Contents lists available at ScienceDirect

Future Generation Computer Systems

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

Sustainable software products—Towards assessment criteria for resource and energy efficiency

Eva Kern [a,b,*], Lorenz M. Hilty [c,d,e], Achim Guldner [b], Yuliyan V. Maksimov [c,f], Andreas Filler [b,g], Jens Gröger [h], Stefan Naumann [b]

[a] Leuphana University Lueneburg, Universitätsallee 1, 21335 Lueneburg, Germany
[b] Institute for Software Systems in Business, Environment, and Administration, Trier University of Applied Sciences, Environmental Campus Birkenfeld, P.O. Box 1380, 55761 Birkenfeld, Germany
[c] Department of Informatics, University of Zurich, Zurich, Switzerland
[d] Technology and Society Lab, Empa Swiss Federal Laboratories for Materials Science and Technology, St. Gallen, Switzerland
[e] KTH Royal Institute of Technology, Stockholm, Sweden
[f] i4Ds Centre for Requirements Engineering, University of Applied Sciences Northwestern Switzerland (FHNW), Windisch, Switzerland
[g] Institute of Technology Management, University of St. Gallen, Dufourstrasse 40a, 9000 St. Gallen, Switzerland
[h] Sustainable Products & Material Flows Division, Oeko-Institut, Schicklerstraße 5-7, 10179 Berlin, Germany

HIGHLIGHTS

- A causal model linking software properties to sustainability conditions is proposed.
- A set of criteria to evaluate a software product's sustainability is presented.
- Application of selected criteria demonstrates practicability of the set of criteria.

ARTICLE INFO

Article history:
Received 7 July 2017
Received in revised form 9 January 2018
Accepted 25 February 2018
Available online 18 April 2018

Keywords:
Environmental criteria for software
Green software
Resource efficiency
Sustainability indicators
Model of software impacts
Energy-aware software

ABSTRACT

Many authors have proposed criteria to assess the "environmental friendliness" or "sustainability" of software products. However, a causal model that links observable properties of a software product to conditions of it being green or (more general) sustainable is still missing. Such a causal model is necessary because software products are intangible goods and, as such, only have indirect effects on the physical world. In particular, software products are not subject to any wear and tear, they can be copied without great effort, and generate no waste or emissions when being disposed of. Viewed in isolation, software seems to be a perfectly sustainable type of product. In real life, however, software products with the same or similar functionality can differ substantially in the burden they place on natural resources, especially if the sequence of released versions and resulting hardware obsolescence is taken into account. In this article, we present a model describing the causal chains from software products to their impacts on natural resources, including energy sources, from a life-cycle perspective. We focus on (i) the demands of software for hardware capacities (local, remote, and in the connecting network) and the resulting hardware energy demand, (ii) the expectations of users regarding such demands and how these affect hardware operating life, and (iii) the autonomy of users in managing their software use with regard to resource efficiency. We propose a hierarchical set of criteria and indicators to assess these impacts. We demonstrate the application of this set of criteria, including the definition of standard usage scenarios for chosen categories of software products. We further discuss the practicability of this type of assessment, its acceptability for several stakeholders and potential consequences for the eco-labeling of software products and sustainable software design.

1. Introduction

This article presents the results of a project on sustainable software design commissioned by the German Federal Envi-

* Corresponding author at: Leuphana University Lueneburg, Universitätsallee 1, 21335 Lueneburg, Germany.
E-mail address: mail@nachhaltige-medien.de (E. Kern).
URL: http://www.green-software-engineering.de (E. Kern).

https://doi.org/10.1016/j.future.2018.02.044

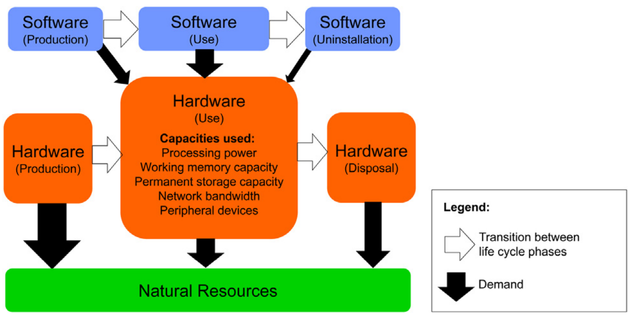

Fig. 1. Life cycles of hardware and software (horizontal dimension) and the resource demand induced by the life cycles (vertical dimension)

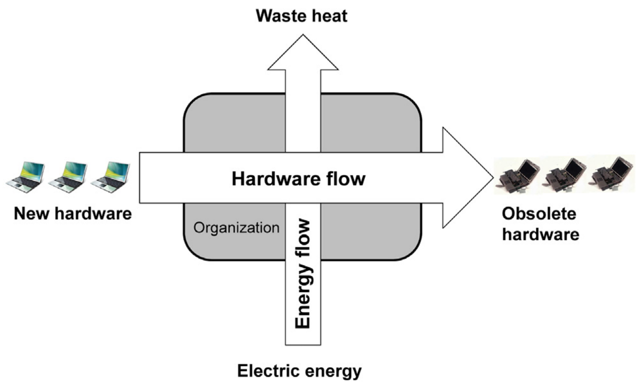

Fig. 2. The two main physical flows to be reduced by sustainable software.

ronment Agency.[1] The project builds on the results of earlier projects [1,2].

The goal of the present project is to develop a method for evaluating the environmental impacts of software products and to provide recommendations to software engineers for developing software with low environmental impact. The evaluation method is intended to support both the procurement of software products with the consideration of environmental criteria and the development of resource-efficient software. In particular, the method is supposed to enable a comparison of two given software products with similar functionality, where the comparison will focus on the impacts of their use on natural resources. Based on the formulation of ambitious minimum standards, the method will help to define criteria for the awarding of an environmental or quality label to sustainable software products. The potential effect of exploiting the software functionality, such as the carbon emissions saved by using videoconferencing software to avoid flights (as demonstrated by Coroama et al. [3]), which may be much larger, are not in the focus of this research. In the given example, we would distinguish among a set of software products instead, all providing videoconferencing functionality, by the amount of natural resources consumed per hour of using them.

Thus, the project makes a contribution to expanding the focus of "Green IT" beyond the hardware level to include the software level as the place where hardware requirements emerge and expand. Since software products are immaterial goods, it is a challenge to capture the indirect material and thus environmental impacts of these products in conceptual and methodological terms.

A product's environmental impacts generally occur through the use of natural resources[2] during the life cycle of the product. We take this life-cycle perspective into account in relation to software products, as well (see Fig. 1, upper part). We further consider that the hardware needed to operate a software product must be produced, supplied with electricity, and disposed of at the end of its useful life (Fig. 1, middle part). Thus, every software product is responsible for a quantifiable fraction of the life cycle of all the hardware products required for its operation (programmable devices of any kind, peripheral devices, and storage media). During production, use and disposal, these hardware products demand a quantifiable part of natural resources.

Because it takes a life-cycle perspective, this approach can be expanded to include social aspects of sustainability, e.g. when producing the raw materials for the hardware (or producing the software) as well as the working conditions in hardware production and disposal; however, our focus is on the environmental aspects.

At the software level, we intentionally limit our perspective to the use phase when developing the criteria. Although the production (development) and disposal (uninstallation) of software also has its indirect environmental impacts, they are not considered in this project. One exception is that we take into account the resources that can be wasted due to a software product's incomplete uninstallability.

Focusing on the use phase of software products is justified for standard software that is installed and run in millions or billions of cases. Minor changes of software properties decided by the developers can have a huge impact in terms of resource demand when the software product is being used, just because of the high multiplication factor that has to be applied due to the large number of installations and executions.

