Reference:
Kern, Eva; Hilty, Lorenz M.; Guldner, Achim; Maksimov, Yuliyan V.; Filler, Andreas; Gröger, Jens; Naumann, Stefan (2018): Sustainable software products-Towards assessment criteria for resource and energy efficiency. In: Future Generation Computer Systems, Volume 86, September 2018, pp. 199-210, ISSN 0167-739X, https://doi.org/10.1016/j.future.2018.02.044

Set of criteria for sustainable software

Lorenz Hilty, Stefan Naumann,
Yuliyan Maksimov, Eva Kern,
Andreas Filler, Achim Guldner
Jens Gröger

Version 01
Effective: 31.05.2017

UFOPLAN research project „Sustainable Software Design - Entwicklung einer Methodik zur Bewertung der Ressourceneffizienz von Softwareprodukten“
code number: 3715 37 601 0
By order of the German Federal Environmental Agency

Project contact persons

Dipl.-Ing. Jens Gröger
Oeko-Institut e.V.
Phone +49 30 40 50 85 378, j.groeger@oeko.de

Prof. Dr. Stefan Naumann
Trier University of Applied Sciences, Environmental Campus Birkenfeld
Phone +49 6782 17 12 17, s.naumann@umwelt-campus.de

Prof. Dr. Lorenz Hilty
University of Zurich
Phone +41 44 635 67 24, hilty@ifi.uzh.ch

Table of contents

Introduction

This document is the outcome of the first of five work packages in the German Federal Environment Agency's UFOPLAN project "Sustainable software design—Development and application of criteria for resource-efficient software products with consideration of existing methods."

The goal of the project is to develop a method for evaluating the environmental impacts of software products. This method is intended to support both the procurement of software products with 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 in terms of their impacts on natural resources. Based on the formulation of ambitious minimum standards, the method will also help to define criteria for awarding an environmental or quality label to sustainable software products.

Thus, the project makes a contribution to expanding the focus of "Green IT" beyond the hardware level to include the software level. Since software products are immaterial goods, one problem arising is to capture the indirect material impacts of these products in conceptual and methodological terms.

A product's environmental impacts generally occur through the use of natural resources[1] during the life cycle of the product. We take on this life-cycle perspective in relation to software products as well (see figure 1). We take into account 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. 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).

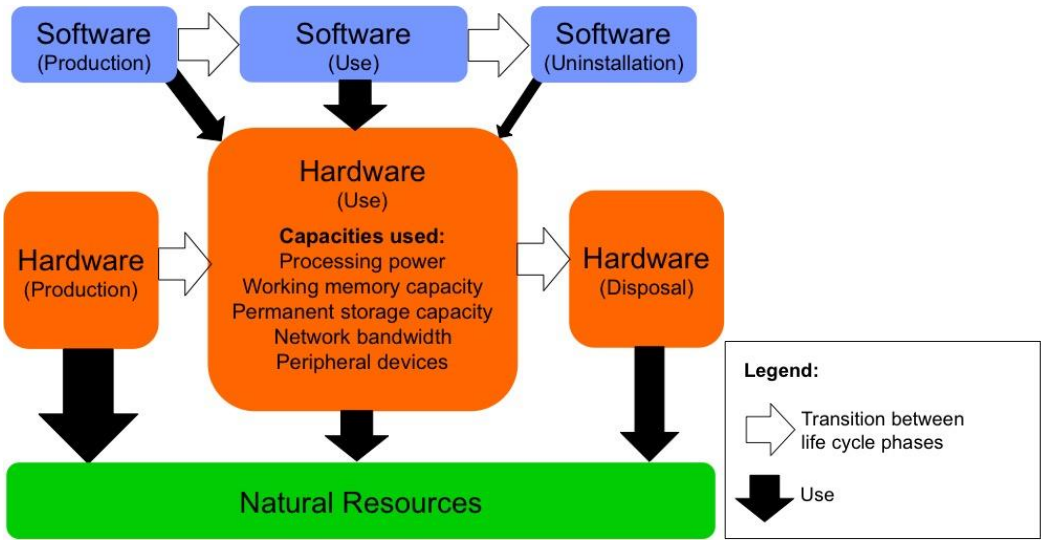


Figure 0-1 Life cycles of hardware and software (horizontal dimension) and the resource use induced by the life cycles (vertical dimension)

Because it takes a life-cycle perspective, this approach can be expanded to include the social aspects of producing the raw materials as well as the working conditions in hardware production and disposal; our focus is on the environmental aspects.

At the software level, we intentionally limit our perspective to the use phase in the following. The goal of the criteria defined here is to evaluate a software product on the basis of characteristics that are observable

[1] Definitions of "resource" and other key terms are provided in the glossary. In this document, we reserve the term "resource" for natural resources and mostly avoid the technical term "hardware resource" by describing hardware resources directly in terms of capacities, i.e., quantifiable aspects of their performance.

in the use phase, be it by the users themselves or by persons conducting special tests. In principle, it would also be possible to take the production phase of software into account by broadening the approach. However, *evaluating* the process of software development seems less important to us than *influencing* it, among other things by making recommendations addressed to those responsible for software development. A software development guide will be prepared in a later phase of this project.

In general, we examine standard application software in this project, in other words, neither system software nor special application software for a small number of users. In the latter case, the resources required for development would surely have to be included.

As a matter of principle, especially the evaluation of widespread software products requires not just a snapshot, but consideration of the use of the software product (including several versions) over longer periods of time. Only from this perspective does the question as to software-induced purchasing of hardware become relevant, for example.

Expressed in abstract terms, our analysis focuses on two flows caused by a software product:

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

If a software product causes significantly lower hardware and energy flows than competing products with similar functionality, then it can be considered "sustainable."[2]

The resource consumption induced by the flow of hardware can be estimated by applying 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. Energy flow can also be evaluated with LCA methods; the various methods for generating electricity have been examined sufficiently; therefore, we also take this data as given.

That is why it is sufficient to use the criteria developed in this project 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, then we analyze exclusively the section of the chain of impacts from the software to the hardware products and their electricity consumption[3], because it is the only part that is specific to our object of investigation.

Practicable criteria are necessary to be able to assess the sustainability of software with reference to the hardware and energy flows it induces. These criteria can then be applied, e.g., to inform those responsible for software development or software procurement or to award an environmental label.

The set of criteria proposed here focuses on environmental impacts resulting from the *operation* of a software product. This does not rule out that the awarding of environmental labels also includes further 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

[2] The functionality of a software product, and thus its *utility*, will not be evaluated here. The goal is exclusively to estimate and evaluate the *amount of resource use* it induces. A given amount of useful work can be related to the amount of resource use induced to determine *efficiency*.

[3] In some cases, it may be necessary to broaden this perspective and take the flow of consumables such as paper or toner through the hardware into account, analogously to the flow of energy. Whether this is the case for a given software product and which consumables are relevant can be decided based on a first screening.

[4] International Labour Organization

research from the outset and not to confound it with other questions. Studies and criteria are available for many of these neighboring questions and can be used to complement our set of criteria.

A tree (a hierarchy) of criteria and indicators is described on the following pages. The leaves of the tree are indicators serving to operationalize the relevant higher-level criterion. This document's table of contents provides a complete overview of the criteria.

An evaluation model whose structure we will lay out in this project will later be applied to combine (arithmetically or logically) the indicators to criteria and then to combine the criteria to higher-level criteria. The actual evaluation by means of weighting, normalization functions, or designating mandatory criteria remains up to the German Federal Environment Agency or other organizations applying our criteria and may change over time.

Weighting of indicators and criteria may vary depending on the class of software in an existing evaluation model. In particular, indicators or criteria can be assigned a weight of zero if they are not applicable or irrelevant to certain classes of software. We have developed a classification of software products that is adapted to the application of our criteria (see also Appendix A).

