and principles. For this reason, we strongly encourage other researchers to contribute to this field of research, and we make our data available (see Sect. 6.5) for reproduction and replication of our results, to stimulate the community towards new and interesting findings.

6.9 Appendix

Regression Model to Predict the EC of Software Elements

The percentages of EC that JM reports on process level (on average 61.9% for the application server and 69.3% for the database server) indicate that we are still unable to explain a relatively large amount of the energy overhead of software execution. We initially attributed this to a lack of accuracy of JM: the tool is based upon a linear model that takes into account only a limited amount of hardware resources [68]. Hence, we hypothesized that this energy estimation gap could be due to unaccounted resources in the linear model. For this reason, we built a special–purpose linear model, trained by using performance data and the energy consumption measured by the WUP. In this Section we briefly explain the techniques we adopted and our results. In order to train the models, we used the values from a single experiment execution (see Sect. 6.4) as training set. We then use the remaining executions as test sets to evaluate the performance of our models.

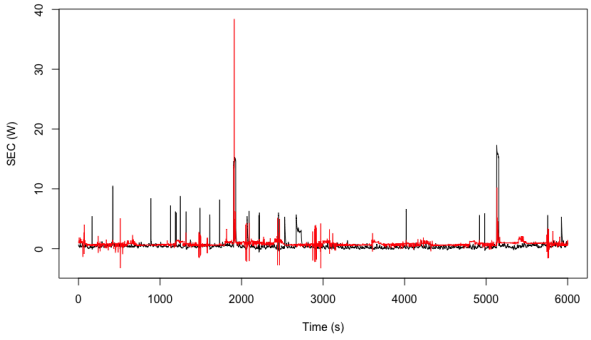
Our first version of the regression model was obtained by means of multiple generalized linear models selection [19]. We selected the SEC as a response and through a genetic algorithm we generated multiple instances of linear models, using resource usage data as predictors (specifically, the used predictors were CPU time, IO bytes/sec, memory private bytes and working set: the other predictors (Sect. 6.3.1) were excluded due to collinearity). We fitted both level-1 and level-2 models, by analyzing interactions between the predictors. The models generated with this method were characterized by strong overfitting to the specific machine. Fig. 6.7 shows why the model performs poorly: if fitted to the application server data, it performs well when predicting data from the same machine (Fig. 6.7a), but is unable to predict the database server data with reasonable error (Fig. 6.7b).

By performing some diagnostics on the model, we found out that there were a number of observations with high leverage (i.e. significantly influencing the regression coefficients). In addition, the data was also characterized by a high number of outliers. Hence, we opted for robust regression [3], a form of regression analysis that gives more reliable results in such conditions. Indeed, such method performed significantly better: in Fig. 6.8 you can see a comparison of the performance of the two models. A large systematic error is still present, but in terms of Mean Absolute Percentage Error (MAPE) we were able to improve from of 12.6 to 2.6.

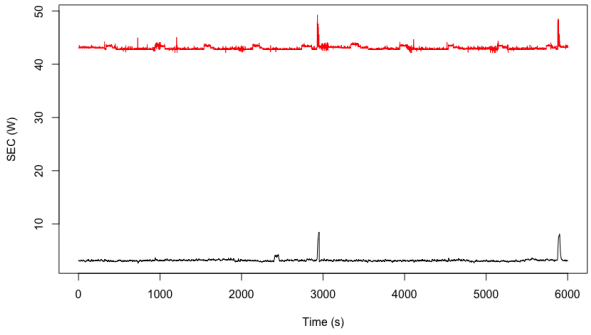
However, the fitted regression models exhibit negative coefficients. If we assume that a software process will use a positive and finite share of the system

resources, this is probably not realistic. Hence, we adopted penalized linear regression [143], a regression technique that enables to specify constraints for the model features. This was done in order to enforce a positive value for the predictors.

At machine-level predictions, i.e., trying to predict the total system EC, the model obtained through penalized regression outperforms JM (see Fig. 6.5). Our model has a MAPE of 0.005 when compared to WUP measurements,

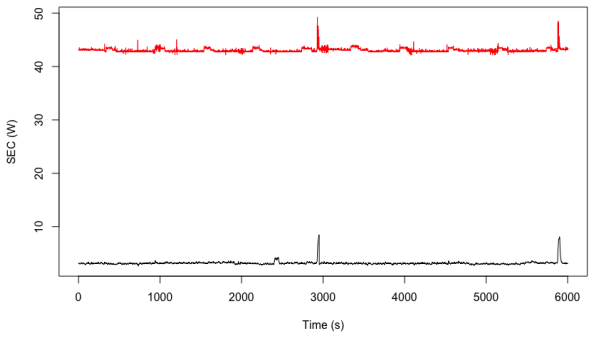


(a) Application server predict set.

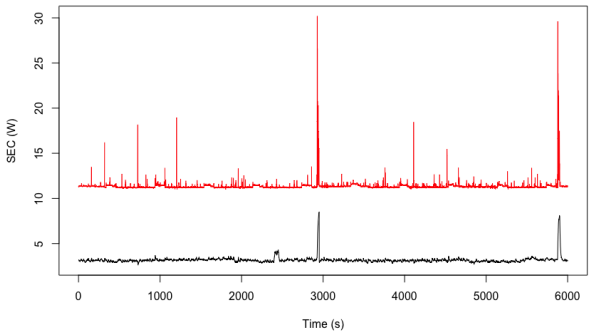


(b) Database server predict set.

Figure 6.7: Overfitting of level 2 linear regression model trained on Application server data.



(a) Non-robust model.



(b) Robust model.

Figure 6.8: Comparison of non-robust and robust regression models, trained on Application server data and predicting Database server data.

as opposed to the 0.08 of JM. Given these promising results, we used the same model to predict the impact at process-level. The prediction values were obtained by using the resource usage data of the single processes (as measured by Perfmon, see Sect. 6.3.1) as an input to the model. The intercept coefficient of the model was subtracted from the prediction, to remove the machine-dependant idle power estimation. Through this technique, we are able to obtain a realistic estimation of the energy impact for each process (in

Table 6.8: Example comparison of the energy estimation for Joulemeter and our special-purpose linear model for the Oracle process.

Total SEC (Wh)	**Joulemeter (Wh)**	**Model (Wh)**
8.239	5.618 (68.18%)	5.724 (69.47%)

Tbl. 6.8 an example for an execution of Oracle is shown).

If we aggregate the estimation obtained using our model for all the processes running in our application, however, we obtain very similar percentages to those computed via JM. Given this validation, we must conclude that JM provides a fairly accurate estimation of the energy consumption of specific processes.

Hence, a relatively high percentage of energy consumption cannot be attributed to specific processes. This is a strong indication that other factors are playing a role. Examples might be OS-level processes and system calls that the profiler is unable to detect as separate processes. Further work must be done to reliably attribute EC to specific software elements.

7

Awakening Awareness on Energy Consumption in Software Engineering

Software producing organizations have the ability to address the energy impact of their ICT solutions during the development process. However, while industry is convinced of the energy impact of hardware, the role of software has mostly been acknowledged by researchers in software engineering. Strengthened by the limited practical knowledge to reduce the energy consumption, organizations have less control over the energy impact of their products and lose the contribution of software towards energy related strategies. Consequently, industry risks not being able to meet customer requirements or even fulfill corporate sustainability goals.

In this paper we perform an exploratory case study on how to create and maintain awareness on an energy consumption perspective for software among stakeholders involved with the development of software products. During the study, we followed the development process of two commercial software products and provided direct feedback to the stakeholders on the effects of their development efforts, specifically concerning energy consumption and performance, using an energy dashboard. Multiple awareness measurements allowed us to keep track of changes over time on specific aspects affecting software development. Our results show that, despite a mixed sentiment towards the dashboard, changed awareness has triggered discussion on the energy consumption of software.

This work was originally published as:

E. Jagroep, J. Broekman, J. M. E. M. van der Werf, S. Brinkkemper, P. Lago, L. Blom, and R. van Vliet. Awakening awareness on energy consumption in software engineering. In *Proceedings of the 39th International Conference on Software Engineering: Software Engineering in Society Track*, pages 76–85. IEEE Press, 2017

7.1 Introduction

Software is acknowledged by academia to be a key driver for the energy consumption of ICT solutions [51,82]. Software and energy, i.e., the area of green software [82], can be related to the environmental dimension of sustainability, which is generally defined as 'the capacity to endure' [83]. One way to address energy consumption in software, i.e. SEC, is through its software architecture [6]. By applying an ECP [60], the SEC can be addressed in the early stages of software engineering. In this way, sustainability can be considered as a Quality Attribute [54], and consequently, it can be included in trade-off analysis and architecture evaluation [69]. However, a necessary prerequisite is that the stakeholders involved in the development are aware of the energy consumed by their software and its causes [61].

Different studies have surfaced investigating means to measure SEC [63, 109] compare releases of software products on their energy consuming characteristics [52, 61], and provide insight to those involved in software development [105]. Another study [112] reports limited knowledge of energy efficiency and lack of knowledge of practices to reduce the SEC among software developers. Additionally, the study reports uncertainty about how software consumes energy, which seems to contrast the findings of [118], that practitioners are aware of energy consumption problems. From these studies, it becomes clear that a gap remains with respect to concrete coding guidelines and practices reaching their target audience [121]. In other words, we see that industry is not yet able to adopt solutions provided by research.

With the growing attention for corporate social responsibility, SPOs [150], such as independent software vendors and open-source foundations, risk not being able to fulfill their corporate sustainability goals and meet (customer) sustainability requirements with their software [61]. Awareness of their software products' energy consumption potentially helps SPOs to mitigate this risk.

In this paper, we present the findings of a multiple-case study on creating and maintaining awareness of the energy consumption of software products among stakeholders during development, as it has the greatest impact [102]. We first introduce an energy dashboard for the ECP based on earlier research [62], which provides insight in the energy consumption between consecutive sprints. For two commercial software product, we then measure how the awareness changed over several sprints. To measure the development of the awareness we create a specialized awareness model for SEC, inspired by the work of [92] and that served as a basis for the surveys held with the stake-

holders after each sprint.

The main contribution of this paper is twofold:

- **Awareness model for SEC:** The model we apply allows us to capture the awareness of stakeholders involved with product development. The scores we obtain allow for analysis on different constructs, which provide insight into the areas that require extra attention in relation to SEC and ECP.
- **Development of awareness over sprints:** Although the stakeholders are better aware of the ECP of their software products, the results show and shortcomings in current state-of-the-art tactics and best-practices to improve software energy efficiency. Furthermore, to maintain awareness, SEC should be supported throughout the whole organization.

The paper is structured as follows: Section 7.2 presents our research questions followed by a discussion of related work (Section 7.3) and the design of our empirical study (Section 7.4). In Sect. 7.5 we present the results of our study which are discussed in Section 7.6, followed by the threats to validity (Section 7.7). Concluding remarks and an outline for future work are provided in Section 7.8.

7.2 Research Questions

To address the issue presented above, our study was structured around the following **main research question (RQ)**:

RQ: *How to create and maintain awareness of the energy consumption perspective in software product development?*

Following 'A Dictionary of Psychology'[1] awareness is part of being 'conscious' that is: "giving due weight to something". In our context, being aware means weighted decisions can be made with respect to SEC, without implying an improvement or deterioration of the SEC. For an SPO, systematically addressing the SEC in the software design requires that awareness is maintained among the stakeholders involved with the software.

As a prerequisite to answer our RQ we need to be able to determine awareness among stakeholders, which leads to our first research sub-question:

[1] http://www.oxfordreference.com/view/10.1093/acref/9780199534067.001.0001/acref-9780199534067

SQ1: *How can we measure awareness on the topic of SEC?*

For this sub-question (SQ1) we look into those aspects that determine awareness and operationalize these in the software engineering context.

Second, to actually create awareness, we require a means to stimulate the stakeholders to actively think about SEC that can be incorporated in the development process. Resulting in the second and third sub-questions:

SQ2: *What stimulus can be used to trigger SEC awareness?*

SQ3: *How can we incorporate SEC in the development process?*

The second sub-question (SQ2) is set to investigate what information is required by the stakeholders and in which form the information should be presented. After determining what stimulus is required, we answer the final sub-question (SQ3) by investigating how the stimulus can be included in the development process, with minimal impact on the process itself, and enable stakeholders to structurally consider the SEC of their software products.

7.3 Background

Green Software: Green software generally refers to energy efficient software. To create green software, the sustainability aspect should be addressed during the early development stages of a product and constantly monitored during the software product lifecycle [102]. For example, applying green practices [121] and selecting the right 'Collection types' [48] potentially reduce the energy consumption up to respectively 25% and 300%. If we position sustainability as a software quality property [60, 83] we can go back further in the lifecycle, i.e. the design phase, where the software architecture allows for precluding qualitative traits of the software [6]. Similar to technical debt [149], early awareness of green software could save a significant amount of costs compared to refactoring the software at a later stage.

Energy Profiling: A recurring theme with green software is monitoring the SEC to support engineers with understanding their code and its energy impact [91]. However, unlike example the mobile domain [113], monitoring is more difficult with software products [51, 61] and different approaches exist to estimate the SEC. Most prominent are power models that profile the software based on resource usage, e.g. [63, 109], but new approaches are surfacing using big data principles [28]. A modeling specialist, for example, could help in building predictive profiling models [72]. In practice, performance is often used

as a proxy for energy efficiency; i.e. less resource usage equals to less energy consumption. However, energy consumption and performance are not always positively correlated [20, 125, 144] and should thus be considered separately.

SEC Awareness: A lack of knowledge on SEC [112] does not imply green software should be neglected altogether. Knowing the difference in energy consumption between releases, e.g. [61], could help practitioners determine whether the energy consumption is reasonable given the work being performed [91]. Being aware of the topic, which fits the 'service awareness' problem area [81], could affect the believes of software engineers that are bound to affect their practice [30]. Following the economic dimension of sustainability [83], software-oriented data analytics [72] provides actionable insights to achieve business goals using green software practices.

Creating awareness requires a stimulus that triggers stakeholders to actively make conscious decisions with respect to SEC. Examples like the 'Eco' programming model [156], RUS [62] and a graphical energy monitoring interface [105] have shown positive effects in this regard. However, creating awareness does not automatically imply a reduced energy consumption. A conscious decision could be to favor a specific quality aspect above sustainability, e.g. color usage to improve the usability [118]. In this case a conscious design trade-off is made [60].

