Green Software and Green Software Engineering – Definitions, Measurements, and Quality Aspects

Eva Kern[1], Markus Dick[2], Stefan Naumann[1], Achim Guldner[1], Timo Johann[2]

[1]Institute for Software Systems, Environmental Campus Birkenfeld, Campusallee, D-55761 Birkenfeld
e.kern@umwelt-campus.de, s.naumann@umwelt-campus.de, a.guldner@umwelt-campus.de
[2]sustainablesoftwareblog@gmail.com - http://sustainablesoftware.blogspot.de/

ABSTRACT

The two big fields of sustainability and Information and Communication Technology (ICT) are Green IT (how can we make ICT itself more sustainable) and Green by IT (how can we encourage sustainability by ICT). Taking a deeper look, software links these two areas: Regarding Green IT, there are a lot of solutions to build and use hardware in a more energy efficient way. But the debate how energy-intensive software might be is just beginning. In contrast, Green by IT is often software-based, e.g. by tools that help to optimize logistics and automate processes to save energy. But until now there are no considerations about the energy saving potential of software itself. Therefore, it is useful to take a closer look at what green software and green software engineering are. In our paper, we will describe a reference model for green and sustainable software, as well as its engineering and also give some definitions. Though, we will just give a short introduction of the model itself and, to distinguish our work from our previous research, zoom in on the sub model "Sustainability Criteria for Software Products". Additionally, we describe a model to measure the energy efficiency of software and give an example of measuring results in our contribution. The next step is to clearly differentiate from other measurement models to position our approach within other efforts of software's energy consumption.

Keywords

Green Software; Green IT; Green by IT; Software Engineering; GREENSOFT Model

1. INTRODUCTION

By now, Sustainable Development (SD) found its way into different fields of our life. Regarding Information and Communication Technology (ICT), especially the aspects Green IT and Green by IT describe the efforts towards Green and Sustainable Information Technologies. In that context, ICT brings new opportunities as well as problems for the goal of SD [14]. Green IT considers the resource and energy consumption of ICT itself, induced during the whole life cycle, and tries to optimize it. Here, current concepts mainly focus on the hardware side, especially regarding data centers. The Green by IT concept pursues the objective of reducing the environmental impacts in other fields by means of ICT solutions [13].

These two concepts are linked by software. Most of the ICT solutions are software-based, so that it seems to be expedient to factor the software side into the ICT for sustainability discussions.

ICT4S 2013: Proceedings of the First International Conference on Information and Communication Technologies for Sustainability, ETH Zurich, February 14-16, 2013. Edited by Lorenz M. Hilty, Bernard Aebischer, Göran Andersson and Wolfgang Lohmann.
DOI: http://dx.doi.org/10.3929/ethz-a-007337628

2. RELATED WORK

2.1 Measurement of Software

Regarding the measurement of software the group around Capra [6, 7] did a lot of research and experiments. Based on their theoretical analyses, they proposed a research roadmap to compare different software applications in their energy consumption. As a first step, they focused on the cause of energy consumption by information systems. Going on, Capra et al. [6] compared the energy consumption of different management information systems (MIS). They found that the software induced power consumption depends on the implementation of the operation systems, the runtime environment, and the software product itself. Hence, it is not possible to give specific recommendations which software is to be chosen to reach the best energy efficiency. Besides these, they came up with the result that the influence of using application development environments on energy efficiency is more distinct for larger than for smaller applications [7]. Apart from the relationship of energy efficiency to the use of external libraries and high level constructs, they compared different open source tools in their size, age, and functional types with the result that the application types that appear to be less energy efficient are those that present higher degrees of functional complexity.

Based on the appraisal that the mean utilization of real-world database servers is far from common benchmarking results, Schall et al. [22] proposed a new benchmarking paradigm extending the already existing TPC-Energy benchmarks. They aimed to comprise the overall power consumption of the System Under Test (SUT). The introduced benchmark can be run with workloads to get a more realistic energy-related result.

The overall approach is also followed by Jwo et al. [18], but within another scope: They presented a model to capture the energy consumption for enterprise applications. In doing so, they wanted to give an organization a better chance to understand its consumption caused by ICT and to optimize their applications, in order to reduce the overall energy consumption.

Assuncao et al. [4] had a look at the infrastructure of Grids and Clouds to reduce their energy footprint. They observed the electricity consumption of an experimental grid infrastructure and its correlations with users' resource reservation requests for a period of six month. All in all, they demonstrated that there are huge possibilities to save energy by switching off unused resources. In the context of Cloud Computing, Chen et al. [8]

presented an energy consumption model and an analysis tool for Cloud environments. The tool characterizes energy consumed by each task and helps to identify the relation between energy consumption, running tasks, the platform configuration, and system performance.

In order to identify power behavior associated with source code, Wang et al. [33] developed "SPAN", a software power analyzer to provide live power phases information of running applications. With it, developers can identify the sections of code consuming the most power in the program.

One of the big players regarding energy consumption of ICT are communication systems running 24/7. In this area, also software and system architectures play a significant role. Zhong et al. [34] presented a model to analyze the energy consumption of a software architecture adapted from the CPU/processor. Getting to know what kind of architecture needs less energy enables one to reduce the overall energy on the communication network.

In order to get an overview, we summed up the described approaches on measuring the energy consumption of ICT in Table 1. A more detailed survey of efforts on power measurement is presented in [33].