- local application
- application with remote data storage
- application with remote processing
- remote service

These four classes are relevant to our approach as they attempt to encompass not only the resource consumption induced by local execution of the software, but also the resource consumption induced remotely, from network infrastructure to dedicated servers to the cloud. Otherwise, the approach would be useless because the criteria could be fulfilled by shifting environmental impacts elsewhere. A detailed impact model describing the various impact paths from software characteristics via hardware capacities to natural resources can be found in Appendix B.

In the following main section of this document, each criterion is characterized by

- a one- to three-digit number, depending on its level,
- a designation (heading),
- a question explaining the criterion,
- a comment following the question, as appropriate.

Each criterion in the lowest position in the hierarchy is operationalized by indicators identified with a lower-case letter.

This tree of criteria and indicators is based on a comprehensive literature research on criteria for evaluating software analyzing more than 130 such criteria from more than 60 sources. A draft of the set of criteria was discussed with experts from the scientific community, public agencies, and industry at a stakeholder workshop on 11 March 2016. The participants' feedback during and after the workshop was taken into consideration during the revision of the document.

Some of the following criteria and indicators refer to a “reference system,” a “standard configuration,” or a “standard usage scenario”; these and other key concepts are defined in the glossary.

1 Resource efficiency

To what extent are hardware capacities used, and therefore, to what extent are natural resources consumed indirectly, when a given function is performed?

This main criterion assumes that a given functionality can be fulfilled by a software product using different amounts of hardware capacities, which indirectly results in different amounts of natural resource consumption required for hardware provision, operation, and disposal.

The ideal is a software product that achieves a given functionality with minimum resource consumption, i.e., that maximizes resource efficiency (see glossary). Functionality is specified by standard usage scenarios (see glossary). The hardware capacities to be made available and those actually used as well as the energy consumed serve as approximations for estimating natural resource consumption.

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?

Hardware capacities are measured in % of the corresponding capacity of a reference system[5]. They can be differentiated according to two dimensions: (Table 1). On one dimension, they are differentiated in local, network and remote capacities. Here, we further distinguish in recommended (1.1.1) and minimum (1.1.2) capacities as well as capacities required in idle mode (1.1.3) and during the execution of a standard usage scenario (1.1.4). On the other dimension, we differentiate according to the type of hardware capacity: processing power, working memory, permanent storage, bandwidth, and display resolution. The matrix is open to the addition of new columns in case new categories of hardware will become relevant in the future.

Table 1-1 Differentiation of hardware capacities in two dimensions. The numbers refer to the criteria explained in the following sections, the letters refer to the indicators.

		Processing power	Working memory	Permanent storage	Bandwidth	Display resolution
Local	recommended minimum idle standard usage	1.1.1 a) 1.1.2 a) 1.1.3 a) 1.1.4 a)	1.1.1 b) 1.1.2 b) 1.1.3 b) 1.1.4 b)	1.1.1 c) 1.1.2 c) 1.1.3 c) 1.1.4 c)	-	1.1.1 d) 1.1.2 d)
Network	recommended minimum idle standard usage	-	-	-	1.1.1 e) 1.1.2 e) 1.1.3 d) 1.1.4 d)	-
Remote	recommended minimum idle standard usage	1.1.1 f) 1.1.2 f) 1.1.3 e) 1.1.4 e)	1.1.1 g) 1.1.2 g) 1.1.3 f) 1.1.4 f)	1.1.1 h) 1.1.2 h) 1.1.3 g) 1.1.4 g)	-	-

Each cell of the matrix in **Table 1-1** shows the associated criterion (e.g., 1.1.1) with the corresponding indicator (e.g. a)) for operationalization. The criteria and indicators will be described in the following

[5] Application of the set of criteria requires that a reference system corresponding to current technical developments is determined periodically. The reference system serves to standardize indicators.

sections, which are numbered accordingly. Not all of the criteria 1.1.1 to 1.1.4 are applicable in all the matrix cells. For this reason, some of the cells remain empty.

Criteria 1.1.5 and 1.1.6 are used for the assessment of hardware efficiency as well. They can be assessed in general terms; they do not require differentiation according to this matrix and do not show up in Table 1-1 for this reason.

When these criteria are to be aggregated later, the principal problem arises that a trade-off between different hardware capacities (local vs. remote, processing power vs. working memory, processing power for data compression vs. bandwidth, etc.) must be made. If it were possible to evaluate the hardware capacities in the form of an ecological footprint, they could be weighted and aggregated in that regard. Assessing this footprint is not part of the work reported here; we refer the reader to existing life cycle inventories for ICT hardware and electric energy as a basis for aggregation.

Table 1-2 Basic definitions for the measurement of the criteria 1.1.3 and 1.1.4.

Identifier	Name	Definition	Comment
FL_i	full load	Upper limit of the capacity i in the reference system.	For processing power, the FL is 100%, for working memory the sum of the installed RAM, for network bandwidth the maximum transmission speed, etc.
BL_i	base load	Average load of the capacity i in the reference system when the software product under study is not installed	
IL_i	idle load	Average load of the capacity i in the reference system when the software product under study is installed, but idle.	Idle load includes base load.
NIL_i	net idle load	$NIL_i = IL_i - BL_i$	
t	time	Time needed to execute the standard usage scenario on the reference system.	Begins with the start of the standard usage scenario and ends when all required actions are executed, including follow-up processes (such as releasing memory or deleting temporary files).
GL_i	gross load	Load of the capacity i in the reference system while executing the standard usage scenario, measured as time-weighted average over t.	
NL_i	net load	$NL_i = GL_i - BL_i$	
AF_i	allocation factor	$AF_i = NL_i / (FL_i - BL_i)$	Allocation factor used to assign a share of the base load GA to the effective load EL (defined below).

AFI_i	allocation factor idle	$AFI_i = NIL_i / (FL_i - BL_i)$	Allocation factor used to assign a share of the base load GA to the effective load idle ELI (defined below).
EL_i	effective load	$EL_i = NL_i + AF_i * BL_i$	
EIL_i	effective load idle	$EIL_i = NIL_i + AFI_i * BL_i$	Used to calculate the indicators for hardware demand of criterion 1.1.3
HD_i	hardware demand	$HD_i = EL_i * t$	Used to calculate the indicators for hardware demand of criterion 1.1.4

For practical purposes, it is sufficient to calculate the allocation factors AF and AFI for criteria 1.1.3 and 1.1.4, in particular for processing power (indicators a. and e.) and working memory (indicators b. and f.). For all other indicators (c., d., g.) the allocation factors can be set to zero, i.e., it can be assumed that EL = NL and EIL = NIL for simplicity.

Figure 1-1 illustrates the process of measuring hardware capacity load by executing a standard usage scenario.