7.4 Research Design

To answer our main research question we have conducted an embedded multiple-case study [152] where we measure SEC awareness and stimulus acceptance (i.e. multiple units of analysis) with two cases in a company developing commercial software products. In this section we describe the design of our study following the guidelines provided in [66, 74, 127, 148, 152].

7.4.1 Energy Consumption and Performance Measurements

To perform energy consumption measurements we applied a software-based approach using Microsoft JM, similar to the approach applied in our earlier research [60, 61]. After calibration, JM allows us estimate the total energy consumed by a system at run time based on the computational resources used with a one second interval between measurements. To determine the SEC we subtract the idle energy consumption of a system from the energy consumed

while running the software, both obtained using JM. The difference is in the energy consumed on the account of the software product, i.e. its SEC.

A prerequisite for valid measurement is to let the hosting server cool down to a stable state (i.e. a state with no active processes without direct instructions from the user [61]) after a reboot. The cooldown time has to be determined for each individual server used in the study.

In turn, the performance of the hardware components was measured using Windows Performance Monitor, a standard tool with the Windows operating system. Developers are in general more familiar with performance aspects, e.g. CPU utilization, and have experience with addressing the performance aspects of a system. To provide insight in the resource usage by the software we calculate a Resource Utilization Score, or RUS [62] – a score for the software based on the relevant performance aspects according to the stakeholders.

SEC and Performance Measurements Protocol: To ensure the validity of the SEC measurements, a simple protocol was followed to perform each run:

1. Restart the environment.
2. Close unnecessary applications.
3. Start performance measurements and setup JM.
4. Remain idle for the duration of the cooldown time.
5. Start JM measurements.
6. Start load test and wait for test to finish.
7. Collect and check data.

Starting the performance measurements upfront (step (3)), allowed us to check whether the system was indeed in a stable state during a run. If this was not the case (checked in step (7)), the run was excluded from further analysis.

7.4.2 On measuring awareness on SEC

In designing the study to answer SQ1, we found that 'awareness' cannot be measured directly due to the inability to quantify the concept [101]. Hence, as a means of indirect measurement, we used the model presented by Matthies [92] meant to transform environmental-detrimental habits into pro-environmental habits, and specialized it to capture changes in awareness on SEC. The resulting model (Fig. 7.1) consists of four stages (norm activation, motivation,

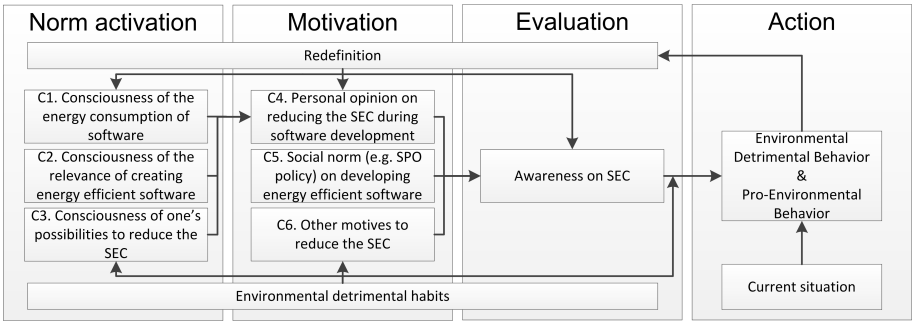


Figure 7.1: Model on transforming environmental behavior with the constructs translated to our study, after Matthies [92].

evaluation and action) and six constructs (C1 through C6) that directly or indirectly affect the weighting of moral, social and other types of costs and benefits, potentially resulting in a behavioral change.

Following the definition of awareness provided in Sect. 7.2, for stage Evaluation we relabeled the activity of 'weighting relevant aspects' into 'awareness on SEC' and defined a survey to measure it. We used the specialized constructs as basis for defining the survey questions[2]. In the Action stage of the model we determine whether behavior has indeed changed.

As illustrated in Tbl. 7.1, we formulated 14 statements based on the constructs. To this aim, we carried out brainstorming sessions with experts in the field of green software engineering (including the authors) combined with related works like [32,112]. While the statements are related to personal experience, it has been recognized that such personal experience has the strongest influence on beliefs with respect to specific topics related to software engineering [30]. (See also Sect. 7.7 for a discussion of the related threats to validity.) Each individual statement can be answered with an option ranging from strongly disagree (-2), disagree (-1), neutral (0), agree (1) to strongly agree (2) – where the number behind every option represents our internal coding scheme.

Next to measuring SEC awareness, the survey also includes statements for measuring the acceptance of the awareness stimulus (i.e. the dashboard, see Sect. 7.4.3). To this aim, we used as basis constructs inspired by the Unified Theory of Acceptance and Use of Technology (UTAUT) [145,147], and included the relevant associated statements. Both constructs and associated

[2]The complete survey is available online at `http://tinyurl.com/gvpf3hg`.

statements are shown in Fig. 7.6.

The first version of the survey was reviewed by ten practitioners in our network (software engineers and IT managers) with varying years of experience. The main feedback was related to the formulation of the statements. However, several warnings were also issued with respect to the length of the survey. As we intended to present the survey to stakeholders multiple times (see Sect. 7.4.4), we have split the survey in three separate sections:

- Survey A, evaluating the awareness with a generic and broad scope (C1 and C4-C6).
- Survey B, evaluating the awareness related to a specific software release (C2 and C3).
- Survey C, focused on the acceptance of the awareness stimulus (in our case, the dashboard).

Each survey section is presented to the participants only when relevant, as exemplified in Fig. 7.3. This allowed us throughout the study to distribute the effort required to the participants while collecting all necessary data.

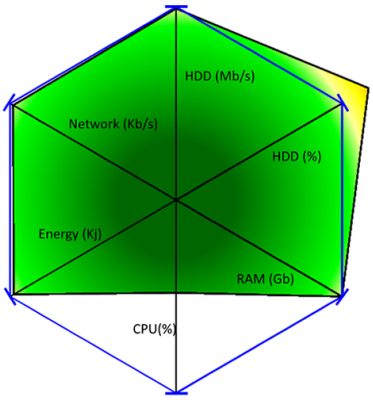
7.4.3 The Stimulus to Trigger SEC Awareness

To answer SQ2, we used as input the work done in [62] and [105], and combined them resulting in an energy dashboard, as shown in Fig. 7.2. This includes a radar chart and an overview of the exact measurements. Data for the dashboard is obtained by following the provided measurement protocol (Sect. 7.4.1).

In particular, the *radar chart* graphically shows the RUS [62] and is meant to enhance the communication of the measurements to stakeholders and highlight key findings. As adopted in [105], visual information coveys familiar indicators and gauges, e.g. percentages and bar charts, that 'non-energy experts' can easily grasp. Regarding the *individual measurements*, the energy dashboard (lower part of Fig. 7.2) includes the *delta* between two releases, which indicates the change with respect to the usage of a specific resource [62]. Calculating deltas requires labeling one release as benchmark and positioning the measurements of a different release in light of this benchmark. In the example of Fig. 7.2, the results show a decrease in energy consumption with 2.15% of the new release (black line) compared to the previous release (blue line), whereas the color of the surface (green, to yellow to red) represent the intensity of the decrease (green) or increase (red) in resource usage. With multiple

Table 7.1: The statements per construct used for the awareness measurements, with its partitioning into surveys A and B.

C1 (A)	1	I want to determine the energy consumption of our software development environment (e.g. by calculating the energy consumed by (test)servers, laptops and other resources).
	2	Before this project, I wondered multiple times about the energy consumption of our software development environment.
C2 (B)	3	I expect that software has a large influence on the energy usage.
	4	I would like to know the energy consumption of our software product.
C3 (B)	5	If I had more time to work on the code, I would be able to reduce the energy consumption.
	6	The applied techniques (programming language, design patterns, etc.) to realize the software product, allow for the reduction of energy consumption.
	7	It is possible to make a trade-off between our current non-functional requirements (e.g. performance) and the energy consumption of our software.
C4 (A)	8	Addressing the energy consumption of our software should gain more attention.
	9	I would like to reduce the energy consumption, if I am allowed to spend time on it.
C5 (A)	10	The energy consumption of our software product is discussed during (in)formal meetings.
	11	If other teams reduce the energy consumption of their software, I would attempt it too.
	12	Reducing the energy consumption would be a benefit to the organization and the customer.
C6 (A)	13	Code optimizations to improve non-functional requirements should be acknowledged and included in our backlog.
	14	The benefits of rewriting the code to reduce the energy exceed the costs.



Version	CPU (%)	RAM (Gb)	HDD (%)	HDD (Mb/s)	Network (Kb/s)	Energy (Kj)	Δ RUS
5.3.12	14.10	1.80	13.64	2.24	89.36	27.03	6424
5.3.13	6.77	1.80	15.81	2.22	87.76	26.45	5543
Delta	-51.99%	+0.00%	+15.91%	-0.89%	-1.79%	-2.15%	-13.71%

Figure 7.2: The energy dashboard: example as presented to the stakeholders of case RS.

consecutive releases, each new release is set as the benchmark for the next, thereby summarizing the effects of the latter release.

7.4.4 On incorporating the Stimulus in Development Process

To answer SQ3, we followed the advice of Devanbu et al. [30] to take practitioners' beliefs into account in designing an experiment. A short investigation with multiple development teams and the experience of the authors learned that Scrum was the common development method in the company, and that software-related dashboards were frequently consulted at the end of each sprint. At this point in time the team typically reflects on the past sprint and decides on corrections, e.g. (re-)prioritize requirements, if required. Accordingly, it was natural to present our dashboard shortly after each sprint.

Fig. 7.3 illustrates the holistic organization of our multiple-case study. The survey and energy dashboard have been incorporated in the Scrum development process used by the involved company. **At the start**, survey A and B are presented to determine the initial awareness of the stakeholders, followed

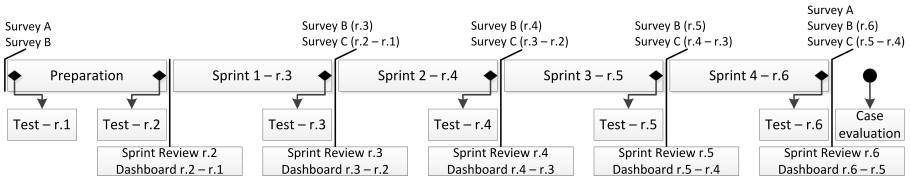


Figure 7.3: The case study organization including sprints, test periods, sprint reviews, energy dashboard presentations and surveys.

by the **preparation** phase where the first two releases of a software product (r.1 and r.2) are tested. The preparation phase ends with the sprint review for release 2 where the first dashboard (dashboard r.2 - r.1) is presented. The order of releases with the dashboard indicates that, e.g., release 2 is compared to release 1, and as such release 1 served as the benchmark for calculating the delta's.

After the preparation phase, we repeat the following procedure for each consecutive sprint. At the end of a sprint, while the new release is being tested by the team, the load test can be performed to collect data for the energy dashboard. In the accompanying sprint review the stakeholders fill in survey B and C, looking back at the past sprint and dashboard presented at the previous sprint review. Afterwards the new dashboard is presented. In the last iteration, survey A, B and C are presented, followed by the final dashboard. The **case evaluation** with the team closes a case and helps to determine whether behavior has changed (i.e. the Action stage of the model in Fig. 7.1).

7.4.5 Case Selection

The cases included in our study were acquired by contacting product managers of multiple software products within the case company; a large international SPO. While we did not have specific criteria for the software products themselves, we did formulate the following inclusion criteria related to the processes, technology, and team:

- An agile development method is implemented including reviews after each development iteration.
- At least six releases are available, four of which will be developed during the case study.

- The software product can be deployed in a production-like environment.
- Automated load tests are available.
- Commitment to participate for the duration of at least five sprints including filling in multiple surveys.

With respect to the load test, we are striving for usage scenarios as these are used most often when practitioners evaluate energy usage [91]. In the end we identified two cases that met all inclusion criteria. The details of these cases are provided below.

Case 1: DG is a commercial software product used by over 300 (mostly governmental) organizations in the Netherlands, counting more than 900 end-users, and generating more than 30 million documents on an annual basis. Although DG has been used in earlier research [60,61], meanwhile the product has moved to a new development team. This assures us of not having biased results with respect to awareness. We identified four developers and one tester as the stakeholders involved with developing DG and commitment was given for releases 8.0.6, 8.0.7, 8.0.7.1, 8.0.8, 8.0.9 and 9.0.0 being developed during the study. The duration of a sprint was three weeks, which meant the case study would last for fifteen weeks.

DG was deployed in a production-like environment encompassing an application and database server, both with a cooldown time of 20 minutes. The load test was designed to simulate a user generating 258 complex documents. Within the limited time for testing we were aiming for at least 40 measurements per DG release. Discussing the measurements with our DG contact learned that, apart from energy consumption, the CPU utilization, memory usage, hard disk usage, network usage and execution time were of interest.

Case 2: Retail System (RS) is a commercial software product for retail stores, e.g. supermarkets, to process the customer transactions of their points of sales, e.g. cash registers. With a customer base of 110 customers in 30 countries, counting more than 20.000 stores and 75.000 points of sales, RS processes more than 20 billion transactions on an annual basis. For RS, the 23 stakeholders participating in our study were located in Belgium and Romania: 16 developers, 1 database developer, 2 technical analysts, 2 testers, 1 software architect and 1 product specialist. Commitment was given for releases 3.11, 3.12, 3.13, 3.14, 3.15 and 3.15.1. Each sprint takes three weeks, resulting in a total case duration of fifteen weeks.

The RS software was deployed on a single server, with a cooldown time of 15 minutes, which corresponds to the most simple production-like setting. Despite its simplicity, our load test was designed to simulate the transactions

for up to 100 points of sales, i.e. multiple supermarkets, in a fixed time-span of three hours. Considering the limited time window, we aimed to perform at least 10 runs for each RS release. With respect to the measurements, our RS contact pointed out the CPU utilization, memory usage, hard disk usage and network usage should be measured.