The purpose of the set of criteria is to evaluate a software product on the basis of characteristics that are observable in its use phase, be it by the users themselves or by persons conducting professional tests. We excluded software production because *assessing* the process of software development seems less important to us than *influencing* it, in particular by making recommendations addressed to those responsible for software development.

The evaluation of widespread software products requires more than a snapshot, ideally an observation of the software product over longer periods of time which cover several versions. From this long-term perspective, questions concerning software-induced purchasing of new hardware become more relevant, for example.

Expressed in abstract terms, our analysis focuses on two essential flows caused by the use of a software product (see Fig. 2):

- the flow of energy through the hardware running the software (electricity to waste heat),
- the flow of hardware through the organization using it (new hardware to electronic waste).

The impact of both flows on natural resources can be determined by using standard life-cycle assessment (LCA) methods. Life cycle inventories for production and disposal of the most important hardware components exist for this purpose, and we take them as given without entering a detailed discussion [4,5]. Energy flow can also be evaluated with LCA methods; the various methods for generating electricity have been examined sufficiently; that is why we take these results for granted as well [6,7].

If a software product causes significantly lower hardware and energy flows than competing products with similar functionality, it can be considered "relatively sustainable".[3]

[1] "Sustainable software design—Development and application of criteria for resource-efficient software products with consideration of existing methods." UFOPLAN project no. 3715 37 601 0.

[2] In this article, we reserve the term "resource" for natural resources and mostly avoid the technical term "hardware resource" by describing hardware resources in terms of their capacities, i.e., quantifiable aspects of their performance such as computing power, storage capacity, or transmission bandwidth.

[3] The functionality of a software product, and thus its *utility*, will not be evaluated here. Our goal is restricted to estimating and evaluating the *amount of resource use*

That is why it is sufficient to address the impact of software on the required hardware capacities. If one imagines a chain of impacts from software characteristics to natural resource use, we exclusively analyze the section of this causal chain from the software characteristics to the hardware products and their electricity consumption, because it is the only part of the causal chain that can vary among software products. In other words, a software product has usually no impact on the way hardware is manufactured or electricity is generated, but only on the amount of hardware and electricity consumed for its running. Exceptions may occur in cases of co-design of hardware and software or when an operating system and the hardware are optimized together. However, our focus is on standard application software. This does not exclude that the criteria or subsets of them can in principle be applied to a broader class of software products.

Additional hardware that is not directly involved in executing the programs of a software product, but indirectly used, such as the Internet nodes routing the traffic generated by the software, may not even be known; in such cases, using average impact factors like "Internet energy intensity" in kWh/GB [8] is better than implicitly assuming a zero impact.

Thus, operational criteria are necessary to be able to assess the sustainability of software with reference to the hardware and energy flows it induces. Then these criteria can be applied, e.g., to inform people responsible for software development or software procurement—or to award an eco-label for software products [9].

The set of criteria proposed here focuses on environmental impacts resulting from the *operation* of a software product. As already mentioned, this does not rule out that the awarding of eco-labels also includes social criteria regarding the process of software development (e.g., compliance with ILO[4] standards when outsourcing programming work), the functionality of the software (e.g., accessibility, or exclusion of particular categories such as violent games), or other aspects. It seems important to us, however, to treat the *impacts of software characteristics on natural resource consumption* as a clearly defined object of research from the outset and not to confound it with other issues. Studies and criteria are available for many of these neighboring issues and can be used to complement our set of criteria [10,11].

A tree of criteria was developed during the first phase of the project. The leaves of this tree are indicators serving to operationalize the parent criterion. An overview of the set of criteria is provided in Section 3. The full set of criteria is available in the supporting information to this article, which will be also published online. We will provide updates at: http://green-software-engineering.de/criteria-catalog, add a revision control on this website and invite for comments and feedback. In addition, we will keep the version to which this article directly refers as supplementary information constant and accessible. The set of criteria is intended to be used as a catalog from which a selection can be made depending on the goal and scope of the assessment task. Direct comparisons of software products are of course only possible if the same selection is used in all cases investigated.

2. Criteria for software sustainability and related approaches in literature

Many fields of research are addressing interactions between ICT and the goal of sustainable development. Hilty and Aebischer provide a general framework for this type of research [12]. As a special case, research on software sustainability is focusing on the software part of ICT systems, looking for interdependencies between software engineering and sustainability issues. A first literature review in this context was done by Penzenstadler et al. in 2012 [13]. Based on their data, at that time "little research coverage on the different aspects of sustainability in software engineering" was found. This seemed to have changed two years later, when a similar study by the almost same authors came to the conclusion: "The topic of SE4S [Software Engineering for Sustainability] has received wide-spread attention in the software engineering community over the past few years". [14] Thus, the interest for the issue is widening.

Addressing a more specific topic, Calero et al. [15] analyze publications dealing with software sustainability measurements. In addition to measurements, metrics are an issue in evaluating software products. Here, Bozzelli et al. [16] describe and classify metrics regarding the so-called "greenness" of software while reviewing existing literature in this context. However, they do not define what is meant by "greenness of software". According to their results, the "research community is focusing on metrics strictly related to energy consumption and saving dimensions".

Nevertheless, many authors working on criteria and metrics for "green" or "sustainable" software have a broader understanding of the issue, e.g. addressing the "impacts on economy, society, human beings, and [the] environment that result from development, deployment, and usage of the software." [1,17] Several other definitions of "sustainable software" [18–20] discuss the issue from different perspectives. Summarizing, all of them address the protection of resources, among other issues. However, a standardized understanding of "green" or rather "sustainable" software is still missing. As set out in the introduction, we will focus on environmental sustainability.

2.1. Strategies on finding criteria for sustainable software

The following strategies on how to find and summarize criteria for sustainability or "greenness" of software products can be identified in literature:

i. Taking existing software quality criteria (such as maintainability) or quality models (such as ISO 25010) and interpreting them in the context of environmental sustainability,
ii. taking existing sustainability criteria (such as energy efficiency) and clustering them to categories (bottom-up approach), and
iii. taking an LCA approach and defining software sustainability criteria for software life cycle phases (top-down approach).

Table 1 summarizes literature that can be categorized into methods (i)–(iii). The list is not intended to be exhaustive.

2.2. Research design

In order to create the set of criteria presented in this article, we used a procedure combining methods (i)–(iii): We first collected available extensions of quality models, findings of literature reviews (including scientific and practical publications), and additional ideas in the context of software sustainability (expert discussions). We then related the collected elements to each other and clustered them (bottom-up approach). This resulted in a structured collection of criteria which was reassessed for consolidating overlapping ideas. Then we mapped the criteria to software life cycle phases and to the sustainability aspects resulting from the causal model of the impact of software on natural resource use that was developed in parallel (Fig. 3). Consequently, only a subset of the consolidated set of criteria has been followed up.

The resulting set of criteria will be presented in Section 3. The practical application of example criteria is demonstrated with existing software products in Section 4.

it induces. A given amount of useful work can be related to the amount of resource use induced to determine *efficiency*.

4 International Labor Organization.

Table 1
Comparison of approaches on criteria for sustainable software.