Table 1 Comparison of the Approaches on measuring energy consumption of ICT

Approach by	Subject	Objective
Wang [33]	Multicore Computer Systems	Identifying run-time factors that determine the power wastage of processors for computing intensive workloads
Assuncao [4]	Grids and Clouds	Reducing the energy footprint of Grids and Clouds
Chen [8]	Grids and Clouds	Identifying the relationship between energy consumption, running tasks, configurations, and performance
Schall [22]	Infrastructure	Comprising the overall power consumption of the SUT
Capra [5, 7]	Open Source	Comparing the energy efficiency of different software
Jwo [18]	Enterprise Applications	Understanding the energy consumption caused by using ICT by the organization
Zhong [34]	Software Architecture	Comparing competing software architectures by an additional dimension
Dick [9]	Server and Application Software	Finding recommendations for Software Engineers

2.2 Energy Efficient Software and Recommendations

The above described measurement methods mainly aim to produce more energy efficient software. Overall, the consideration of energy efficient software gives a rise to green and sustainable software engineering. In addition to the mentioned recommendations in the measurement context, there are different checklists and guidelines available, e.g. [24], [23], [11], and [20].

Moreover, Steigerwald [27] (Intel) described software methodologies, designs, and software development tools that help to improve the energy efficiency of application software, focusing on mobile platforms. Additionally, they considered Computational Efficiency (methods to improve application performance), Data Efficiency (minimizing data movement and using the memory hierarchy effectively), and advantages of the resources offered by the operation system to save energy.

2.3 Green Software and its Engineering

In most of the cases, the reason for establishing energy efficient software and systems is to achieve a longer battery life or to reduce costs. On top of that, moving to the ecological part of sustainability, there is the potential to decrease energy and resource consumption to support a SD.

A first impression on how software influences the life cycle of the hardware by requiring more and more resources is given in [15]. The so called "Software Bloat" denotes the effect that the availability of more powerful hardware in the near future, relaxes software developers' efforts to produce highly efficient code. This is due to the fact, that new hardware is quickly available and inexpensive and will hopefully overcompensate inefficient code. As a result, hardware is replaced by new hardware before its useful lifetime ends.

A methodology to measure and incrementally improve the sustainability of software projects is presented by Albertao et al. [2]. It is advisable to implement sustainable aspects continuously, divided into the following phases: Assessment Phase, Reflection Phase, and Goal Improvement Phase. In order to make the different sustainability issues manageable, he pointed out properties of a quality model that are further developed in a later work [3]. These aspects are also integrated in the Quality Model for Green and Sustainable Software, described in section 5 of this paper.

Agarwal et al. [1] took a look behind the definition of Green and Sustainable Software Engineering [21] and discussed possibilities and benefits of green software. As one of the results, they claimed more efficient algorithms will take less time to execute and thus overall support sustainability. Additionally, they presented methods to develop software in a sustainable way and compared these to conventional methods.

Based on the life cycle of software, Taina proposed metrics [29] and a method to calculate the carbon footprint of software [28]. To do so, he analyzed the impacts of each software development phase for a generic project. The resulting carbon footprint is mainly influenced by the development phase, but also by the way how it is delivered and how it will be used by the customers. The main problem regarding the calculation is that detailed data is required, which is often not available.

3. THE GREENSOFT MODEL

In order to classify and sort some aspects of green and sustainable software and its engineering, we developed the GREENSOFT Model, shown in Figure 1.

The GREENSOFT Model contains four parts: The life cycle of a software product, some criteria and metrics that represent sustainability aspects that are directly and indirectly related to the software product, procedure models for the different phases, and some recommendations for action, as well as tools.

The life cycle of software is geared to Life Cycle Thinking (LCT) according to the "from cradle to grave" principle [31]. The intention is to estimate the ecological, social, and economic impacts that already occur in early stages during the software's whole life cycle. Details of our approach are given in [9].

Additionally, the model comprises Sustainable Criteria and Metrics for software products. Especially regarding measurements of common effects of software products, there exist quality models and standardized metrics. By means of these quality aspects, software can be revised. Indeed models and characteristic numbers of directly and indirectly related effects need to be developed by research initiatives. We will present a first approach to do so in section 4.

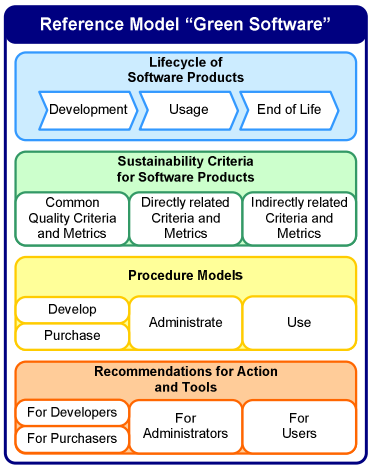

Figure 1 The GREENSOFT Reference Model [21]

The third sub model contains procedure models, based on the different usage types: Developers, purchasers, administrators, and users. The proposed models can be implemented to support the optimization of the different processes focusing on green and sustainable software engineering. In that way, software engineering should become green and sustainable in its production, support, and application processes. Since the models are general, they can be adapted to different contexts.