7.4.6 Data Analysis Procedure

The number of parcticipants per case (n=5 for DG, and n=22 for RS), led us to follow a qualitative approach to analyze our data [94] using awareness (survey A and B) and acceptance (survey C) scores calculated using the survey results. We calculate scores by adding up individual scores into a statement score, adding up the statement scores to score a construct, and finally adding up the construct scores resulting in a score for awareness and acceptance. Following the coding scheme of our data, i.e. from -2 to +2, a negative score indicates disagreement with the statements and vice versa. To accurately show changes over time, we only include the results of participants that filled in survey A at the start and end of the study and missed at most one combination of survey B and C.

7.5 Results

In this section we report on the execution and the results of our multi-case study, for both cases. Detailed figures including scores per statement, like Fig. 7.6, are provided online[3].

7.5.1 Study Execution

With DG we deviated from the case description by excluding release 9.0.0 from the study. Even with support of the team, successful deployment was not possible and, consequently, release 8.0.9 was the final release included in the study. Due to time constraints and the planning of the DG team we could not compensate for this exclusion, resulting in a data gap for sprint review r.5. With the included releases we managed to perform the required 40 valid runs and collect all survey data with the exception of survey B (r.3) and survey C (r.2 - r.1) of two team members due to holidays. Despite the missing data, following our analysis procedure, the input of all five team members was included in the score calculations.

[3]Scores per statement available at `http://tinyurl.com/gvpf3hg`.

With RS, given the geographical distribution, an online survey tool was used to conduct the surveys, with the following survey response rates: 96% (SR-r.1), 65% (SR-r.3), 61% (SR-r.4), 57% (SR-r.5) and 65% (SR-r.6). Following the data analysis procedure, the RS scores were calculated based on the survey results of twelve team members. Additionally, we managed to perform nine runs per release, instead of the required ten, due to sharing of the test resources and issues with Performance Monitor. The available resources for load testing were also used for production testing which simply had a higher priority. Investigation into Performance Monitor pointed out data is automatically deleted when a specific threshold is reached for the available hard disk space. An issue we solved with a simple reconfiguration. Despite missing one run we still obtained sufficient data per release to produce valid energy dashboards.

The evaluation for both cases took place approximately two weeks after 'sprint 4' with representatives of the team.

7.5.2 Survey A and B

The final score on awareness for the DG team decreased from +23 to +3 at the end of the case study. On construct level the results (Fig. 7.4) show an increase with respect to consciousness of the energy consumption of software (C1) and the social norm on developing energy efficient software (C5). On the other hand the stakeholders indicate that the relevance of creating energy efficient software (C2) has decreased together with the consciousness of one's possibilities to reduce the SEC (C3), the personal opinion on reducing the SEC (C4) and the other motives to reduce the SEC (C6). The scores on C2 and C3 (collected with survey B), resemble the patterns of the Gartner Hype Cycle with a change from strongly positive to strongly negative and back to a more neutral score, i.e. zero.

The awareness scores for RS changed from +4 to -16 during the case study, and on construct level (Fig. 7.5) resemble the trends found with DG. C2 and C3 again show the Gartner Hype Cycle pattern, C4 and C6 decrease over time with C4 even becoming negative and C5 changed from negative to a positive score. The only discrepancy is with C1 where RS stakeholders became more negative.

7.5.3 Survey C

With respect to the acceptance of the energy dashboard, in general the statement scores of the DG stakeholders (Fig. 7.6) are increasingly negative over sprint reviews – resulting in a change of the total score from 14 to -24. The

only exception is with the effort expectancy (EE) construct, where the final score of +5 (obtained by adding up the EE scores for SR r.6) implies that the dashboard is considered user friendly. The attitude towards technology (AT) follows in a second place with a score moving from +6 to 0. Scores concerning the performance expectancy (PE), social influence (SI) and behavioral intention (BI) become increasingly negative, namely -3 to -5, -2 to -13 and +8 to -11 respectively.

For RS (Fig. 7.7), the total dashboard acceptance score shifts from -79 to -118 and we again only find a positive final score (+6) with the EE construct. The other construct scores imply a consistent negative acceptance of the energy dashboard resulting in final scores of -26 (AT), -35 (BI), -18 (PE) and -45 (SI).

7.6 Discussion

In this section we discuss the results in light of our research sub-questions. To this aim, we follow a method similar to sentiment analysis [111].

7.6.1 SQ1: Measuring awareness on SEC

With respect to C1, the sentiment switched from negative (-2) to positive (+5) with DG (Fig. 7.4), indicating an increased willingness to determine the energy consumption in relation to software, and became more negative (-13 to -18) with RS (Fig. 7.5). The case evaluations learned that both teams were more aware of the SEC and would keep the topic in mind during development, hence implying a change in behavior. However, the focus remains on software functionality in line with product strategy and customer demand.

Software architecture: With C2 and C3, stakeholders from both cases increasingly want to know the SEC of their software product, which seems in contrast with the RS scores on C1. Investigating the individual statements for RS, we found a potential explanation in the negative score concerning the influence of software on energy consumption (i.e. -15 to -6). In contrast with DG, where the case ends with a neutral score on this statement. Rephrased, while there is an interest in the SEC, with RS there is doubt on the impact of the software on energy consumption. One RS developer stated from previous experiences with mobile software that the effectiveness of any effort is clouded by the different layers of the software stack (e.g. operating system and middleware). While the move towards a more neutral sentiment indicates a greater acknowledgement of the role of software, hardware and other aspects are still believed to have a greater impact on the energy consumption.

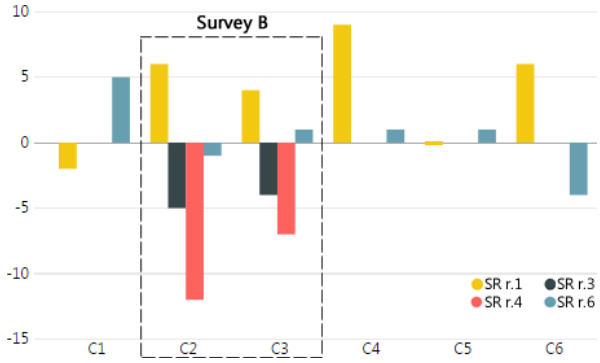


Figure 7.4: DG case: The awareness scores per construct over sprint reviews.

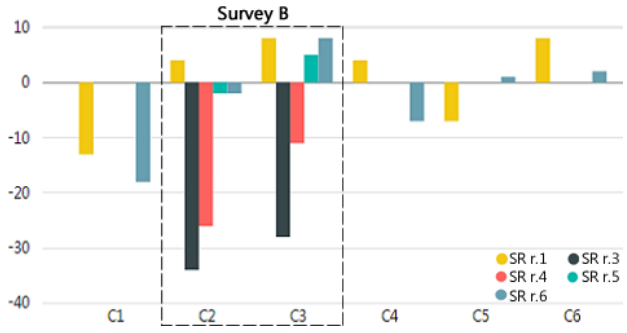


Figure 7.5: RS case: The awareness scores per construct over sprint reviews.

When asked, RS stakeholders appointed software architects as the main actors in relation to SEC along with hardware manufacturers, and a declining role for software developers. In contrast, DG stakeholders consistently appoint software developers and architects as the main actors. These findings provide a plausible reason for the sentiment on C3 and remained negative sentiment on specific statements for this construct. For example, given the large group of developers involved, the RS stakeholders maintain a negative sentiment towards the statement on spending more time to reduce the SEC. However, driving the positive C3 score, RS stakeholders acknowledge the importance of the applied techniques (+6) and trade-offs with non-functional requirements (+4).

The results for C6 also confirm the focus on software architecture. For

example, stakeholders reconsidered the statement on including code optimizations to reduce the SEC in the backlog (+6 to -3 with DG, and +11 to +6 for RS). A possible cause is that the stakeholders increasingly positioned SEC in relation to non-functional requirements of the software [6], which are typically not included in the backlog. This finding could explain the stakeholders' negative sentiment towards the statement that the benefits of rewriting the code to reduce the SEC exceed the costs.

Learning curve: The statements of C3, on the extent to which the applied techniques and making trade-offs with non-functional requirements can contribute to reducing the SEC, suggest a learning curve took place. Starting with positive scores, stakeholders from both cases appear to underestimate the complexity involved with reducing the SEC. After the first dashboard presentation, the scores became negative with stakeholders realizing the complexity involved. The movement towards a neutral or even positive score follows after stakeholders better comprehend the SEC. Interesting to note is that DG stakeholders indicate the SEC is not discussed during (in)formal meetings (C5), whereas a lively discussion on SEC took place during multiple sprint reviews. The RS scores move from -17 to -8 on the latter statement, and the evaluation learned that SEC is discussed by specific groups, e.g. lead developers and architects, within the team.

SEC on the team agenda: Given the developments with C1, C2 and C3, a decline with C4 was expected. With DG a slight positive sentiment remains as the stakeholders are willing to address the SEC when allowed to spend time on this aspect. However, in line the discussion so far, RS becomes negative (+4 to -7) on this statement.

Stakeholders from both cases indicated extra attention towards SEC is only required when reduced energy consumption becomes a requirement from the customer or a strategic driver within the organization. Here we also find a difference between cases; the DG team could obtain a strategic advantage by being delivering a more 'sustainable' product, whereas RS did not identify this advantage within their markets. A finding that fits the survey results and potentially shows a difference between the markets involved in the study.

Willingness: The statement on seeing other teams address the SEC (C5) moves from a negative (-2) to a positive (+2) sentiment with DG and increases from +1 to +3 with RS. Additionally, both cases acknowledged that reducing the SEC benefits the organization and the customer, however customer demand and market trends lead in determining the future of the product. The results indicate a willingness to address the SEC when sustainability goals are formulated and clear benefits can be identified. However, even without strategic goals an SPO could benefit from promoting energy efficient software

development. An overall reduction of the total cost of ownership for a product allows an SPO to, e.g., maintain a more competitive pricing model.

7.6.2 SQ2: Stimulus to trigger SEC awareness

Solely based on the acceptance results (cf. Fig. 7.6 and Fig. 7.7), our conclusion would be that the dashboard does not appear the right means to trigger awareness with respect to SEC: while the dashboard is easy to use, shown by the effort expectancy (EE), it does not stimulate to perform target behavior (AT) or to positively influence technology usage (BI).

Confronting: Interpreting the scores in light of the awareness scores, we can only partially explain the negative sentiment found in the scores. If stakeholders do not know how to reduce the SEC or become aware of the limitations imposed by, e.g., the technology used, presenting the dashboard could frustrate and result in a negative sentiment towards the dashboard. To this end, the dashboard does trigger awareness, be it in a confronting manner. This suggests that awakening awareness would be effective when combined with software engineering knowledge on how to decrease the SEC.

Despite potentially being confronting, 'Using the dashboard is a good idea' (AT) is one of few positive statements for RS. Also DG stakeholders indicate they like working with the dashboard and that the dashboard makes work more interesting. However, DG stakeholders did not find using the dashboard a good idea. While we cannot explain the unexpected decline with the latter, the results potentially indicate the dashboard to contain useful elements in this context.

Presentation frequency: Apart from being confronting, the frequency of presenting the dashboard, i.e. after every sprint, could also have contributed to the negative sentiment. For example, while the statement concerning the

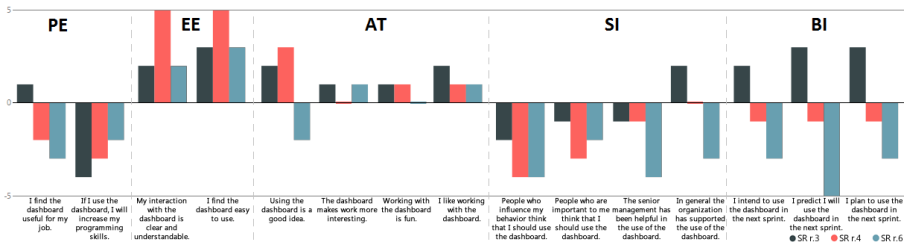


Figure 7.6: DG case: The scores on the individual acceptance statements (survey C), grouped per construct.

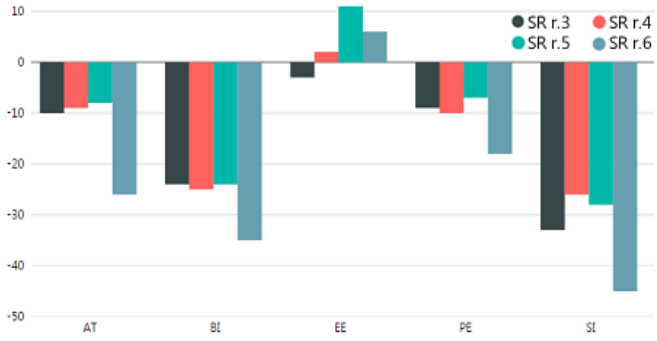


Figure 7.7: RS case: The acceptance scores per construct over sprint reviews.

usefulness for the job starts positive in both cases, consecutive measurements indicate a negative sentiment. During the evaluation, stakeholders indicated that the SEC should not be inspected after every sprint, unless there is a concrete motive to do so. Instead, similar to other quality attributes, a periodical evaluation of the SEC is considered to be sufficient.

Target audience: with DG a contrasting finding is found with the PE statements (Fig. 7.6): while the dashboard is considered increasingly less useful for the job, stakeholders disagree less with using the dashboard as a means to increase programming skills. The diversity of roles included in the study, i.e. also non-developers, potentially explains this contrast, which would imply the dashboard in its current form is role-specific and should be tailored for its target audience. Combined with the SI results, indicating that stakeholders find little support with others in using the energy dashboard, an SPO should carefully consider what information to present on a dashboard and to whom.

7.6.3 SQ3: incorporating SEC in the development process

The survey results indicate that the stakeholders are able to make weighted decisions with respect to SEC, i.e. are aware of the topic. However, for an SPO the ability to fulfill corporate social responsibility goals with software products requires that awareness is maintained. Based on the results, we can provide the following related recommendations:

Formulate sustainability goals: Stakeholders indicated that a reduced energy consumption benefits the organization and the customer. When posi-

tioned as a strategic goal for the organization, stakeholders can justify efforts to address the SEC of their products. Additionally, by embedding SEC in the organization, an SPO can also experience benefits from the social norm that motivate stakeholders (C5). An effect that can be potentially amplified by means of, e.g., gamification.