Approach by	Objectives	Outcomes	Criteria (Examples)
Method (i)			
Albertao [21] Albertao et al. [22]	Assessing properties of software for environmental, economic and social aspects; Introducing a set of metrics to assess the sustainability of software products, demonstration how to use the metrics	Sustainability performance metrics and strategy how to improve follow-up releases by using the metrics	Modifiability, Reusability, Dependability, Usability, Efficiency, and Predictability
Calero et al. [18] Calero et al. [23]	Extending the ISO 25010 quality model by including sustainability aspects; definition of "greenability"	Model for software sustainability that can be added to the ISO software product quality model	Energy efficiency, Resource Optimization, Capacity Optimization, Perdurability
Method (ii)			
Taina [24]	Developing the criteria set "green software factors"	Framework for green quality factors: related to software development and execution	Feasibility (Carbon Footprint, Energy, Travel, ...), Efficiency (CPU-intensity, Idleness, ...), Sustainability (Reduction, Beauty, ...)
Kern et al. [25]	Summarizing existing approaches in a "quality model" for green and sustainable software	Quality model to classify green software and its engineering and exemplary corresponding metrics	Feasibility, Social Aspects, Portability, Efficiency, Reflectivity, Product Sustainability
Method (iii)			
Abenius [26]	Evaluation of "Green IT", especially "Green Software", and pointing out possibilities to use software in a more energy-saving way	Examples of actions towards Green IT, mapped to software life cycle phases	Choice of Material, Reuse Refurbish Recycle, Production Logistics
Naumann et al. [1]	Mapping potential effects of software to sustainable development	Life cycle model for software products including effects relevant to sustainability	Working Conditions, Manuals, Data Medium, Download Size, Accessibility, Hardware Requirements, Backup Size

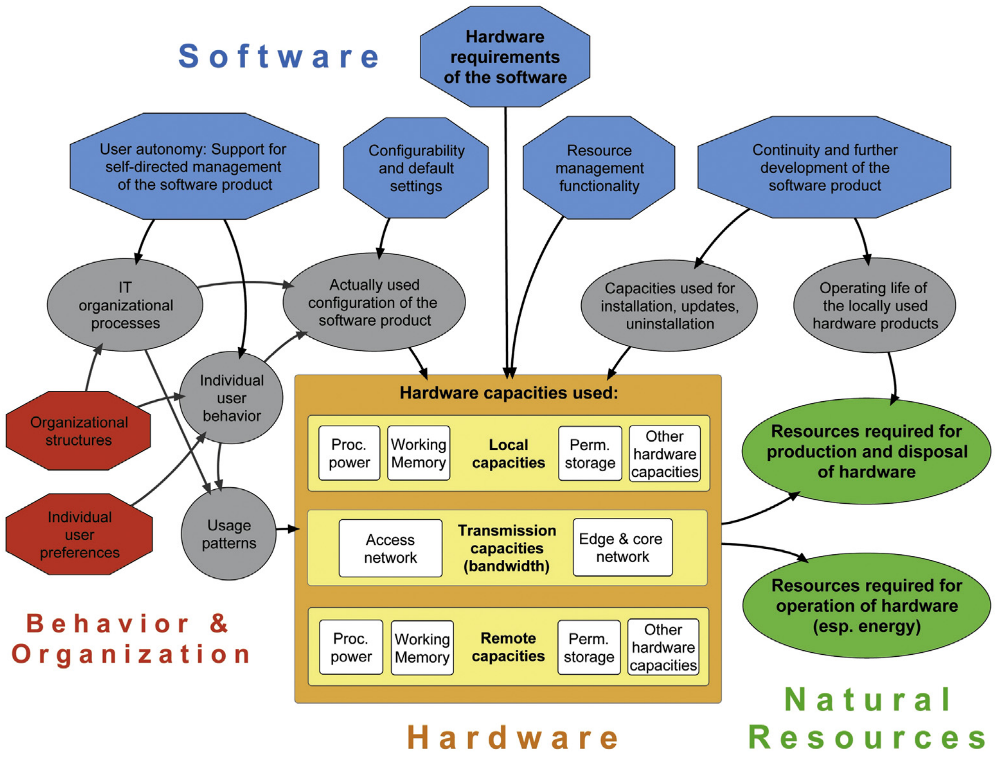
Software
Hardware requirements of the software
User autonomy: Support for self-directed management of the software product
Configurability and default settings
Resource management functionality
Continuity and further development of the software product
IT organizational processes
Actually used configuration of the software product
Capacities used for installation, updates, uninstallation
Operating life of the locally used hardware products
Individual user behavior
Organizational structures
Individual user preferences
Usage patterns
Hardware capacities used:
Proc. power
Working Memory
Local capacities
Perm. storage
Other hardware capacities
Access network
Transmission capacities (bandwidth)
Edge & core network
Proc. power
Working Memory
Remote capacities
Perm. storage
Other hardware capacities
Resources required for production and disposal of hardware
Resources required for operation of hardware (esp. energy)
Behavior & Organization
Hardware
Natural Resources

Fig. 3. The model describing the causal chains leading from software properties (blue) to the natural resources required for using the software (green). A link means "has impact on". The central node "Hardware capacities used" is structured into several types of local, transmission and remote hardware capacities procured and operated to run the software product. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

3. A set of criteria to assess the resource efficiency of software products

As a guideline to structure the criteria collected from the literature, we developed a causal model describing the principal mechanisms by which a software product can influence the demand for natural resources (inspired by [27]). Each of these mechanisms is a direct path (representing a causal chain) from observable properties of the software product to the use of hardware capacities and, finally, to the demand for natural resources that is induced by producing and running the hardware. This causal model is shown in Fig. 3. We will first explain the model and then provide an overview of the set of criteria we formulated on its basis.

Table 2
Software properties (corresponding to the five top nodes of Fig. 3) and example criteria. The numbers in brackets refer to the set of criteria provided as supporting information and described in Table 3.

Software property	Rationale	Example criterion
Support for self-directed management of the software product (user autonomy)	If the organization or individual using the software product wants to save resources, the software should support this intention.	Uninstallability [3.2].
Configurability and default settings	The default settings for configuring the software product may have a substantial influence on the resources used.	This aspect can be viewed as overarching, it affects several criteria by creating awareness for default settings, e.g., Resource Management [1.3].
Hardware requirements	The hardware required to run the software product have a crucial influence on the hardware capacities the user will procure and on the load and energy consumption when using the product.	Electricity consumption for a standard usage scenario and a standard configuration [1.2].
Resource management functionality	This is the capability of the software product to manage the hardware capacities (and thus the natural resources needed to provide them) in a way that avoids wasteful use.	Resource Management [1.3].
Continuity and further development of the software product	Backward compatibility of a software product mitigates the obsolescence effect that can be created by new versions.	Backward compatibility [2.1].

The observable properties of the software product are represented by the five (blue) nodes in the upper part of the graph. The most important property is the hardware requirements, i.e., the amount of hardware capacities the software product requires (both declared by the producer and actually measured, as partly shown in Section 4).

The amount of natural resources (including energy resources) demanded for running the software is represented by the two (green) oval nodes labeled "Resources required for production and disposal of hardware" and "Resources required for operation of hardware" (lower right part of the figure).

Each path leading from a software product property to the natural resource demand represents a possible way for software developers to influence the natural resource consumption that will be caused by their products. All paths happen to cross the node "Hardware capacities used"—except one.

This exceptional path starts at "Continuity and further development of the software product", which has a direct impact on the "Operating life of the locally used hardware products", which then affects the "Resources required for production and disposal of hardware". The idea behind this causal chain is that the evolution of a software product has an influence on hardware obsolescence. If, for example, a new version is more demanding with regard to hardware capacities, it can shorten the useful life of the hardware equipment in use even when this is still fully functional, which then increases the amount of natural resources required to produce the hardware per unit of used hardware capacity $*$ time.

All the other paths from software properties to natural resources cross the (orange) node "Hardware capacities used". Note that we assume that there are – in the general case – three spheres of hardware capacities involved in running application software:

- local capacities provided by end consumer devices,
- transmission capacities provided by network infrastructure, which includes hardware such as routers, switches and links (e.g., optical fibers), and
- remote capacities provided by servers.

Even if the software product under test runs only locally, it may require and access remote and transmission capacities which have to be assessed as well.