Finally, recommendations for action and tools are made available to the different stakeholders. These parts comprise checklists, guidelines, best practice examples, software tools, as well as other tools (like paper-based data collection sheets). These support stakeholders with different professional skill levels in applying green or sustainable techniques in general, when developing, purchasing, administrating, or using software products. Possible roles are software developers, acquirers of software, administrators, and professional and private users [21].

The different aspects of green and sustainable software engineering are summed up in the following definition: "*Green and Sustainable Software Engineering* is the art of developing green and sustainable software with a green and sustainable software engineering process. Therefore, it is the art of defining and developing software products in a way, so that the negative and positive impacts on sustainable development that result and/or are expected to result from the software product over its whole life cycle are continuously assessed, documented, and used for a further optimization of the software product." [9]

4. QUALITY MODEL FOR GREEN AND SUSTAINABLE SOFTWARE

In order to decide if a software is green and sustainable or not, appropriate criteria and metrics are required. A first approach of these is covered by the Quality Model for Green and Sustainable Software, presented in the following section.

4.1 Criteria

Overall, the different aspects can be classified in common, directly, and indirectly related criteria and metrics, as pictured in our Quality Model for Green and Sustainable Software (Figure 2). Regarding the GREENSOFT-Model, described in section 3, the model goes into the sub model "Sustainability Criteria and Metrics".

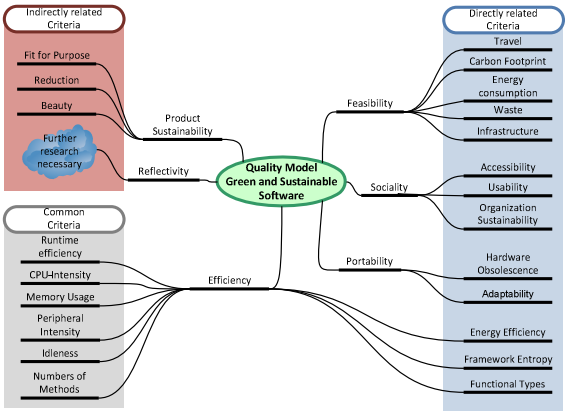

Figure 2 Quality Model for Green and Sustainable Software

In the following, we will present a first approach to developing a Quality Model for Green and Sustainable Software. In order to create the model, we will discuss different publications regarding quality aspects, criteria, and indicators for sustainable software. These will be compared, combined, and summed up in the proposed model. This first approach intends to get an impression and overview of possible criteria. As a next step, the aspects should be prioritized, completed by tangible examples and discussed properly.

4.1.1 Common Quality Criteria

The common criteria arise out of the well-known and standardized quality aspects for software, issued by the International Organization for Standardization [16]. The proposed Quality Model takes aspects, like *Efficiency,* into account as well as e. g. *Reusability*, *Modifiability*, and *Usability*. These quality aspects reach into the field of SD [3].

The quality aspects belonging to efficiency, next to *Energy Efficiency*, are *Runtime Efficiency*, *CPU-Intensity*, *Memory Usage*, *Peripheral Intensity*, *Idleness*, and *Number of Methods*. In this case, *Idleness* describes how often the system is idle. This aspect is only relevant to certain types of software systems, such as virtual servers [29]. The total *Number of Methods* reflects the size of applications [7].

4.1.2 Directly related Criteria

In contrast to these efficiency criteria, which can be considered as indicators for *Energy Efficiency* [19, 33], the energy efficiency of a system itself can be ranked against a reference system [10, 32]. Hence, the quality aspect *Energy Efficiency* is assigned to the directly related criteria [21]. It is the same for the criteria *Framework Entropy* and *Functional Types*, presented by Capra et al. [7]. The *Framework Entropy* represents the grade of usage of external libraries and high level constructs. It is high, if the usage of external sources is high. The different types of applications are indicated by the criterion *Functional Types*, since different functional types of applications have different levels of energy efficiency [7].

Another aspect of the directly related criteria is *Hardware Obsolescence* [3, 15, 21]. It considers the amount of hardware that needs to be replaced before the sustainably worthwhile life-time is reached, due to software with higher system requirements. Against the background that the energy supply changes to more and more regenerative energy sources and the development to more energy efficient hardware, the life-time will be extended increasingly. The software development needs to keep up with these changes.

The hardware obsolescence comes along together with the quality aspect *Adaptability*. Both aspects belong to the criterion *Portability*. Here, *Adaptability* describes the possibilities to adapt the software to the specific consumer needs. In case a user just needs to have the basic functionality, it is not necessary to run large software and consume a huge amount of energy and resources. Additionally, the performance requirements and the amount of used memory are lower.

The quality aspect *Feasibility* valuates the effect of the product development or rather the software engineering process regarding SD. The substantial issues of this characteristic comprise *Travel* [3], the *Carbon Footprint*, the *Energy Consumption*, the *Waste* of resources during the software development phase [21, 29], and the *Infrastructure*. Traveling includes the business trips, belonging to a software project and, where appropriate, the daily trips to work. The *Carbon Footprint* specifies the amount of CO_2 emissions caused by the software development during its life cycle [28]. The aspect *Energy Consumption* indicates the consumed energy caused by software development. *Waste* counts the resources consumed during software development for activities without any additional value for the customer or the end user [29]. The aspect *Infrastructure* contemplates the conditions providing the usage of the software product.

