Future Generation Computer Systems 155 (2024) 402–418

Contents lists available at ScienceDirect

Future Generation Computer Systems

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

# Development and evaluation of a reference measurement model for assessing the resource and energy efficiency of software products and components—Green Software Measurement Model (GSMM)

Achim Guldner[a,*], Rabea Bender[a], Coral Calero[b], Giovanni S. Fernando[c], Markus Funke[c], Jens Gröger[d], Lorenz M. Hilty[e], Julian Hörnschemeyer[f], Geerd-Dietger Hoffmann[g], Dennis Junger[h], Tom Kennes[i], Sandro Kreten[j], Patricia Lago[c], Franziska Mai[a], Ivano Malavolta[c], Julien Murach[a], Kira Obergöker[a], Benno Schmidt[k], Arne Tarara[g], Joseph P. De Veaugh-Geiss[l], Sebastian Weber[a], Max Westing[a], Volker Wohlgemuth[h], Stefan Naumann[a]

[a] *Trier University of Applied Sciences, Umwelt-Campus Birkenfeld, PO-Box 1380, Birkenfeld, 55761, Germany*
[b] *University of Castilla-La Mancha, Alarcos Research Group, Paseo de la Universidad, 4, Ciudad Real, 13071, Spain*
[c] *Vrije Universiteit Amsterdam, Software and Sustainability Research Group, NU Building, De Boelelaan 1111, Amsterdam, 1081 HV, The Netherlands*
[d] *Oeko-Institut, Borkumstrasse 2, Berlin, 13189, Germany*
[e] *University of Zurich, Department of Informatics, Binzmühlestrasse 14, CH-8050, Zurich, 49076, Germany*
[f] *RegioShopper, Marie-Curie-Str. 3, Osnabrück, 49076, Germany*
[g] *Green Coding Berlin, Jablonskistr. 24, Berlin, 10405, Germany*
[h] *HTW Berlin, University of Applied Sciences, Industrial Environmental Informatics Unit, Wilhelminenhofstr. 75A, Berlin, 12459, Germany*
[i] *Nationale Nederlanden, Pr. Beatrixlaan 35, Den Haag, 2595 AK, The Netherlands*
[j] *capacura, Haus Auel 1, Lohmar, 53797, Germany*
[k] *Bochum University of Applied Sciences, Am Hochschulcampus 1, Bochum, 44801, Germany*
[l] *KDE e.V., Prinzenstraße 85 F, Berlin, 10969, Germany*

## ARTICLE INFO

*Keywords:*
Measurement reference model
Green software
Sustainable software
Energy-efficient software
Green coding
Software-induced energy consumption
Software-induced resource consumption

## ABSTRACT

In the past decade, research on measuring and assessing the environmental impact of software has gained significant momentum in science and industry. However, due to the large number of research groups, measurement setups, procedure models, tools, and general novelty of the research area, a comprehensive research framework has yet to be created. The literature documents several approaches from researchers and practitioners who have developed individual methods and models, along with more general ideas like the integration of software sustainability in the context of the UN Sustainable Development Goals, or science communication approaches to make the resource cost of software transparent to society. However, a reference measurement model for the energy and resource consumption of software is still missing. In this article, we

* Corresponding author.
*E-mail addresses:* a.guldner@umwelt-campus.de (A. Guldner), r.bender@umwelt-campus.de (R. Bender), coral.calero@uclm.es (C. Calero), g.s.fernando@student.vu.nl (G.S. Fernando), m.t.funke@vu.nl (M. Funke), j.groeger@oeko.de (J. Gröger), hilty@ifi.uzh.ch (L.M. Hilty), julian.hoernschemeyer@mailbox.org (J. Hörnschemeyer), didi@green-coding.berlin (G.-D. Hoffmann), dennis.junger@htw-berlin.de (D. Junger), tom.kennes@nn.nl (T. Kennes), sandro.kreten@capacura.de (S. Kreten), p.lago@vu.nl (P. Lago), f.mai@umwelt-campus.de (F. Mai), i.malavolta@vu.nl (I. Malavolta), murach@t-online.de (J. Murach), k.obergoeker@umwelt-campus.de (K. Obergöker), benno.schmidt@hs-bochum.de (B. Schmidt), arne@green-coding.berlin (A. Tarara), joseph@kde.org (J.P. De Veaugh-Geiss), seb.weber@umwelt-campus.de (S. Weber), m.westing@umwelt-campus.de (M. Westing), volker.wohlgemuth@htw-berlin.de (V. Wohlgemuth), s.naumann@umwelt-campus.de (S. Naumann).
*URLs:* https://green-software-engineering.de/ (A. Guldner), https://iss.umwelt-campus.de/ (R. Bender), https://alarcos.esi.uclm.es/ (C. Calero), https://s2group.cs.vu.nl (G.S. Fernando), https://s2group.cs.vu.nl (M. Funke), https://www.oeko.de/en (J. Gröger), https://www.ifi.uzh.ch/en/isr/people/people/hilty.html (L.M. Hilty), https://regioshopper.de (J. Hörnschemeyer), https://www.green-coding.berlin/ (G.-D. Hoffmann), https://www.linkedin.com/in/dennismjunger/ (D. Junger), https://www.nn.nl (T. Kennes), https://www.linkedin.com/in/sandro-kreten-1a8964165/ (S. Kreten), https://s2group.cs.vu.nl (P. Lago), https://green-software-engineering.de/ (F. Mai), https://s2group.cs.vu.nl (I. Malavolta), https://green-software-engineering.de/ (K. Obergöker), https://www.hs-bochum.de (B. Schmidt), https://www.green-coding.berlin/ (A. Tarara), https://eco.kde.org/ (J.P. De Veaugh-Geiss), https://green-software-engineering.de/ (M. Westing), http://wohlgemuth.f2.htw-berlin.de/ (V. Wohlgemuth), https://www.green-software-engineering.de/ (S. Naumann).

https://doi.org/10.1016/j.future.2024.01.033
Received 17 August 2023; Received in revised form 1 December 2023; Accepted 31 January 2024
Available online 16 February 2024

jointly develop the *Green Software Measurement Model (GSMM)*, in which we bring together the core ideas of the measurement models, setups, and methods of over 10 research groups in four countries who have done pioneering work in assessing the environmental impact of software. We briefly describe the different methods and models used by these research groups, derive the components of the GSMM from them, and then we discuss and evaluate the resulting reference model. By categorizing the existing measurement models and procedures and by providing guidelines for assimilating and tailoring existing methods, we expect this work to aid new researchers and practitioners who want to conduct measurements for their individual use cases.

## 1. Introduction

The expansion of the Information and Communications Technology (ICT) sector in recent years has led to remarkable growth in its environmental impact. Recent reports conclude that approximately 2–3 %, or 1.0–1.7 GtCO$_2$e,[1] of the total global greenhouse gas emissions stem from the ICT industry's energy use [1]. Moreover, estimates indicate that global ICT energy use could exceed 20 % of total energy and emit up to 5.5% of the world's carbon emissions by 2025. This would have a large negative impact on the environment [2,3]. This is even more relevant with the increased development and usage of new technologies like artificial intelligence and especially machine learning based systems or distributed ledgers, which consume significant amounts of computational resources and energy [4,5].

While hardware consumes energy and resources, it is software that triggers its usage and thus can influence the consumption. The unquestionable influence of software on energy consumption is fostering research in the field of "green software" [6–9]. Several works discuss definitions of software sustainability and how to address it by better understanding and measuring how software drives energy and resource consumption [10–14]. However, software products become complex very fast and, as stated by Pang et al. [15], programmers do not have much experience with software-induced energy consumption. Lago et al. [16] conducted a survey about the experience and beliefs of IT practitioners and researchers on current and desired practices in architecting for sustainability. They conclude that the main obstacles preventing the inclusion of sustainability issues in software development projects are business motivation, short-term thinking, a lack of agreement on why sustainability is important, and an absence of knowledge on what concrete measures can be taken. Developers used to have to rely on Q&A websites, blog posts, or videos when trying to optimize for energy consumption, but these resources are not supported by empirical evidence and may even be incorrect [17,18].

Fortunately, research and developer communities have produced and tested guidelines, tools, and criteria to aid developers. For example, Pereira et al. [19] analyzed 27 software languages with the aim of helping software engineers decide which language to use from an energy efficiency perspective. Chowdhury et al. [20,21] proposed a model which is based on dynamic traces of system calls and CPU utilization in order to estimate the energy consumption of software. Moreover, the influence of software architecture on energy consumption has been addressed by Guamán and Pérez [22] and Cabot et al. [23]. Additionally, as stated by Manotas et al. [18], energy concerns have been largely ignored during, e. g., the maintenance phase. Cruz et al. [24] investigated whether improving energy efficiency by applying energy efficiency standards has a negative impact on maintainability, and Calero et al. [25] used a hardware measurement device to analyze the relationship between maintainability and software energy efficiency. Estimations of the energy consumption of software of mobile applications have also been the object of study [26,27].

What is missing is a comprehensive measurement reference model that aids stakeholders in the life cycle of a software product in order to gauge the growing supply of available measurement methods and models. This should then enable them to develop, plan, conduct, and analyze measurements for their software, using a selected method or creating their own. To do so, in this article, we present a reference model by structuring existing research to provide recommendations, processes, and tools for practitioners to assess, and thus minimize, the environmental impact of software. Section 2 presents the need for a measurement reference model, while Sections 3 and 4 outline the contemporary understanding, approach, and available methods for measurements in the context of Green Software Engineering and Green Coding. In Section 5, we present and describe the measurement reference model. Section 6 discusses the model by exemplarily mapping the existing methods from Section 4. Finally, we summarize our findings and provide an outlook with the overarching goal of the continuous improvement of the measurement reference model in Section 7.

## 2. Problem statement

With the rising resource and energy consumption induced by software, it is necessary for stakeholders in the software life cycle to be able to measure and continuously monitor this consumption in order to minimize software-induced environmental impacts and to enhance resource efficiency. Consequently, it is necessary to develop applicable and valid measurement methods that are actually usable in developer and user communities. Therefore, the aim of our proposed Green Software Measurement Model (GSMM) is to provide a framework that contains essential elements for measuring software and to present existing measurement methods. In this way, measuring and improving the resource and energy efficiency of software can become part of the daily work of the involved stakeholders, such as developers, administrators, and users.

Currently, there is no consensus on measurement setups, methods, or techniques for data analysis. With each researcher applying their own methods, often with little to no documentation or publicly available data (e. g., in the form of replication packages), it is difficult and sometimes outright impossible to check or compare results obtained across studies, to replicate analyses, or to re-use data. To solve this problem, we propose establishing a reference model for measurement and analysis methods to assess the resource and energy efficiency of software.

In this paper, we collect 8 methods from international groups dealing with energy and resource measurements and analyses of software. We assess them and other procedure models to compare, generalize, extract, and categorize a comprehensive GSMM. This model should allow the categorization of existing measurement methods and the derivation of adapted methods for individual measurement use cases, such as for software types, hardware and software setups, and also individual components of software systems, e. g., along the software stack.