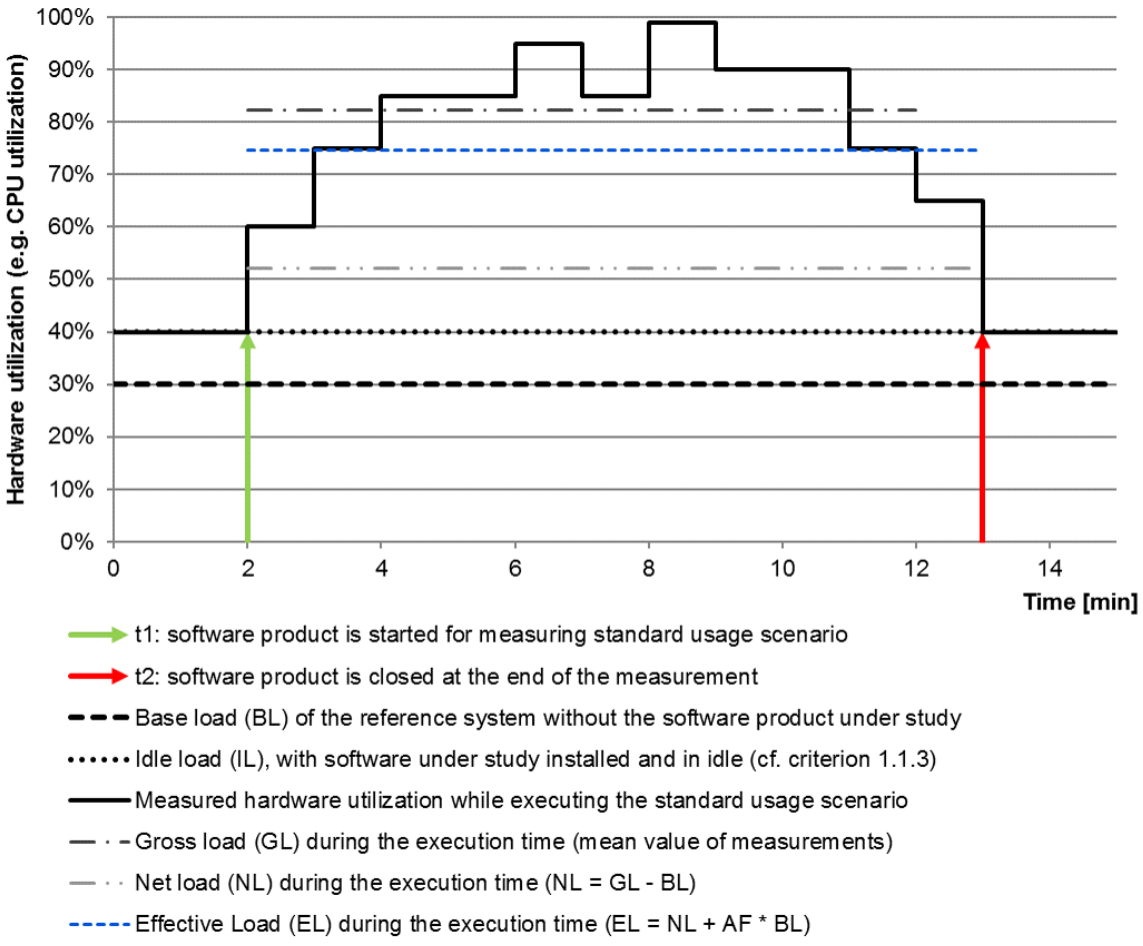


Figure 1-1 Exemplary measurement process of hardware capacity load

1.1.1 Recommended system requirements and resulting hardware requirements (including peripheral devices)

Which system requirements does the manufacturer recommend for operating the software product?

Indicators:

a) Recommended local processing power as specified by the manufacturer in % of the processing power of the reference system
b) Recommended local working memory as specified by the manufacturer in % of the working memory of the reference system
c) Recommended local permanent storage as specified by the manufacturer in % of the permanent storage of the reference system
d) Recommended display resolution as specified by the manufacturer in % of the display resolution of the reference system
e) Recommended network bandwidth as specified by the manufacturer in % of the network bandwidth of the reference system
f) Recommended server processing power as specified by the manufacturer in % of the processing power of the reference system
g) Recommended server working memory as specified by the manufacturer in % of the server working memory of the reference system
h) Recommended server permanent storage as specified by the manufacturer in % of the server permanent storage of the reference system

1.1.2 *Minimum* system requirements and resulting hardware requirements (including peripheral devices)

What are the minimum system requirements for operating the software product?

Indicators:

a) Minimum local processing power as specified by the manufacturer in % of the processing power of the reference system
b) Minimum local working memory as specified by the manufacturer in % of the working memory of the reference system
c) Minimum local permanent storage as specified by the manufacturer in % of the permanent storage of the reference system
d) Minimum display resolution as specified by the manufacturer in % of the display resolution of the reference system
e) Minimum network bandwidth as specified by the manufacturer in % of the network bandwidth of the reference system
f) Minimum server processing power as specified by the manufacturer in % of the processing power of the reference system
g) Minimum server working memory as specified by the manufacturer in % of the server working memory of the reference system
h) Minimum server permanent storage as specified by the manufacturer in % of the server permanent storage of the reference system

1.1.3 Hardware utilization in *idle mode* assuming a standard configuration

What is the level of utilization of the available hardware capacities by the software product in idle mode?

Indicators:

a) Measurement of average processor utilization in idle mode under the standard configuration
b) Measurement of average working memory utilization in idle mode under the standard configuration
c) Measurement of average permanent storage utilization in idle mode under the standard configuration
d) Measurement of average bandwidth utilization for network access in idle mode under the standard configuration
e) Measurement of average server processor utilization in idle mode under the standard configuration
f) Measurement of average server working memory utilization in idle mode under the standard configuration
g) Measurement of average server permanent storage utilization in idle mode under the standard configuration

Average processor load (indicators a. and e.) and average working memory load (indicators b. and f.) are calculated as effective idle load EIL (see **Table 1-2**).

1.1.4 Hardware utilization during *normal use* assuming a standard configuration and a standard usage scenario

What is the average utilization of the available hardware capacities during operation of the software product?[6]

It should be noted here that utilization of hardware capacities is understood as a variable integrated over time. If, for example, program A requires twice as much processing power, working memory, or bandwidth as program B to accomplish a given standard usage scenario, but makes the processor, memory, or bandwidth available again after half the period of time required by B, then according to this criterion, A is not less efficient than B. (This is not the case for criteria 1.1.1 to 1.1.3.) Thus, the use of acceleration technologies is not penalized by this criterion.

Indicators:

a) Measurement of average processor utilization when running the standard usage scenario under the standard configuration
b) Measurement of average working memory utilization when running the standard usage scenario under the standard configuration
c) Measurement of average permanent storage utilization when running the standard usage scenario under the standard configuration
d) Measurement of average bandwidth utilization for network access when running the standard usage scenario under the standard configuration
e) Measurement of average server processor utilization when running the standard usage scenario under the standard configuration
f) Measurement of average server working memory utilization when running the standard usage scenario under the standard configuration

[6] Average capacity utilization determines which free hardware capacities can be used by other software products during operation of the software product.

g) Measurement of average server permanent memory utilization when running the standard usage scenario under the standard configuration

Hardware demand for processor load (indicators a. and e.) and average working memory load (indicators b. and f.) are calculated as defined in **Table 1-2**.

1.1.5 Economical use of hardware through adaptability and support for users when adapting the software product

Does the software product use only those hardware capacities required for running the functions demanded by the individual user? Does the software product provide sufficient support when users adapt it to their needs?[7]

Indicators:

a) Does the software automatically minimize the required capacities and/or are there relevant options available during installation? (Scale: yes/no)
b) If users choose an option, can they change the decision for or against installation options at any later point in time? (Scale: yes/ no)
c) Black box test whether hardware-intensive modules can be disabled (Scale: can permanently be disabled/can temporarily be disabled/cannot be disabled)
d) Is it possible (without drawbacks) to disable peripheral devices that are not needed temporarily or permanently or to avoid providing them at all? (Scale: can be disabled temporarily and permanently/can be disabled only temporarily/cannot be disabled)
e) Will files used only for installing the product be deleted after installation?

1.1.6 Online delivery

Can the software product (including all programs, data, and documentation including manuals) be purchased, installed, and operated without transporting physical storage media (including paper) or other materials goods (including packaging)?

Indicators:

a) Can the software be delivered and updated online?
b) Is it supported that the user organization can store the software product and its updates on a local server, avoiding transferring them for every single user?

1.2 Energy efficiency

How much electricity does the hardware consume when the software product is used to execute a standard usage scenario?[8]

The consumption of electric energy is a consequence of the utilization of hardware capacities. How to measure hardware utilization has already been described in section 1.1.4 above. In parallel to those measurements, the electrical power demanded by the hardware should be measured (or estimated) as well,

[7] No utilization of capacities by functionality temporarily or permanently not demanded by the user.

[8] Use of electricity is a consequence of the use of hardware capacities already discussed in section 1.1. This implies that this criterion is redundant. However, the redundancy is desired since energy can be measured separately and not all sub criteria of hardware efficiency (1.1) are operationalizable.

at least for the entirety of hardware used locally, for data transmission in the network or remotely, respectively.