Out of the perspective of the social aspect of SD, there might be some more qualities that require special measurement and valuation methods. As a first step, the aspects are called *Accessibility*, *Usability* and *Organization's Sustainability*. In contrast to the first two aspects, the sustainability of the organization itself is considered by the last one. It covers the social situation, including e.g. working conditions or payment of offshore workers (e.g. developers, type setters), which have consequences for the workers and their families [21]. In order to get the required data, a sustainability report can be consulted in case of a big company. Indeed, the acquisition of theses aspects might be difficult in case of small and medium sized enterprises. Overall, there needs to be more research in this context to revise this first proposal of an idea to bring the social aspects into the discussion for criteria for green and sustainable software.

4.1.3 Indirectly related Criteria

Using software can gain reduction of resources and energy in other branches or application domains. Although it is occasionally difficult to identify and qualify the usage effects and systemic effects, there might be effects that are measureable or appreciable. This applies to software, which aims to accomplish a purpose regarding SD. Examples for such kinds of software are software products enabling smart technologies. In this case, the expected effects can be estimated during product development, e.g. by life cycle assessment. Besides the area of environmental sustainability, regarding the social compatibility, a qualifying consideration is getting slightly harder and requires methods that are more complex [21].

Product Sustainability summarizes the effects of software on other products and services. Hence, it covers effects of use as well as systemic effects. The following aspects are presented by Taina [29]. *Fit for Purpose* takes the fitness of software for its specific operation purpose into account. For example, an optimized heater produces 20% less emission. If the optimized heating control system contributes 10% to that, the value for the aspect *Fit for Purpose* is 10%. The contribution of software within its application domain to reduce the emissions, waste, etc. is indicated by the aspect *Reduction*. *Beauty* aims to value the software regarding SD. In the case of a piece of software that raises the awareness for the climate change by 15%, the value for beauty is 15% [29]. Indeed, since a reference system is required to have a comparative figure, the exact quantification of these aspects might be difficult.

According to Taina [29], the quality *Reflectivity* refers to the quality aspect of efficiency. It specifies the manner how the usage of software influences the resources that are indirectly needed by other persons or systems. Since the aspect *Reflectivity* does not belong to the first order effects or the efficiency effects of ICT in the proper meaning, but counts as an effect of use, it is pointed out as standalone quality of the indirectly related criteria. However, more research is required in this field because user behavior and usage scenarios need to be taken into account.

4.2 Metrics

The different aspects need to become measureable by means of metrics and indicators. In general, software engineering is already aware of these aspects, even so the awareness of these qualities needs to be emphasized in the context of green and sustainable software engineering.

There are already a few metrics available to gather some quality aspects: Albertao et al. indicated *Estimated System Lifetime* as a potential index for hardware obsolescence. According the aspect Travel, *Work from Home Days*, *Long-Haul Roundtrips* and *Teleconferencing* are possible metrics [3].

Regarding efficiency, the following metrics might be useful:

Aspect	→ Metric
• Energy Efficiency	→ Energy / Unit of Work
• CPU-Intensity	→ CPU Cycle Count
• Memory Usage	→ Memory Consumption
• Peripheral Intensity	→ Peripheral Usage Time
• Idleness	→ Idle Time

In contrast to other criteria, especially the directly related criteria, energy efficiency is easily quantifiable. Hence, from this aspect, it is achievable to extrapolate metrics to validate and verify software products. On closer consideration of efficiency, it is not possible to define the benefit of software in an exact way. Whereas this benefit depends on specific use cases and can be defined in different ways. Additionally, most of the software products contain much functionality, so that more than one benefit might be possible. Contrary to that, the cost regarding energy efficiency is simply the energy consumption. Certainly, the system boundary needs to be clearly defined in this context.

Here, different approaches exist: Energy benchmarks, energy measurement with individual use case scenarios (black box), and energy measurements with code instrumentation (white box). Common benchmarks for energy efficiency are EnergyBench [30] according micro controller, SpecPower [26] according server hardware or TPC-Energy according databases [22, 32]. Overall, all of the benchmarks produce a standardized workload

on a given system and measure the energy consumption simultaneously. Though, just in case of TPC-Energy, a specific metric regarding the energy efficiency of software exists.

Another approach is to measure the energy consumption by means of specified use cases and compare different types or configurations of software [10]. In this case, an appropriate use case scenario for the type of software is required. This scenario is applied to the different systems and the energy consumption is measured simultaneously. Based on the results, software systems can be compared regarding their energy efficiency for every use case. This method is a black box measurement, since the software system is measured and compared as a whole. Certainly, it does not denote an insight into the software itself to point out the reason of the energy consumption or the specific part of the software that is responsible for the results. One possible approach for a black box measurement method is presented in section 5.

In contrast to the black box method, the white box method takes a look at the code by code instrumentation. In that way, internal data can be acquired and taken as indicators to create metrics. An example for a generic metric is

$$\frac{Useful\ Work\ Done}{Consumed\ Energy}$$

In a following step, this metric can be further granulated to consider the fact that a software system consists of different sub systems, measured separately or as a whole. Hence, it is possible to optimize subsystems, consuming a huge amount of energy, in an early state of the development phase. Additionally, it can be helpful to find requirements that can be validated, to create metrics.

4.3 Discussion

In order to find criteria and metrics to compare different software products regarding ecological and sustainable aspects, several conditions need to be reflected on.