In order to structure the terminology used in the area of measurements of the resource and energy efficiency of software, and with regard to the missing standardization in the field of Green Software Engineering, we created a glossary,[2] based, i. a., upon the ontology created in Mancebo et al. [28]. The main goals are to provide a terminological overview and a means for disambiguation, as well as point out the synonymous usage of terms in the literature.

[1] 1 GtCO$_2$e = $10^9$ tonnes of carbon dioxide equivalent

[2] https://gitlab.rlp.net/green-software-engineering/gsmm/-/blob/main/english/glossary.md

## 3. Related work

Sustainable Software Engineering is still lacking a unified terminology. In general, it aligns software engineering with the so-called dimensions of sustainability (e. g., social, economic, and environmental) according to Calero et al. [29] and Purvis et al. [30]. The environmental dimension is often referred to as *Green Software Engineering (GSE)*. However, GSE encounters hurdles in the standardization of its terminology as well. This results in terms such as *Green Coding* and *Sustainable Software Engineering (SSE)* being used interchangeably, an issue previously addressed by Calero and Piattini [31].

Naumann et al. [10] introduced the GREENSOFT Model, a reference model that structures and classifies elements of GSE and identifies measurements as a central necessity. A number of approaches and tools for measuring the energy consumption of software have been published. One of the earliest was described in 2011 by Wang et al. [32], who introduced SPAN, a system for analyzing the energy consumption of CPUs using performance monitoring counters and their correlation with measured energy consumption. Building on SPAN, they also introduced SAFARI for assessing energy consumption at the function level [33]. Also proposed in 2011, SEEP utilizes symbolic execution and platform-specific energy profiles to give estimates of the energy consumption of code during development [34]. Hindle et al. [35] describe GreenMiner, a test setup for mobile devices which ran automated tests of mobile apps on the measured phone using a standard usage scenario-based approach.

Newer, integrated frameworks include *Scaphandre*, *EnergAt*, *CodeCarbon*, and the *Experiment-Impact-Tracker*. Scaphandre is a measurement and visualization tool built on the Intel RAPL interface.[3] It supports the measurement of CPU power consumption for server or container-based solutions [36]. EnergAt is a model for obtaining application-specific energy measurements in a setting where multiple applications are collocated on the same device [37]. CodeCarbon [38] and the Experiment-Impact-Tracker [39] also use RAPL as well as nvidia-smi[4] to gather information about energy consumption, with a focus on machine-learning applications (see also Guldner et al. [40] for a comparison of these two trackers with the SERENA method, described in Sections 4 and 6). Ournani [41] and Schade [42] also provide an overview of software energy measurement tools, both software-based and hardware-based, and Jay et al. [36] compare a set of software-based measurement tools and investigate how measurements obtained through them correlate to those taken with an external power meter.

The relevance of software for ICT-related energy consumption has also been acknowledged in energy efficiency and sustainability label specifications such as the American *EnergyStar* or German *Blue Angel* labels. The EnergyStar specifies a number of requirements for energy measurements, including ambient temperature, humidity, and characteristics of the power meter used [43,44]. Testing has to occur in accordance with relevant standards (e. g., EN 62623:2013) and includes power consumption in various states, as defined by measurement standards, as well as a maximum power test using the Linpack and SPECviewperf benchmarks. The Blue Angel is a German sustainability label owned by the Federal Ministry for the Environment. It offers a number of ICT-related specifications and since 2020 a dedicated specification for software centered around a number of quality criteria and prescribing the measurement and reporting of software-induced energy consumption [45].[5]

## 4. Modeling approach and referenced measurement methods

It is apparent from the literature that measurements of software-induced energy and resource consumption are necessary to assess and reduce the environmental impact of software. Thus, we introduce the GSMM to allow for the implementation of measurements in a structured and standardized way and to define minimum requirements for measurement methods, as well as guidelines for all associated processes, experiment setups, and expected outcomes. We base its development on existing reference models, such as the GREENSOFT model, as well as on measurement methods from related work, from the authors' own research, and from practitioner groups. They form a basis of the GSMM and range from easy-to-apply methods designed to help developers or users quickly capture metrics with onboard means, to sophisticated and integrated measurement setups for scientific experiments.

It should be noted that we see model creation and method development as iterative processes. Since this process is described linearly in the article, we first briefly introduce the existing methods and then present the GSMM in Section 5. We then describe the existing models in detail and show how they can be mapped to the GSMM in Section 6. This, in turn allows for the continued development of the GSMM, creating a structural coupling between the GSMM and available as well as novel methods. Therefore, we highly encourage other researchers and practitioners within the (Green) Software Engineering community to use, discuss, and contribute to the model.[6]

- *Software Energy and Resource EfficieNcy Analysis (SERENA)* (see Section 6.1)[7] is a measurement method developed at Umwelt-Campus Birkenfeld based on the work of Kern et al. [46]. It enables the resource and energy efficiency assessment of software by measuring the consumption of the entire physical system, using a power meter and resource-monitoring tools running on the system. It also includes the OSCAR script for analyzing the measurements.[8]
- The *Green Software Measurement Process (GSMP)* [47] (see Section 6.2) details all activities and roles necessary to perform measurements and analyses of the energy consumption of software. GSMP is the methodological component of the *Framework for Energy Efficiency Testing to Improve Environmental Goal of the Software (FEETINGS)* [28], which also has a conceptual and a technological component, a hardware device built to capture the energy consumption of the computer and several hardware components, and ELLIOT, a tool to analyze the captured data.
- The *Sustainability Assessment Framework (SAF) Toolkit* [48] and the *Green Lab* (see Section 6.3) are both developed by the Software and Sustainability (S2) group at VU Amsterdam. The SAF Toolkit provides various tools to define, guide, uncover, and eventually operationalize sustainability-quality concerns in the form of a concrete measurement plan and report. The Green Lab is the experimental platform for conducting empirical studies on the quality of software, with a special emphasis on its energy efficiency. The Green Lab is supported by several open source tools, which help researchers in following established guidelines for the design and running of measurement-based experiments [49,50].
- The *Green Metrics Tool* (see Section 6.4)[9] is a versatile industry energy measurement tool from Green Coding Berlin. It isolates applications in containers for precise measurement of power, network, disk, and memory consumption, and supports various metrics. The tool is architecture-agnostic, it can be used for GUI applications, and it separates benchmark runs into distinct life

[3] https://web.archive.org/web/20230623133912/https://01.org/blogs/2014/running-average-power-limit-%E2%80%93-rapl [2023-11-06]

[4] https://developer.nvidia.com/nvidia-system-management-interface [2023-11-06]

[5] https://www.blauer-engel.de/en/productworld/resources-and-energy-efficient-software-products [2023-11-06]

[6] Available at https://gitlab.rlp.net/green-software-engineering/gsmm/-/tree/main/english

[7] https://gitlab.rlp.net/green-software-engineering/serena

[8] https://gitlab.rlp.net/green-software-engineering/oscar

[9] https://github.com/green-coding-berlin/green-metrics-tool

cycle steps. It offers a detailed dashboard and an API for data analysis, focusing on machine-dependent factors to understand energy consumption impacts.

- The *Cloud Energy Usage Estimation Model* (see Section 6.5) developed by Green Coding Berlin is a machine learning model that estimates energy usage in environments where controlled measurements are not feasible. Based on research by Rteil et al. [51], it uses the SPECPower dataset to create an XGBoost model for estimating the AC power draw of servers. The model is particularly useful in settings (e. g., cloud) where detailed CPU information is not available.
- *Software Footprint* (see Section 6.6)[10] was developed by the Oeko-Institut in conjunction with the eco-label Blue Angel for software as a means of checking the energy consumption of software without great technical effort. The low-threshold tool provides software developers with information on their local development computer about how much energy the execution of the software consumes.
- The *Emission Estimation Framework* from the Sustainable Digital Infrastructure Alliance (SDIA) (see Section 6.7) is a mathematical model to estimate software energy consumption focusing on CPU usage measurements [52]. The model is grounded in several essential assumptions including the reliability of the Thermal Design Power (TDP) as a proxy for CPU energy consumption and fixed energy allocation between the CPU and other hardware components. This makes the framework applicable to all software types and systems as a pragmatic estimation method.
- The *Container Overhead Measurement Methodology* (see Section 6.8) introduced in Kreten [53] enables pinpointing efficiency deficiencies within container configurations and environments, emphasizing a low-cost, practical approach. It is well-suited for assessing resource and power consumption of containerized software using standard server rack Power Distribution Units (PDUs). Furthermore, it includes a tool for assessing efficient horizontal container scaling.

The application areas of the GSMM are the categorization of existing models and methods that aim at assessing the energy and resource consumption of software and the creation of new models. Thus, as a first step in the creation of the GSMM, we identified the elements and scope of the introduced methods:

- What can be measured and what are the requirements?
- What is the purpose of the model (e. g., which stakeholders use the measurement results for which goal and at which point in the software life cycle)?
- What measures can be assessed, and how are the metrics calculated and evaluated?
- What are the models components, i. e., which phases does the model establish for someone who wants to measure software in accordance with it?

## 5. Green software measurement model

To derive the framework model, we categorized the measurement methods (bottom-up approach) to describe a generic measurement model that is widely applicable. Furthermore, we considered quality criteria and guidelines, as well as requirements, limits, and constraints for measurement methods, and created a list of metadata to describe the measurements from our experiences executing the measurements (top-down approach). Fig. 1 shows an overview of the main components of the GSMM, acting as the reference model framework.

In the center of the considerations for a measurement is the **measured object**, i. e., the software product(s) or part(s) thereof to be assessed, and the necessary parameters for their execution, as well as the **measurement goals** that should be achieved; see (A). Then, the desired **metrics** can be determined, which define the required **measurements** to be conducted; see (B). The measurements, in turn, impose prerequisites for the **measurement procedure** (C), as well as the **measurement setup** (D), which define the measurement method, the System under Test (SuT), the necessary tools, the measurement hardware, etc. When these prerequisites are met, the measurement(s) can be executed and the data gathered **evaluated** (E). The resulting output of a measurement according to the GSMM are the **measurement data**, e. g., power and hardware measurements from the executed scenarios, the **metadata** describing the measured object, measurement procedure, setup, produced data, etc., and the **analysis report** produced with a data evaluation model, e. g., to optimize the software product. In the following, we provide details about the individual components of the GSMM and the generated data, and illustrate exemplary categories for measurements.