Another important feature of the model is that some of the causal chains reach the "Hardware capacities used" via (red) behavioral and organizational aspects of using the software product. This includes everything that may influence the "actually used configuration of the software product" or the "Usage patterns". This reflects that the actual configuration and the way of using the software product both influence the hardware capacity it requires. This is where users and the organizational structures in which they act come into play as influencers of the software-induced resource demand. Software developers cannot (and probably should not want to) determine user behavior and organizational structures, but they do define the decision space for organizations and individual users to optimize resource consumption when using the software. If this decision space is too constrained to allow users who intend to save resources to do so, this is less sustainable than providing such options. The gray nodes are intermediate nodes in the causal network.

In Table 2, we briefly explain the five software properties used in our model and provide examples for the criteria addressing them. Please note that the criteria numbers shown in brackets refer to Table 3 and to the full set of criteria that is provided as supporting information to this article.

To provide an overview of the full set of criteria, Table 3 shows the levels one and two of the hierarchy. There are three main criteria, "Resource efficiency", "Potential hardware operating life" and "User autonomy". They have different numbers of sub-criteria. Some of the sub-criteria are further refined to a third level. The leaf criteria are operationalized by indicators which can be directly used to measure or qualitatively assess properties of the software. Third-level criteria and indicators are not shown in Table 3 to keep it short. Please refer to the supporting information for the full documentation.

The next section describes the application of four example criteria to demonstrate their practicability in the real word.

4. Exemplary application of the criteria set

In order to show how the set of criteria can be used to compare software products with similar functionality in terms of resource efficiency, we will demonstrate the application of selected criteria.

The example criteria, for which we will demonstrate the operationalization here, were selected using the following requirements: at least one from each of the main criteria (level 1), potentially high relevance in terms of natural resource use, and different methodological challenges for applying the criteria. This led to the following selection:

- "Electricity consumption for a standard usage scenario and a standard configuration" [1.2],
- "Default settings supporting resource conservation" [1.3.3],
- "Backward compatibility" [2.1], and

Table 3
Levels one and two of the criteria tree. For the full description of the criteria set please refer to the supporting information to this article.

1	Resource efficiency 1.1 Hardware efficiency: Which hardware capacities must be available for operating the software product and what is the degree of capacity utilization during operation? 1.2 Energy efficiency: How much electricity does the hardware consume when the software product is used to execute a standard usage scenario? 1.3 Resource management: Does the software product have an energy management feature, and how effective is it when using the product in a standardized context?
2	Potential hardware operating life 2.1 Backward compatibility: Does the manufacturer of the software product guarantee that the current release can be executed on a reference system that is *n* years old? 2.2 Platform independence and portability: Can the software product be executed on different currently prevalent productive system environments (hardware and software), and can users switch between them without disadvantages? 2.3 Hardware sufficiency: Does the amount of hardware capacity used remain constant over time as the software product is developed further and additional functions are added?
3	User autonomy 3.1 Transparency and interoperability: Can users understand resource-relevant aspects of the software product with a reasonable amount of time and effort? Are they free to re-use data they produced with this software product with other software products? 3.2 Uninstallability: Can the software product be uninstalled easily, without leaving traces, and without avoidable disadvantages? 3.3 Maintenance functions: Does the software product provide easy-to-use functions permitting users to repair damage to data and programs? 3.4 Independence of outside resources: Can the software product be operated as independently as possible of resources not subject to the users' control? 3.5 Quality of product information: Does the information provided about the software product support its resource-efficient use?

- "Uninstallability of programs" [3.2.1].

For criteria with quantitatively measurable indicators (only the first one in our example), we use a measurement setup (following ISO/IEC 14756, as introduced in [28], see Fig. 4) to record the usage of the hardware capacities and energy consumption of a reference computer system to derive the hardware utilization and energy consumption induced by the software products. To demonstrate the applicability of the method, we used the measurement method of previous work, which used different measurement scenarios and focused on energy issues. The measurement setup is briefly described in Section 4.1.

The assessment of criteria for which no measurable indicators can be defined depends on observations of the software products' behavior, expert and user opinions, visual inspections, black-box tests and reviews of software manuals and other documents.

To evaluate the practicability of the criteria, we established a measurement procedure for each indicator. We then conducted case studies with 11 software products from the product groups "word processors", "web browsers", "content management systems", and "database systems". Thus, the four chosen software types represent three different software architecture patterns: local applications, applications with remote data processing, and server applications (see Table 4). The definition of the software architecture patterns is a result of the aforementioned prior research activities, a review of relevant literature, and expert interviews.

Based on the decision to select software products representing the different software architecture patterns, we chose the specific products for the case studies. Table 4 shows the resulting selection of software product groups. In order to find the products that should be tested, the following aspects were considered:

- High installation or user count
- Long useful life
- Different user groups
- Different devices used to run the products
- Different operating systems used to run the products
- Different licenses.

The specific selection is based on statistics on private and professional usage and market shares of the products. We chose from a large range of software product groups to test the applicability of the indicators over a large scope.

4.1. Example criterion "Electricity consumption for a standard usage scenario and a standard configuration"

To measure the energy consumption of software for the first sub-criterion of criterion 1.2, which is "Electricity consumption for a standard usage scenario and a standard configuration", we let the system execute the same task with a set of comparable software products and monitor the consumption behavior of the system under test (SUT) at the hardware level in a standardized test environment.

Fig. 4 depicts an exemplary measurement setup, introduced in [29], that allows recording the utilization of hardware capacities induced by a software product and the resulting energy consumption. Previous work showed the comparison of the energy consumption of different configurations of a web content management system, using caching and compression technologies, and of different ways of using web browsers on two well-known tools, based on black box measurements [29]. In addition, we applied white box measurements to sorting applications and multi user web applications [30].

The software product is installed on the SUT, which can be a desktop computer or server. The workload generator then performs the tasks defined in the usage scenario (see below). The power supply of the SUT is monitored by a power meter. The SUT itself collects the data on the utilization of its hardware capacities. All data is aggregated in a centralized data storage and then analyzed.

In order to produce comparable results, we record several measurements: the baseline consumption of the operating system, the consumption of the operating system plus the software product in idle mode, and a standard usage scenario. To ensure a fair appraisal system, the standard usage scenario needs to be devised for each software product group in a way that it does not favor one product over the other. Thus, the scenarios that we describe here are merely suggestions that we used to prove the viability of the method. The creation of the scenarios, which are later used e.g. for awarding eco-labels, should be devised by the respective entity, like the certifier.

For a non-interactive product group (e.g. database systems), we propose to use or devise a benchmark that puts load on the system. This can be done in several steps (low, medium, and high load) to ensure an equitable comparison. For an interactive product group (e.g. word processors), we propose to create a standard usage scenario in a way that it emulates a user as realistically as possible. To do so, we propose the following steps: First, we analyze which tasks users typically carry out with the software product and which functions of the software are used most frequently in this process. Additionally, we consider expert opinions on functionalities which may induce high energy demand or high resource utilization. With this information, we define individual actions that then are scheduled in a flow chart. We test the scenario with each software product of the product group to ensure that the execution of the whole scenario is possible with each one. This way, when the energy consumption and hardware utilization of the SUT is

Table 4
Selection of software products for the case studies.