Set up knowledge bank: The overall impression, confirmed by the results, is that there is limited knowledge on how to actually address the SEC. Additionally, the results indicate stakeholder increasingly becoming aware of the possibilities in relation to SEC, which highlights the importance of having an ECP [60] to guide decision-making. A knowledge bank on this topic, containing e.g., tactics, patterns and best practices, provides concrete guidelines, and potentially makes green software practices more cost effective. Relating the knowledge bank to the views in the ECP helps to make trade-offs on the different aspects related to software design and strengthens the relation with sustainability goals.

Quantify SEC: Despite the limited acceptance, the energy dashboard proved useful to show the effects of development efforts. Quantification makes SEC more concrete, hence enabling informed decision making. Additionally, the energy dashboard can highlight specific achievements and potentially encourage development efforts to further improve the software quality. For example, the reduced CPU utilization by solving a long term bug in RS (see Fig. 7.2) was put in the spotlight using the energy dashboard. When an energy dashboard, or similar stimulus, is to be implemented, an SPO needs to keep the target audience in mind as this could greatly affect acceptance.

7.7 Threats to Validity

This section presents the threats to validity as required by [66, 127, 148]. For our study, the threats to validity can be related to two separate aspects; performing SEC measurements and the case study itself. With respect to the former, we applied the methodology as described in [61] and as such were confronted with the same threats to validity. Specifically the reliability of JM, measurement interval, operating system effects, energy consumption overhead and measurement tooling were of concern, and countered in the same manner [61]. In this section, we focus on the latter aspect, the case study itself.

For the **internal validity**, uncontrolled factors that might affect our results, we acknowledge that stakeholders could have modified their behavior in response to being observed, i.e. the *Hawthorne effect*. Additionally, following the *principal-agent problem*, the motivation of the stakeholders to address

SEC could be affected by their (in)ability to choose the technology being used and the fact that they do not pay the energy bill. To minimize the effects we respectively minimized the intrusiveness of our protocol and focused on the personal experience with our survey statements.

External validity addresses the extent to which the results can be generalized. While our *inclusion criteria* could exclude SPOs from participating in our study, we argue that RS and DG are representative for cases found in the software industry. However, we do acknowledge the limited ability to generalize findings beyond the two specific cases, and the limited *number of participants* should be considered to, at best, provide insight in SEC-related awareness in software engineering. Additionally, further investigation is required into *cultural differences*. Although the cases were separate enough within the case company, thereby not affected by company-wide cultural aspects, we were not able to investigate differences between countries which would require more case studies.

The **construct validity**, the degree to which the measures capture the concepts of interest, is focused around the survey. First, the *definition of awareness* allowed us to create a survey and perform the case study. However, as it is a field on its own, we do not aim to provide a general definition of the term. The resulting *survey statements* are systematically developed and valid proxies for awareness. Other statements might be included depending on the context, such as the software and the organization under study. To minimize this threat we carefully designed our study around the model in Fig. 7.1, and established a chain of evidence that allowed us to relate the individual statements to the six awareness constructs.

Finally, **reliability** considers the extent to which research is dependent on the specific researchers. By describing, in detail, the *protocol* that was followed as well as the forthcoming analysis, we provide openness in the research that is performed. Deviations from the protocol were described as such. With respect to the sentiment analysis being researcher dependent, we support our claims with the data obtained from the study.

7.8 Conclusions

In this paper we present the results of a embedded multi-case study performed with two cases regarding two commercial software products. To provide an answer to our main research question "how to create and maintain awareness of the energy consumption perspective in software product development?", we first investigated three sub-research questions.

To measure awareness (SQ1) we constructed a survey based on six constructs that directly and indirectly affect awareness. Although the survey is specific for our purposes and by no means a generic solution, the survey data allowed us to express awareness using a score. As a stimulus to trigger awareness (SQ2), an energy dashboard was created including those measurements the stakeholders consider relevant in relation to the software product in their daily practice. The survey and dashboard were combined in the overall multi-case study organization, hence incorporating SEC awareness creation and the related impact in the software development process (SQ3).

With respect to creating awareness, we found the stakeholders of both cases to actively discuss the topic during the case study and able to make weighted decisions with respect to SEC. In other words, appropriate stimuli help *awaken SEC awareness*. To *maintain SEC awareness*, our results show that organizational policy is required to support creating green software products strengthened with a knowledge bank to stimulate informed decision making on software design.

In future work, we plan to automate the testing activity in case study organization to further lower the threshold to perform SEC measurements. Also we plan to investigate the SEC with individual development efforts, e.g. commits, to distill guidelines for green software knowledge banks.

Part IV

Concluding the Research

8 Conclusions

The aim of this dissertation has been to integrate energy consumption in software engineering and bring SPOs in control of this qualitative aspect of their software products. Being in control enables SPOs to deliver green software products whilst still meeting other (quality) requirements for their software. Our research was formalized in the following main research question:

MRQ - ***How can software producing organizations be in control of the energy consumption of their software products?***

To answer this question, our research was scoped into three parts concerning specific aspects related to green software products. Each part is addressed individually through multiple research questions. Combined, the answers on the research questions provide an answer to the main research question. In this chapter, we provide an overview of the main contributions, discuss the implications of our research on the field and reflect on the research that has been conducted for this dissertation. The chapter is concluded with the limitations of our research and an agenda for future research to continue improving the greening of ICT research field.

8.1 Contributions

RQ1: What are effective measures to quantify the energy consumption of software products?

This question is addressed in Part I, 'Quantifying Software Energy Consumption', and answered in Chapters 2 and 3. A summary of the answers is provided below:

> **RQ1.1: Can energy profilers be used to accurately estimate the energy consumption of software?**
>
> This question was investigated through an experiment involving fourteen energy profilers. Out of these initial fourteen energy profilers, only two

energy profilers actually produced valid energy consumption estimations due to calibration and measurement issues. A finding that shows the immaturity of the tools available in the field. With regard to energy consumption, we found significant differences between the estimations provided by the energy profilers and the actual energy consumption. Concerning the timeliness of the measurements, energy profilers allow the user to relate specific actions, e.g. user input, to software activity. Hence, in their current form, energy profilers can be used to, at best, provide a general sense of the energy consumption behavior of software. Additional measures, like hardware-based measurements, are still required to obtain accurate energy consumption figures.

RQ1.2: How can we effectively express the resource utilization for executing a software product in relation to the software energy consumption?

We provide an answer on this research question by constructing the Resource Utilization Score (RUS). The RUS delivers a single score to characterize the resource utilization of a software product and is based on relevant dimensions (e.g. CPU and memory utilization) identified by a stakeholder. By allowing performance dimensions to be included, we acknowledge the relation of energy consumption with hardware. Additionally, as the RUS is a generalization of the surface of a radar chart, we are able to visualize measurements and quickly identify key findings.

$$RUS = \sum_{i=1}^{n} \sum_{i<j}^{n} W_{i,j}(P_i \cdot P_j) \tag{8.1}$$

To calculate the RUS (Eq. 8.1), a separate score is calculated for each combination of dimensions. The weight (W) assigned to combination *i,j* with values (P) P_i and P_j provides a score for pair *i,j*. The weight allows a stakeholder to stress the importance of specific dimensions that are more relevant in a particular context. Combining the scores for each pair provides the RUS. Using the RUS, we are able to compare systems on the selected dimensions provided that the systems are tested on the same hardware.

RQ2: How can energy consumption concerns be addressed in software architecture?

This question is addressed in Part II, 'Energy Consumption in Software Architecture', and answered in Chapters 4 and 5. A summary of the answers is provided below:

RQ2.1: How can we position energy consumption within the scope of software architecture in the context of product software?

We answer this research question by positioning sustainability as a quality attribute for software. With the quality attribute we identified measurable, low-level elements to quantitatively evaluate the energy consumption of a software product (Fig. 8.1). To relate energy consumption to software architecture, we constructed an energy consumption perspective according to the format provided by Rozanski and Woods [126]. We identified key questions for each viewpoint related to software design, specified the activities to apply the energy consumption perspective, described concerns and provided architectural tactics. As an initial validation of the perspective, a case study was performed in which we successfully applied the energy perspective to a commercial software product.

RQ2.2: How can we create an in-depth energy profile of a software product?

We proposed the Stubbed Energy Profiling Method, or StEP Method, as an answer to this question. The method is used to determine the energy consumption of software on the level of architectural elements, or even line of code, and encompasses the creation of multiple stubbed versions of

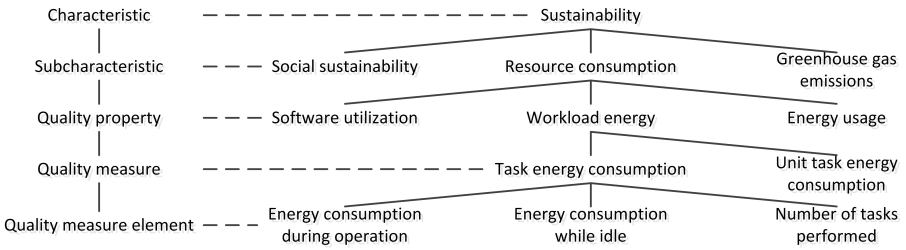


Figure 8.1: A partial breakdown of the sustainability characteristic following the format of the ISO 25010 standard.

a software product. When compared to a non-stubbed benchmark, the difference in energy consumption can be attributed specifically to the stubbed software elements (Fig. 8.2). In the experiment performed with the method, we quantified the energy consumption of the individual tasks performed by a software product in relation to its main functionality. Using this information we identified the source code that is the prime target when solving energy related concerns. The method is positioned as a detailed elaboration of how to create an energy profile, as proposed for answering RQ2.1.

RQ3: How can energy awareness be embedded in the software engineering process?

This question is addressed in Part III, 'Energy Awareness in Software Engineering', and answered in Chapters 6 and 7. A summary of the answers is provided below:

RQ3.1: How can we reliably compare the energy consumption of large-scale software products across different releases?

To answer this research question, we investigated how to reliably measure the energy consumption of a software product. In contrast to RQ2.1 and RQ2.2, the experiment focused on a suitable application in an industrial context. We extensively documented our energy profiling method and applied the profiling method to compare releases of a software product. In the experiment that was performed, the method allowed the stakeholders to establish a better causation link between energy consumption and software change. In addition to the experiment, we investigated a regression model to attribute energy consumption to individual pro-

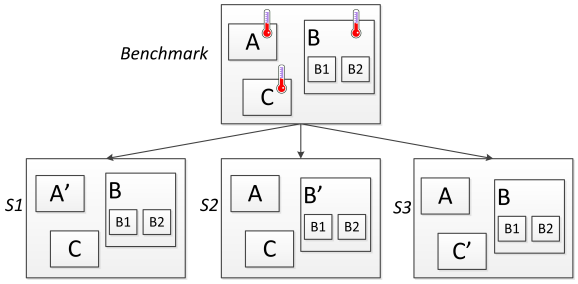


Figure 8.2: Visualization of the Stubbed Energy Profiling Method concept.

cesses. Unfortunately, despite promising improvements, we were unable to create an accurate, software-based model at the process level.

RQ3.2: What is the added value for a software producing organization to perform energy consumption measurements on software products?

Followed by the experiment performed for RQ3.1, multiple interviews were conducted with product stakeholders to answer RQ3.2. We identified that the added value of energy consumption measurements is on both operational and strategic level. On operational level, energy consumption measurements, among others, provide a new technique to identify inefficiencies in software. This improves the skills of a software engineer and evolution of software product, thereby positively contributing to the social and technical dimensions of sustainability.

On strategic level, the SPO is able to plan the evolution of a software product in more detail, i.e. also in terms of sustainability, and thereby gains the opportunity to increase the success of the product. In general, including sustainability on the product roadmap is expected to improve product quality and the balance in relation to other quality attributes. However, stakeholders acknowledged that there must be awareness on the potential of green software products.

RQ3.3: How to create and maintain awareness of the energy consumption perspective in software product development?

A case study was performed with two cases regarding two commercial software products, to provide an answer on this question. First, a survey was constructed to measure awareness of which the data allowed us to express awareness as a score. Second, as a stimulus to trigger awareness, we created an energy dashboard based on the results related to RQ1.2. Combining the survey and dashboard in the design of the case studies, allowed us to incorporate awareness creation in the software engineering process and measure the evolution of awareness over time.

We found that appropriate stimuli help awaken awareness on the energy consumption of software. To maintain awareness, organizational policy is required to support green software products and stimulate informed decision making on sustainable software design. With respect to actual green software engineering, we found willingness among the stakeholders to address software energy consumption which could be strengthened by

providing a knowledge bank with guidelines. The stakeholders also explicitly positioned energy consumption as a non-functional aspect, which implies that an architectural approach is required.

8.2 Implications

The research results have multiple implications for the research communities involved with greening of ICT, software engineering and software architecture, and the software industry in general. In this section we discuss the implications for Part I to III in relation to the objective and research questions identified for our research.

8.2.1 Quantifying Software Energy Consumption

Measuring the energy consumption of software products was found to be a serious barrier for industry to tackle energy concerns. Data center managers, for example, monitor the energy consumption of their facilities but were found hesitant to measure the energy consumption on software level. Specifically, the usability and value of the results was questioned, i.e. 'how can we be sure we actually measure the effects of the software and not a configuration setting of the operating system or load balancing?', considering the required investments to perform such measurements. The barrier was also perceived with less complex environments, e.g. simplified test environments, where similar questions were asked. Therefore, at the start of this research, significant effort was put into determining how to measure software energy consumption. Provided that this was possible to begin with.

During the research project, multiple methods were encountered that provided insight in the energy consumption of the software. In essence, we are able to characterize the measurement methods according to three dimensions:

- *Granularity*: Considers the scope of the measurements, e.g. application level or process level, as required by stakeholders in relation to specific concerns.
- *Investment*; Considers the costs (time and money) of applying a measurement method. Stakeholders often operate from a limited budget and should aim for a short 'Return of Green Investments' period [16].
- *Accuracy*; Considers how closely the method provides measurements that reflect the actual energy consumption. Stakeholders should determine to what extent measurements can deviate, if at all, to remain useful.