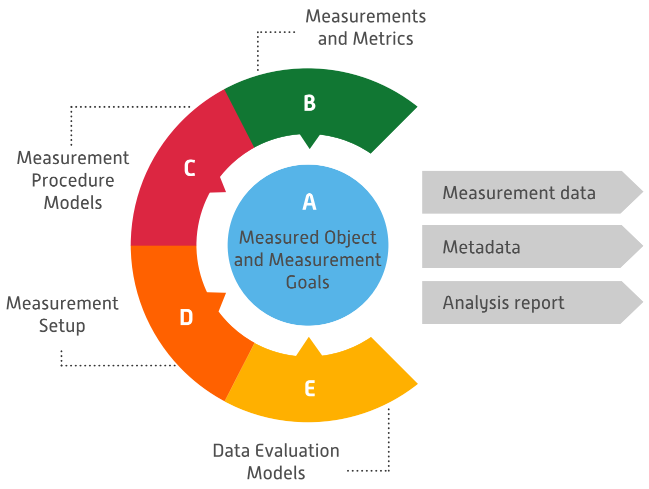


**Fig. 1.** Components of the GSMM.

### 5.1. Measured object and measurement goals (A)

Looking at the available measurement and assessment methods referenced in Section 4, they all have in common that as a first step they require a defined measurement goal as well as a software product, or parts thereof, to be measured. This is sometimes called the software entity. Common measurement goals include the comparison of the software entity with itself over the development process, e. g., within a CI/CD pipeline, between releases, or when introducing new features. Furthermore, comparisons between different implementations, libraries, configurations, etc., and between different products performing similar tasks (e. g., within software product groups like browsers, media players, databases) are possible—and it is even feasible to compare individual functionalities or software features across product groups (e. g., there are many software products which provide a feature to “edit text”). When defining the features to be measured, one approach found in the methods is to consider which functions, libraries, or API calls would typically be used, and which could induce a high resource load or energy consumption. Examples include computational or network-intensive functions, functionalities that take a relatively long time to be executed, functions that execute known complex implementations. Furthermore, the methods differentiate between measurements that are carried out over a long period of time (“long-term”), those that are repeated in certain intervals (e. g., with each major release), or those conducted only once (e. g., to determine which implementation is more efficient). The measurement goals and specified parameters should be recorded as measurement metadata.

[10] https://blog.oeko.de/energieverbrauch-von-software-eine-anleitung-zum-selbermessen/

**Table 1**
Overview of examples of relevant metrics.

| Measure/Metric | Exemplary unit | Prerequisites and required measures | Description |
|---|---|---|---|
| Run time and space complexity | None | Knowledge of the algorithm | Big-O notation is a good indicator when comparing algorithms, not easily applicable to larger software products |
| Duration | $s$ | Recording timestamps, usage scenario | Relatively easy to measure, good introductory metric, already widely used |
| Mean power draw | $W$ | Duration, power meter or internal power logger,[a] usage scenario | Useful to estimate influences of code changes, energy calculation (see below) necessary for comparison purposes |
| Energy[b] | $Wh$ or $J$ | Power, duration, baseline, usage scenario | Robust indicator when comparing software products, algorithms, etc., ideally integrated in development phase, potentially too complex to assess |
| CPU usage[b] | % | Resource logger,[c] usage scenario, baseline | Important indicator since CPU is one of the largest local energy consumers, easy to measure, expandable to recording individual CPU cores to evaluate parallelization |
| RAM usage[b] | $MB$ or % | Resource logger, usage scenario, baseline | Important considering hardware obsolescence, easy to measure, usually less significant for energy usage |
| Permanent storage[b] | $MB$ | Resource logger, usage scenario, baseline | Important considering hardware obsolescence, easy to measure, usually less significant for energy usage |
| Network traffic | $MB$ | Resource logger, usage scenario, baseline | Important indicator since transport and remote data processing might have large resource and energy impacts, easy to measure, potentially usable to estimate remote energy usage |
| GPU usage[b] | % | Resource logger, usage scenario, baseline | Important indicator since GPU is one of the largest local energy consumers, easy to measure if hardware provides data, only necessary if software uses GPU |
| GRAM usage[b] | % | Resource logger, usage scenario, baseline | Important considering hardware obsolescence, easy to measure when hardware provides data, only necessary if software uses GPU |
| Mean GPU power draw | $W$ | Supported hardware w. internal power logger, usage scenario | Some GPUs provide internal power measurements, necessary for GPU energy calculation (see below) |
| GPU energy[b] | $Wh$ or $J$ | Mean GPU power draw, usage scenario, baseline | Analogous to overall energy consumption, robust indicator when GPU is used by software, easy to measure when hardware supports power measurements |
| Useful work | varied | usage scenario | Necessary when calculating energy efficiency (see below), performance metrics (regression error, test accuracy, F1-score, IoU, etc.) can indicate useful work done of ML models |
| Energy efficiency factor | $\frac{\text{item}}{J}$ | Useful work and energy | Especially useful for comparisons, needs additional computation and possibly the recording of additional metrics |

[a] External power meters or PDUs, e. g., from Janitza https://www.janitza.com/energy-and-power-quality-measurement-products.html [2023-11-06], GUDE https://gude-systems.com/en/cat/power-distribution-units/ [2023-11-06], internal power loggers like RAPL and nvidia-smi.
[b] Software-induced metrics calculated by subtracting the according baseline measurements from the scenario measurements.
[c] Resource loggers include collectl (https://collectl.sourceforge.net/ [2023-11-06]), Windows performance monitor (https://techcommunity.microsoft.com/t5/ask-the-performance-team/windows-performance-monitor-overview/ba-p/375481 [2023-11-06]), wireshark (https://www.wireshark.org/ [2023-11-06]), and nvidia-smi.

### *5.2. Measurements and metrics (B)*

As a next step, the referenced methods usually derive the required measurements from the defined goals and measured object directly. The available metrics are dependent on several factors and prerequisites, such as the used hardware and software, but the measurement goals also have an influence on the selection of the appropriate metrics. While developers may want, e. g., detailed information on changes to the efficiency, RAM usage, and parallelization of a code block they just altered, execution duration and network traffic metrics may suffice for a user who wants to work with an app on their end device. In the following, we list the most widely measured metrics from the methods in Table 1 and discuss their usefulness and measurement complexity.

Besides the listed measures and metrics, methods like SERENA (Section 6.1) and GMT (Section 6.4) also include further criteria, such as CPU and GPU temperatures (which show a correlation to the power draw [40] and may be an easy-to-measure indicator for optimization success if the hardware provides the data), fan-speeds, clock speeds, and environment data (ambient temperature, humidity, etc.). Additionally, it may be useful to gather and analyze the data on a more detailed level, e. g., measuring individual CPU core usage to determine parallelization effectiveness or logging CPU usage and network traffic on a container-level for server software in data centers.

The available metrics also impact the prerequisites for the measurement procedure models and measurement setup. Developers can use common run time and space requirements (big O notation) as asymptotic requirements of an algorithm without any scenarios or logging. With them, developers can quickly decide to use one over the other to reduce the complexity of their software product and thus the environmental impact when designing software products. However, this also depends on the context, since “race to idle” does not always imply a lower energy consumption of the whole system as shown, e. g., by Pereira et al. [19] and Oliveira et al. [54]. All further metrics require at least a usage scenario and some kind of resource logger.

Regarding energy efficiency metrics, it is necessary to define “useful work”, as described, e. g., in Johann et al. [55]. This, of course, depends strongly on the software product and is not always feasible to define. Examples from the methods are the number of created, read, changed, deleted, or transmitted data points, the number of executed operations, or benchmarks. The benefit of these metrics is that they make different implementations directly comparable. If the items cannot be easily defined, e. g., when measuring a complete software product like a word processor, a possibility to compare the efficiency of one software over the other is to make their outcomes as equal as possible (e. g., create the same PDF document with the word processors) and then perform, for

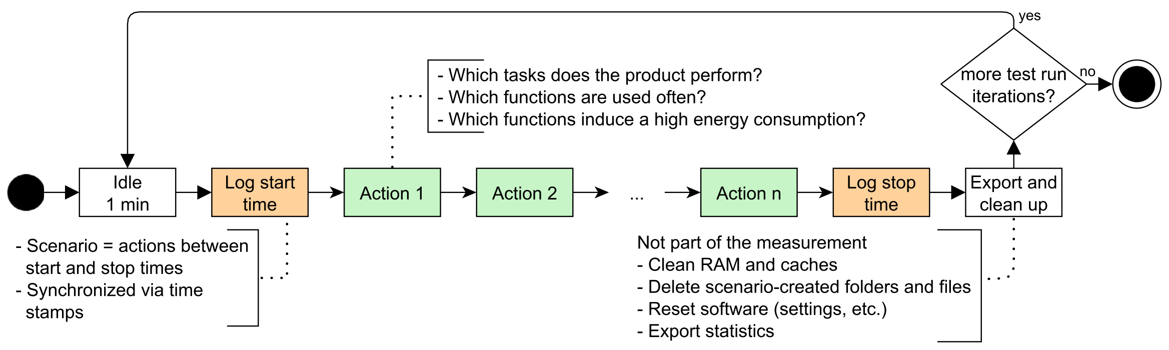

yes
- Which tasks does the product perform?
- Which functions are used often?
- Which functions induce a high energy consumption?
more test run iterations?
no
Idle 1 min
Log start time
Action 1
Action 2
...
Action n
Log stop time
Export and clean up
- Scenario = actions between start and stop times
- Synchronized via time stamps
Not part of the measurement
- Clean RAM and caches
- Delete scenario-created folders and files
- Reset software (settings, etc.)
- Export statistics

**Fig. 2.** Actions in usage scenario.

instance, a t-test as described in Kern et al. [46] to test if the means of the samples are different and thus determine the more efficient software.

### 5.3. Measurement procedure models (C)

The determined goals, measurements, and metrics establish the prerequisites that the measurement procedure and the resulting setup need to satisfy. Typical steps from the methods that need to be addressed in component C are the definition of the measurement method, including repetitions, which tools (e. g., workload generator) to use or build, and which scenarios to devise (baseline without starting the software, idle mode of the inspected software, usage, load, long-term, etc.). When creating the scenarios from the specified measured object, there are also several aspects to take into account, including the kind (idle, load, etc.) and duration of the scenario, pre- and post-scenario operations like the handling of log files, and which actions to perform in which order. The scenario can then be constructed as in Fig. 2 or along already existing tests, such as unit, integration, UI, end-to-end tests, etc.

Many use cases from the referenced methods stipulate a scenario that is somehow automated and executed repeatedly to ensure reliable results. In statistics, a minimum sample size of 30 is recommended to ensure that the sampling distribution is close to the distribution of the population (as described in Kern et al. [46]). Of course, in some cases, such as exceptionally long scenarios of a system in production, or if a developer just wants to check a new implementation, it may be advisable to reduce the number of repetitions to bring the measurement time to a reasonable level. In these cases, however, one should check the graphs for outliers and calculate, e. g., the per-second standard deviation in the measurement. The workload can be generated via software tools, like automation software or scripts, on the SuT, or directly from using the software, e. g., in continuous long-term measurements. If automation tools are used, baseline and idle measurements should be acquired with these tools active to later subtract their consumption from the results.