#	Product group	Software architecture pattern	Platform	Products and Licenses
1	Word processors	Local application	Desktop/ Mobile	One proprietary and one open source word processor were selected
2	Web browsers	Application with remote data processing	Desktop/ Mobile	One proprietary and two open source web browsers were selected
3	Content Management Systems	Application with remote data processing	Desktop/ Server	Three open source browsers were selected
4	Database systems	Server application	Server	One proprietary and two open source database systems were selected

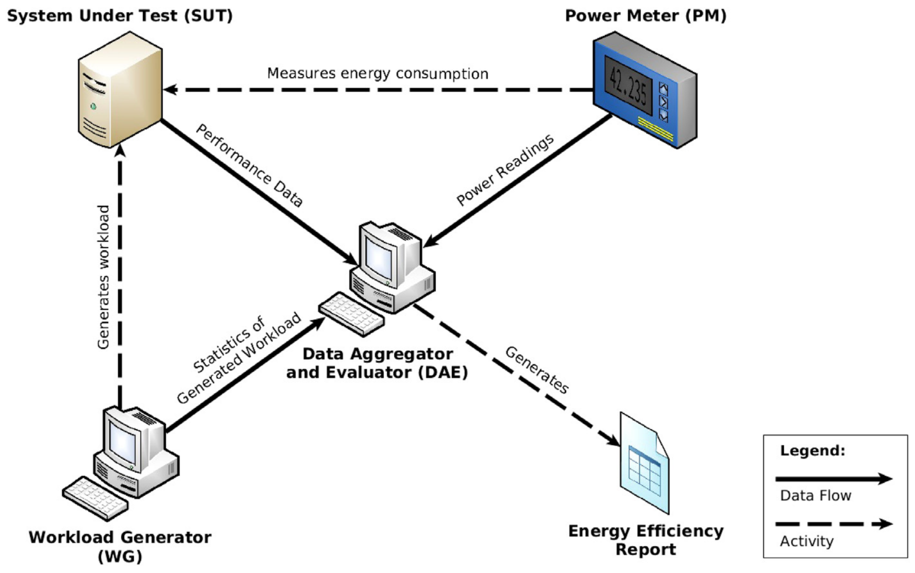

Fig. 4. Exemplary setup for measuring hardware utilization and energy consumption of a software system.

measured while performing the scenario, the same useful work is done with each software product and we can compare the results for each one of the scenarios.

After the design of a standard usage scenario, we prepare the SUT. In order to reduce side effects from remnants of previous installations, we overwrite the whole hard drive with a predefined standardized disk image including the desired operating system, before installing the software product. In this system, all possible background processes – such as automatic updates, virus scanners, indexing- and backup processes – are deactivated. With this standardized configuration, we measure the baseline consumption of the operating system several times and average the results in order to ensure that the fewest possible number of other processes are interfering with the scenario measurements.

By means of a macro software, we repeat the measurement of the standard usage scenario several times in order to generate a representative sample. In the following, we present exemplary results of the measurements from the case studies. Table 5 provides some technical details.

For each measurement, we record the power input and hardware utilization data, averaged per second. We store the measurement data together with the log data from the load generator in a database, in order to be able to analyze which action causes which resource consumption. By way of example, we present the results of the indicator "measured power input" (an indicator for criterion 1.2.1) for two word processors executing the same standard usage scenario (see Fig. 5).

The measured values are averaged per second over the 30 repetitions. It can be seen that the two word processors take a different amount of time to finish the same scenario within the 10-min. interval.

From these measurements, we calculate the average consumed electrical energy [Wh] per standard scenario and decide whether the two candidates' mean values differ significantly via a t-test.

In this case, word processor 2 uses significantly more electrical energy (16.72 Wh) than word processor 1 (14.43 Wh) on average (confidence interval: 95%). Similarly, we also calculate and compare the work that the software demands from the available hardware capacities, such as "processor work" (processor load integrated over time) to evaluate other criteria, in particular 1.1, "Hardware efficiency": Which hardware capacities must be available for operating the software product and what is the degree of capacity utilization during operation?

4.2. Further example criteria

All considerations about the evaluation of the criteria are made with the default settings of the software product to be assessed in order to locate the optimization potentials.

4.2.1. Example criterion "default settings supporting resource conservation"

Criterion 1.3.3 (a sub-criterion of 1.3 Resource management) requires that "the default settings of the software product are selected in such a way that they also take the goal of resource conservation into account". This is assessed by means of observing the default settings of the software product and a reviewer's assessment. In this case, it is important to check settings both during and after the software installation and, if necessary, to measure the default settings with a separate scenario and compare the results with those of the other software products in the same group.

Table 5
Details of the measurement procedure.

System Under Test (SUT)	We use two SUT, a desktop computer for local software (e.g. word processors) and a server for distributed software (e.g. content management systems).
Sampling rate	All data is collected with a sampling rate of one kilohertz. (1000 data points/s)
Scenario length	The time interval of each scenario is set to 10 min.
System configuration	We use two standard configurations of hardware and operating system, one for each SUT (client and server). All software products of one product group are installed on the same reference system.
Synchronization	The analysis software synchronizes all measured data by means of time stamps.
Sample size	All measurements are repeated 30 times and then averaged. Assuming a normally distributed population [28], and given the controlled test environment described in the text, 30 measurements are usually a sufficient sample size, as "the sampling distribution will tend to be normal regardless of the population distribution in samples of 30 or more" [31].

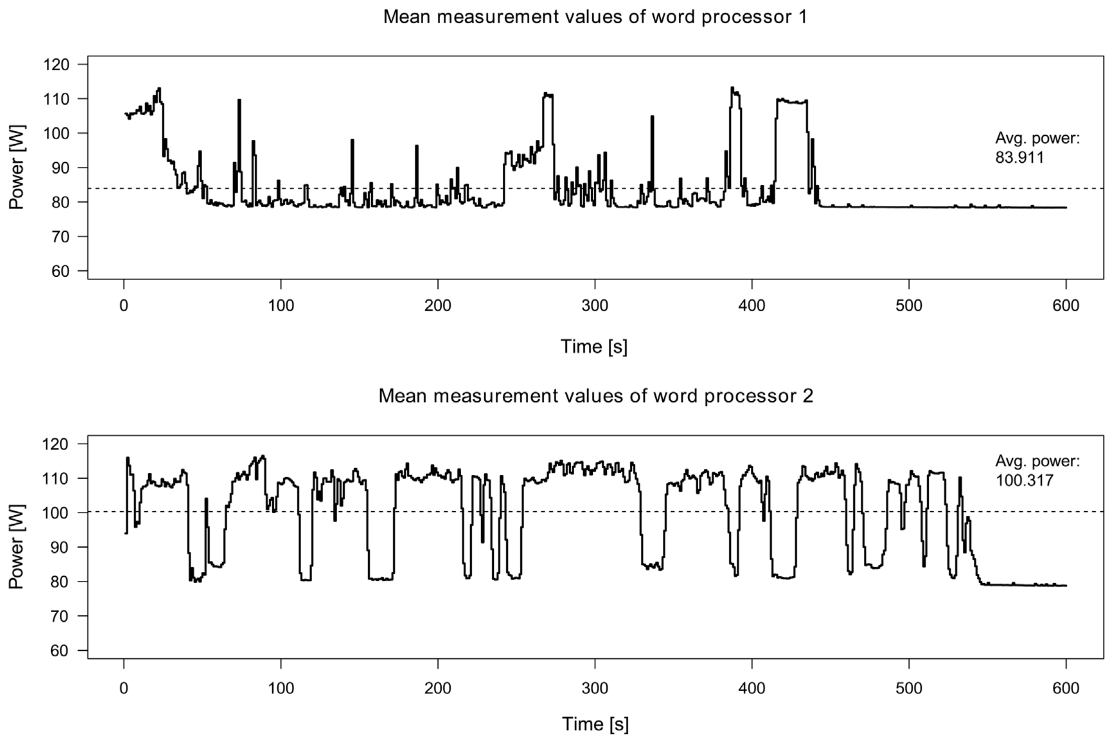

Fig. 5. Comparison of the power input of two word processors.

As mentioned in Table 2, this criterion is a special case because it is implicitly linked to other criteria and indicators addressing a broad variety of settings relevant for resource demand. Examples of such settings are sleep mode settings (indicator for 1.3.2, energy options (indicator for 1.3.1), data compression and transfer options.)