The intention of the *Social Aspects* is to include the enterprise in the ration of software. With the fact that that sustainability comprises environmental, economic, and social aspects in mind, these suggestions might be comprehensible. Indeed, this limits the range of possibilities, e.g. regarding the kinds of software. In the case of open source, it is virtually impossible to value the sustainability of programming communities. Besides huge enterprises, these also include small enterprises and single persons that do not have a sustainability report in most of the cases. Hence, the comparability is limited to a small range of software.

In general, it needs to be discussed, if open source projects are examined separately, since the situation is nearly the same regarding the measurement of the energy consumption during the development process. A lot of programmers work on many different work stations and they use various items of equipment, as well as a tool to organize the collaboration. Hence, it might be difficult – or rather nearly impossible – to collect all of the data needed. On the other hand, those projects have an advantage regarding criteria, like *Numbers of Methods*, *Framework Entropy,* and *Functional Types*. It might be easier to get this data or rather the source code itself, what might compensate the restrictions in the acquisition of data.

A solution to cover the different kinds of software (e.g. server based applications, Cloud Computing, etc.) might be to further develop the criterion *Functional Types*, presented by Capra et al. [7]. With it, the different kinds of software with all their characteristics, limitations, and properties will be considered.

Taken as a whole, if software is rated in its quality regarding the dimensions of sustainability, the considered framework needs to be defined in the first place. Another strategy is to differ between the green qualities and the sustainable ones. That is to say to divide the different aspects into the three pillars of sustainability: social, economic, environmental; conceivably supplemented by the potential cross-cutting concerns.

5. MEASUREMENT OF SOFTWARE INDUCED ENERGY CONSUMPTION

When measuring the energy consumption of software, one problem is to decide, what and how to measure. The first point is connected to the fact that software normally does not work by itself, but depends on the hardware. The operating system, and background processes also induce energy consumption and have to be subtracted. A second problem is that there is no absolute data about the energy consumption of a special software and that software has a broad variety of tasks. Consequently, we suggest two ways for evaluating measurement results: At first, you can compare software products with other software of similar tasks, e.g. web browsers of different vendors. Secondly, you can compare different versions of the same software product, in order to see whether the energy consumption changed significantly.

5.1 Measurement Method

Regarding the second point, we developed a test rig and a measurement method. Figure 3 demonstrates the overall architecture.

The SUT is the computer hosting the application program, whose induced energy consumption will be evaluated. It is defined by the combination of hardware components that make up the computer system, the operating system, the runtime environment, and the application program.

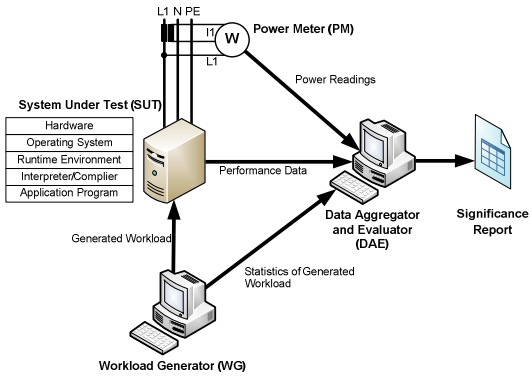

Figure 3 Architecture of the measurement setup [10]

The Workload Generator (WG) is a computer program that generates the statistically reproducible workload that is applied to the SUT. Depending on the type of application program, it is either run directly on the SUT, e.g. for stand-alone applications or on one or more separate computers, e.g. as clients that access an interactive transaction-based application on a remote server.

The Power Meter (PM) is an appropriate energy or power meter whose readings can be read out remotely to be aggregated and evaluated by the Data Evaluator and Aggregator (DAE). It is only connected to the SUT. The energy consumption of clients

that access a remote server is not considered. Instead, clients require their own measurement.

The DAE collects all data and evaluates it. This includes the power or energy readings, the performance readings from the SUT, and the workload execution statistics from the WG. After the data is collected and aggregated, it will be evaluated. In the case of two competing SUTs, a statistical significance report is generated that states, which SUT has the lower energy consumption and therefore is more energy efficient. If there is only one SUT, the measured data could be used to be visualized in graphs, in order to support developers in finding software components that should be optimized.

5.2 Exemplary Measurement Results

Based on a three-layered architecture for client-server-applications, we looked at the energy and resource consumption of a Web Content Management System (WCMS), by comparing two different configurations of the WCMS Joomla![TM]: The first configuration uses a file based cache for results of database queries and generated HTML fragments. This configuration is compared to the base-configuration that does not use the caching function. With this, the database is requested for every page view and the content is always created dynamically.

The workload was generated with Apache JMeter, simulating 67 contemporary users. A Supermicro Server (P4BP8-G2) with two Intel Xeon dual core CPUs CPUs (@2.4 GHz, 4 GiB RAM, 40 GB HDD) running Ubuntu GNU/Linux Server (SMP 10.04 LTS, Kernel 2.6.32-32-generic-pae) was used for the SUT. The tested application was Joomla! (1.5.23, http://www.joomla.org/) with PHP 5.3.2, MySQL 5.1 and Apache httpd 2.2.14.