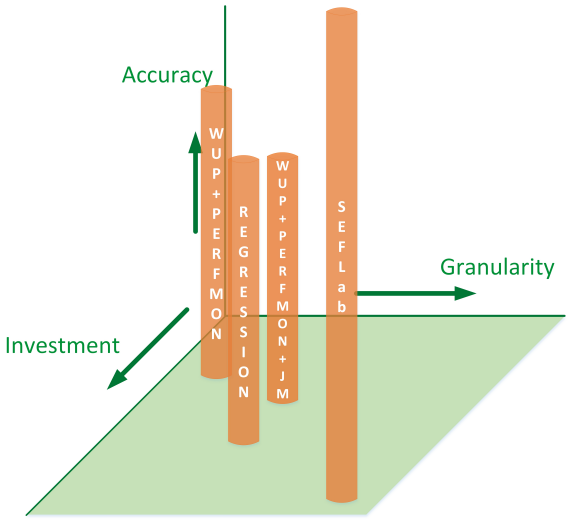


Figure 8.3: Positioning of the applied energy consumption measurement methods.

The synergy between these dimensions determines the positioning of a method. For example, a higher budget often means more detailed and accurate measurements can be obtained. In addition, each context (e.g. data center or laptop) imposes its own requirements and limitations for the measurement method that can be applied. Each stakeholder should determine what level of detail and accuracy are required for a concern, and achievable within a specific budget and context.

In Fig. 8.3, we positioned the methods applied throughout the research project accordingly on the aforementioned dimensions. Note that no scale is provided for the dimensions, since we cannot uniformly quantify different levels. For example, granularity encompasses both software and hardware. The two extremes for a scale would encompass the most and least fine-grained combinations, being line of code and application level for software, and hardware component and data center level for hardware. Although we can identify the combinations in between the extremes, their respective positions on the scale is unknown and could depend on the scope of the concern. Similar examples can be provided for accuracy (e.g. involving error % and timeliness) and investment (e.g. time and money). Thus, each method in Fig. 8.3 is positioned in relativity to the other methods.

The *SEFLab* (Chapter 2) provided highly accurate and fine-grained measurements. On the other hand, such a method requires significant investments and does not extrapolate easily to more complex environments. The method applied in Chapter 3 (*WUP + PERFMON*) encompassed a WattsUp? Pro device and Windows Performance Monitor. This combination required significantly less investments compared to the SEFLab and could be applied in different hardware environments. Albeit, less fine-grained and at the cost of accuracy.

In Chapters 4 to 7, we expanded the combination above with Microsoft Joulemeter (*WUP + PERFMON + JM*) to improve the level of detail (i.e. granularity). The energy profiler adds measurements on process level at the cost of the overall accuracy of the method. In addition, having to operationalize the energy profiler increased the effort, i.e. investment, to apply the method.

Finally, the regression model presented in Chapter 5 (*REGRESSION*) builds on the WattsUp? Pro and Windows Performance Monitor to provide process level measurements. The model outperformed Joulemeter on system level and matched the results on process level, thereby increasing accuracy at the same level of granularity. However, in its current state, the increased accuracy did not offset the investment required to build a special-purpose linear model. Hence, we found the *WUP + PERFMON + JM* method to best suit our purposes.

Comparing software products

Complementary to the measurement methods provided and demonstrated in this dissertation, the RUS allows for the comparison of software based on dimensions selected by a stakeholder. This enables informed decision-making in relation to the product. For example, calculating the RUS based on regression tests allows for quick judgment on the impact of a new release. Note that the insights do not automatically imply a software change. If the RUS shows considerable change on a performance dimension, a sustainability decision could also encompass scaling up or down in terms of hardware resources.

Within the software industry, the RUS provides a first step towards comparing software products with one another in terms of their sustainability. When comparing different products, e.g. cloud services, that provide a similar service, the score could function as an evaluation criteria for the self-regulation concept [116].

8.2.2 Energy Consumption in Software Architecture

When discussing the energy consumption of software products, the discussion often turned towards the question on where to start. Software used to be considered as a 'black box' with respect to energy consumption, where the additional layers in the software stack (e.g. operating system) only complicated the issue of assigning energy consumption to specific software elements. As a consequence, addressing the energy consumption could be characterized as a trial and error process with respect to where to apply changes and determining the effectiveness thereof.

With the energy consumption perspective we provide a starting point to tackle energy concerns for the software. The perspective enables industry to structurally consider the sustainability aspect of software products and open the 'black box' to identify, measure and analyze energy concerns related to architectural elements. Additionally, the perspective relates sustainability to other quality aspects to facilitate trade-offs decisions. As the perspective can be applied early on in the product life cycle, i.e. during development, the costs associated with addressing software sustainability are reduced.

For the research community, the perspective provides an umbrella for research related to the energy consumption of software. For example, sustainable patterns and tactics enrich the perspective, whereas new measurement methods enhance the ability to create an energy profile of a software product. By including the deployment viewpoint, the perspective paves the road to expand to more complex hardware environments (e.g. including virtualized components) whilst maintaining focus on the software. Additionally, the perspective itself is subject for improvement based on new insights and breakthroughs in the research field.

The Stubbed Energy Profiling Method provides a concrete means to create an energy profile, as required for applying the perspective. The method shows the benefits of a synergy between hardware and software; controlled software adjustments provide insight into the energy consumption of specific software elements, while the actual measurements are performed using hardware. Hence, the method does not solely rely on software-based estimations or expensive hardware to perform detailed measurements. Note that the accuracy and level of detail of the Stubbed Energy Profiling Method depend on the time taken to stub the software (investment). Thus, the positioning of the method in Fig. 8.3 varies per application.

8.2.3 Energy Awareness in Software Engineering

Following the line with respect to Part I and II, i.e. the barrier to measure and lack of starting point, it should be no surprise that industry was not actively involved with the energy consumption of software. We observed that market requirements concerning the energy consumption of ICT were solved using a hardware-based approach, without even considering the contribution of the software. The example provided in Chapter 1, showed that software can potentially account for 66% of the power consumption of servers and thereby have a significant impact towards achieving sustainability goals.

With the energy dashboard, we provided a concrete means to relate software engineering activities to energy consumption. Awareness among software engineers and software architects ensures an SPO that energy concerns are on the radar among those that can influence this aspect. The provided architectural approach enables stakeholders to address concerns on different levels of a software product, and find a balance between the efforts needed towards improving energy efficiency and the expected gains. With regard to these expected gains, green software minimizes the required hardware resources to deploy a software product. Hence, green software allows SPOs to rationalize on hardware investments for more efficient scaling and minimize facility costs.

Awareness also helps to bridge the gap between research and industry. When there is a concrete need from industry (which requires awareness on the topic), research is able to actively participate in satisfying this need. Thus awareness potentially opens the door for research in an industrial context. Actively participating in industry provides insight into the decision-making process concerning green software engineering and potentially motivates the adoption of green software engineering practices. Additionally, the research community is better able to generalize the results compared to, e.g., a controlled laboratory setting.

8.3 Reflection

In this section we reflect on the research project in general and more specifically on the research process, methods used and observations in industry.

Experimentation and software energy consumption

The main research method applied throughout this dissertation is the controlled experiment. To apply the method, we consistently followed the guidelines provided in literature, e.g. [148] and [66], and the process as shown in

Fig. 8.4. Although useful, we did experience a gap in applying the method. In our view, specifically the experiment planning, or design [66], activity does not address the complexity of experiments performed when software is the object of the study.

The experiment process is often explained using human participants as part of the experiment. However, with energy consumption measurements the experiment involves software. To successfully apply the method with software energy consumption, specific activities are required during experiment design. In particular, more attention should go towards ensuring all independent variables are under control. These can be directly related to the software and the hardware that executes the software, but also extend to less evident variables like hardware infrastructure and room temperature.

A different aspect of experiment design is the number of repetitions required to validly determine the energy effect of the treatment. Components like the memory (RAM) store information that is frequently accessed; an automatic measure to improve the efficiency, that affects the experiment. To circumvent such measures, in contrast to human participants, software allows the experimenter to fully restore the environment to its initial situation using, e.g., clones or images. As the researcher can be confronted with extremely small energy consumption differences across measurements, the inability to revert to an initial state could invalidate the measurement.

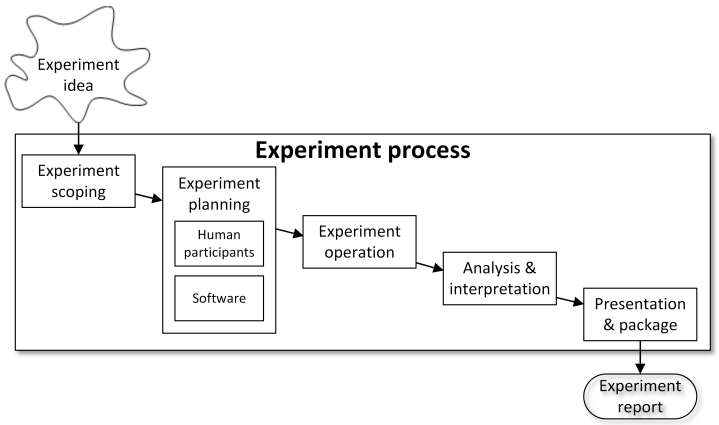


Figure 8.4: The experiment process adjusted to acknowledge a difference between human participants and software as experiment objects, after [148].

From an organization point of view, experimentation with energy consumption could require exceptional authorizations for the experimenter. To apply a hardware-based measurement approach, a stakeholder requires physical access to the hardware. This could be a problem for environments where potentially privacy sensitive data is processed. On the other hand, a software-based approach often requires administrator rights to install and configure the tooling. Among others, this opens a door for malicious tooling to be installed without supervision of environment administrators. Hence, experimentation with software energy consumption requires specific arrangements within an organization. Ideally, the experimenter has commitment from top management to overcome any hurdles for performing experiments.

Synergy between hardware and software

In the starting days of ICT, the limited capabilities of hardware forced software developers to reflect on each line of code for a program. Developers were aware of the limited resources available and software optimization was a standard exercise. However, the continuing evolution of hardware reduced the resource scarcity and changed the mindset away from efficient software. Decades later, motivated by sustainable development, we again are required to increase the efficiency of software with respect to resource utilization.

The focus on software products helps to restore balance between hardware and software optimization and, in our opinion, further advancements should consider both in synergy. During this research project, multiple aspects were observed where synergy between software and hardware would have provided a more sustainable solution. For example, adding encryption to a product can be done using a hardware- or software-based solution. An optimized hardware chip could provide a more sustainable solution compared to software encryption, which ideally is determined during software design. In this same line, hardware manufacturers could include specialized, high quality sensors in their systems to perform energy consumption measurements. The latter would even further lower the threshold to perform measurements and potentially improve the quality of the measurements.

Collaboration with software industry

An important aspect of this research is the fact that it has been conducted in close collaboration with the software industry. Centric is a large SPO from the Netherlands with approximately 5.000 employees and a product portfolio containing over 60 software products. The products include public administration software and tax applications, and span several markets like the public sector, retail and health care. The scale and number of the software products,

triggered the organization to investigate how to improve the sustainability of their ICT solutions through the software.

Without close collaboration, we would not have had close access to the cases as presented in Chapter 4 and 7 and the experiments would most likely not have been performed with non-trivial commercial software products. Additionally, the setting allowed us to investigate sustainability in an industrial context, which improved the ability to generalize the results. In this light, we can only encourage other researchers in the field to intensify their collaboration with industry.

The close collaboration with industry also helped with the valorization of the research results. Specifically, the results helped to tackle the problem of unawareness regarding sustainability and the potential of software. While it will certainly not be the case that the least energy consuming solution for the software is implemented, the strategy can now be justified using the social, technical and economical dimensions. Hence, green software is not an unknown topic anymore within the participating SPO and stakeholders are able to better address concerns in relation to the sustainability of their software products.

Future of green software products

Following the growing importance of sustainability and Corporate Social Responsibility, we expect that a change is eminent in the ICT industry. Specifically, we think players in the industry, e.g. SPOs, are increasingly expected to either improve their own sustainability or enable their customers to achieve sustainability goals. Hence, the industry expected to more actively address sustainability through policies and concrete goals. The position of SPOs in the supply chain, e.g. supplier of software and often buyer of large volumes of hardware, also provides the opportunity to actively involve other links in the chain in a top-down approach. From the bottom up, where the actual changes are made, a direct connection is created with strategic goals which helps to prioritize sustainable undertakings.

Specific for software products, a complementary change is expected of which the first outings are already perceived. Driven by entrepreneurship and software engineering as an increasingly common skill, we are quick to find functionally equivalent software products available for usage. Consider the multitude of smartphone apps available that provide the same functionality. In this case, a distinction between apps is made on the battery impact; a qualitative property of the software. For enterprise software, currently the most apparent quality properties are performance and, given recent developments, security. It is our hope that sustainability is added to the list when software qualities are the decisive factor for the industry.

8.4 Limitations and Future Research

In this final section we discuss the main limitations of our work and identify some directions for future research.

Validation of the research artifacts

The research artifacts presented in this dissertation are constructed in the context of a large SPO in the Netherlands. This makes our results prone to specific situational factors like company size, culture and affinity with sustainability. We did include an international element in the study presented in Chapter 7, however these were branches of the SPO and the data did not allow us to draw conclusions with respect to, e.g., cultural differences. Additionally, the artifacts are the results of exploratory research and are only validated in either an experimental or case study setting.

Further research is needed to increase the maturity of the artifacts and mitigate the limitations. Specifically, we can identify the need for empirical evaluation in different contexts to improve the existing artifacts. Applying the artifacts across SPOs in different countries contributes to the overall quality and general applicability. When applied across market segment, we are able to identify segment specific (sustainability) requirements that should be accounted for in the artifacts. Finally, the organization size could affect the ability to apply the artifacts and the corresponding strategic opportunities.

Measurement method

Throughout the research, apart from Chapter 2 where extra investment was required, we consistently applied a method that was sufficient to provide accurate results for our purposes. The method enables stakeholders to perform energy consumption measurements on software products according to a systematic methodology and requires relatively little investments. However, the method was mostly applied in a simplified test environment and thus should be considered as a starting point for performing energy consumption measurements. Additionally, despite our results, we acknowledge that the method will not suffice for all purposes.

Further research is required to improve the measurement method in terms of the level of detail and its applicability in more complex environments. With respect to the level of detail, we do not refer to detailed measurements on the software product, where the Stubbed Energy Profiling Method should suffice, but rather the different layers of the software stack. The results in Chapter 5 showed significant energy consumption overhead that could not be attributed to the software product under test. An in-depth analysis on process level,

for example, could show the energy effects of the operating system, which potentially affect design decisions for the software product.