Indicators:

a) Measurement of the energy consumed on the local device for running the standard usage scenario under the standard configuration
b) Estimation of the energy consumed in the network for the data traffic caused by running the standard usage scenario under the standard configuration (a current estimate of network energy intensity in kWh/GB from literature may be used, if necessary differentiated among types of access network)
c) Measurement of the energy consumed by servers for the remote processing and storage for running the standard usage scenario under the standard configuration (if measurement not possible, an estimate based on factors for average energy intensity of data center services from literature may be used)

The electric energy consumed is the integral of electric power over the time needed for the execution of the standard usage scenario. Departing from the specifications provided to measure hardware load (section 1.1.4), only net indicators will be used for the energy measurements (indicators a. and c.), i.e., only the quantity that exceeds the level of the electric base load. This is done to increase practicability (calculating an allocation factor for electricity may be difficult because a true upper limit for electric power is sometimes not known). It also adds to the clarity of the results of the energy measurements if base load energy is not included when comparing software products.

1.3 Resource management

To what extent does the software product contribute to efficient management of the resources it uses during operation?

Since the extent to which a given software product is used may vary, adaptive demand for hardware capacities that is supported by the software product contributes to resource conservation. Hardware capacities not in use can potentially be used by other processes or reduce their energy consumption. Both options contribute indirectly to natural resource conservation.

In contrast to criteria 1.1 and 1.2, this criterion refers to adapting the demand for hardware capacities at the program's runtime, in particular the transition to less energy-consuming modes, dependent on the current user requirements or the available hardware capacities or energy. In other words, while resource efficiency in the various modes was addressed by criteria 1.1 and 1.2, the focus here is on the ability to switch between modes depending on context.

1.3.1 Adaptation of hardware capacities used to current demand

Does the software product have the feature to release hardware capacities (and reduce energy consumption as a consequence) when it doesn't temporarily use these capacities?

Indicators:

a) Does the software product have different modes which have a measurable effect on energy consumption?
b) Does the software product dynamically change to a more energy saving mode when possible (e.g. sleep mode)?
c) In case the user has to make energy-relevant settings, are these settings concentrated in one place and easily understandable for the user?[9]

1.3.2 Adaptation of hardware capacities used to current supply

Is the software product able to dynamically adapt its demand for hardware capacities and energy when the supply is changing? (e.g., when the available bandwidth is decreasing or battery is low)

Indicators:

a) Does the software product switch to a more economical mode when less hardware capacity or energy is available, avoiding errors or loss of data? (no restrictions, slower execution, error during execution)
b) Is the full software functionality available in if the energy management of lower system layers or connected client systems is activated?[10]

1.3.3 Default settings supporting resource conservation

Are the default settings of the software product selected in such a way that they also take the goal of resource conservation into account?[11]

Indicators:

a) Reviewer's assessment whether the default settings of the software product are selected in such a way that they also take the goal of resource conservation into account

1.3.4 Feedback on use of hardware capacities and energy

Can the local and remote hardware capacities used by the software product and their resulting energy consumption be monitored, and are the displayed values correct?

Indicators:

a) Are the hardware capacities in use, data flow, and energy consumption displayed? (Scale: yes/to some extent/no)
b) Assessment by the reviewer whether the display is correct

[9] Examples: Background/sleep settings, animations, computing-intensive processes such as indexing etc., cache sizes, ability to select the time at which processes are executed to take advantage of ecologically more beneficial energy (demand shaping).

[10] In particular server-based software should avoid that activating the energy management on client side hampers the functionality. For example, no session information should be lost if the client computer enters sleep mode.

[11] Example: Default setting for printing: Double-sided printing if the printer has this capability?

2 Potential hardware operating life

To what extent are hardware replacement cycles decoupled from software replacement cycles?[12]

Software imposes requirements on the hardware on which it is executed. The faster these requirements increase as the software product is developed further, and the more specific they are, the more they limit the use of hardware products already in existence. If existing hardware products cannot be used, or can no longer be used, to execute the given software product, then this shortens the operating life of the hardware.

The ideal is a software product whose development dynamics permit operators to manage their hardware products independently of these dynamics, i.e., decouple hardware management from software management.

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?[13]

Indicators:

a) Initially use the specification by the manufacturer (hardware, older operating systems, older frameworks), since no standard configurations have been defined for previous years.
b) When this criterion has been applied for a long enough time period, so that the standard usage scenario can also be executed on earlier standard configurations as well: Can the standard usage scenario still be executed with the current release of the software product on a configuration that was the standard configurations n years ago (n still needs to be specified)?

[12] Decoupling software and hardware replacement cycles amounts to long potential hardware operating life. Basic assumption: Every software product requires a system environment as the platform on which it is executed. The system environment is defined as the sum of the hardware and software components of the ICT system that are required for executing the software product. The software product itself can be part of the system environment of other software products. Example: A web browser requires an operating system, additional system software, and hardware as a system environment, and at the same time it constitutes the system environment for a web application. From the perspective of a given software product, the following question is crucial to understand its influence on hardware operating life: when the software product is replaced by a newer version, which requirements to the lowest level—the hardware—does this generate via the intermediate levels of the system environment?

[13] Thus, the software product can be executed on a standard hardware configuration that has already been in operation for *n* years.

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?[14]

Indicators:

a) Manufacturer specifications (compatible with various operating systems, runtime environments).
b) Execute standard usage scenario on various currently prevalent productive system environments and check for portability of data and software settings.

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?

This criterion rewards software manufacturers who make it easy for their customers to continue to use their existing hardware. It intentionally does not take into account whether functionality is expanded. Sufficiency means that the amount of resources required will *not* increase even if the utility they provide increases (which is possible, after all, because of increasing efficiency).

The ideal is a software product that fulfills more and more requirements from one version to the next, but nonetheless does not increase its hardware requirements.

This criterion can be applied only when products have already been assessed several times, i.e., when at least one previous result is available.

Indicators:

a) intertemporal comparisons with the following imaginable results:

1. "very good": To date, new versions have resulted in a decrease in the hardware capacities required.
2. "good": To date, new versions have resulted in no increase in the amount of hardware capacities required.
3. "sufficient": Although to date, new versions have increased the amount of hardware capacities required, the increases have not overcompensated for the efficiency improvements due to technical factors as exhibited by the succession of reference systems over time.
4. "insufficient": Because of new versions, the required hardware capacities have increased faster than technical efficiency.

[14] We recommend that this criterion should not be considered one of the minimum requirements because in principle, there could be very resource-efficient software that runs on just one platform. Nonetheless, platform independence is to be considered beneficial since it gives users more freedom when optimizing procurement of hardware and system software.

3 User autonomy

Does the manufacturer of the software product respect user autonomy in dealing with the purchased product?

This main criterion assumes that a relevant number of users is interested in using software in a resource-efficient way. If they can do so without functional disadvantages, they will try to work with a small amount of hardware capacity (which they generally pay for) and keep energy consumption low (which is also financially relevant or at least impacts the battery life of mobile devices). However, users can do so only if they are not forced to consume unnecessary amounts of resources and if they understand how they can avoid unnecessary resource consumption.

The ideal is a software product that respects the freedom of users to decide about utilizing hardware capacities (and thus indirectly about using resources) when using the product, as far as possible.

The following criteria are to be evaluated from the perspective of target groups that are not technical specialists; in other words, they will generally not be fulfilled simply by the fact that an expert can fulfill them. Criterion 3.1.2 is an exception in this regard.

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?

3.1.1 Transparency of data formats and data portability

Is sufficient documentation provided for the data formats (file or data stream formats) used by the software product to enable interoperability? Do the data formats comply with open standards enabling further use of the data with another software product?[15]

To apply this criterion, it must first be defined which standards are considered open standards at the time of awarding a label.

Indicators:

a) Review of manuals and technical data sheets, comparison with known open standards
b) Check of compliance with known and open standards.

[15] This is decisive to prevent customer lock-in (dependence on the software product), which may force unnecessary resource consumption, both in the case of retaining an inefficient product and in the case of switching to a different product, which may require resources as well.