Similar to software testing, usage scenarios can be categorized according to the scenario technique into (i) black-box measurements, where within the scenario some functionalities of the software product are executed without regarding the implementation, function calls, used libraries, etc.; and (ii) white-box measurements, where the impact of individual parts of the implementation can be assessed and possibly optimized. While white-box measurements usually require more effort to be put into the measurement procedure (e. g., setting up automated logging of individual actions, like library calls in Guldner et al. [40], or even code blocks as described in Verdecchia et al. [56]), they can also provide developers with in-depth insight into the consumption hot spots of their software. Black-box measurements on the other hand only require logging start and end-timestamps of the measurement and they are useful, for instance, when assessing a complete software product or tracking them over longer time periods such as over update cycles.

Another way of classifying scenarios is by their kind. Here, we differentiate between idle, standard usage, and load scenarios, as well as baselines. These scenario kinds are detailed in Kern et al. [46]. Idle scenarios can point to consumption hotspots when the software is simply being executed without (user) interactions, e. g., through repetitive background tasks, such as update services, indexing, etc. Standard usage scenarios are usually the most relevant. Here, the software is executed as if it was used as intended. If the scenario includes timestamps for individual actions (like GUI interactions, function calls, etc.), it can help to identify consumption hotspots in functionalities, startup and shutdown processes, etc. Load scenarios for distributed systems are useful for stress-testing the architectures with many (simulated) users, but can also consist of benchmarks, e. g., for databases, API calls, etc. Further usage scenarios can be defined along the life cycle of the software product and tailored to each software type. Examples from the referenced methods are installation and de-installation, the boot process, e. g., for operating systems, the training or inference phase of ML models, etc. The baseline is a special kind of scenario, and it is required by some methods. Here, only the necessary components to run the software product are active (device hardware and software stack), but the software product itself is not executed. Baselines can help when calculating software-induced consumption by subtracting the baseline measurements from, e. g., the usage scenario, resulting in only the overhead consumption induced by executing the scenario.

### 5.4. Measurement setup (D)

Using the specifications from the measurement method, metrics, and measured object, the measurement setup identifies the necessary hardware and software stack for the measurement in order to tailor them to the use case. This includes the hardware and software for the measurements themselves (SuT), the metering devices (if any, in case of software-based logging), required tools for logging, etc. The parameters that should be noted in the metadata here include the necessary hardware setup as well as methods used for data acquisition, tools, meters, etc. Furthermore, especially with power meters and measurement devices, their accuracy, sampling rate, settings, etc. should be noted to track possible errors in the evaluation.

Similar to the automation tools, there is a plethora of power and hardware usage loggers available, depending on the measurement method, operating system, and hardware setup. The data acquisition can be done using software-based logging (e. g., using a resource logger as in the Software Footprint Tool, the COMM method in Section 6, a tracker like CodeCarbon [38], or manually logging them in a bash script) and hardware-based logging (usually via an external power meter, e. g., in the SERENA, GSMP, Green Lab, and GMT methods in Section 6). In some cases, such as experiments involving several different measured objects on one SuT, it is also advisable to consider the creation of system images before and after installing the software to ensure a “clean” system environment.[11]

[11] E. g., with a tool like clonezilla https://clonezilla.org/ [2023-12-01]

When using hardware for an SuT that is not a desktop PC, there are additional challenges, e. g., how to generate repeatable workloads on a mobile device like a smartphone or an embedded sensor node, or how to assess the resource consumption of a cloud-based compute-node where the developer usually does not have direct access to the hardware.

### 5.5. Data evaluation models (E)

Once the data is recorded, the data evaluation model defines the methods used to analyze and evaluate the measurement data and prepare the measurement report. In the referenced methods, this usually includes calculating and visualizing the metrics for all conducted measurements. Depending on the measurement goals and scenarios, it may be useful to calculate summary statistics such as mean values for the hardware usage and standard deviations for the scenario repetitions, or inferential statistics such as t-tests to compare the mean values of measurements, along with visualizations for the data such as tables, boxplots, or bar plots. Energy efficiency metrics and the energy consumption of the scenarios or actions should be calculated as the integral of the power draw over time, e. g., as detailed in Guldner et al. [57]. With baseline measurements, the software-induced consumption and hardware usage can be calculated.

The results should be logged in a meaningful way. For example, the energy consumption can be given in Joules for short measurements or when calculating the energy efficiency factor (e. g., transferred items per Joule), or in kWh for long and resource-intensive measurements like training ML-models. While simple evaluations can be performed using a spreadsheet software, tools exist to automate the evaluation (e. g., the GMT or OSCAR; see Sections 4 and 6).

### 5.6. Generated data

A measurement, according to the GSMM, outputs the raw *measurement data* in a format that is usable for the evaluation model and ideally also available and re-usable for future assessment. The result of the evaluation is the *analysis report* as the main outcome from the experiment. To increase transparency, reproducibility, and create the possibility to find measurement errors, *metadata* should be recorded for all experiments, which comprehensively describes the measurement for future reference. Especially when the experiments contain processing of large amounts of data (e. g., in machine-learning scenarios) or when many metrics are recorded over a long period of time, data storage and management also become relevant. Here, a data repository, version control system, or the automated management of the data with a tool such as DVC[12] or in a database could be considered. The metadata should include the following information[13]:

- the SuT, e. g., measurement device used, hardware and software setups, etc.;
- the measured object, e. g., version, software type, product group, system boundaries (especially in case of a distributed software);
- the scenario, e. g., its duration, repetitions, the pertaining life cycle phase, etc.;
- the evaluation, e. g., analysis method used, scripts or tools, etc.;
- quality issues, e. g., sampling frequency, threats to validity, measurement accuracy, etc.; as well as
- general data, e. g., date, time, stakeholders, etc.

One of the metadata items that is recommended to be recorded for each measurement is the threats to validity of the experiment, e. g., following the categorization in Wohlin et al. [49, p. 68] and applied in Ardito et al. [58]. Recording the threats to external, internal, construct, and conclusion validity enables reflection from an outside point of view, and allows for improvements of the method and future experiments (see also Cruz [59], who provides a guide to set up energy efficiency experiments and reduce errors).

## 6. Description and mapping of referenced measurement methods

Typical applications of the GSMM are the development of new measurement methods or the categorization of existing ones. Examples of this categorization are whether or not the methods are useful for assessing the energy and resource efficiency of software, which GSMM components they cover, and how they relate their methodology to each component (which is described in more detail in the model's repository). The GSMM is based on a number of established measurement methods, which we briefly introduced in Section 4. In this section, we put the GSMM into practice by revisiting these methods, describing them in more detail, and mapping the GSMM components to them. They serve to cross-check the model and, depending on the application context, they can also be used directly to measure the resource requirements of software. For an overview of how the methods can be categorized into the GSMM, see Table 2.

### 6.1. Software energy and resource EfficieNcy analysis (SERENA)

For SERENA, auditable measured objects (A) include full software products for local, distributed, and server systems, as well as mobile apps, operating systems, container systems, and software parts such as individual algorithms, libraries, or API calls. Explored measurement goals include comparisons between software entities, e. g., for optimization purposes, as well as individual measurements, e. g., to just make the consumption transparent or to acquire the Blue Angel eco-label.[14] With SERENA, all measures and metrics (B) from Table 1 can be recorded and analyzed. Additionally, several ideas for the energy efficiency factor of artificial intelligence models are included, like "correctly classified test data points per Joule" and some further metrics such as GPU temperature [40]. Furthermore, several criteria from the criteria catalog for sustainable software products depend on measurements according to SERENA.[15]

SERENA's procedure model (C) stipulates separate measurements for the baseline, idle, and usage scenarios. Standard usage scenarios (SUS) are often defined as a series of actions that are intended to represent a typical use of the software entity. The SUS is automated and repeatedly executed. From the resulting measurement values, the baseline measurement values (adjusted to the measurement duration) is subtracted to get the energy and resource consumption induced by the execution of the scenario. Optionally, further scenarios, e. g., along the software life cycle, can be measured, such as load scenarios, benchmarks, boot process, or steps for the creation of AI-based systems like data preparation, training, etc. For software with remote data storage and execution, two hardware systems were used and measured separately, one for the server and one for the clients (which in some cases were simulated, e. g., by scripting multiple, repeated calls to an API, website, etc.). Mostly black-box measurements were conducted for complete software products; however, white-box measurements, e. g., on code-block level are also possible [56] with SERENA. Furthermore, long-term black-box measurements, e. g., for in-production systems, are possible.

The measurement setup (D), as well as the analysis method (E) for full software products on local and client–server setups, is described in Kern et al. [46] and is expanded to other setups, such as containerized setups [53], or IoT devices and single board computers for AI/ML applications [60]. Fig. 3 shows an overview of the method used

[12] https://dvc.org/ [2023-11-06]

[13] An extensive list is available at https://gitlab.rlp.net/green-software-engineering/gsmm/-/blob/main/english/metadata_collection.md

[14] For details, see e. g., https://eco.kde.org/handbook/ [2023-11-06]

[15] https://gitlab.rlp.net/green-software-engineering/criteria-catalog

**Table 2**
Categorization of exemplary measurement methods.

| Measurement procedure model | Measured object and measurement goals | Main measurements and metrics | Measurement setup | Data evaluation model |
|---|---|---|---|---|
| Software Energy and Resource EfficieNcy Analysis (SERENA) | all software types/parts | energy consumption, network traffic, hardware usage | energy measurement by power meter, hence everything that can be plugged into an outlet | analysis report |
| Green Software Measurement Process (GSMP) | all software types/parts except mobile apps | energy consumption, hardware usage | energy measurement by power meter, hence everything that can be plugged into an outlet, additionally measurement of individual hardware components | analysis report |
| Sustainability Assessment Framework (SAF) Toolkit and Green Lab | all software types/parts | energy consumption, network traffic, hardware usage | hardware within the testfarm and Green Lab only | analysis report |
| Green Metrics Tool | containerized software only | energy consumption, network traffic, hardware usage | energy measurement by power meter, hence everything that can be plugged into an outlet | dashboard-based analysis |
| SPECPower XGBoost Model | only complete virtual machine or bare metal | estimation of energy consumption | CPU usage based estimation | ML-based estimation |
| SDIA Framework and Linear Approximation | all software types/parts | CPU-based energy consumption | everything with a known CPU | mathematical estimation |
| Oeko-Institut's Software Footprint tool | all software types/parts, simple measurements | CPU-based energy consumption | CPU usage-based estimation | energy and CO2 estimation |
| Container Overhead Measurement (COMM) | containerized software only | network traffic, hardware usage | docker-based software | none (raw data only) |