As a starting point, the reviewer can look out for hardware-intensive modules identified in criterion 1.1.5 "Economical use of hardware through adaptability and support for users when adapting the software product". In some cases, it may also be necessary to rely on other assessment methods like reviewing manuals or visual inspections (e.g., in case of web browsers: which is the default page the browser opens at every start?).

For the case studies with software products from the "word processor" group, we found that for word processor 1, the default is to install all office tools (like word processor, spreadsheet program, presentation program, etc.) and some extensions (approximately 140 megabytes in size, including, e.g., dictionaries for the spell checker). Word processor 2 also includes the standard office tools. In addition, it installs several programs, plugins, and add-ons (approximately 1000 megabytes in size).

4.2.2. *Example criterion "backward compatibility"*

Criterion 2.1 requires that "the manufacturer of the software product guarantee[s] that the current release can be executed on a reference system that is n years old". The maximum number n is the result.

To apply this criterion, two indicators are evaluated:

(a) Use the specification by the manufacturer (hardware, old operating systems, old frameworks), since no standard configurations have been defined for previous years.
(b) When this criterion has been applied for at least one year, execute the standard usage scenario on earlier standard configurations as well. Can the standard usage scenario still

be executed with the current release of the software product under the standard configurations from n years ago?

For indicator (a), we review the manufacturer's specification to identify how many years have passed since the most current version of the minimum requirements (hardware, operating systems, frameworks, etc.) to operate the software product was released. For example: If the minimum requirements are Windows 2000 and php 5 (released in 2006), this would yield $n = 11$ today (in 2017). For the case studies with software products from the "word processor" group, we found that for word processor 1, the current release can be executed on a system that is 8 years old. For word processor 2 we found $n = 7$. In the following iterations of the assessment of the software product, when a new version is available, the evaluation of indicator (b) can start by experimentally testing whether the standard usage scenario can be executed on earlier standard configurations as well. This may then provide more robust results.

4.2.3. Example criterion "uninstallability of programs"

Criterion 3.2.1 requires that "the user receives sufficient support to uninstall the program without leaving traces". In order to test this criterion, we propose a black-box test that shows if after installing and uninstalling of the software product under study, the condition will be identical to that prior to the installation.

To achieve this, we first make a copy of the standardized disk image with exclusively the desired operating system. Then, we install the software product, perform the standard usage scenario and uninstall the software product again, following the instructions in the user manual (if available). We then create another disk image and compare it to the image we created before the installation. This way, we find all files and changes to files that remain after uninstalling. The reviewer then traces which files were created by the software product. Additionally, we search for remaining entries in the registry of Windows.

For the case studies with software products from the "word processor" group, we found that after uninstalling word processor 1, there were no related files left except user-generated documents. After uninstalling word processor 2 however, there were 14 related empty folders and 1 file remaining on the disk image and over 100 entries in the registry. The manufacturer also provides a manual for completely removing the registry entries. Indeed, this can only be accomplished with administrator rights and may be difficult for unexperienced users.

As exemplarily in Section 4 demonstrated, we verified all criteria during our case studies. In conclusion, our studies show that all criteria can be utilized with varying effort. As described, we chose these examples to be presented in the paper, because they are a cross section of the criteria set, potentially have a high impact in terms of resource consumption, and pose different methodological challenges. The general goal is not to establish a fixed methodology, but to show the viability of the criteria. The methods should be adapted by the entities that use the set of criteria e.g. for awarding eco-labels.

5. Potential use cases for the criteria set

In this section, we discuss the set of criteria and its potential future application from the perspective of different stakeholders.

5.1. Software user perspective

Software users are those who use software products without usually needing skills in developing or administrating software products. Thus, this group comprises everyone using a desktop PC, laptop, smartphone or similar end-user devices in private or professional contexts. A software user is not necessarily identical with the software purchaser (see Section 5.2).

From the perspective of software users, the criteria for sustainable software products provide information about the environmental impacts of the products. They inform about the idea how to characterize and evaluate the sustainability of a software product, and give hints how to use and configure software products to achieve higher energy and resource efficiency. Informing about these issues by exemplary demonstrations of this set of criteria (like the one presented in Section 4) might, hopefully, lead to more transparency and awareness for the topic.

Additionally, the set of criteria could be the basis for the development of a label for green software products (see Section 5.5). Such a label can be seen as an information medium supporting transparency in the relation between software usage and environmental impacts by presenting the information created by applying the set of criteria in a maximally aggregated form. The awareness for environmental impacts of software can support "greener" user behavior with regard to ICT products, especially software. Besides protecting the environment, the behavioral changes can also result in economic advantages for the users.

5.2. Software purchaser perspective

Software purchasers are those who care about searching for and buying new software products for their organizations. In private households, purchasers and users are often identical. In a company, the purchaser may be responsible for ordering vast amounts of software products. In any case, someone must decide which software product is to be bought because there are competitive products with similar functionality.

If a purchaser is interested in sustainability issues, he or she can include our set of criteria or a selected part of it to evaluate candidate products or rely on test results provided by independent sources (NGO's, journalists) that have applied the criteria or by a public authority providing an eco-label.

Besides that, the set of criteria can inspire purchasers to include additional requirements for new software products in their calls for tenders. Such requirements can become part of procurement guidelines.

In the long run, our set of criteria can help companies reducing their CO2 footprints by purchasing sustainable software products. At the same time, there may be economic benefits by reducing hardware capacities and obsolescence.

5.3. Software administrator perspective

The software administrators do not only use the software products, but also care about configuration and related technical aspects. Thus, the set of criteria can be a source of inspiration and deeper understanding, a guideline and an argumentation aid for administrators to configure software products in an environment-friendly way. In large organizations with thousands of users, the impacts of an increased administrators' awareness for software sustainability issues can be huge both in environmental and in economic terms.

5.4. Software developer perspective

The group of software developers includes both individual developers and software companies.

For a software company, creating sustainable software and following marketing trends may be conflicting goals (e.g., creating customer lock-in effects or monitoring user behavior to sell this data might not work with sustainable software). However, both the individual developer as well as a future-oriented company, can use the idea of sustainable software to create a unique selling point. Considering our criteria in software development expands the spectrum of non-functional requirements for the software products. As soon as sustainability becomes a highly rated requirement for software purchasers, software manufacturers who are able to deliver sustainable products will have an advantage.

Recommendations for software developers that can be directly or indirectly extracted from the criteria may be transformed into a guideline for resource-efficient software development. The recommendations and the guideline can help to spread the ideas of caring about sustainability issues in software engineering.

5.5. *Software certifier perspective*

Software certifiers are people or organizations who are involved in developing and awarding eco-labels or sustainability labels for software products.

Software certifiers may use the set of criteria as a basis for awarding a label. In order to create guidelines on how to implement test procedures, the method of exemplary application (Section 4) could be formalized. To do so, our set of criteria provides a reliable basis.

Providing a label for sustainable software products extends the number of certified products in the context of responsible consumption. It extends the spectrum of software properties that can be certified, e.g., quality, security, and usability. As a consequence, the portfolio of software certifiers grows, strengthening their role in contributing to market transparency and supporting users in responsible behavior.

6. Discussion

The previous section already shows possible use cases for the presented set of criteria and assigns them to the different user perspectives. However, in order to be able to assess the criteria of sustainable software products in such a structured way as presented, we had to set priorities in (i) the scope, (ii) the application area, and (iii) product selection. This can be interpreted both as strength and as limitation.

As described in the introduction, we decided to focus on the usage phase of software. This reduces the complexity of gathering software products as immaterial goods and provides a starting point that is easier to handle. Nevertheless, we are aware of the connection between the different life cycle phases. Thus, we definitely speak out in favor of addressing the other phases and the connection between them in future work. Overall, the set of criteria presents a balanced selection of possible criteria since it is the result of literature reviews and working sessions of a team of seven researchers and practitioners with different backgrounds. Additionally, the selection has been evaluated by external experts.