3.1.2 Transparency and interoperability of the programs

Are application programming interfaces (APIs) clearly documented, and are dissemination and further development of the program supported? Do the interfaces comply to open standards to enable interoperability?

Weighting of indicators may be highly dependent on context. The effects of open source code and licensing models on resource use cannot be assessed in terms of a general rule.

Indicators:

a) If APIs exist: Review of the documentation of the interfaces on the basis of the documentation of the software product and its APIs
b) Is the source code open?
c) Is the software released under a license that allows it to further develop it?

3.1.3 Continuity of the software product

Can the software product be used for longer periods of time without serious negatives (in particular IT security problems) occurring, and does the user have the option to avoid unnecessary updates?[16]

Indicators:

a) How long is the time period for which the supplier guarantees future support for the product, including security updates?
b) Does the manufacturer respond promptly when security gaps (vulnerabilities) become known?
c) Can the user influence the frequency of updates by configuring the software product and when doing so differentiate between security updates and other updates?
d) Is it possible to receive differential updates only?[17]

3.1.4 Transparency of task management

Does the software product inform users that it is automatically launching or running tasks in the background that are possibly not being used?

Indicators:

a) On the basis of the installation and the execution of standard usage patterns, test which processes are automatically launched by the software product and whether it informs users of this (Scale: informs users of all such processes/informs users of some such processes/does not inform users)
b) If the software product is automatically launched at system start ("autostart"): does it inform users that this is the case?
c) If the user carries out an action that can be understood as ending the program, but at least one of the tasks remains active: does the software product inform the user that this is the case?

[16] A high frequency of updates causes resource consumption and makes it more difficult to maintain transparency. It is difficult to define the "necessity" of updates objectively; however, it makes at least sense to differentiate between security-relevant (and thus doubtless necessary) updates and other updates; this is addressed by indicator b).

[17] This avoids replacing the entire program, which can cause significant resource consumption if performed frequently.

3.2 Uninstallability

Can the software product be uninstalled easily, without leaving traces, and without avoidable disadvantages?

3.2.1 Uninstallability of programs

Does the user receive sufficient support to uninstall the program without leaving traces?

Indicators:

a) Uninstallation of the software and comparison with the condition prior to installation, which must be identical.

3.2.2 Capability to erase data

Does the user receive sufficient support when erasing data generated during operation of the software product as desired?

This criterion is intended specially to avoid the case that compliance with high IT security standards following uninstallation of the software product can be guaranteed only by physically destroying hardware.

Indicators:

a) After erasing of the data explicitly stored by the user and comparison with the condition prior to installation, are the two states identical in relevant respects?
b) Does the software product provide transparency about the places where it stores data?
c) Is the user supported in erasing data stored on remote storage devices without leaving traces?

3.3 Maintenance functions

Does the software product provide easy-to-use functions permitting users to repair damage to data and programs?

3.3.1 Recoverability of data

Can the data be recovered in its last condition following an abnormal termination?

Indicators:

a) Does the manufacturer provide specifications and can they be validated by means of a test?
b) Can the user set the periodicity at which changes are automatically saved?

3.3.2 Self-recoverability

Can the installed instance of the software product be recovered following the occurrence of an inconsistent state?

Indicators:

a) Manufacturer specifications and review by means of a test

3.4 Independence of outside resources

Can the software product be operated as independently as possible of resources not subject to the users' control?

3.4.1 Offline capability

To what extent does the software product avoid forced connectivity that is not necessary for providing the functionality?[18]

Indicators:

a) Testing on the basis of the standard usage scenario (Scale: offline operation possible/possible with limitations/impossible)

3.5 Quality of product information

Does the information provided about the software product support its resource-efficient use?

3.5.1 Comprehensibility and manageability of product documentation, licensing conditions, and terms of use

Is all the information easy for users to understand?

Indicators:

a) Inspection by reviewers; test with actual users

[18] Examples of unnecessarily forced connectivity: establishing a connection to the license server, repeated download of fonts required.

3.5.2 Resource relevance of product information

Does the product information include everything that users need to minimize resource consumption by the software product in a structured form, and is the information correct?

The long-term goal is to develop standardized product descriptions for resource-relevant product information. As soon as a satisfactory standard exists in this regard, compliance with it can be included as an indicator.

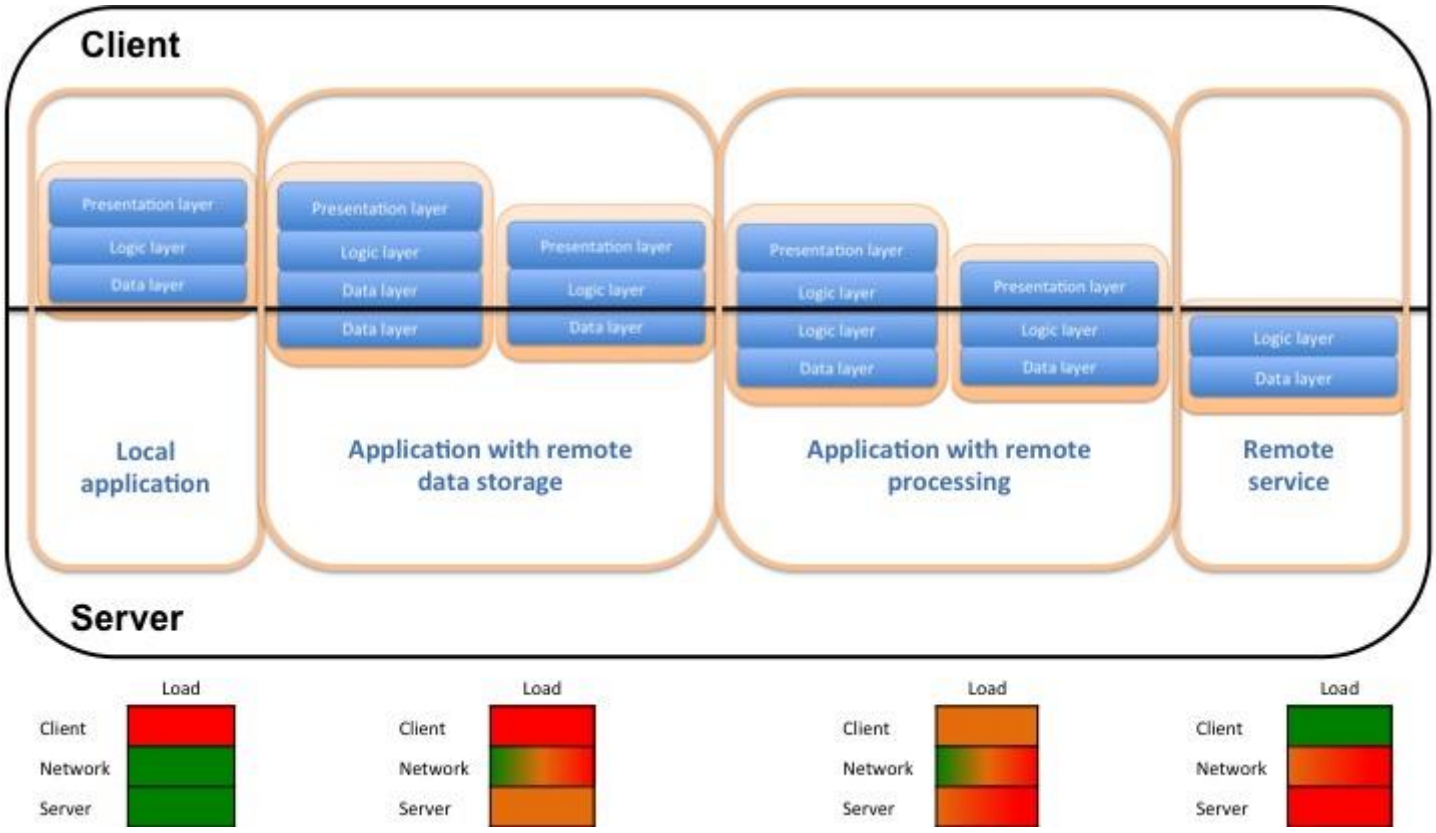
Indicators:

a) Qualitative assessment of completeness and comprehensibility
b) Does the product information refer to the current version of the product?
c) Inspection whether the information is correct (information is conclusive / partially conclusive / non-conclusive)