Understanding the synergy between different layers of the software stack also positively contributes towards the applicability of the method. For example, the in-depth analysis on process level could expose the path of processes from application to hardware in heavily virtualized environments. The measurement method should be flexible to allow detailed measurements in more complex hardware environments, which in this case implies adjustments should be made with respect to the process level measurements.

Green guidelines

The artifacts enable stakeholders to structurally influence the sustainability of their software. However, to enable also requires that guidelines are provided, i.e. what are green coding practices in relation to a software product. Given the state of the research field, relatively little concrete guidelines are provided in this dissertation which is a limitation of our research.

Further research is required to provide concrete green guidelines. The guidelines should address specific implementation and design choices from the perspective of a software engineer and architect, i.e. how to implement function X in a sustainable manner. Additionally, the guidelines should specify the constraints and limitations for applying the guideline. In essence, the strive should be to build a body of knowledge for green software engineering. The artifacts provided in this dissertation form a basis for the guidelines and could help structure the body of knowledge in the research field.

8.5 To Conclude

We entered new and relatively unexplored terrain with respect to software production, to facilitate the contribution of software towards sustainability. This dissertation provides concrete tools to support green software engineering and discusses the dynamics involved with satisfying all four sustainability dimensions. We hope this dissertation forms a foundation to further mature this field, and contributes to sustainable developments in the ICT industry.

For the coming years, our outlook for the greening of ICT research field is positive. Supported by international policies and agreements, increased awareness provides a perspective for industry-wide standards and benchmarks similar to houses, electronic equipment and cars. If developments sustain, we hope this dissertation contributes to establish energy labels for software in the near future.

Bibliography

[1] B. Alcott. Jevons' paradox. *Ecological Economics*, 54(1):9–21, 2005.

[2] N. Amsel and B. Tomlinson. Green tracker: A tool for estimating the energy consumption of software. In *CHI '10 Extended Abstracts on Human Factors in Computing Systems*, pages 3337–3342. ACM, 2010.

[3] R. Andersen. *Modern methods for robust regression.* Sage, 2008.

[4] D. Ardagna, B. Panicucci, M. Trubian, and L. Zhang. Energy-aware autonomic resource allocation in multitier virtualized environments. *IEEE Transactions on Services Computing*, 5(1):2–19, 2012.

[5] S. Barlowe and A. Scott. O-charts: Towards an effective toolkit for teaching time complexity. In *2015 IEEE Frontiers in Education Conference (FIE)*, pages 1–4, 2015.

[6] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice.* SEI Series in Software Engineering. Pearson Education, 2012.

[7] L. Bass, M. Klein, and F. Bachmann. Quality attribute design primitives and the attribute driven design method. In F. van der Linden, editor, *Software Product-Family Engineering*, pages 169–186. Springer Berlin Heidelberg, 2002.

[8] C. Becker, R. Chitchyan, L. Duboc, S. Easterbrook, B. Penzenstadler, N. Seyff, and C. Venters. Sustainability Design and Software: The Karlskrona Manifesto. In *Software Engineering (ICSE), 2015 IEEE/ACM 37th IEEE International Conference on*, volume 2, pages 467–476. IEEE, 2015.

[9] J. Beckett and R. Bradfield. Power efficiency comparison of enterprise-class blade servers and enclosures. Technical report, Dell, 2011.

[10] G. Bekaroo, C. Bokhoree, and C. Pattinson. Power measurement of computers: analysis of the effectiveness of the software based approach. *Int. J. Emerg. Technol. Adv. Eng*, 4(5):755–762, 2014.

[11] W. Bekkers, I. van de Weerd, M. Spruit, and S. Brinkkemper. A framework for process improvement in software product management. In *Systems, Software and Services Process Improvement*, volume 99 of *Communications in Computer and Information Science*, pages 1–12. Springer Berlin Heidelberg, 2010.

[12] A. Beloglazov, J. Abawajy, and R. Buyya. Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. *Future Generation Computer Systems*, 28(5):755–768, 2012.

[13] A. Berl, E. Gelenbe, M. Di Girolamo, G. Giuliani, H. De Meer, M. Q. Dang, and K. Pentikousis. Energy-efficient cloud computing. *The Computer Journal*, 53(7):1045–1051, 2010.

[14] B. W. Boehm. A spiral model of software development and enhancement. *Computer*, 21(5):61–72, 1988.

[15] E. Bondarev, M. R. V. Chaudron, and E. A. de Kock. Exploring performance trade-offs of a jpeg decoder using the deepcompass framework. In *Proceedings of the 6th International Workshop on Software and Performance*, pages 153–163. ACM, 2007.

[16] P. Bozzelli, Q. Gu, and P. Lago. A systematic literature review on green software metrics. Technical report, VU University Amsterdam, 2013.

[17] D. J. Brown and C. Reams. Toward energy-efficient computing. *Communications of the ACM*, 53(3):50–58, 2010.

[18] Brundtland Commission. Report of the world commission on environment and development: Our common future, 1987. Online; accessed 03-01-2017.

[19] K. P. Burnham and D. R. Anderson. *Model selection and multimodel inference: a practical information-theoretic approach*. Springer, 2003.

[20] T. Cao, S. M. Blackburn, T. Gao, and K. S. McKinley. The Yin and Yang of Power and Performance for Asymmetric Hardware and Managed Software. In *Proceedings of the 39th Annual International Symposium on Computer Architecture*, pages 225–236. IEEE Computer Society, 2012.

[21] C. Cappiello, M. Matera, and M. Picozzi. A ui-centric approach for the end-user development of multidevice mashups. *ACM Transactions on the Web*, 9(3):11:1–11:40, 2015.

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

[23] R. Carpa, O. Gluck, L. Lefevre, and J.-C. Mignot. Improving the energy efficiency of software-defined backbone networks. *Photonic Network Communications*, 30(3):337–347, 2015.

[24] G. G. Casta, A. Nez, P. Llopis, and J. Carretero. E-mc2: A formal framework for energy modelling in cloud computing. *Simulation Modelling Practice and Theory*, 39:56–75, 2013. S.I.Energy efficiency in grids and clouds.

[25] F. Chasin. Sustainability: Are we all talking about the same thing. In *Proceedings of the 2nd International Conference on ICT for Sustainability*, pages 342–351. Atlantis Press, 2014.

[26] H. Chen, B. Luo, and W. Shi. Anole: A case for energy-aware mobile application design. In *2012 41st International Conference on Parallel Processing Workshops*, pages 232–238, 2012.

[27] X. Chen and C. Phillips. An evolutionary based dynamic energy management framework for ip-over-dwdm networks. *Sustainable Computing: Informatics and Systems*, 4(2):94–105, 2014.

[28] S. A. Chowdhury and A. Hindle. Greenoracle: Estimating software energy consumption with energy measurement corpora. In *Proceedings of the 13th International Conference on Mining Software Repositories*, pages 49–60. ACM, 2016.

[29] V. Coroama and L. M. Hilty. Energy consumed vs. energy saved by ict - a closer look. In V. Wohlgemut, B. Page, and K. Voigt, editors, *Environmental Informatics and Industrial Environmental Protection: Concepts, Methods and Tools*, pages 353–361, 2009.

[30] P. Devanbu, T. Zimmermann, and C. Bird. Belief & evidence in empirical software engineering. In *Proceedings of the 38th International Conference on Software Engineering*, pages 108–119. ACM, 2016.

[31] T. Do, S. Rawshdeh, and W. Shi. ptop: A process-level power profiling tool. In *Proceedings of the 2nd Workshop on Power Aware Computing and Systems (HotPower09)*, 2009.

[32] R. E. Dunlap. The new environmental paradigm scale: From marginality to worldwide use. *The Journal of Environmental Education*, 40(1):3–18, 2008.

[33] C. Ebert and S. Brinkkemper. Software product management - an industry evaluation. *Journal of Systems and Software*, 95:10–18, 2014.

[34] K. M. Eisenhardt. Agency theory: An assessment and review. *The Academy of Management Review*, 14(1):57–74, 1989.

[35] H. Esmaeilzadeh, T. Cao, X. Yang, S. Blackburn, and K. McKinley. What is happening to power, performance, and software? *IEEE Micro*, 32(3):110–121, 2012.

[36] J. Espadas, A. Molina, G. Jiménez, M. Molina, R. Ramirez, and D. Concha. A tenant-based resource allocation model for scaling software-as-a-service applications over cloud computing infrastructures. *Future Generation Computer Systems*, 29(1):273–286, 2013.

[37] European Commission. Paris agreement, 2016. Online; accessed 03-01-2017.

[38] A. M. Ferreira and B. Pernici. Managing the complex data center environment: an integrated energy-aware framework. *Computing*, pages 1–41, 2014.

[39] M. A. Ferreira, E. Hoekstra, B. Merkus, B. Visser, and J. Visser. Seflab: A lab for measuring software energy footprints. In *2013 2nd International Workshop on Green and Sustainable Software (GREENS)*, pages 30–37, 2013.

[40] A. Field. *Discovering Statistics Using SPSS*. SAGE Publications, 2007.

[41] F. Fotrousi and S. A. Fricker. Software analytics for planning product evolution. In A. Maglyas and A.-L. Lamprecht, editors, *Software Business: 7th International Conference, ICSOB 2016, Proceedings*, pages 16–31. Springer International Publishing, 2016.

[42] S. A. Fricker. Software product management. In A. Maedche, A. Botzenhardt, and L. Neer, editors, *Software for People: Fundamentals, Trends and Best Practices*, pages 53–81. Springer Berlin Heidelberg, 2012.

[43] D. Ganesan, M. Lindvall, D. McComas, M. Bartholomew, S. Slegel, and B. Medina. *Architecture-Based Unit Testing of the Flight Software Product Line*, volume 6287 of *Lecture Notes in Computer Science*, pages 256–270. Springer, 2010.

[44] V. Grassi, R. Mirandola, E. Randazzo, and A. Sabetta. Klaper: An intermediate language for model-driven predictive analysis of performance and reliability. In A. Rausch, R. Reussner, R. Mirandola, and F. Plášil, editors, *The Common Component Modeling Example: Comparing Software Component Models*, pages 327–356. Springer Berlin Heidelberg, 2008.

[45] K. Grosskop and J. Visser. Identification of application-level energy optimizations. In *Proceedings of the First International Conference on ICT for Sustainability*, pages 101–107. Atlantis Press, 2013.

[46] Q. Gu, P. Lago, and S. Potenza. Aligning economic impact with environmental benefits: A green strategy model. In *Proceedings of the First International Workshop on Green and Sustainable Software*, pages 62–68. IEEE Press, 2012.

[47] A. Gupta, T. Zimmermann, C. Bird, N. Nagappan, T. Bhat, and S. Emran. Detecting energy patterns in software development. Technical report, Microsoft Research Microsoft, 2011.

[48] S. Hasan, Z. King, M. Hafiz, M. Sayagh, B. Adams, and A. Hindle. Energy profiles of java collections classes. In *Proceedings of the 38th International Conference on Software Engineering*, pages 225–236. ACM, 2016.

[49] A. Hevner and S. Chatterjee. *Design Science Research in Information Systems*, pages 9–22. Springer, 2010.

[50] L. M. Hilty and B. Aebischer. Ict for sustainability: An emerging research field. In L. M. Hilty and B. Aebischer, editors, *ICT Innovations for Sustainability*, pages 3–36. Springer International Publishing, 2015.

[51] A. Hindle. Green mining: a methodology of relating software change and configuration to power consumption. *Empirical Software Engineering*, pages 1–36, 2013.

[52] A. Hindle, A. Wilson, K. Rasmussen, E. J. Barlow, J. C. Campbell, and S. Romansky. Greenminer: A hardware based mining software repositories software energy consumption framework. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 12–21. ACM, 2014.

[53] S.-Y. Ihm, A. Nasridinov, J.-H. Lee, and Y.-H. Park. Efficient duality-based subsequent matching on time-series data in green computing. *The Journal of Supercomputing*, 69(3):1039–1053, 2014.

[54] ISO. Systems and software engineering – systems and software quality requirements and evaluation (SQuaRE) – system and software quality models. ISO 2510:2011, International Organization for Standardization, 2011.

[55] E. Jagroep, J. Broekman, J. M. E. M. van der Werf, S. Brinkkemper, P. Lago, L. Blom, and R. van Vliet. Awakening awareness on energy consumption in software engineering. In *Proceedings of the 39th International Conference on Software Engineering: Software Engineering in Society Track*, pages 76–85. IEEE Press, 2017.

[56] E. Jagroep, G. Procaccianti, J. M. van der Werf, S. Brinkkemper, L. Blom, and R. van Vliet. Energy efficiency on the product roadmap: An empirical study across releases of a software product. *Journal of Software: Evolution and Process*, 29(2), 2017.

[57] E. Jagroep, I. van de Weerd, S. Brinkkemper, and T. Dobbe. Implementing software product portfolio management. In *2011 Fifth International Workshop on Software Product Management (IWSPM)*, pages 67–76, 2011.

[58] E. Jagroep, I. van de Weerd, S. Brinkkemper, and T. Dobbe. Framework for implementing product portfolio management in software business. In G. Ruhe and C. Wohlin, editors, *Software Project Management in a Changing World*, pages 193–221. Springer Berlin Heidelberg, 2014.

[59] E. Jagroep, A. van der Ent, J. M. E. M. van der Werf, S. Brinkkemper, L. Blom, and R. van Vliet. Hunt the energy guzzler in my software: An architectural approach. *Submitted for publication.*

[60] E. Jagroep, J. M. van der Werf, S. Brinkkemper, L. Blom, and R. van Vliet. Extending software architecture views with an energy consumption perspective. *Computing*, 99(6):553–573, 2017.

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

[62] E. Jagroep, J. M. E. M. van der Werf, J. Broekman, S. Brinkkemper, L. Blom, and R. van Vliet. A resource utilization score for software energy consumption. In *Proceedings of the 4th International Conference on ICT for Sustainability*, pages 19–28. Atlantis Press, 2016.

[63] E. Jagroep, J. M. E. M. van der Werf, S. Jansen, M. Ferreira, and J. Visser. Profiling energy profilers. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, pages 2198–2203. ACM, 2015.