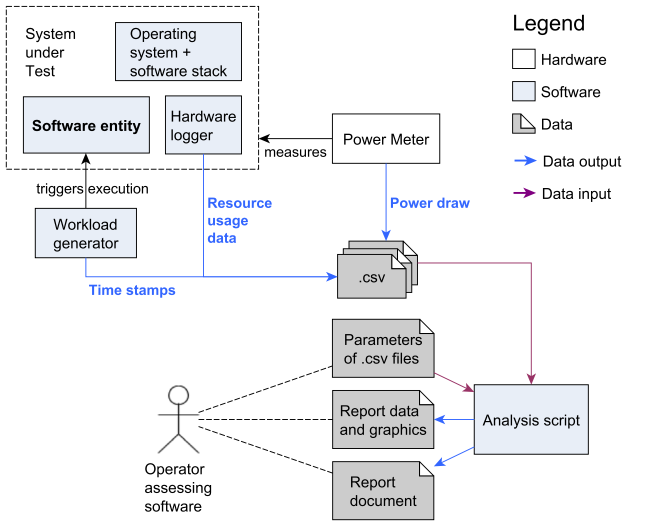


**Fig. 3.** Overview of SERENA for local software with hardware-based logging of power draw.

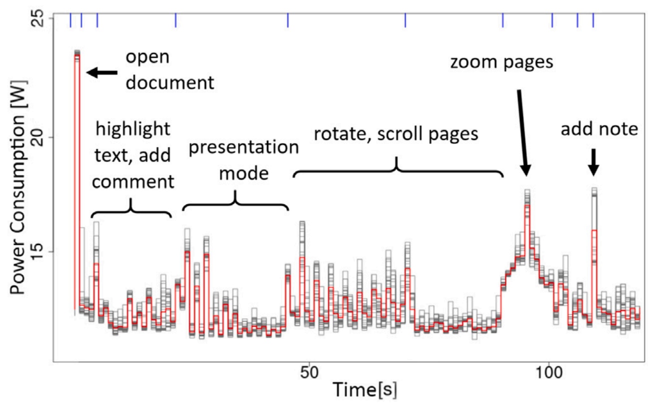


**Fig. 4.** Evaluation of Okular's standard usage scenario with OSCAR (annotated with scenario actions).

for local software with hardware power meter. Power meters from Janitza are used in the lab at Umwelt-Campus Birkenfeld which deliver power draw as a per-second average. Thus, a prerequisite for using SERENA is that the software can be executed on a device that can be plugged into the measured outlets if the hardware power meter is to be used. For logging, "performance monitor" on Windows is used, and collectl on GNU/Linux systems, as well as nvidia-smi for GPU metrics. To automate the scenarios, either a workload generator is used such as Actiona and command-line tools such as Bash scripts, or, e. g., for measurements of software parts the logging in the source code may be directly integrated.

Fig. 4 exemplarily shows the evaluation for the measurements of KDE's document reader Okular,[16] generated using the data evaluation model (E) OSCAR. For this purpose, the timestamps of the start and end of the measurement runs are used to superimpose the results (gray lines represent measured values of individual runs, red line visualizes per-second average over all measurement runs). Furthermore, OSCAR generates descriptive statistics for the metrics under consideration (power consumption, RAM, CPU, network and memory utilization) and can also be used to generate t-tests to compare the consumption of scenarios with each other.

As data output according to the GSMM (see Section 5.6), the raw measurement data and OSCAR analysis reports are provided in replication packages for all publications, e. g., in a Git repository. This also includes relevant metadata for all phases. As an outlook for the

16 https://okular.kde.org/

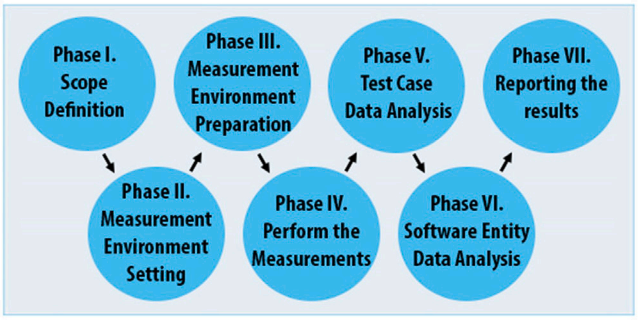

Phase I. Scope Definition
Phase II. Measurement Environment Setting
Phase III. Measurement Environment Preparation
Phase IV. Perform the Measurements
Phase V. Test Case Data Analysis
Phase VI. Software Entity Data Analysis
Phase VII. Reporting the results

**Fig. 5.** Process for evaluating the energy efficiency of software.

SERENA method, it is planned to better integrate the measurement of distributed and cyber–physical systems, and, furthermore, to integrate the OSCAR analysis script and user interface into the development process of software products.

### 6.2. Green Software Measurement Process (GSMP)

The *GSMP*, depicted in Fig. 5, includes guidelines that describe the process of performing measurements. It also includes the roles participating in the process (client, measurement analyst, measurement performer, and data analyst).

During the initial phase (Scope Definition) it is necessary to specify the requirements for the evaluation of energy efficiency and to identify and describe the software to be analyzed. This is equivalent to the measured object and measurement goals (A) of GSMM. Phase II (Measuring Environment Setting) requires selecting the measuring instrument, the Device Under Test (DUT), i. e., the hardware where the tests cases will be run, selecting the set of measures and metrics (B) to be used and to check that no other software is running in the background and all services and processes that may affect the baseline measurement of consumption are stopped. This corresponds to the measurement procedure (C) and measurement setup (D) of the GSMM. Phase III is focused on the measurement environment preparation (D), checking that no other software is running in the background, setting the number of times each measurement is repeated, and installing the required software and services. The procedure also establishes that once the measurements for one of the software entities are completed the DUT must be reset to its initial state. This procedure is repeated for the different software entities. In phase IV, the measurements are performed and the raw energy consumption data is collected with the measuring instrument (measurement data). Additionally, metadata about the DUT is gathered.

Phases V and IV are the GSMP's data evaluation model (E). The test case data analysis is the focus of phase V and consists of turning the raw data into useful information for the analysis. Moreover, the descriptive statistics for each test case are obtained. In phase VI the data is then analyzed, before a laboratory package is produced in phase VII that reports the results and promotes replicability of the empirical study conducted. This corresponds to the analysis report of GSMM. GSMP can be used whether the energy consumption is obtained by means of a software estimator or by a hardware device. Depending on the selected measurement instrument, the measures could be different (CPU, memory, GPU, etc.). GSMP also supports the possibility of recovering other measures such as the performance of certain hardware components or different kinds of measurements that are necessary for further analysis, e. g., information about the executed source code (Total Lines of Code or Complexity).

*GSMP* is the methodological component of *FEETINGS*. The framework also contains a terminological component (an ontology to define all the terminology used in *FEETINGS*) and a technological component, composed by a hardware measurement device (*EET*-Energy Efficiency Tester) and a tool (*ELLIOT*) to analyze the data recovered by EET. In addition to the total energy consumption of the DUT (Device Under Test), *EET* supports the measurement of different hardware components: CPU, hard disk, GPU, and monitor. The sampling frequency of *EET* is around 100 Hz, which provides very reliable consumption information. The main objective of the ELLIOT software tool is to provide a visual environment that allows researchers to process the data collected by *EET*, analyze them, and generate an appropriate visualization of the results obtained.

When using FEETINGS, and then EET, it is only possible to measure local software running on the DUT. So, the measurements are limited to local software, web applications, client, or server applications. Usually the recovered raw data and the processed results from the studies are made public on repositories. FEETINGS and GSMP can therefore be seen as instantiations of the GSMM, applied to software able to be installed locally on a DUT.

### 6.3. SAF toolkit and Green lab

While the SAF Toolkit was developed to guide decision-making on sustainability from the perspective of software architecture, the experimental platform Green Lab was founded to address the challenges in the field of software energy efficiency research by providing a transparent platform for conducting concrete experiments. Notably, both methods can be utilized together. Green Lab experiments can be built upon the SAF Toolkit and derived measurements can be fed back. Thus, the SAF Toolkit, together with the Green Lab, can be seen as an instantiation of the GSMM, providing tools to identify the components in their use case and eventually executing the planned experiment in a controlled environment.

#### 6.3.1. SAF toolkit

The SAF reflects on the four dimensions of sustainability according to Lago et al. [61], i. e., Technical, Economic, Social, and Environmental. While the environmental dimension may encompass the broader impacts of software activities on our natural ecosystem, for the GSMM this dimension aims to address the ecological concerns, i. e., the resource and energy efficiency of software-intensive systems. The SAF is composed of several components to provide a holistic assessment of all different sustainability dimensions. In the context of the GSMM framework, the application areas of the toolkit are in the design as well as monitoring phase of a concrete experiment. Below, we focus on the two most relevant components.

**Decision Maps (DM)** [62] are a visual notation to frame and illustrate sustainability-relevant design and quality concerns, as well as their expected impact. In the context of GSMM, Decision Maps help to define the measured object and determine concrete measurement concerns (A). Concerns can range from very specific, such as *execution time* or *network traffic*, to a more high level, such as $CO_2$ footprint or *maintainability*. Impacts can be further classified into three types: (i) *Immediate* impacts, referring to instantly observable changes; (ii) *Enabling* impacts, arising from use over time; and (iii) *Systemic* impacts, referring to persistent changes and requirements, such as economic structural changes. As an example, while the *execution time* can be observed immediately and has a direct impact on the system, the $CO_2$ *footprint* must to be observed over time.

After the concerns are established, the **Sustainability-Quality (SQ) Model** [63] defines the actual measurement plan. The SQ model consolidates the definitions about the identified concerns and results in a collection of Quality Attributes (QAs) categorized into the four sustainability dimensions. For instance, *execution time* relates to the technical dimension, while *energy efficiency* relates to the environmental dimension. To be measurable, the SQ model operationalizes the identified QAs by assigning a set of metrics and measurements (B) to each QA. After an experiment has been executed, the SQ model serves as data

| QUALITY ATTRIBUTE | DEFINITION | TEC | ENV | ECO | SOC | METRIC/UNIT |
|---|---|---|---|---|---|---|
| Execution Time (ET) | the time it takes to execute the predefined set of tests | X | | | | $ET = ET_{system} - ET_{baseline}$ (Duration in $s$) |
| Energy Efficiency (EE) | the total energy consumed to execute a predefined set of tests | | X | | | $EE = EE_{system} - EE_{baseline}$ (Energy in *Joule*) |

**Fig. 6.** Sustainability-Quality (SQ) Model example. **TEC** = Technical; **ENV** = Environmental; **ECO** = Economic; **SOC** = Social.

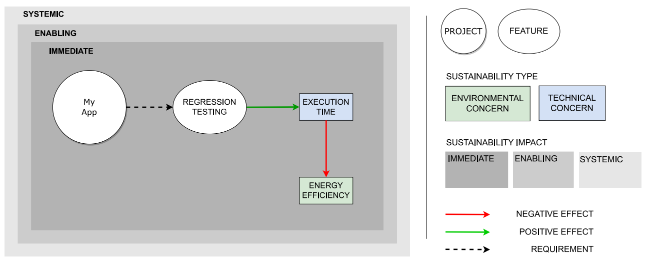


**Fig. 7.** Decision Map (DM) example.

evaluation model (E) and manages the generated data (see Section 5.6). Next to the metric and unit, the SQ model can include the analysis report, i. e., in form of a dedicated measurement column keeping track of all measurement data that has been recorded. This allows for future evaluations of the complete experiment.