Appendix A: Classification of application software

Classification of application software in terms of the software architecture

Focus: Division of work between client and server

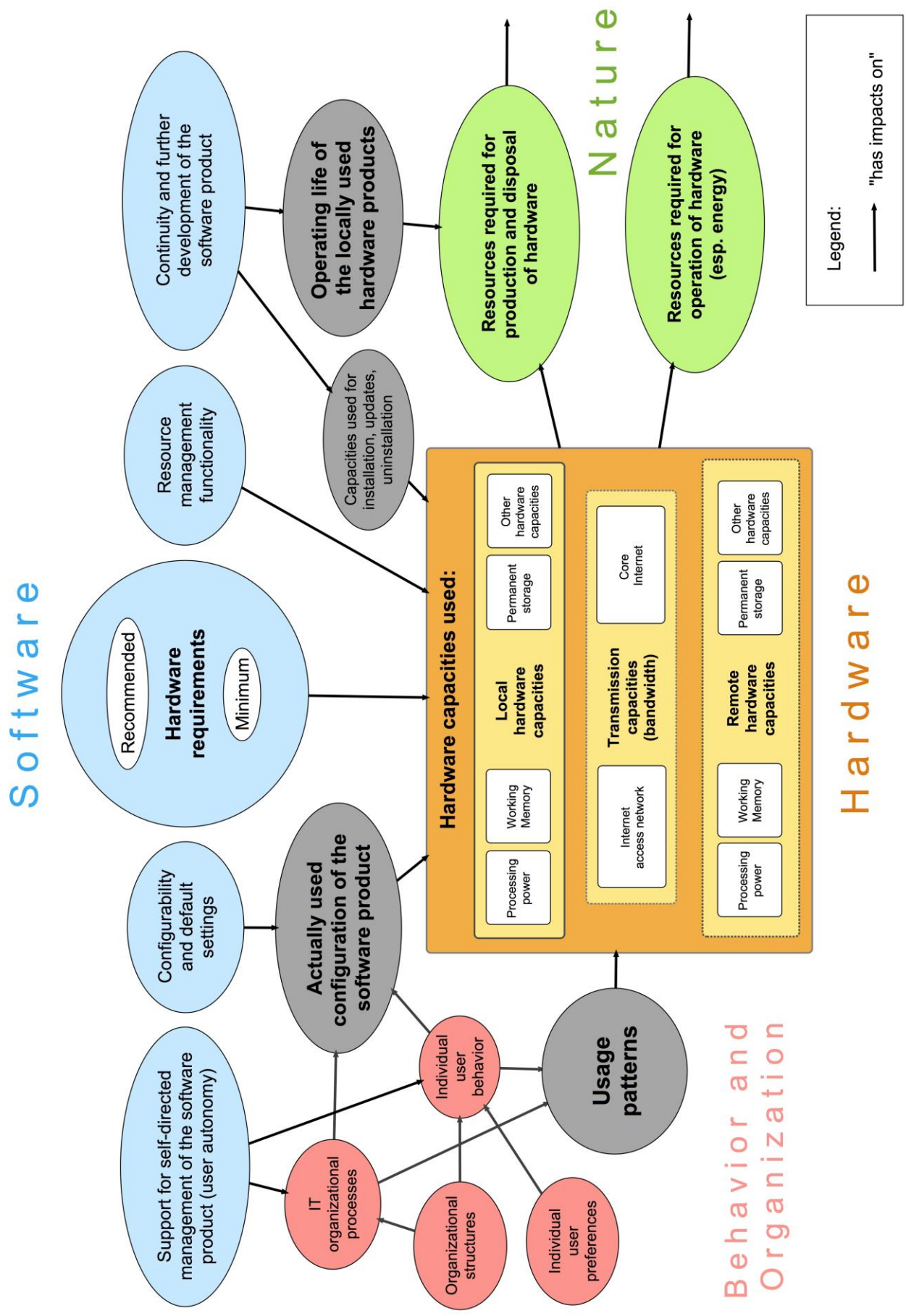


Explanations:

- Presentation layer: the parts of the software product responsible for interaction with users.
- Logic layer: the functional core of the software product; it includes all processing mechanisms and access to the data layer.
- Data layer: the parts of the software product responsible for storing and accessing data, e.g., in a database.

Colors indicate ranges of percentage of the total load caused by using the software product between zero (green) and 100% (red).

Appendix B: Impact model



Appendix C: Glossary

Energy efficiency: Generally, the amount of "useful work" divided by the amount of energy it requires. In the context of this document, "useful work" is operationalized as the successful execution of standard usage scenarios.

Hardware: The material goods required to run programs or to store or transport data.

Hardware capacity: Quantifiable characteristic of a hardware system which represents its performance limit on a given dimension of performance (e.g., working memory capacity, computing power, bandwidth).

Hardware system: Delimitable unit of hardware that performs defined functions.

Indicator: An empirically determinable quantity that provides insight into a matter that cannot be measured directly. The indicators proposed in this document have different levels of measurement. In some cases, researchers will have to settle for an ordinal scale (e.g., "insufficient", "sufficient", "good", "very good", or even merely "fulfilled", "not fulfilled") to avoid giving the false impression of non-existent precision arising from a cardinal scale.

Reference system: A hardware system that is defined as generally customary in terms of its most important capacities (e.g., working memory, processor performance) during a defined period of time (e.g., one year). The purpose of the reference system is to be able to express indicators such as "minimum local memory" in relation to a reference value (currently "customary" memory).

Resource: In the context of this document, a natural resource, in particular a raw material, a form of energy, or also the capacity of an environmental medium to absorb emissions. To differentiate natural resources from technical ones, especially hardware resources, the more precise term "hardware capacities" is used here for the latter. Since using hardware capacities always results in using natural resources, this distinction (which ultimately amounts to a definitionally difficult differentiation between the ecosphere and the technosphere) is not of decisive importance here.

Resource efficiency: Generally, the amount of "useful work" divided by the amount of resources it requires. In the context of this document, "useful work" is operationalized as the successful execution of standard usage scenarios.

Software: Programs and data in digital form.

Software product: A delimitable unit of programs and data for which a license is available.

Standard configuration: A set of conditions, defined as a reference, under which a given software product is run; it includes the parameter settings selected during installation or operation, the system software provided, potentially additional software products required for operation, as well as the reference system at the hardware level.

Standard usage scenario: A usage scenario that is used for testing a software product and is supposed to be as representative as possible for the customary use case.

Usage pattern: Abstracted form of a sequence of interactions with a given software product.

Usage scenario: Description of a usage pattern which is generally machine executable.

Appendix D: Bibliography

Abdullah, Rusli; Abdullah, Salfarina; Din, Jamilah; Tee, Mcxin; others (2015): A Systematic Literature Review of Green Software Development in Collaborative Knowledge Management Environment. In: International Journal of Advanced Computer Technology (IJACT) 9, S. 136.

Abdullah, Rusli; Abdullah, Salfarina; Tee, Mcxin (2014): Web-based knowledge management model for managing and sharing green knowledge of software development in community of practice. In: Software Engineering Conference (MySEC), 2014 8th Malaysian. IEEE, S. 210–215.

Afgan, Naim Hamdia (2010): Sustainability paradigm: intelligent energy system. In: Sustainability 2 (12), S. 3812–3830.

Afzal, Shehla; Saleem, M. Faisal; Jan, Fahad; Ahmad, Mudassar (2013): A Review on Green Software Development in a Cloud Environment Regarding Software Development Life Cycle:(SDLC) Perspective. In: International Journal of Computer Trends and Technology (IJCTT) 4 (9), S. 3054–3058.