[64] E. Jagroep, J. M. E. M. van der Werf, R. Spauwen, L. Blom, R. van Vliet, and S. Brinkkemper. An energy consumption perspective on software architecture. In D. Weyns, R. Mirandola, and I. Crnkovic, editors, *Software Architecture: 9th European Conference, ECSA 2015, Proceedings*, pages 239–247. Springer International Publishing, 2015.

[65] S. Jansen, S. Brinkkemper, J. Souer, and L. Luinenburg. Shades of gray: Opening up a software producing organization with the open software enterprise model. *Journal of Systems and Software*, 85(7):1495–1510, 2012.

[66] N. Juristo and A. M. Moreno. *Basics of Software Engineering Experimentation*. Springer, 2010.

[67] G. Kalaitzoglou, M. Bruntink, and J. Visser. A practical model for evaluating the energy efficiency of software applications. In *Proceedings of the 2nd International Conference on ICT for Sustainability*, pages 77–86. Atlantis Press, 2014.

[68] A. Kansal, F. Zhao, J. Liu, N. Kothari, and A. A. Bhattacharya. Virtual machine power metering and provisioning. In *Proceedings of the 1st ACM Symposium on Cloud Computing*, pages 39–50. ACM, 2010.

[69] R. Kazman, M. Klein, M. Barbacci, and T. Longstaff. The architecture tradeoff analysis method. In *4th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS) 1998*, pages 68–78. IEEE, 1998.

[70] E. Kern, M. Dick, S. Naumann, and A. Filler. Labelling sustainable software products and websites: Ideas, approaches, and challenges. In *Proceedings of the 3rd International Conference on ICT for Sustainability*, pages 82–91. Atlantis Press, 2015.

[71] E. Kern, M. Dick, S. Naumann, A. Guldner, and T. Johann. Green software and green software engineering - definitions, measurements, and quality aspects. In *Proceedings of the First International Conference on ICT for Sustainability*, pages 87–946. Atlantis Press, 2013.

[72] M. Kim, T. Zimmermann, R. DeLine, and A. Begel. The emerging role of data scientists on software development teams. In *Proceedings of the 38th International Conference on Software Engineering*, pages 96–107. ACM, 2016.

[73] A. Kipp, T. Jiang, M. Fugini, and I. Salomie. Layered green performance indicators. *Future Generation Computer Systems*, 28(2):478–489, 2012.

[74] B. Kitchenham, H. Al-Khilidar, M. Babar, M. Berry, K. Cox, J. Keung, F. Kurniawati, M. Staples, H. Zhang, and L. Zhu. Evaluating guidelines for reporting empirical software engineering studies. *Empirical Software Engineering*, 13(1):97–121, 2008.

[75] B. A. Kitchenham, S. L. Pfleeger, L. M. Pickard, P. W. Jones, D. C. Hoaglin, K. E. Emam, and J. Rosenberg. Preliminary guidelines for empirical research in software engineering. *IEEE Transactions on Software Engineering*, 28(8):721–734, 2002.

[76] R. Koller, A. Verma, and A. Neogi. WattApp: an application aware power meter for shared data centers. In *Proceedings of the 7th international conference on Autonomic computing*, pages 31–40. ACM, 2010.

[77] S. Kounev, F. Brosig, N. Huber, and R. Reussner. Towards self-aware performance and resource management in modern service-oriented systems. In *2010 IEEE International Conference on Services Computing*, pages 621–624, 2010.

[78] H. Koziolek, D. Domis, T. Goldschmidt, and P. Vorst. Measuring architecture sustainability. *IEEE Software*, 30(6):54–62, 2013.

[79] K. Kritikos, B. Pernici, P. Plebani, C. Cappiello, M. Comuzzi, S. Benrernou, I. Brandic, A. Kertész, M. Parkin, and M. Carro. A survey on service quality description. *ACM Computing Surveys*, 46(1):1:1–1:58, 2013.

[80] P. Kruchten, R. L. Nord, and I. Ozkaya. Technical debt: From metaphor to theory and practice. *IEEE Software*, 29(6):18–21, 2012.

[81] P. Lago and T. Jansen. Creating environmental awareness in service oriented software engineering. In *Service-Oriented Computing*, volume 6568 of *Lecture Notes in Computer Science*, pages 181–186. Springer Berlin Heidelberg, 2011.

[82] P. Lago, R. Kazman, N. Meyer, M. Morisio, H. A. Müller, F. Paulisch, G. Scanniello, B. Penzenstadler, and O. Zimmermann. Exploring initial challenges for green software engineering: summary of the first GREENS workshop, at ICSE 2012. *ACM SIGSOFT Software Engineering Notes*, 38(1):31–33, 2013.

[83] P. Lago, S. A. Koçak, I. Crnkovic, and B. Penzenstadler. Framing sustainability as a property of software quality. *Communications of the ACM*, 58(10):70–78, 2015.

[84] D. Li, S. Hao, W. G. J. Halfond, and R. Govindan. Calculating source line level energy information for android applications. In *Proceedings of the 2013 International Symposium on Software Testing and Analysis*, pages 78–89. ACM, 2013.

[85] J. Li, F. Tao, Y. Cheng, and L. Zhao. Big data in product lifecycle management. *The International Journal of Advanced Manufacturing Technology*, 81(1):667–684, 2015.

[86] B. Littlewood, P. Popov, and L. Strigini. Modeling software design diversity: A review. *ACM Comput. Surv.*, 33(2):177–208, 2001.

[87] Y. Liu, C. Xu, and S.-C. Cheung. Characterizing and detecting performance bugs for smartphone applications. In *Proceedings of the 36th International Conference on Software Engineering*, pages 1013–1024. ACM, 2014.

[88] K. Lundfall, P. Grosso, P. Lago, and G. Procaccianti. The green practitioner: A decision-making tool for green ict. In *Proceedings of the 3rd International Conference on ICT for Sustainability*, pages 74–81. Atlantis Press, 2015.

[89] D. Magalhães, R. N. Calheiros, R. Buyya, and D. G. Gomes. Workload modeling for resource usage analysis and simulation in cloud computing. *Computers & Electrical Engineering*, 47:69–81, 2015.

[90] J. Mair, D. Eyers, Z. Huang, and H. Zhang. Myths in power estimation with performance monitoring counters. *Sustainable Computing: Informatics and Systems*, 4(2):83–93, 2014.

[91] I. Manotas, C. Bird, R. Zhang, D. Shepherd, C. Jaspan, C. Sadowski, L. Pollock, and J. Clause. An empirical study of practitioners' perspectives on green software engineering. In *Proceedings of the 38th International Conference on Software Engineering*, pages 237–248. ACM, 2016.

[92] E. Matthies. How can psychologists better put across their knowledge to practitioners? suggesting a new, integrative influence model of pro-environmental everyday behaviour. *Umweltpsychologie*, 9(1):62–81, 2005.

[93] I. Meedeniya, B. Buhnova, A. Aleti, and L. Grunske. Architecture-driven reliability and energy optimization for complex embedded systems. In G. T. Heineman, J. Kofron, and F. Plasil, editors, *Research into Practice – Reality and Gaps: 6th International Conference on the Quality of Software Architectures*, pages 52–67. Springer Berlin Heidelberg, 2010.

[94] M. B. Miles and A. M. Huberman. *Qualitative data analysis: An expanded sourcebook*. Sage, 1994.

[95] M. P. Mills. The cloud begins with coal: an overview of the electricity used by the global digital ecosystem. Technical report, Digital Power Group, 2013.

[96] R. Mittal, A. Kansal, and R. Chandra. Empowering developers to estimate app energy consumption. In *Proceedings of the 18th annual international conference on Mobile computing and networking*, pages 317–328. ACM, 2012.

[97] A. Morales Ruiz, W. Daniels, D. Hughes, and C. Grothoff. Cryogenic: enabling power-aware applications on linux. In *Proceedings of the 2nd International Conference on ICT for Sustainability*. Atlantis Press, Atlantis Press, 2014.

[98] H. Mosley and A. Mayer. Benchmarking national labour market performance: A radar chart approach. WZB Discussion Paper FS I 99-202, Wissenschaftszentrum Berlin für Sozialforschung (WZB), 1999.

[99] S. Murugesan. Harnessing green it: Principles and practices. *IT Professional*, 10(1):24–33, 2008.

[100] G. J. Myers, C. Sandler, and T. Badgett. *The art of software testing*. John Wiley & Sons, 2011.

[101] T. Nagel. What is it like to be a bat? *The Philosophical Review*, 83(4):435–450, 1974.

[102] S. Naumann, M. Dick, E. Kern, and T. Johann. The greensoft model: A reference model for green and sustainable software and its engineering. *Sustainable Computing: Informatics and Systems*, 1(4):294–304, 2011.

[103] S. Naumann, E. Kern, M. Dick, and T. Johann. Sustainable software engineering: Process and quality models, life cycle, and social aspects. In L. M. Hilty and B. Aebischer, editors, *ICT Innovations for Sustainability*, pages 191–205. Springer, 2015.

[104] A. Noureddine, A. Bourdon, R. Rouvoy, and L. Seinturier. A preliminary study of the impact of software engineering on greenit. In *Green and Sustainable Software (GREENS), 2012 First International Workshop on*, pages 21–27, 2012.

[105] A. Noureddine, S. Islam, and R. Bashroush. Jolinar: Analysing the Energy Footprint of Software Applications (Demo). In *Proceedings of the 25th International Symposium on Software Testing and Analysis*, pages 445–448, 2016.

[106] A. Noureddine and A. Rajan. Optimising energy consumption of design patterns. In *Proceedings of the 37th International Conference on Software Engineering - Volume 2*, pages 623–626. IEEE Press, 2015.

[107] A. Noureddine, R. Rouvoy, and L. Seinturier. A review of energy measurement approaches. *SIGOPS Operating Systems Review*, 47(3):42–49, 2013.

[108] A. Noureddine, R. Rouvoy, and L. Seinturier. Unit testing of energy consumption of software libraries. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, pages 1200–1205. ACM, 2014.

[109] A. Noureddine, R. Rouvoy, and L. Seinturier. Monitoring energy hotspots in software. *Automated Software Engineering*, 22(3):291–332, 2015.

[110] H. H. Olsson and J. Bosch. From requirements to continuous reprioritization of hypotheses. In *Proceedings of the International Workshop on Continuous Software Evolution and Delivery*, pages 63–69. ACM, 2016.

[111] B. Pang and L. Lee. Opinion mining and sentiment analysis. *Foundations and Trends in Information Retrieva*, 2(1-2):1–135, 2008.

[112] C. Pang, A. Hindle, B. Adams, and A. E. Hassan. What do programmers know about software energy consumption? *IEEE Software*, 33(3):83–89, 2016.

[113] A. Pathak, Y. C. Hu, and M. Zhang. Where is the energy spent inside my app?: Fine grained energy accounting on smartphones with eprof. In *Proceedings of the 7th ACM European Conference on Computer Systems*, pages 29–42. ACM, 2012.

[114] K. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee. A design science research methodology for information systems research. *Journal of Management Information Systems*, 24(3):45–77, 2007.

[115] B. Pernici, M. Aiello, J. vom Brocke, B. Donnellan, E. Gelenbe, and M. Kretsis. What IS can do for environmental sustainability: a report from CAiSE11 panel on green and sustainable is, 2012. Online; accessed 03-01-2017.

[116] B. Pernici, D. Ardagna, and C. Cappiello. Business process design: Towards service- based green information systems. In A. Mazzeo, R. Bellini, and G. Motta, editors, *E-Government Ict Professionalism and Competences Service Science*, pages 195–203. Springer US, 2008.

[117] G. Pinto and F. Castor. On the implications of language constructs for concurrent execution in the energy efficiency of multicore applications. In *Proceedings of the 2013 Companion Publication for Conference on Systems, Programming, & Applications: Software for Humanity*, pages 95–96. ACM, 2013.

[118] G. Pinto, F. Castor, and Y. D. Liu. Mining questions about software energy consumption. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 22–31. ACM, 2014.

[119] G. Pinto, F. Castor, and Y. D. Liu. Understanding energy behaviors of thread management constructs. *SIGPLAN Not.*, 49(10):345–360, 2014.

[120] M. Poess and R. O. Nambiar. Energy cost, the key challenge of today's data centers: a power consumption analysis of tpc-c results. *Proceedings of the VLDB Endowment*, 1(2):1229–1240, 2008.

[121] G. Procaccianti, H. Fernàndez, and P. Lago. Empirical evaluation of two best practices for energy-efficient software development. *Journal of Systems and Software*, 117:185–198, 2016.

[122] G. Procaccianti, P. Lago, and G. A. Lewis. A catalogue of green architectural tactics for the cloud. In *2014 IEEE 8th International Symposium on the Maintenance and Evolution of Service-Oriented and Cloud-Based Systems*, pages 29–36, 2014.

[123] G. Procaccianti, P. Lago, and G. A. Lewis. Green architectural tactics for the cloud. In *2014 IEEE/IFIP Conference on Software Architecture*, pages 41–44, 2014.

[124] G. Procaccianti, P. Lago, A. Vetrò, D. M. Fernández, and R. Wieringa. The green lab: Experimentation in software energy efficiency. In *Proceedings of the 37th International Conference on Software Engineering - Volume 2*, pages 941–942. IEEE Press, 2015.

[125] K. K. Rangan, G.-Y. Wei, and D. Brooks. Thread motion: Fine-grained power management for multi-core systems. In *Proceedings of the 36th Annual International Symposium on Computer Architecture*, pages 302–313. ACM, 2009.

[126] N. Rozanski and E. Woods. *Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives*. Addison-Wesley, 2011.

[127] P. Runeson and M. Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical software engineering*, 14(2):131–164, 2009.

[128] M. Sabharwal, A. Agrawal, and G. Metri. Enabling green it through energy-aware software. *IT Professional*, 15(1):19–27, 2013.

[129] C. Sahin, M. Wan, P. Tornquist, R. McKenna, Z. Pearson, W. G. J. Halfond, and J. Clause. How does code obfuscation impact energy usage? *Journal of Software: Evolution and Process*, 28(7):565–588, 2016.

[130] E. Saxe. Power-efficient software. *Communications of the ACM*, 53(2):44–48, 2010.