Fig. 7 depicts a practical example of a DM identifying an arbitrary software system and its accompanied feature (example based on [64]). As automated regression testing might have a positive impact on the overall test-execution time, a negative environmental impact on the energy efficiency due to higher computational power might need to be considered. Such identified concerns (i. e., QAs) are further defined in the SQ model and operationalized by identifying concrete metrics, as shown in Fig. 6.

### *6.3.2. Green lab*

The Green Lab utilizes a mixed approach to conduct experimentation for energy-efficient software by identifying energy hotspots through a two-phase process, which consists of candidate hotspot identification and hotspot verification [65]. By providing a controlled environment and a dedicated Energy Lab, the Green Lab aims to contribute to the understanding and advancement of energy-efficient software development and to help researchers in the field overcome the challenges they face.

Auditable measured objects (A) are all software products that can be run on the Green Lab's physical servers (with varying architectures and OSs), a test farm composed of more than 20 Android smartphones/tablets (covering several generations and multiple technical specifications), and several robots. Servers, mobile devices, and robots are enriched with state-of-the-art power monitors/profilers (e. g., Monsoon power monitor, Watts Up Pro Meter, Intel RAPL, Android BatteryManager, INA219, etc.).

In terms of measures and metrics (B) and procedure model (C), The Green lab also provides a set of open source tools for conducting energy efficiency research on a variety of platforms according to well-known guidelines for empirical software engineering research [49,50]. The main tools used in experiments within the Green Lab include:

- Android Runner (AR) [66]: a tool for automatically executing measurement-based experiments on native and web apps running on Android devices. It aims to streamline the execution of these experiments, making them more automatic, customizable, and replicable. AR produces an output in the form of measurements (generated by profilers) and logs (produced by the Android OS during the experiment).
- Robot Runner (RR) [67]: a tool for executing measurement–based experiments on robotics software. RR is a plugin-based, Robot Operating System (ROS)-dependent tool that automatically sets up, starts, and resumes user-defined experiments. The primary output of RR is a populated run table which includes factors and required measures, such as CPU usage, memory usage, and energy consumption. RR allows plugin developers to focus on producing measures and providing a default aggregation policy, while RR takes care of the experiment orchestration. RR is the first tool of its kind in the robotics domain, with similar tools existing for energy efficient mobile development, like Android Runner, GreenMiner, and PETrA.
- Experiment Runner (ER) [68]: a platform-independent tool to automatically execute measurement-based experiments. The experiments are user-defined and compatible with plugins, both included or user-made. ER can be and is used in combination with the aforementioned Green Lab machines to conduct experiments related to power consumption.

In the Green Lab, the typical measurement setup (D) involves at least two physical machines. One machine serves to orchestrate the experiment (it is usually deployed either on a virtual machine running on a server or on a Raspberry Pi, depending on the nature of the experiment), while the other machine hosts the subjects of the experiment (e. g., another server for experiments on server-side software, a smartphone for experiments on mobile apps, a robot for experiments on robotics software, etc.). The power monitor/profiler (e. g., a WattsUp Pro meter) is directly connected to the subject machine and measures/profiles its power consumption with a given sample rate (e. g., from 1 Hz with the WattsUp Pro meter to 5 kHz with the Monsoon power monitor). The orchestrator machine is responsible for collecting low-level measures from the power monitor/profiler, to transform them into higher-level metrics (e. g., energy), and to persist them according to an open file format, such as comma-separated values or JSON, depending on the needs of the experiment. The interested reader can refer to [69, Sec. 3.4] for a concrete example of the infrastructure used in the Green Lab.

For the data evaluation model (E), in order to promote transparency, reproducibility, and applicability of scientific experiments on software energy efficiency, the researchers managing the lab make all the collected measurement data and the metadata about the measurement setups used in the Green Lab publicly available in a GitHub repository.[17] Analysis reports are generally shared with the research community via scientific publications, e. g., [70–73].

## *6.4. Green metrics tool*

The Green Metrics Tool (GMT) is a versatile energy measurement tool created by Green Coding Berlin.[18] By isolating applications through containers in the measurement setup (D), it enables precise measurements of a multitude of factors like power, network, disk, and memory consumption. By leveraging existing infrastructure files, currently limited to Docker Compose files, it seamlessly integrates as a drop-in

[17] https://github.com/s2-group

[18] https://www.green-coding.berlin/

measurement solution. The containers and networks are orchestrated according to the supplied infrastructure file and, as proposed in the GSMM under measurement procedure models (C), a usage scenario is executed. As each component of the software is encapsulated in a container, the measurements can be component specific and all aspects of the application can be precisely measured.

In order to collect metrics, the GMT utilizes a flexible plugin infrastructure that can attach different so-called "metric-providers" to capture the performance and energy metrics from the containers and host system. This gives great flexibility when deriving the measurements and metrics (B). At the time of writing the tool supports CPU Utilization (system-wide and at container-level), CPU energy (system-wide, using RAPL), CPU clock speeds, DRAM usage at container-level, DRAM energy (system-wide, using RAPL), network transferred data at container-level, temperature, fan-Speed, AC energy (system-wide using IPMI, PowerSpy2, SPECPower XGBoost model estimation, or SDIA model estimation), and DC energy (system-wide, using PicoLog).

In general, the GMT can run on GNU/Linux, macOS, and Windows, while GNU/Linux is the preferred architecture as the docker containers can be monitored in more detail. There is no limitation on what type of application can be measured as long as it can be containerized on one of the architectures listed above (A). This includes also GUI applications like for instance browsers. As the GMT does a software life cycle assessment, it automatically separates each benchmark run into distinct steps: Baseline, Install, Boot, Idle, Run time, Remove. Through this setup, each step of the software life can be measured precisely and independently (C).

The GMT can be run on various hardware configurations in reference to the measurement setup (D). When developing the usage scenario files or when precise measurements are not the key metric, a local environment is sufficient. For local developer setups without power-meters or RAPL access, Green Coding Berlin supplies a free-to-use measurement cluster. This has a specialized, no interruption GNU/Linux distribution installed and various hardware components for exact measurements.

As described in the GSMM, once the measurement is complete, there are various ways to analyze the data (E). The GMT comes with a dashboard that enables detailed analytics and comparisons between different runs. It is also possible to query an API or connect external dashboards like Grafana.[19] To enable a meaningful interpretation of the data, the GMT collects as many machine-dependent factors as feasible.

### 6.5. Specpower xgboost model

As it is not always possible to run jobs in a controlled environment, Green Coding Berlin has developed a machine learning model that can estimate energy usage based on readily available input parameters. The model is based on a paper by Interact DC and the University of South London [51] and utilizes the SPECPower dataset[20] to develop an XGBoost model for estimating the AC power draw of a server. This model accepts a variety of input parameters (e. g., from Table 1), with only CPU utilization being mandatory. Optional parameters include CPU chips, CPU threads, CPU cores, CPU frequency, CPU architecture, CPU make, release year, RAM, TDP, and vHost ratio.

The optional nature of these parameters allows the model to function in constrained environments, such as cloud-based measurement setups (D), where information regarding the CPU name or release year might not be readily accessible. Although supplying more parameters enhances the model's precision, it is designed to maintain broad applicability. The model's output can be either the power draw or the accumulated energy since the last output.

The model is operating system and architecture agnostic and does not impose a specific functional unit. To accommodate the reality of shared (virtualized) environments, the model employs a vHost-ratio input variable. This correction factor can be utilized when the user is a guest on the system and only a portion of the energy should be attributed to them.

An internal validation script[21] demonstrates that the open source model's in-sample accuracy is comparable to the Interact DC model, with an error of approximately 10 W (∼10 %).

### 6.6. Oeko-Institut's software footprint tool

The Software Footprint Tool runs in parallel with the measured software and captures the CPU load. The measured object (A) can be any software run locally, identified by its process-name and ID. Further child-processes created by the software under test are also accounted for. Required metrics (B) are the maximum power draw of the device and the operating system's process statistics. The tool consists of a Python script[22] for logging the CPU processing times and for calculating the energy consumption and $CO_2$ emissions.

First the maximum power draw is measured with an external power meter (D) using a workload script that exposes the CPU to the maximum load. The max power draw is then entered in the `softwarefootprint` script and the script is started, passing it the name of the process to be monitored as a command line argument (C). It then logs the actively consumed CPU times and share of used processor cores over time. This makes the measurement independent of which background processes are running on the computer.

The underlying computational method (E) for calculating energy consumption assumes that (i) the maximum power consumption of the computer $P_{max}$ is reached at maximum processor utilization and that (ii) the software's electrical power draw correlates with its processor usage. If, for example, a software uses only half of the processor power (50 %) or half of all available CPU cores over a time period $t$, the energy consumption of the software is calculated to be half the maximum power draw multiplied by the execution time ($W = 50\ \% \times P_{max} \times t$) (E). The values calculated in this way only match the measured power draw of the SuT in case of the full utilization of all CPU cores. If the SuT is only used partially, its energy consumption is a combination of idle power and workload power. The calculation logic of the `softwarefootprint` method assigns the analyzed software a share of the idle power draw that corresponds to the share of CPU utilization.

With the tool, software developers are able to test their software initially and to observe further in the development process whether the energy consumption or the CPU time used changes positively or negatively. Also various use cases of existing software can be tested. As an example, the loading of different web pages with the Firefox browser is shown in Fig. 8. Measurements were taken over a defined period of 60 s. The tool logged the cumulative CPU times and calculated how much energy was consumed by calling the respective web page during this time. A practical use case now consists of either comparing different websites with each other or optimizing one's own website to achieve the lowest possible CPU runtime.

Due to its simplicity, the method also has some weaknesses. First, the measurement results are purely estimates and do not reflect the actual energy consumption of the computer. Second, the figures obtained with the method are only valid on the particular SuT hardware. Third, the method considers only the power draw from CPU usage, and omits other important load drivers like network, etc. The tool is therefore suitable for optimizing the performance of a software product, but less suitable for comparing software across platforms.

[19] https://grafana.com/ [2023-12-01]
[20] https://www.spec.org/power_ssj2008/
[21] https://github.com/green-coding-berlin/spec-power-model/blob/main/interact_validation.py
[22] https://github.com/OekoJ/softwarefootprint

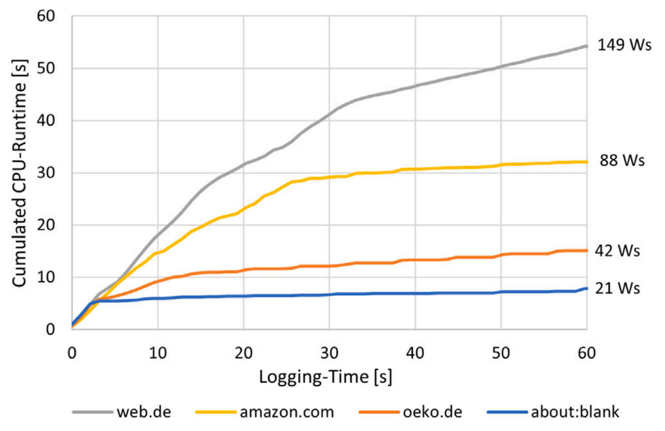


**Fig. 8.** Cumulated CPU runtime and energy consumption of different website calls (start page).

### 6.7. *SDIA mathematical framework and linear approximation*

The SDIA developed a set of formulas describing a general mathematical framework to derive a linear approximation through only CPU usage measurements (B), one of the most common proxies used in various tools and implementations. The measurements of the CPU usage and linear approximation thus serve as a rule-of-thumb and give an estimation methodology in even the most data-scarce scenarios, since most practical scenarios allow for measuring CPU usage. The linear approximation can be used for any type of software and SuT, as long as the Thermal Design Power (TDP)[23] of the CPU is known (D). Given *ceteris paribus*, one would measure CPU before, throughout, and after running a particular program and deduce the energy consumed related to the program afterward.