Agarwal, Shalabh; Nath, Asoke; Chowdhury, Dipayan (2012): Sustainable Approaches and Good Practices in Green Software Engineering. In: IJRRCS 3 (1), S. 1425–1428. Available online scholarlyexchange.org, zuletzt geprüft am 15.03.2012.

Ahmad, Ruzita; Baharom, Fauziah; Hussain, Azham (2014): A Systematic Literature Review on Sustainability Studies in Software Engineering. In: Proceedings of KMICe. Knowledge Management International Conference (KMICe) 2014. Malaysia, 12 – 15 August 2014.

Albertao, Felipe (2004): Sustainable Software Engineering. Carnegie Mellon University Silicon Valley. Available online www.scribd.com, zuletzt aktualisiert am 04.09.2008, zuletzt geprüft am 30.11.2010.

Albertao, Felipe; Xiao, Jing; Tian, Chunhua; Lu, Yu; Zhang, Kun Qiu; Liu, Cheng (2010): Measuring the Sustainability Performance of Software Projects. In: IEEE Computer Society (Hg.): 2010 IEEE 7th International Conference on e-Business Engineering (ICEBE 2010), Shanghai, China. Technical Committee on Electronic Commerce (TCEC), S. 369–373. Available online doi.ieeecomputersociety.org, zuletzt geprüft am 04.03.2011.

Amsel, Nadine; Ibrahim, Zaid; Malik, Amir; Tomlinson, Bill (2011): Toward sustainable software engineering (NIER track). In: Proceedings of the 33rd International Conference on Software Engineering. ACM, S. 976–979.

Ardito, Luca; Morisio, Maurizio (2014): Green IT - Available data and guidelines for reducing energy consumption in IT systems. In: Sustainable Computing: Informatics and Systems 4 (1), S. 24–32.

RAL-UZ 161, 2012-07: Basic Criteria for Award of the Environmental Label Energy-Conscious Data Centers. Available online www.eco-institut.de

Berkhout, Frans; Hertin, Julia (2001): Impacts of Information and Communication Technologies on Environmental Sustainability: speculations and evidence. Report to the OECD. Hg. v. Organisation for Economic Co-operation and Development OECD. Brighton. Available online www.oecd.org, zuletzt geprüft am 02.03.2011.

Bouwers, Eric; van Deursen, Arie; Visser, Joost (2013): Evaluating usefulness of software metrics: an industrial experience report. In: Proceedings of the 2013 International Conference on Software Engineering. IEEE Press, S. 921–930.

Bozzelli, Paolo; Gu, Qing; Lago, Patricia (2013): A systematic literature review on green software metrics. Technical Report: VU University Amsterdam.

Calero, C.; Bertoa, M.F; Angeles Moraga, M. (2013a): A systematic literature review for software sustainability measures. In: Green and Sustainable Software (GREENS), 2013 2nd International Workshop on, S. 46–53.

Calero, Coral; Bertoa, Manuel F.; Moraga, Maria Ángeles (2013b): Sustainability and Quality: Icing on the Cake. In: RE4SuSy@RE. Citeseer.

Calero, Coral; Moraga, M.; Bertoa, Manuel F. (2013c): Towards a software product sustainability model. In: arXiv preprint arXiv:1309.1640.

Calero, Coral; Moraga, Maria Ángeles; Bertoa, Manuel F.; Duboc, Leticia (2015): Green Software and Software Quality. In: Coral Calero und Mario Piattini (Hg.): Green in Software Engineering: Springer, S. 231–260.

Capra, E.; Francalanci, C.; Slaughter, S. A. (2012): Measuring Application Software Energy Efficiency. In: IT Professional, S. 54–61.

Capra, Eugenio; Francalanci, Chiara; Slaughter, Sandra A. (2011): Is software green? Application development environments and energy efficiency in open source applications. In: Information and Software Technology 54, S. 60–71.

Dick, Markus; Naumann, Stefan (2010): Enhancing Software Engineering Processes towards Sustainable Software Product Design. In: Klaus Greve und Armin B. Cremers (Hg.): EnviroInfo 2010: Integration of Environmental Information in Europe. Proceedings of the 24th International Conference on Informatics for Environmental Protection, October 6 - 8, 2010, Cologne/Bonn, Germany. Aachen: Shaker, S. 706–715.

EPA ENERGY STAR (2014): ENERGY STAR Program Requirements Product Specification for Computers: Eligibility Criteria, Version 6.1. Environmental Protection Agency. Available online www.energystar.gov.

EPA Office of Air and Radiation, Climate Protection Partnerships Division (2015): National Awareness of ENERGY STAR for 2014. Analysis of CEE Household Survey. Hg. v. U.S. Environmental Protection Agency. Available online www.energystar.gov.

Erdmann, Lorenz; Hilty, Lorenz M.; Goodman, James; Arnfalk, Peter (2004): The Future Impact of ICTs on Environmental Sustainability. Technical Report EUR 21384 EN. Hg. v. Carlos Rodríguez Casal, Christine Van Wunnik, Luis Delgado Sancho, Jean Claude Burgelman und Paul Desruelle. European Commission; Joint Research Centre; IPTS - Institute for Prospective Technological Studies. Seville (Technical Report Series, EUR 21384 EN). Available online ftp.jrc.es, zuletzt geprüft am 26.07.2011.

Europäische Union (Hg.) (2011a): Beschluss der Kommission vom 6. Juni 2011 zur Festlegung der Umweltkriterien für die Vergabe des EU-Umweltzeichens für Notebooks. (Bekannt gegeben unter Aktenzeichen K(2011) 3736)Text von Bedeutung für den EWR. Available online eur-lex.europa.eu.

Europäische Union (2011b): Beschluss der Kommission vom 9. Juni 2011 zur Festlegung der Umweltkriterien für die Vergabe des EU-Umweltzeichens für Tischcomputer. (Bekannt gegeben unter Aktenzeichen K(2011) 3737)Text von Bedeutung für den EWR. Available online eur-lex.europa.eu.

Finkbeiner, Matthias; Schau, Erwin M.; Lehmann, Annekatrin; Traverso, Marzia (2010): Towards life cycle sustainability assessment. In: Sustainability 2 (10), S. 3309–3322. Available online www.mdpi.com.

Fujitsu Technology Solutions (Hg.) (2010): Green Label-Kategorien bei Fujitsu Technology Solutions. White Paper.

Fujitsu Technology Solutions (Hg.) (2012): Green Label Levels at Fujitsu Technology Solutions. White Paper. Available online globalsp.ts.fujitsu.com, zuletzt aktualisiert am 25.04.2012, zuletzt geprüft am 02.01.2013.

GeSI, Global e-Sustainability Initiative; The Climate Group (2008): SMART 2020: Enabling the low carbon economy in the information age.

Gröger, Jens; Köhn, Marina; Albers, Erik; Löhr, Patrik; Lohmann, Wolfgang; Naumann, Stefan (2015): Nachhaltige Software. Dokumentation des Fachgesprächs „Nachhaltige Software“ am 28.11.2014. Hg. v. Umweltbundesamt. Öko-Institut e.V. Dessau-Roßlau. Available online www.umweltbundesamt.de.

Gröger, Jens; Quack, Dietlinde; Grießhammer, Rainer; Gattermann, Marah (2013): TOP 100 - Umweltzeichen für klimarelevante Produkte: Freiburg. Available online www.ecodialog.de.

Held, Alexandra (2010): Entwicklung und Operationalisierung von Kriterien zur Bewertung der Nachhaltigkeit von Softwareprodukten. Abschlussarbeit zur Erlangung des akademischen Grades Master of Science eingereicht am Umwelt-Campus Birkenfeld. Masterarbeit. Fachhochschule Trier, Standort Umwelt-Campus Birkenfeld, Hoppstädten-Weiersbach. ISS Institut für Softwaresysteme in Wirtschaft, Umwelt und Verwaltung.