Defining standard usage scenarios and taking this scenario as a basis for the measurements done (see Section 4.1), allows us to compare the energy consumption and hardware requirements of software products of the same product classification. However, as described, the standard usage scenarios are one viable approach to the assessment of the software products, especially for interactive product groups. Nevertheless, a certifier (as described in Section 5.5) who uses the criteria catalog must decide which scenarios are to be established within the community. To evaluate the exemplary criteria, we used a measurement setup including specific hardware and software tools (Fig. 4). However, the portability of the measurement method is ensured since the results are repeatable by using comparable tools.

In order to be able to test the suggested criteria, we had to select software products the criteria could be applied to. We chose 11 products representing (a) the defined software architecture patterns and (b) popular products of these classes. It turned out that there are some limitations in the application of the set of criteria that are caused by the software architecture. For example, local applications do not transfer data in the network. Thus, we do not need to estimate the energy consumed in the network for the data traffic. This (energy consumed in the network) is one of the indicators of the criterion 1.2 Energy efficiency (see Table 3). Overall, we are satisfied that the set of criteria can be applied to additional software products representing one of the software architecture patterns.

7. Conclusion and outlook

In this article, we described how we developed a set of criteria for software sustainability and demonstrated the application of a subset of them. This research is one step towards awarding a sustainability label for software products by comparing products with similar functionality. If it turns out that one of the products causes less pressure on natural resources than others in its class, then it can be labeled “sustainable”. The basic causal model and the criteria we developed go beyond energy demand at runtime by also covering the mechanisms that drive the increasing demand for hardware capacities, including software-induced obsolescence.

The criteria were developed with a combined approach including an extensive literature search, the selection and transfer of existing software quality criteria and the derivation of software-specific criteria from general sustainability indicators. The pilot application of core criteria addressing electricity consumption, default settings, backward compatibility and uninstallability of software products revealed that there are significant differences between products that may look quite similar at a first glance. We conclude that the criteria have a high relevance and offer practical benefit to any stakeholder who wants to distinguish similar software products with respect to their resource efficiency or environmental impact.

So far, the criteria are unweighted. The set of criteria provides no statement about what is more relevant and which minimum standards must be met to characterize sustainable software. When the criteria set should be further developed into an eco-label or procurement requirements, it will have to be expanded to a rating system. One can imagine that sustainability properties of software can finally be described, e.g., on a single scale from 0 to 100. This would make it easy to rank products according to their environmental friendliness. However, this step towards standardization should be done by experts of the certification field since it is a political decision leading to issues of environmental policy.

Next to moving forward in creating a standardized label for sustainable software products, future work contains an extension of the measurements, e.g. evaluating further software products and including mobile devices as SUT. A vision for the future is to integrate the measurements of software sustainability directly in the development process and when releasing new versions of software products. Thus, in future, we will pay attention to providing recommendations for software engineers and an integration of the knowledge on sustainability characteristics for software products into teaching and education.

Acknowledgments

There have been many contributors to shape the set of criteria. The authors are thankful to each of them. We specifically would like to thank Marina Köhn, Dr. Hans-Jürgen Baumeister (both German Environment Agency), and Prof. Dr. Benno Schmidt from Bochum University of Applied Sciences.

This work was supported by the German Environment Agency under project number 3715 37 601 0.

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.future.2018.02.044.

References

[1] S. Naumann, M. Dick, E. Kern, T. Johann, The GREENSOFT model: A reference model for green and sustainable software and its engineering, SUSCOM 1 (4) (2011) 294–304.

[2] L. Hilty, W. Lohmann, S. Behrendt, M. Evers-Wölk, K. Fichter, R. Hintemann, Green Software: Final Report of the Project: Establishing and Exploiting Potentials for Environmental Protection in Information and Communication Technology (Green IT). Report Commissioned By the Federal Environment Agency, Berlin, Förderkennzeichen 3710 95 302/3 (23) (2015).

[3] V.C. Coroama, L.M. Hilty, M. Birtel, Effects of internet-based multiple-site conferences on greenhouse gas emissions, Telemat. Inform. 29 (4) (2012) 362–374.

[4] P.A. Wäger, R. Hischier, R. Widmer, The material basis of ICT, in: L.M. Hilty, B. Aebischer (Eds.), ICT Innovations for Sustainability: Advances in Intelligent Systems and Computing, Springer, Switzerland, 2015, pp. 209–221.

[5] R. Hischier, V.C. Coroama, D. Schien, M.A. Achachlouei, Grey energy and environmental impacts of ICT hardware, in: L.M. Hilty, B. Aebischer (Eds.), ICT Innovations for Sustainability: Advances in Intelligent Systems and Computing, Springer, Switzerland, 2015, pp. 171–189.

[6] E.G. Hertwich, T. Gibon, E.A. Bouman, A. Arvesen, S. Suh, G.A. Heath, J.D. Bergesen, A. Ramirez, M.I. Vega, L. Shi, Integrated life-cycle assessment of electricity-supply scenarios confirms global environmental benefit of low-carbon technologies, Proc. Natl. Acad. Sci. 112 (20) (2015) 6277–6282.

[7] R. Turconi, A. Boldrin, T. Astrup, Life cycle assessment (LCA) of electricity generation technologies: overview, comparability and limitations, Renew. Sustain. Energy Rev. 28 (2013) 555–565.

[8] V.C. Coroama, L.M. Hilty, Assessing Internet energy intensity a review of methods and results, Environ. Impact Assess. Rev. 45 (2014) 63–68.

[9] E. Kern, M. Dick, S. Naumann, A. Filler, Labelling sustainable software products and websites: Ideas, approaches, and challenges, in: V.K. Johannsen, S. Jensen, V. Wohlgemuth, C. Preist, E. Eriksson (Eds.), Proceedings of EnviroInfo and ICT for Sustainability 2015: 29th International Conference on Informatics for Environmental Protection (EnviroInfo 2015) and the 3rd International Conference on ICT for Sustainability (ICT4S 2015) Copenhagen, September 7–9, 2015, Atlantis Press, Amsterdam, 2015, pp. 82–91.

[10] M. Al Hinai, R. Chitchyan, Building social sustainability into software: Case of equality, in: Requirements Patterns (RePa), 2015 IEEE Fifth International Workshop on, 2015, pp. 32–38.

[11] T. Johann, W. Maalej, Position Paper: The Social Dimension of Sustainability in Requirements Engineering, in: Proceedings of the 2nd International Workshop on Requirements Engineering for Sustainable Systems, 2013.

[12] L.M. Hilty, B. Aebischer, ICT for sustainability: An emerging research field, in: L.M. Hilty, B. Aebischer (Eds.), ICT Innovations for Sustainability: Advances in Intelligent Systems and Computing, Springer, Switzerland, 2015, pp. 3–36.

[13] B. Penzenstadler, V. Bauer, C. Calero, X. Franch, Sustainability in software engineering: A systematic literature review, 2012.

[14] B. Penzenstadler, A. Raturi, D. Richardson, C. Calero, H. Femmer, X. Franch, Systematic Mapping Study on Software Engineering for Sustainability (SE4S) - Protocol and Results, Irvine, 2014.

[15] C. Calero, M.F. Bertoa, M. Angeles Moraga, A systematic literature review for software sustainability measures, in: 2nd International Workshop on Green and Sustainable Software (GREENS), 2013, pp. 46–53.

[16] P. Bozzelli, Q. Gu, P. Lago, A systematic literature review on green software metrics, 2013.