Approximations, estimations, and statistics are merely as powerful as the validity and robustness of the underpinning assumptions. For the SDIA framework, these assumptions consist of:

- TDP is an accurate proxy for measuring the maximum amount of energy a CPU can consume.
- There is a fixed energy allocation between the CPU and the other hardware components, relative to the total energy consumption. In other words, this model assumes the consumption of memory, networking, storage, etc. to be a fixed percentage of the CPU energy consumption.
- When the server does not run at full capacity, it assumes that the energy consumption decreases linearly with the energy consumption of the CPU.
- This model ignores the idle consumption, such that linearity holds up until completely powering off.

Concerning the GSMM, the approximation can be categorized as direct measurements of the CPU usage metric while a program runs. All other things being equal, CPU usage can then be converted into the consumed energy through the linear approximation, thus targeting (E) within Fig. 1. CPU usage can be measured by a variety of tools and programs. In the SDIA test setup, RAPL was used.[24]

### 6.8. *Container overhead measurement methodology (COMM)*

The Container Overhead Measurement Methodology (COMM), as proposed by Kreten et al. [74], is a framework designed for the measurement goal (A) of evaluating containerized software deployed using Docker technology. COMM aims to pinpoint inefficiencies within container configurations and their operational environments, offering a systematic approach to performance assessment for software within containers. In terms of Measurements and Metrics (B), as delineated in Table 1, COMM currently omits consideration of GPU-centric metrics and persistent storage performance. It primarily focuses on the consumption of CPU resources and memory utilization. Within its procedure model (C), COMM, which is based on SERENA (see Section 6.1), uses measurements for the baseline, idle, and usage scenarios. In order to find efficiency gaps in container configurations, measurements are repeated 10 times for 60 s for various settings that are suitable for the SUS. Different settings of containers, such as the layer system, storage, network, or logging drivers, can be examined.

The configuration of the measurement environment (D) includes one or several SuT, which are connected to SMTP-compatible power meters, such as commonly available server rack Power Distribution Units (PDUs). In this setup, the functions of workload generation as well as data aggregation and evaluation are encapsulated within a singular Python script. While a default workload generator is provided, it can be substituted with alternative tools for specific usage scenarios, such as automation tools for repetitive tasks or network stress tools for web service simulations. This flexibility allows for customization according to the unique demands of the application being tested. Similar to SERENA, COMM produces raw data outputs that can readily be integrated with the OSCAR data evaluation model (E). Moreover, COMM includes a Python-based analysis tool, facilitating the examination of scaling within container environments. This tool aids in determining the optimal scaling points regarding CPU and RAM usage for deployment in container orchestration platforms, thereby enhancing the efficiency of container cluster management.

## 7. Discussion and outlook

The GSMM aims to enhance software sustainability, particularly in the realm of measuring software-induced consumption. However, it is important to note that the methods encompassed within the model primarily address the core components of complex software systems. As of now, these methods are not yet fully applicable to complex architectures and distributed systems. Despite this limitation, the GSMM is suitable for projections and simulations, as demonstrated by several of the measurement methods. The GSMM serves as a reference model, synthesized from a combination of various measurement methods (bottom-up approach) and a structured analysis of software measurement requirements (top-down approach). Table 2 shows an overview of the organization of the analyzed measurement methods under the GSMM umbrella, highlighting its current focus and limitations.

The measurement categorization and metadata collection in the GSMM git repository provide a more detailed breakdown, indicating potential areas for future expansion of the model's applicability. Currently, the measurement methods can be broadly divided into two categories: those that can measure nearly any software type, and those specifically designed for measuring containerized software. This distinction reflects the current focus on more straightforward software structures as opposed to complex or distributed systems. In general, this observation aligns with the division between comprehensive methods and specialized methods, e. g., for containers, mobile apps, embedded systems, etc.

While almost all models concentrate on measuring or estimating energy consumption, there are notable exceptions, such as COMM. Furthermore, there is significant variation in the measurement setup and data evaluation models across different methods, which could be further developed to address the challenges of complex architectures and distributed systems in future iterations of the GSMM.

By delineating the main components for assessing software's energy and resource usage, the GSMM simplifies the process for developers

[23] The maximum amount of heat a chip can dissipate before it starts to wear down or malfunction.

[24] https://sdialliance.org/blog/sdia-digital-environmental-footprint-reaches-a-key-milestone/

and researchers to either devise new measurement models or choose, tailor, refine, and expand upon existing methods. This is achieved, for instance, by incorporating metadata collection to enhance the reusability of experiments. This ensures comparability and fosters a cohesive and comprehensive approach within the Green Software Community.

It must be taken into account that software is always connected to the underlying hardware and that this can also be a distributed system. There, the software is spread over several nodes or triggers complex data exchange processes over several network types. A certain comparability of different products can be achieved via metrics such as “useful work done per Joule”, provided that this “useful work” can be specified in a comparable way. It should also always be considered that used technologies such as databases, frameworks, and SDKs are error sources that can hinder exact comparisons [75]. Accordingly, the model defines guard rails as to what such measurement procedures can entail, such as keeping all other influencing factors equal and only exchanging the part that is of interest.

Our contribution presents several concrete procedures that can be applied directly. Further procedures can also be developed with the help of the GSMM. We could not include all currently available methods in this article due to the diversity of hardware and software solutions (desktop, distributed, mobile, cloud applications, etc.). There is not only one concrete procedure, but the GSMM contains the essential components for measuring the resource and energy efficiency of software in a variety of ways.

This collaborative approach promotes the expansion of knowledge on existing methods, tutorials, guidelines, metrics, and tools, all aimed at measuring and assessing the resource and energy efficiency of software. Moreover, by offering a structured framework and an open repository for universal contribution, the GSMM empowers the scientific community and software developers, promotes a shared understanding, clarifies terminology, and we think it is a step toward integrating energy and resource measurements into software practice. In terms of applicability, the GSMM is versatile, accommodating various system architectures, software and hardware types, and usage scenario types. Since its primary focus is on the direct effects of software, such as its energy consumption, the GSMM itself does not account for secondary and tertiary effects of software, but it can be extended (e. g., via the SAF or GREENSOFT model) to include these factors as well.

Next steps are (i) to integrate and test other models, and (ii) to apply these models to additional software products in order to set up a database of measurement results for several software solutions. This also includes tips and tricks regarding standard usage scenarios, automation, data processing, as well as expanding the open repository. With regard to the further development of the model, minimum standards can be developed so that a measurement methodology will be considered robust and directionally reliable. In a further step, the GSMM can also be adapted to include approaches to consider the software product context of the software stack, the data center environment, and also the integration into the “energy smart grid” (energy aware software).

## CRediT authorship contribution statement

**Achim Guldner:** Conceptualization, Data curation, Formal analysis, Investigation, Methodology, Resources, Software, Validation, Visualization, Writing – original draft, Writing – review & editing. **Rabea Bender:** Visualization. **Coral Calero:** Investigation, Methodology, Resources, Writing – original draft. **Giovanni S. Fernando:** Investigation, Methodology, Resources, Validation, Writing – original draft. **Markus Funke:** Investigation, Methodology, Resources, Validation, Writing – original draft. **Jens Gröger:** Investigation, Methodology, Resources, Validation, Writing – original draft. **Lorenz M. Hilty:** Investigation, Validation. **Julian Hörnschemeyer:** Investigation, Validation, Writing – review & editing. **Geerd-Dietger Hoffmann:** Investigation, Methodology, Resources, Validation, Writing – original draft. **Dennis Junger:** Investigation, Validation, Writing – review & editing. **Tom Kennes:** Investigation, Methodology, Validation, Writing – original draft. **Sandro Kreten:** Investigation, Methodology, Resources, Validation. **Patricia Lago:** Investigation, Methodology, Validation, Writing – original draft. **Franziska Mai:** Investigation, Validation. **Ivano Malavolta:** Investigation, Methodology, Resources, Validation, Writing – original draft. **Julien Murach:** Investigation, Validation. **Kira Obergöker:** Investigation, Validation, Writing – review & editing. **Benno Schmidt:** Investigation, Validation, Writing – review & editing. **Arne Tarara:** Investigation, Methodology, Resources, Software, Validation, Writing – original draft. **Joseph P. De Veaugh-Geiss:** Investigation, Validation, Writing – review & editing. **Sebastian Weber:** Conceptualization, Investigation, Validation, Writing – original draft, Writing – review & editing. **Max Westing:** Investigation, Validation, Writing – original draft. **Volker Wohlgemuth:** Investigation, Methodology, Validation. **Stefan Naumann:** Conceptualization, Investigation, Methodology, Validation, Writing – original draft, Writing – review & editing.

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Data availability

No data was used for the research described in the article.

## Acknowledgments

This work was supported by the Rhineland-Palatinate Ministry for Science and Health, Germany (research training group ”AI-CPPS“); the German Federal Ministry for the Environment, Nature Conservation, Nuclear Safety, Germany, and Consumer Protection project “KIRA” (grant 67KI32013B); the German Federal Ministry for Economic Affairs and Climate Action project, Germany “EASY” (grant 01MD22002D); the Spanish Ministry of Science and Innovation, Spain and the European Regional Development Fund ERDF (State Plan for Scientific, Technical and Innovation Research 2021–2023) project “OASSIS” (grant PID2021-122554OB-C31); JCCM (Junta de Comunidades de Castilla-La Mancha, Spain) and the ERDF project “EMMA” (grant SBPLY/21/180501/000115); the EU Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie (grant 871342) “uDEVOPS”; the SustainableCloud project (OCENW.M20.243) from the research program Open Competition by the Dutch Research Council; (NWO); and the LETSGO project from the Netherlands Enterprise Agency (RVO).

## References

[1] A. Ross, L. Christie, Energy Consumption of ICT, POSTNOTE 677, The Parliamentary Office of Science and Technology, Westminster, London SW1A 0AA, 2022, URL https://post.parliament.uk/research-briefings/post-pn-0677/. (Accessed 21 November 2023).

[2] J. Vidal, ‘Tsunami of data’ could consume one fifth of global electricity by 2025, 2017, URL https://www.climatechangenews.com/2017/12/11/tsunami-data-consume-one-fifth-global-electricity-2025/.