Hilty, Lorenz; Lohmann, Wolfgang; Behrendt, Siegfried; Evers-Wölk, Michaela; Fichter, Klaus; Hintemann, Ralph (2015): Grüne Software. Schlussbericht zum Vorhaben: Ermittlung und Erschließung von Umweltschutzpotenzialen der Informations- und Kommunikationstechnik (Green IT). Studie im Auftrag des Umweltbundesamtes, Berlin, Förderkennzeichen 3710 95 302/3 (im Druck).

Horne, Ralph E. (2009): Limits to labels: The role of eco-labels in the assessment of product sustainability and routes to sustainable consumption. In: International Journal of Consumer Studies 33 (2), S. 175–182. Available online 19-659-fall-2011.wiki.uml.edu.

Kern, Eva; Dick, Markus; Naumann, Stefan; Guldner, Achim; Johann, Timo (2013): Green Software and Green Software Engineering – Definitions, Measurements, and Quality Aspects. In: Lorenz M. Hilty, Bernard Aebischer, Göran Andersson und Wolfgang Lohmann (Hg.): ICT4S ICT for Sustainability. Proceedings of the First International Conference on Information and Communication Technologies for Sustainability, ETH Zurich, February 14-16, 2013. Zürich: ETH Zurich, University of Zurich and Empa, Swiss Federal Laboratories for Materials Science and Technology, S. 87–94. Available online e-collection.library.ethz.ch.

Koçak, Sedef Akınlı; Calienes, Giovanna Gonzales; Alptekin, Gülfem Işıklar; Bener, Ayşe Başar (2013): Requirements Prioritization Framework for Developing Green and Sustainable Software using ANP-based Decision Making. In: EnviroInfo, S. 327–335.

Koçak, Sedef Akınlı; Alptekin, Gülfem Işıklar; Bener, Ayşe Başar (2014): Evaluation of Software Product Quality Attributes and Environmental Attributes using ANP Decision Framework. In: Proceedings of the Third International Workshop on Requirement Engineering for Sustainable Systems (pp. pp. 37-44). Karlskrona: Central Europe Workshop Proceedings. Available online ceur-ws.org

Lago, Patricia; Jansen, Toon; Jansen, Marten (2010): The service greenery-integrating sustainability in service oriented software. In: International Workshop on Software Research and Climate Change (WSRCC), co-located with ICSE, Bd. 2.

Lago, Patricia; Koçak, Sedef Akinli; Crnkovic, Ivica; Penzenstadler, Birgit (2015): Framing sustainability as a property of software quality. In: Communications of the ACM 58 (10), S. 70–78.

Lami, Giuseppe; Fabbrini, Fabrizio; Fusani Mario (2012): Software Sustainability from a Process-Centric Perspective. In: D. Winkler, R.V O'Connor und R. Messnarz (Hg.): EuroSPI 2012, CCIS 301: Springer, S. 97–108.

Mazijn, B.; Doom, R.; Peeters, H.; Vanhoutte, G.; Spillemaeckers, S.; Taverniers, L. et al. (2004): Ecological, Social and Economic Aspects of Integrated Product Policy. Integrated Product Assessment and the Development of the Label `Sustainable Development´ for Products. CP/20. SPSD II - Part I - Sustainable production and consumption patterns. Available online www.bernardmazijn.be.

Naumann, Stefan; Dick, Markus; Kern, Eva; Johann, Timo (2011): The GREENSOFT Model: A Reference Model for Green and Sustainable Software and its Engineering. In: SUSCOM 1 (4), S. 294–304. DOI: 10.1016/j.suscom.2011.06.004.

Penzenstadler, Birgit; Mahaux, Martin; Salinesi, Camille (2013): RE4SuSy: Requirements Engineering for Sustainable Systems. In: Journal of Systems and Software.

Prakash, Siddharth; Manhart, Andreas; Stratmann, Britta; Reintjes, Norbert (2008): Environmental product indicators and benchmarks in the context of environmental labels and declarations. Öko-Institut e.V.; Ökopol GmbH.

Schipper, Irene (2015): TCO Certified Smartphones versusFairphone. A comparison of sustainability criteria. Hg. v. GoodElectronics Network Südwind. Stichting Onderzoek Multinationale Ondernemingen (SOMO), Centre for Research on Multinational Cooperations, Netherlands. Amsterdam.

Schmidt, Benno (2014): Strategien für eine integrativ-nachhaltige Software-Entwicklung. Hochschule Bochum, Fachbereich Geodäsie. Bochum (14-02).

Schmidt, Benno; Wytzisk, Andreas; Plödereder, In E.; Grunske, L.; Schneider, E.; Ull, D.; others (2014): Software Engineering und Integrative Nachhaltigkeit. In: Erhard Plödereder, Lars Grunske, Eric Schneider und Dominic Ull (Hg.): INFORMATIK 2014. Big Data – Komplexität meistern. – Proceedings. 2. Workshop "Umweltinformatik zwischen Nachhaltigkeit und Wandel (UINW) / Environmental Informatics between Sustainability and Change"), 26.09.2014, im Rahmen der INFORMATIK 2014, 22.-26.09.2014 in Stuttgart. P-232: Lecture Notes in Informatics (LNI), S. 1935–1945.

Scholl, Gerd; Simshäuser, Ulla (2002): Machbarkeitsuntersuchung für Umweltzeichen-Analyse der Möglichkeiten zur Akzeptanzerhöhung des Umweltzeichens" Blauer Engel" für Haushaltsgroßgeräte (" Weiße Ware") bei potenziellen Zeichennehmern: Umweltbundesamt. Available online www.umweltbundesamt.de.

Stieß, Immanuel; Birzle-Harder, Barbara (2013): Der Blaue Engel – ein Klassiker mit Potenzial: eine empirische Studie zu Verbraucherakzeptanz und Marktdurchdringung des Umweltzeichens.

Sundblad, Yngve; Lind, Torbjörn; Rudling, Jan (2002): IT product requirements and certification from the users' perspective. In: Proceedings of WWDU 2002 Conference, S. 280–282. Available online http://cid.nada.kth.se/pdf/CID-176.pdf

International Standard ISO/IEC 25010:2011, 01.03.2011: Systems and software engineering -- Systems and software Quality Requirements and Evaluation (SQuaRE) -- System and software quality models.

Taina, Juha (2011): Good, Bad, and Beautiful Software - In Search of Green Software Quality Factors. In: CEPIS UPGRADE XII (4), S. 22–27. Available online www.cepis.org, zuletzt geprüft am 09.01.2012.

TCO Development (Hg.) (2012): TCO Certified Notebooks 4.0. Available online tcodevelopment.com.

Teufel, J.; Rubik, F.; Scholl, G.; Stratmann, B.; Graulich, K.; Manhart, A. (2009): Untersuchung zur möglichen Ausgestaltung und Marktimplementierung eines Nachhaltigkeitslabels zur Verbraucherinformation. In: Project report of the Öko-Institut e. V. in cooperation with the Institut für ökologische Wirtschaftsforschung (IÖW) GmbH. Freiburg: Öko-Institut e. V. Available online download.ble.de.

Umweltbundesamt (Hg.) (2013): Ökodesign-Richtlinie <Computer und Computerserver>. Available online www.umweltbundesamt.de.

Vergabegrundlage für Umweltzeichen RAL-UZ 100, 2014-06: Vergabegrundlage für Car-Sharing.

Vergabegrundlage für Umweltzeichen RAL-UZ 78a, 2014-11: Vergabegrundlage für Umweltzeichen - Computer.

Walch, Isabelle (2015): Standard ECMA-370. TED - The ECO Declaration. Hg. v. ecma INTERNATIONAL.

Waller, Lars (2015): TCO Certified Desktops 5.0 - Critera Document and Certification Process. TCO Development. Available online tcodevelopment.com, zuletzt aktualisiert am 11.11.2015.

Warschun, Mirko; Rühle, Jens (2008): Zwischen ÖkoLabels, grüner Logistik und fairem Handel Lebensmitteleinzelhandel auf der Suche nach Wegen zur Nachhaltigkeit.