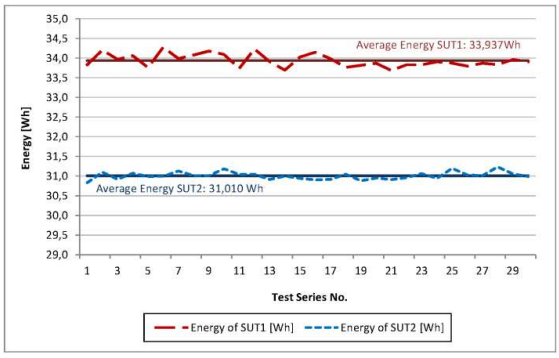

Figure 4 Results of the energy measurement of Joomla! without (upper line) and with (lower line) HTML Cache over 30 test series

The result of the evaluation of the measured power data can be interpreted as follows: On average, the SUT supporting HTML-Cache consumes less energy (N = 30, mean = 31.009 Wh, standard deviation = 0.096 Wh) than without HTML-Cache (N = 30, mean = 33.937 Wh, standard deviation = 0.163 Wh). These results are depictured in Figure 4. The difference was significant with p-value = 6,62e-53 < 0,001. In this case, using the HTML-Cache saves approx. 8,6% of electrical energy. Additionally, approx. 19% of reserve capacity could be generated.

5.3 Discussion of the Measurement Method

A plenty of different approaches for the energy measurement of software were developed and presented in the last decade, as was already discussed in the related work section above.

Often, the presented approaches are very hardware-centric, as they try to develop a sophisticated and well-fitting power model for different kinds of processors and hardware platforms, e.g. ranging from [25] for embedded systems to prolong battery life up to [33]. The idea behind the latter approach is to estimate the power consumption of a computer system by using performance counters and a power model that is suitable for the used hardware instead of applying a power metering device to the power lines of the computer system. A similar approach was also presented by Kansal [19], who estimated the power consumption of several virtual machines running on one server hardware.

All these approaches do not provide a method that can be used by software developers to improve the energy efficiency of their software, because in principle they developed a software implemented replacement for a power metering device.

A method and a SDK introduced by Steigerwald [27] (Intel) was advertised to support software developers in creating energy efficient applications. They use counters and events embedded into the source code of the application to match activities of the application to a power graph recorded by a power meter. This approach is suitable if one does not make intensive use of concurrent software execution, because then it is not possible to distinct which part of the software accounts for which amount of power consumption.

This is also a common problem when measuring a web server/application or any other kind of MIS. Here, it is usually necessary to measure the system under realistic operational conditions, because design flaws or bottlenecks in software design may increase the energy demand more likely under higher workloads with concurrent requests. Hence, it is necessary to simulate realistic workloads that are expected to occur during system operation. These workloads usually consist of a huge amount of parallel requests, executed in a well defined time frame.

Such an approach is used by the TPC benchmarks and their corresponding TPC energy metric [32]. The benchmarks are designed to imitate a typical workload of an online shop system, serving several shops in parallel with their typical user transactions. Capra et al. [6] made use of the same TPC benchmark to measure the energy consumption of different Database Management Systems (DBMS). They did however not apply the TPC energy metric. To measure the other kinds of software (ERPs, CRMs), they developed their own suitable benchmarks. As was discussed by Dirlewanger [12], the TPC benchmarks can be represented with the workload model introduced by ISO/IEC 14756 [17] that forms the basis of our measurement method. In contrast to the methods used by TPC or Capra et al., the ISO method validates that the generated workloads meet specific predefined statistical parameters within a given delta (e.g. the relative frequency of tasks). This ensures that different measurement experiments that generate the same type of workload are comparable to each other.

Compared to the approach of Steigerwald [27] on non-concurrent software, it is with Capra's and our approach on concurrent software not possible to measure the energy consumption of a single method call. Thus, for a software developer it is difficult to decide, which parts of the software should be optimized to increase its energy efficiency. Such a decision may require additional statistical data retrieved, e.g. by analyzing performance counters, server request logs, or runtime efficiency ratings. The latter is part of ISO 14756 and can

therefore be calculated alongside with our energy efficiency approach with minimal additional expenditure.

Nevertheless, it is possible to study effects of software optimization on mean energy consumption on higher abstraction levels above the method level, e.g. the introduction of application caches to reuse created objects, reducing disk or network IO, or DBMS requests.

Where Capra et al. had difficulties to define a benchmark suitable for different ERPs due to the differences in workflows or UI design, we decided to measure software systems mainly as standalone products. Hence, our method and the proposed energy efficiency metric do not rely on a comparison with other software products of the same kind. However, this is not a limitation in general, because it is possible to calculate Capra's energy efficiency metric with our approach, if it is possible to provide suitable benchmarks and sufficient measurement values of software products of the same kind.

One limitation that is common to all discussed approaches that rely on executable software artifacts to measure or estimate their energy consumption is that you need executables before you can get analyzable data. Hence, these methods are not applicable in early design phases, when only class diagrams, sequence diagrams, etc. are available. In these early development stages, adoptions of techniques known from software performance engineering combined with power models to estimate power consumption of several design decisions may be more suitable. To our knowledge, such approaches do currently not exist and will require extensive further research.

6. CONCLUSION & OUTLOOK

Concluding, we found that software plays an important role regarding ICT and sustainability. But it has to be taken into account that software also induces energy consumption. In our paper we presented a model to classify green software and its engineering and showed some aspects of how the energy efficiency of software can be measured. If ICT and especially software should make a contribution to sustainability, it is necessary that software engineers also take the energy consumption into account. To do so, we presented a measurement method for software induced energy consumption.