[17] M. Dick, S. Naumann, N. Kuhn, A model and selected instances of green and sustainable software, in: What Kind of Information Society? Governance, Virtuality, Surveillance, Sustainability, Resilience: 9th IFIP TC 9 International Conference, HCC9 2010 and 1st IFIP TC 11 International Conference, CIP 2010, Held As Part of WCC 2010, Brisbane, Australia, September 20–23, 2010 Proceedings, Springer, Berlin, Heidelberg, 2010, pp. 248–259.

[18] C. Calero, M. Moraga, M.F. Bertoa, Towards a software product sustainability model, 2013, arXiv preprint arXiv:1309.1640.

[19] B. Penzenstadler, Towards a definition of sustainability in and for software engineering, in: Proceedings of the 28th Annual ACM Symposium on Applied Computing, 2013, pp. 1183–1185.

[20] M. Mahaux, P. Heymans, G. Saval, Discovering sustainability requirements: An experience report, in: Requirements Engineering: Foundation for Software Quality: 17th International Working Conference, REFSQ 2011, Essen, Germany, March 28–30, 2011 Proceedings, Springer, Berlin, Heidelberg, 2011, pp. 19–33.

[21] F. Albertao, Sustainable Software Engineering, available at http://www.scribd.com/doc/5507536/Sustainable-Software-Engineering#about. (Accessed 14 April 2017).

[22] F. Albertao, J. Xiao, C. Tian, Y. Lu, K.Q. Zhang, C. Liu, Measuring the sustainability performance of software projects, in: 2010 IEEE 7th International Conference on e-Business Engineering, ICEBE 2010, Shanghai, China, 2010, pp. 369–373.

[23] C. Calero, M.Á. Moraga, M.F. Bertoa, L. Duboc, Green software and software quality, in: C. Calero, M. Piattini (Eds.), Green in Software Engineering, Springer, 2015, pp. 231–260.

[24] J. Taina, Good, bad, and beautiful software - In search of green software quality factors, in: J.-C. Lopez-Lopez, G. Sissa, L. Natvig (Eds.), Green ICT: Trends and Challenges, 2011, pp. 22–27.

[25] E. Kern, M. Dick, S. Naumann, A. Guldner, T. Johann, Green software and green software engineering –definitions, measurements, and quality aspects, in: L.M. Hilty, B. Aebischer, G. Andersson, W. Lohmann (Eds.), ICT4S ICT for Sustainability: Proceedings of the First International Conference on Information and Communication Technologies for Sustainability, ETH Zurich, February 14–16, 2013, ETH Zurich, University of Zurich and Empa, Swiss Federal Laboratories for Materials Science and Technology, Zürich, 2013, pp. 87–94.

[26] S. Abenius, Green it & green software - time and energy savings using existing tools, in: EnviroInfo 2009: Environmental Informatics and Industrial Environmental Protection: Concepts, Methods and Tools: Proceedings of the 23rd International Conference Environmental Informatics - Informatics for Environmental Protection, Sustainable Development and Risk Management, September 09–11, 2009, HTW Berlin, University of Applied Sciences, Germany, Shaker, Aachen, 2009, pp. 57–66.

[27] C. Som, L.M. Hilty, A.R. Köhler, The precautionary principle as a framework for a sustainable information society, J. Bus. Ethics 85 (2009) 493–505.

[28] W. Dirlewanger, 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, 2006.

[29] M. Dick, E. Kern, J. Drangmeister, S. Naumann, T. Johann, Measurement and rating of software-induced energy consumption of desktop PCs and servers, in: W. Pillmann, S. Schade, P. Smits (Eds.), Innovations in Sharing Environmental Observations and Information: Proceedings of the 25th International Conference EnviroInfo October 5–7, 2011, Ispra, Italy, Shaker, Aachen, 2011, pp. 290–299.

[30] T. Johann, M. Dick, E. Kern, S. Naumann, How to measure energy-efficiency of software: Metrics and measurement results, in: IEEE (Ed.), Proceedings of the First International Workshop on Green and Sustainable Software, GREENS 2012: Held in Conjunction with ICSE 2012, the International Conference on Software Engineering, June 2–9, Zurich, Switzerland, IEEE Computer Society, 2012, pp. 51–54.

[31] A.P. Field, Discovering Statistics using SPSS, third ed., Sage, Los Angeles, 2009.

Eva Kern is a doctoral student at the Leuphana University of Lüneburg in cooperation with the Trier University of Applied Sciences, Environmental Campus Birkenfeld. She deals with the question how to evaluate and communicate environmental issues of software and authored several international conference papers regarding green software. Before starting her doctoral studies, she worked as a research assistant for different research & development projects, including the project "GREENSOFT - Green Software Engineering". Eva Kern graduated in computer science and media (degree: M.Sc.) at the Trier University of Applied Sciences in 2013.

Lorenz M. Hilty is a professor at the Department of Informatics at the University of Zurich, Head of Group at Empa Swiss Federal Laboratories for Materials Science and Technology, and affiliated professor at CESC Center for Sustainable Communications at KTH Royal Institute of Technology, Stockholm. He got his Ph.D. in computer science from the University of Hamburg and has been researching sustainability aspects of ICT for more than 20 years. He authored more than 100 peer-reviewed articles in this field. Lorenz is the initiator of the international conference series ICT4S, Information and Communication Technologies for Sustainability.

Achim Guldner, M.Sc. is a researcher in the UFOPLAN - Sustainable Software Design project at the Trier University of Applied Sciences, Environmental Campus Birkenfeld. Since 2013 he works as research assistant in several R&D Projects, focusing on improving energy efficiency of and through ICT and especially software systems. Before, he worked as scientific assistant at the Institute for Software Systems at the Environmental Campus Birkenfeld, where he mainly focused on the improvement of data quality via statistical methods. He graduated in applied computer sciences in 2013.

Yuliyan V. Maksimov is a Ph.D. student at Blekinge Institute of Technology BTH in Sweden in cooperation with the School of Engineering at the University of Applied Sciences and Arts Northwestern Switzerland FHNW. Currently he works in the field of Telecommunication Systems at BTH and in the area of Requirements Engineering at FHNW. Before starting his Ph.D. studies, he worked as a researcher in the field of Informatics and Sustainability at the University of Zurich in Switzerland. He holds a M.Sc. in Informatics and a Diploma in Information Systems from the Hochschule Konstanz University of Applied Sciences HTWG in Germany.

Andreas Filler is doctoral researcher at the Center for Digital Health Interventions at University of St. Gallen and ETH Zurich (Switzerland) as well as Ph.D. candidate at the University of Bamberg (Germany) in the context of digital health interventions. In his former position as research associate at the Institute for Software Systems at Trier University of Applied Sciences (Germany) he worked on information systems for sustainable development. Andreas studied OnlineMedia (Dipl.-Inf.) and Computer Science in Media (M.Sc.) at Furtwangen University of Applied Sciences (Germany). Within the last years he worked in several national and EU funded research projects.

Jens Gröger is working as a Senior Researcher in the Sustainable Products & Material Flows Division located at Oeko-Institut in Berlin, Germany. He is the main responsible expert for Information and Communication Technologies (ICT) as well as for Green Public Procurement (GPP) at Oeko-Institut. He is leading projects for the development of a number of award criteria for the German Ecolabel “Blue Angel” as well as Green Public Procurement criteria. Jens Gröger holds a degree in energy and process engineering and is a certified technician in communication electronics.

Stefan Naumann is a full professor for fundamentals in computer science, mathematics, and environmental and sustainability informatics at the Trier University of Applied Sciences (Environmental Campus Birkenfeld, Germany) and a directorate member of the Institute for Software Systems. His research interests are sustainable development in conjunction with online communities and the environmental impacts of information technology, especially of software. He studied computer science at the Universities of Kaiserslautern and Saarbrücken (Germany) and received his doctorate in natural sciences (Dr. rer. nat.) from the University of Hamburg in 2006. Stefan Naumann authored over 50 peer-reviewed articles in this field.