[131] S. Schubert, D. Kostic, W. Zwaenepoel, and K. Shin. Profiling software for energy consumption. In *International Conference on Green Computing and Communications*, pages 515–522, 2012.

[132] H. Schütz, S. Speckesser, and G. Schmid. Benchmarking labour market performance and labour market policies: Theoretical foundations and applications. WZB Discussion Paper FS I 98-205, Wissenschaftszentrum Berlin für Sozialforschung (WZB), 1998.

[133] C. Seo, G. Edwards, S. Malek, and N. Medvidovic. A framework for estimating the impact of a distributed software system's architectural style on its energy consumption. In *Seventh Working IEEE/IFIP Conference on Software Architecture (WICSA 2008)*, pages 277–280, 2008.

[134] C. Seo, S. Malek, and N. Medvidovic. Component-level energy consumption estimation for distributed java-based software systems. In M. R. V. Chaudron, C. Szyperski, and R. Reussner, editors, *Component-Based Software Engineering: 11th International Symposium, CBSE 2008, Karlsruhe, Germany, October 14-17, 2008. Proceedings*, pages 97–113. Springer Berlin Heidelberg, 2008.

[135] W. Shang, Z. M. Jiang, B. Adams, A. E. Hassan, M. W. Godfrey, M. Nasser, and P. Flora. An exploratory study of the evolution of communicated information about the execution of large software systems. *Journal of Software: Evolution and Process*, 26(1):3–26, 2014.

[136] K. Singh, M. Bhadauria, and S. A. McKee. Real time power estimation and thread scheduling via performance counters. *ACM SIGARCH Computer Architecture News*, 37(2):46–55, 2009.

[137] D. Sobhy, R. Bahsoon, L. Minku, and R. Kazman. Diversifying software architecture for sustainability: A value-based perspective. In B. Tekinerdogan, U. Zdun, and A. Babar, editors, *Proceedings of the 10th European Conference on Software Architecture*, pages 55–63. Springer, 2016.

[138] P. Somavat, V. Namboodiri, et al. Energy consumption of personal computing including portable communication devices. *Journal of Green Engineering*, 1(4):447–475, 2011.

[139] B. Steigerwald and A. Agrawal. *Green Software*, pages 39–62. John Wiley & Sons, Ltd, 2012.

[140] Y. Sun, Y. Zhao, Y. Song, Y. Yang, H. Fang, H. Zang, Y. Li, and Y. Gao. Green challenges to system software in data centers. *Frontiers of Computer Science in China*, 5(3):353–368, 2011.

[141] J. Taina. How green is your software? In *Software Business*, volume 51 of *Lecture Notes in Business Information Processing*, pages 151–162. Springer Berlin Heidelberg, 2010.

[142] S. te Brinke, S. Malakuti, C. Bockisch, L. Bergmans, and M. Akşit. A design method for modular energy-aware software. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, pages 1180–1182. ACM, 2013.

[143] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, 58(1):267–288, 1996.

[144] A. E. Trefethen and J. Thiyagalingam. Energy-aware software: Challenges, opportunities and strategies. *Journal of Computational Science*, 4(6):444–449, 2013.

[145] V. Viswanath, M. G. Morris, G. B. Davis, and F. D. Davis. User acceptance of information technology: Toward a unified view. *MIS Quarterly*, 27(3):425–478, 2003.

[146] T. Vogelsang. Understanding the energy consumption of dynamic random access memories. In *2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 363–374, 2010.

[147] M. D. Williams, N. P. Rana, Y. K. Dwivedi, and B. Lal. Is UTAUT really used or just cited for the sake of it? A systematic review of citations of utaut's originating article. In *ECIS 2011 Proceedings*, 2011.

[148] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wessln. *Experimentation in Software Engineering*. Springer, 2012.

[149] L. Xiao, Y. Cai, R. Kazman, R. Mo, and Q. Feng. Identifying and quantifying architectural debt. In *Proceedings of the 38th International Conference on Software Engineering*, pages 488–498. ACM, 2016.

[150] L. Xu and S. Brinkkemper. Concepts of product software. *European Journal of Information Systems*, 16(5):531–541, 2007.

[151] Q. Yang, J. J. Li, and D. M. Weiss. A survey of coverage-based testing tools. *The Computer Journal*, 52(5):589–597, 2009.

[152] R. Yin. *Case Study Research: Design and Methods*. Applied Social Research Methods. SAGE Publications, 2009.

[153] G. Zhang, K. Zhang, X. Zhu, M. Chen, C. Xu, and Y. Shao. Modeling and analyzing method for cps software architecture energy consumption. *Journal of Software*, 8(11):2974–2981, 2013.

[154] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, and L. Yang. Accurate online power estimation and automatic battery behavior based power model generation for smartphones. In *Proceedings of the Eighth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, pages 105–114. ACM, 2010.

[155] B. Zhong, M. Feng, and C.-H. Lung. A green computing based architecture comparison and analysis. In *Proceedings of the 2010 IEEE/ACM International Conference on Green Computing and Communications & International Conference on Cyber, Physical and Social Computing*, pages 386–391. IEEE Computer Society, 2010.

[156] H. Zhu, C. Lin, and Y. Liu. A programming model for sustainable software. In *Software Engineering (ICSE), 2015 IEEE/ACM 37th IEEE International Conference on*, volume 1, pages 767–777, 2015.

Publication List

E. Jagroep, A. van der Ent, J. M. E. M. van der Werf, S. Brinkkemper, L. Blom, and R. van Vliet. Hunt the energy guzzler in my software: An architectural approach. *Submitted for publication*

E. Jagroep, J. Broekman, J. M. E. M. van der Werf, S. Brinkkemper, P. Lago, L. Blom, and R. van Vliet. Awakening awareness on energy consumption in software engineering. In *Proceedings of the 39th International Conference on Software Engineering: Software Engineering in Society Track*, pages 76–85. IEEE Press, 2017

E. Jagroep, G. Procaccianti, J. M. van der Werf, S. Brinkkemper, L. Blom, and R. van Vliet. Energy efficiency on the product roadmap: An empirical study across releases of a software product. *Journal of Software: Evolution and Process*, 29(2), 2017

E. Jagroep, J. M. van der Werf, S. Brinkkemper, L. Blom, and R. van Vliet. Extending software architecture views with an energy consumption perspective. *Computing*, 99(6):553–573, 2017

E. Jagroep, J. M. E. M. van der Werf, J. Broekman, S. Brinkkemper, L. Blom, and R. van Vliet. A resource utilization score for software energy consumption. In *Proceedings of the 4th International Conference on ICT for Sustainability*, pages 19–28. Atlantis Press, 2016

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

E. Jagroep, J. M. E. M. van der Werf, R. Spauwen, L. Blom, R. van Vliet, and S. Brinkkemper. An energy consumption perspective on software architecture. In D. Weyns, R. Mirandola, and I. Crnkovic, editors, *Software*

Architecture: 9th European Conference, ECSA 2015, Proceedings, pages 239–247. Springer International Publishing, 2015

E. Jagroep, J. M. E. M. van der Werf, S. Jansen, M. Ferreira, and J. Visser. Profiling energy profilers. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, pages 2198–2203. ACM, 2015

E. Jagroep, I. van de Weerd, S. Brinkkemper, and T. Dobbe. Framework for implementing product portfolio management in software business. In G. Ruhe and C. Wohlin, editors, *Software Project Management in a Changing World*, pages 193–221. Springer Berlin Heidelberg, 2014

E. Jagroep, I. van de Weerd, S. Brinkkemper, and T. Dobbe. Implementing software product portfolio management. In *2011 Fifth International Workshop on Software Product Management (IWSPM)*, pages 67–76, 2011

Summary

The rising energy consumption of the ICT industry has triggered a quest for more energy efficient ICT solutions. Over the years, the role of software as the true consumer of power and its potential contribution to reach sustainability goals has increasingly been acknowledged. At the same time, it is shown to be a complex problem to address. This research focuses on enabling software producing organizations, e.g. independent software vendors and open source foundations, to deliver green software products. Therefore, the main research question in this PhD dissertation is:

How can software producing organizations be in control of the energy consumption of their software products?

This research question is answered in three parts. A prerequisite for the research is the ability to measure the energy consumption of software products. Therefore, the first part of the dissertation presents the *evaluation of energy profilers*. Software producing organizations can use the evaluation as guidance to select an energy profiler based on its functional capabilities and the required accuracy of the measurements. To further improve the reliability of energy consumption measurements, the performance dimension was included to create a *resource utilization score* for software energy consumption. This score allows for objective comparison of application configurations and versions, and visualization of measurements to enhance communication. The results were evaluated through multiple experiments involving different hardware platforms and software products.

The second part of this dissertation relates energy consumption to software architecture and proposes sustainability as a quality characteristic of software products. This enables objective quantification of software energy consumption and making better informed trade-offs with current quality characteristics as defined in, e.g., the ISO25010 standard. The proposed *energy consumption perspective* guides the software architect in the analysis of energy efficacy on

architectural level, which enables the identification of the true energy guzzlers in a software product. To support the perspective, this dissertation presents the *stubbed energy profiling method* a concrete method to unravel software energy consumption using stubs. The energy consumption perspective and the stubbed energy profiling method were evaluated in a case study and an experiment, both using commercial software products.

The last part of the dissertation studies energy consumption, and its consequences and effects in an industrial context to create awareness on the topic. In two empirical studies, the *software energy profiling method* was applied on consecutive releases of a commercial software product. This provided an in-depth analysis of the energy consumption of software components, and identified the added value of energy consumption measurements for multiple stakeholders in a software producing organization. In addition, an exploratory case study was performed on how to create and maintain awareness among stakeholders involved with the development of software products. Quantifying and presenting the energy consumption of the software triggered discussion, and helped to create awareness for considering energy aspects when planning software releases.

Combined, the results and findings from all three parts provide a software producing organization the knowledge and tools to be in control of the energy consumption of their software products throughout the complete software lifecycle.

Samenvatting

De groeiende energiebehoefte van de ICT industrie heeft een zoektocht ontketend naar duurzame, energievriendelijke, ICT oplossingen. In toenemende mate wordt de rol van software als energiegebruiker erkend en daarmee de bijdrage van software in het behalen van duurzaamheidsdoelstellingen zoals het klimaatakkoord van Parijs. Tegelijkertijd blijkt het energiegebruik van software een complex vraagstuk. Dit onderzoek richt zich op het in staat stellen van software producenten, bijvoorbeeld independent software vendors en open-source organisaties, om duurzame software producten te ontwikkelen. De hoofdonderzoeksvraag voor deze dissertatie is dan ook:

Hoe kunnen software producerende organisaties de controle krijgen over het energiegebruik van hun software producten?

De onderzoeksvraag wordt in drie delen beantwoord. Het eerste deel van deze dissertatie richt zich op het kunnen meten van het energiegebruik van software producten; een voorwaarde voor dit onderzoek. Hiertoe wordt een *evaluatie van energy profilers* gepresenteerd. Software producenten kunnen de evaluatie als leidraad gebruiken bij het selecteren van een energy profiler op basis van de geboden functionaliteit en nauwkeurigheid van de metingen. Om de betrouwbaarheid van de metingen te verbeteren, worden ook performance aspecten meegenomen waardoor het energiegebruik kan worden uitgedrukt in een *resource utilization score*. De score stelt belanghebbenden in staat om objectieve vergelijkingen te maken tussen verschillende versies en configuraties van applicaties, en biedt handvatten voor het visualiseren van metingen ter bevordering van de communicatie. De energy profilers en de score zijn geëvalueerd in meerdere experimenten waarin gebruik is gemaakt van verschillende hardware platformen en software producten.

Het tweede deel van deze dissertatie relateert energiegebruik aan software architectuur en positioneert duurzaamheid als kwaliteitsattribuut van software producten. Hierdoor kan het energiegebruik van software objectief worden

gekwantificeerd en kunnen beter onderbouwde trade-offs worden gemaakt met huidige kwaliteitsattributen, bijvoorbeeld zoals in de ISO25010 standaard. De gepresenteerde *energy consumption perspective* stelt de software architect in staat om energie-analyses uit te voeren op architectuur-niveau, en daardoor de belangrijkste energiegebruikende elementen van een product te identificeren. Ter ondersteuning van de perspective wordt tevens de *stubbed energy profiling method* gepresenteerd; dit is een concrete methode om het energiegebruik van software te ontrafelen middels stubs. De energy consumption perspective en de stubbed energy profiling method zijn geëvalueerd in een case study en een experiment, welke beiden zijn uitgevoerd met commerciële software producten.

Het laatste deel van de dissertatie is gericht op het creëren van bewustwording over het energiegebruik van software in een industriële context. In twee empirische studies is de *software energy profiling method* toegepast op achtereenvolgende releases van een commercieel software product. Naast gedetaileerd inzicht in het energiegebruik van software componenten, is inzicht verkregen in de toegevoegde waarde van energiemetingen voor verschillende belanghebbenden binnen een organisatie. Daarnaast is een exploratieve case uitgevoerd om bewustwording te creëren en behouden bij de betrokkenen in het software ontwikkelproces. Het kwantificeren en presenteren van het software energiegebruik heeft geleid tot levendige discussies over het onderwerp en het in acht nemen van het energiegebruik tijdens het plannen van toekomstige ontwikkelingen.

De resultaten en bevinding van de drie delen bieden een software producent de handvatten en middelen om controle te krijgen over het energiegebruik van hun software producten, gedurende de gehele levenscyclus van de software.

Curriculum Vitae

Erik Jagroep was born March 7th, 1987, in Leidschendam, the Netherlands. From 2005 to 2008, he studied Information Science at Utrecht University, where he received his Bachelor of Science degree. In the years that followed, he continued with the master Business Informatics, which resulted in a Master of Science degree in 2011.

He started his PhD research in June 2011 as an external researcher at Centric Netherlands B.V. At first in the role of quality consultant and product manager, focused on improving the quality of the software products and associated processes, promoting his research throughout the organization. The evolution of Corporate Social Responsibility within the software industry increasingly provides the opportunity to put his scientific knowledge in practice. After five years he also became product owner of a company-wide initiative to develop generic, re-usable software components. His most recent challenge involves a transition towards a software as a service based portfolio and the development of value adding products and services.

The research and educational activities of Erik Jagroep focus on software architecture, sustainable development and green software products. In addition he provides lectures and coaching in these areas.