Taking a look at the future of ICT, the aspects of Green IT might also be adopted by the software side. Thus, there should be answers to questions of the energy efficiency of software, especially regarding embedded systems. In this context, Cloud Computing needs to pay attention to the energy and resource consumption.

Overall, Green IT, green software, or green software engineering, meaning concepts of moving to ICT for Sustainability, are no standalone problems. Rather should every process of the hard- and software development implicate aspects of SD. The concepts need to be inherently entrenched into development and manufacturing processes. In this context, the arising questions are "What is a green, or rather sustainable software product? What are the criteria? What do metrics for these look like?" This makes it necessary to provide more tools and methods for software developers. It should become standard during the whole life cycle of software products. Ideally, even costumers should understand Green IT as a "must-have" requirement.

Hence, it is necessary to reach a certain standardization in the field of energy efficient software. This point might also support the development of green and sustainable software. In this context, the end user needs to be involved in a way that e.g. the interaction regarding energy options is better. An interface between standard software and the operation system without performance restrictions might be a future vision.

7. ACKNOWLEDGMENTS

This paper evolved from the research and development project "Green Software Engineering" (GREENSOFT), which is sponsored by the German Federal Ministry of Education and Research under reference 17N1209. The contents of this document are the sole responsibility of the authors and can under no circumstances be regarded as reflecting the position of the German Federal Ministry of Education and Research.

Sustainable Software Blog is a private initiative of computer scientists interested in Sustainable Informatics and Sustainable Development. Some of them are former employees of the GREENSOFT project.

8. REFERENCES

[1] Agarwal, S., Nath, A., and Chowdhury, D. 2012. Sustainable Approaches and Good Practices in Green Software Engineering. *IJRRCS 3,* 1, 1425–1428.

[2] Albertao, F. 2004. *Sustainable Software Engineering*. http://www.scribd.com/doc/5507536/Sustainable-Software-Engineering#about. Accessed 30 November 2010.

[3] Albertao, F., Xiao, J., Tian, C., Lu, Y., Zhang, K. Q., and Liu, C. 2010. Measuring the Sustainability Performance of Software Projects. In *2010 IEEE 7th International Conference on e-Business Engineering (ICEBE 2010), Shanghai, China*, 369–373.

[4] Assuncao, M. D. de, Orgerie, A.-C., and Lefévre, L. 2010. An Analysis of Power Consumption Logs from a Monitored Grid Site. In *2010 International Conference on Green Computing and Communications & 2010 EEE/ACM International Conference on Cyber, Physical and Social Computing*. IEEE Publishing Services, 61–68.

[5] Capra, E. and Merlo, F. 2009. Green IT: Everything starts from the Software. In *17th European Conference of Information Systems*, 62–73.

[6] Capra, E., Formenti, G., Francalanci, C., and Gallazzi, S. 2010. *The Impact of MIS Software on IT Energy Consumption*. 18th European Conference on Information Systems, 7-9 June 2010, Pretoria, South Africa. http://web.up.ac.za/ecis/ECIS2010PR/ECIS2010/Content/Papers/0073.R1.pdf. Accessed 25 October 2010.

[7] Capra, E., Francalanci, C., and Slaughter, S. A. 2011. Is software green? Application development environments and energy efficiency in open source applications. *Information and Software Technology* 54, 60–71.

[8] Chen, F. F., Schneider, J., Yang, Y., Grundy, J., and He, Q. 2012. An energy consumption model and analysis tool for Cloud computing environments. In *Proceedings of the First International Workshop on Green and Sustainable Software 2012. held in conjunction with The International Conference on Software Engineering, June 2-9, Zurich, Switzerland*, IEEE, Ed. IEEE Computer Society, 45-50.

[9] Dick, M. and Naumann, S. 2010. Enhancing Software Engineering Processes towards Sustainable Software Product Design. In *EnviroInfo 2010: Integration of*

Environmental Information in Europe. *Proceedings of the 24th International Conference EnviroInfo, October 6 - 8, 2010, Cologne/Bonn, Germany*, K. Greve and A. B. Cremers, Eds. Shaker, Aachen, 706–715.

[10] Dick, M., Kern, E., Drangmeister, J., Naumann, S., and Johann, T. 2011. Measurement and Rating of Software-induced Energy Consumption of Desktop PCs and Servers. In *Innovations in sharing environmental observations and information*. *Proceedings of the 25th International Conference EnviroInfo October 5 - 7, 2011, Ispra, Italy*, W. Pillmann, S. Schade and P. Smits, Eds. Shaker, Aachen, 290–299.

[11] Dick, M., Kern, E., Johann, T., Naumann, S., and Gülden, C. 2012. Green Web Engineering - Measurements and Findings. In *Man Environment Bauhaus: Light up the Ideas of Environmental Informatics*. *Proceedings of the 26th International Conference EnviroInfo; Federal Environment Agency, Dessau, 2012*, H.-K. Arndt, G. Knetsch and W. Pillmann, Eds. Shaker Verlag, Aachen, 599–606.

[12] Dirlewanger, W. 2006. *Measurement and rating of computer systems performance and of software efficiency*. *An introduction to the ISO/IEC 14756 method and a guide to its application*. Kassel University Press, Kassel.

[13] Erdmann, L., Hilty, L. M., Goodman, J., and Arnfalk, P. 2004. *The Future Impact of ICTs on Environmental Sustainability*. *Technical Report EUR 21384 EN*. Technical Report Series EUR 21384 EN. European Commission; Joint Research Centre; IPTS - Institute for Prospective Technological Studies, Seville.

[14] Hilty, L. M. 2005. *Information systems for sustainable development*. Idea Group Publishing, Hershey, Pa.

[15] Hilty, L. M. 2008. *Information technology and sustainability*. *Essays on the relationship between ICT and sustainable development*. Books on Demand, Norderstedt.

[16] International Organization for Standardization. 2005. *Software engineering - Software product Quality Requirements and Evaluation (SQuaRE) - Guide to SQuaRE* 35.080 35.080, ISO/IEC 25000:2005 (E).

[17] International Organization for Standardization. 2010. *Information technology -- Measurement and rating of performance of computer-based software systems* 35.080 35.080, ISO/IEC 14756:1999.

[18] Jwo, J.-S., Wang, J.-Y., Huang, C.-H., Two, S.-J., and Hsu, H.-C. 2011. An Energy Consumption Model for Enterprise Applications. In *2011 IEEE/ACM International Conference on Green Computing and Communications*. IEEE Publishing Services, 216–219.

[19] Kansal, A., Zhao, F., Liu, J., Kothari, N., and Bhattacharya, A. A. 2010. Virtual machine power metering and provisioning. In *Proceedings of the 1st ACM symposium on Cloud computing*. ACM, Indianapolis, Indiana, USA, 39–50. doi:10.1145/1807128.1807136.

[20] Larsson, P. 2012. *Energy-Efficient Software Guidelines*. http://software.intel.com/en-us/articles/energy-efficient-software-guidelines/?wapkw=energy-efficient+software+guidelines. Accessed 27 June 2012.

[21] Naumann, S., Dick, M., Kern, E., and Johann, T. 2011. The GREENSOFT Model: A Reference Model for Green and Sustainable Software and its Engineering. *SUSCOM* 1, 4, 294–304. doi:10.1016/j.suscom.2011.06.004.

[22] Schall, D., Hoefner, V., and Kern, M. 2011. Towards an Enhanced Benchmark Advocating Energy-Efficient Systems. In *Performance Evaluation and Benchmarking*. *Third TPC Technology Conference, TPCTC 2011, Seattle, WA, USA, August 29. - September 3., 2011*. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg.

[23] Shenoy, S. S. and Eeratta, R. 2011. Green software development model: An approach towards sustainable software development. In *India Conference (INDICON), 2011 Annual IEEE*, 1-6.

[24] Software Improvement Group. 2011. *What can you do to make your software application more energy efficient?* http://www.sig.eu/en/News%20&%20publications/Publications/1078/__What_can_you_do_to_make_your_software_application_more_energy_efficient__.html. Accessed 9 January 2012.

[25] Sinha, A. 2001. *Energy Efficient Operating Systems and Software*. *Submitted to the Department of Electrical Engineering and Computer Science in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Electrical Engineering and Computer Science*. Dissertation, Massachusetts Institute of Technology.

[26] SPEC - Standard Performance Evaluation Corporation. *SPECpower_ssj2008*. http://www.spec.org/power_ssj2008/. Accessed 17 February 2012.

[27] Steigerwald, B., Chabukswar, R., Karthik, K., and Jun, D. V. 2007. *Creating Energy-Efficient Software*. http://software.intel.com/en-us/articles/creating-energy-efficient-software-part-1/. Accessed 19 January 2011.

[28] Taina, J. 2010. How Green Is Your Software? In *Software Business*. *First International Conference, ICSOB 2010, Jyväskylä, Finland, June 21-23, 2010. Proceedings*. Lecture Notes in Business Information Processing 51, 151–162. DOI:10.1007/978-3-642-13633-7_13.

[29] Taina, J. 2011. Good, Bad, and Beautiful Software - In Search of Green Software Quality Factors. *CEPIS UPGRADE* XII, 4, 22–27.

[30] The Embedded Microprocessor Benchmark Consortium. *EnergyBench™ Version 1.0 Power / Energy Benchmarks*. http://www.eembc.org/benchmark/power_sl.php. Accessed 24 September 2012.

[31] Tischner, U., Dietz, B., Maßelter, S., Schmincke, E., Prösler, M., Rubik, F., and Hirschl, B. 2000. *How to do EcoDesign? A guide for environmentally and economically sound design*. Verlag form, Frankfurt am Main.

[32] Transaction Processing Performance Council. 2010. *TPC-Energy Specification,* TPC-Energy 1.2.0. http://www.tpc.org/tpc_energy/spec/TPC-Energy_Specification_1.2.0.pdf. Accessed 2 September 2011.

[33] Wang, S., Chen, H., and Shi, W. 2011. SPAN: A software power analyzer for multicore computer systems. *SUSCOM* 1, Mar. 2011, 23–34. doi:/10.1016/j.suscom.2010.10.002.

[34] Zhong Ming Feng, B. and Lung, C.-H. 2010. A Green Computing Based Architecture Comparison and Analysis. In *International Conference on Green Computing and Communications & 2010 EEE/ACM International Conference on Cyber, Physical and Social Computing*. IEEE Publishing Services, 386–391