[3] A.S.G. Andrae, Prediction studies of electricity use of global computing in 2030, Int. J. Sci. Eng. Investig. (IJSEI) 8 (86) (2019) 27–33, URL http://www.ijsei.com/papers/ijsei-88619-04.pdf.

[4] R. Verdecchia, J. Sallou, L. Cruz, A systematic review of green AI, WIREs Data Min. Knowl. Discov. 13 (4) (2023) e1507, http://dx.doi.org/10.1002/widm.1507.

[5] J. Sedlmeir, H.U. Buhl, G. Fridgen, R. Keller, The energy consumption of blockchain technology: Beyond myth, Bus. Inf. Syst. Eng. 62 (6) (2020) 599–608, http://dx.doi.org/10.1007/s12599-020-00656-x.

[6] B. Penzenstadler, A. Raturi, D. Richardson, C. Calero, H. Femmer, X. Franch, Systematic mapping study on software engineering for sustainability (SE4s), in: 18th International Conference on Evaluation and Assessment in Software Engineering, EASE ’14, ACM, 2014, pp. 1—14, http://dx.doi.org/10.1145/2601248.2601256.

[7] B. Fernandes, G. Pinto, F. Castor, Assisting non-specialist developers to build energy-efficient software, in: 39th International Conference on Software Engineering Companion, (ICSE-C), in: ICSE-C '17, IEEE Press, 2017, pp. 158–160, http://dx.doi.org/10.1109/ICSE-C.2017.133.

[8] E. Kern, Green computing, green software, and its characteristics: Awareness, rating, challenges, in: From Science to Society, Springer, Cham, 2018, pp. 263–273.

[9] D. Junger, M. Westing, C. Freitag, A. Guldner, K. Mittelbach, S. Weber, S. Naumann, V. Wohlgemuth, Potentials of green coding - findings and recommendations for industry, education and science, in: INFORMATIK 2023 - Designing Futures: Zukünfte Gestalten, Gesellschaft für Informatik e.V., Bonn, 2023, pp. 1289–1299, http://dx.doi.org/10.18420/inf2023_137.

[10] S. Naumann, M. Dick, E. Kern, T. Johann, The GREENSOFT model: A reference model for green and sustainable software and its engineering, Sustain. Comput.: Inform. Syst. 1 (4) (2011) 294–304, http://dx.doi.org/10.1016/j.suscom.2011.06.004.

[11] C. Venters, C. Jay, L. Lau, M. Griffiths, V. Holmes, R. Ward, J. Austin, C. Dibsdale, J. Xu, Software sustainability: The modern tower of babel, in: 3rd International Workshop on Requirements Engineering for Sustainable Systems, vol. 1216, 2014, pp. 7–12, URL https://ceur-ws.org/Vol-1216/paper2.pdf.

[12] C. Becker, R. Chitchyan, L. Duboc, S. Easterbrook, B. Penzenstadler, N. Seyff, C. Venters, Sustainability design and software: The karlskrona manifesto, in: 37th IEEE International Conference on Software Engineering, 2015, pp. 467–476, http://dx.doi.org/10.1109/ICSE.2015.179.

[13] C. Calero, M. Piattini, Green in Software Engineering, 2015, pp. 1–327, http://dx.doi.org/10.1007/978-3-319-08581-4.

[14] A. Fonseca, R. Kazman, P. Lago, A manifesto for energy-aware software, IEEE Softw. 36 (6) (2019) 79–82, http://dx.doi.org/10.1109/MS.2019.2924498.

[15] C. Pang, A. Hindle, B. Adams, A.E. Hassan, What do programmers know about software energy consumption? IEEE Softw. 33 (3) (2016) 83–89, http://dx.doi.org/10.1109/MS.2015.83.

[16] P. Lago, D. Greefhorst, E. Woods, Architecting for sustainability, in: EnviroInfo 2022, Gesellschaft für Informatik e.V., Bonn, 2022, p. 199, URL https://dl.gi.de/handle/20.500.12116/39397.

[17] D. Li, W.G.J. Halfond, An investigation into energy-saving programming practices for android smartphone app development, in: Proceedings of the 3rd International Workshop on Green and Sustainable Software, in: GREENS 2014, ACM, 2014, pp. 46–53, http://dx.doi.org/10.1145/2593743.2593750.

[18] I. Manotas, C. Bird, R. Zhang, D. Shepherd, C. Jaspan, C. Sadowski, L. Pollock, J. Clause, An empirical study of practitioners' perspectives on green software engineering, in: 38th International Conference on Software Engineering, ICSE '16, ACM, 2016, pp. 237–248, http://dx.doi.org/10.1145/2884781.2884810.

[19] R. Pereira, M. Couto, F. Ribeiro, R. Rua, J. Cunha, J.P. Fernandes, J. Saraiva, Energy efficiency across programming languages: How do energy, time, and memory relate? in: 10th International Conference on Software Language Engineering, in: SLE 2017, ACM, 2017, pp. 256–267, http://dx.doi.org/10.1145/3136014.3136031.

[20] S. Chowdhury, A. Hindle, Greenoracle: estimating software energy consumption with energy measurement corpora, in: Proceedings of the 13th International Conference on Mining Software Repositories, 2016, pp. 49–60, http://dx.doi.org/10.1145/2901739.2901763.

[21] S. Chowdhury, S. Borle, S. Romansky, A. Hindle, Greenscaler: training software energy models with automatic test generation, Empir. Softw. Eng. 24 (2019) 1573–7616, http://dx.doi.org/10.1007/s10664-018-9640-7.

[22] D. Guamán, J. Pérez, Supporting sustainability and technical debt-driven design decisions in software architectures, in: Information Systems Development: Crossing Boundaries Between Development and Operations (DevOps) in Information Systems, AIS, 2021, p. na.

[23] J. Cabot, R. Capilla, C. Carrillo, H. Muccini, B. Penzenstadler, Measuring systems and architectures: A sustainability perspective, IEEE Softw. 36 (03) (2019) 98–100, http://dx.doi.org/10.1109/MS.2019.2897833.

[24] L. Cruz, R. Abreu, J. Grundy, L. Li, X. Xia, Do energy-oriented changes hinder maintainability? in: 2019 IEEE International Conference on Software Maintenance and Evolution, ICSME, 2019, pp. 29–40, http://dx.doi.org/10.1109/ICSME.2019.00013, arXiv:1908.08332.

[25] C. Calero, M. Polo, M. Moraga, Investigating the impact on execution time and energy consumption of developing with spring, Sustain. Comput.: Inform. Syst. 32 (2021) http://dx.doi.org/10.1016/j.suscom.2021.100603.

[26] Apple Inc., Energy efficiency and the user experience, 2018, URL https://developer.apple.com/library/archive/documentation/Performance/Conceptual/EnergyGuide-iOS/index.html.

[27] S.J. Chinenyeze, X. Liu, Architecting green mobile cloud apps: Key considerations for implementation and evaluation of mobile cloud apps, in: C. Calero, et al. (Eds.), Software Sustainability, Springer, 2021, pp. 183–214, http://dx.doi.org/10.1007/978-3-030-69970-3_8.

[28] J. Mancebo, C. Calero, F. Garcia, M.A. Moraga, I. Garcia-Rodriguez de Guzman, FEETINGS: Framework for energy efficiency testing to improve environmental goal of the software, Sustain. Comput.: Inform. Syst. 30 (2021) http://dx.doi.org/10.1016/j.suscom.2021.100558.

[29] C. Calero, M.Á. Moraga, M. Piattini, Introduction to software sustainability, Softw. Sustain. (2021) 1–15.

[30] B. Purvis, Y. Mao, D. Robinson, Three pillars of sustainability: in search of conceptual origins, Sustain. Sci. 14 (2019) 681–695, http://dx.doi.org/10.1007/s11625-018-0627-5.

[31] C. Calero, M. Piattini, Puzzling out software sustainability, Sustain. Comput.: Inform. Syst. 16 (2017) http://dx.doi.org/10.1016/j.suscom.2017.10.011.

[32] S. Wang, H. Chen, W. Shi, SPAN: A software power analyzer for multicore computer systems, Sustain. Comput.: Inform. Syst. 1 (1) (2011) 23–34, http://dx.doi.org/10.1016/j.suscom.2010.10.002.

[33] S. Wang, Y. Li, W. Shi, L. Fan, A. Agrawal, Safari: Function-level power analysis using automatic instrumentation, in: 2012 International Conference on Energy Aware Computing, IEEE, 2012, pp. 1–6, http://dx.doi.org/10.1109/iceac.2012.6471014.

[34] T. Hönig, C. Eibel, W. Schröder-Preikschat, B. Cassens, R. Kapitza, Proactive energy-aware system software design with SEEP, Softwaretechnik-Trends 33 (2) (2013) 6–7, http://dx.doi.org/10.1007/s40568-013-0021-5.

[35] A. Hindle, A. Wilson, K. Rasmussen, J. Barlow, H. Campbell, S. Romansky, Greenminer: A hardware based mining software repositories software energy consumption framework, in: 11th Working Conference on Mining Software Repositories, 2014, http://dx.doi.org/10.1145/2597073.2597097.

[36] M. Jay, V. Ostapenco, L. Lefèvre, D. Trystram, A.C. Orgerie, B. Fichel, An experimental comparison of software-based power meters: focus on CPU and GPU, in: CCGrid 2023 - 23rd IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing, IEEE, Bangalore, India, 2023, pp. 1–13, URL https://inria.hal.science/hal-04030223.

[37] H. Hè, M. Friedman, T. Rekatsinas, Energat: Fine-grained energy attribution for multi-tenancy, 2023, arXiv:2307.05647.

[38] V. Schmidt, K. Goyal, A. Joshi, B. Feld, L. Conell, N. Laskaris, D. Blank, J. Wilson, S. Friedler, S. Luccioni, CodeCarbon, 2021, p. 20, URL https://github.com/mlco2/codecarbon. Accessed 21 November 2023.

[39] P. Henderson, J. Hu, J. Romoff, E. Brunskill, D. Jurafsky, J. Pineau, Towards the systematic reporting of the energy and carbon footprints of machine learning, J. Mach. Learn. Res. (2020) Arxiv: 2002.05651.

[40] A. Guldner, S. Kreten, S. Naumann, Exploration and systematic assessment of the resource efficiency of machine learning, in: INFORMATIK 2021 - Computer Science and Sustainability, in: Lecture Notes in Informatics (LNI), 2021, pp. 287—299, http://dx.doi.org/10.18420/informatik2021-023.

[41] Z. Ournani, Software Eco-Design: Investigating and Reducing the Energy Consumption of Software (Ph.D. thesis), University of Lille, 2021, URL https://tel.archives-ouvertes.fr/tel-03429300.

[42] T. Schade, Greencoding-measuring-tools, 2023, URL http://github.com/schaDev/GreenCoding-measuring-tools. Accessed 21 November 2023.

[43] EnergyStar, Energy star version 8.0 computers draft test method update, 2022, URL https://www.energystar.gov/sites/default/files/asset/document/ENERGY%20STAR%20Draft%20Test%20Method%20for%20Computers.pdf. (Accessed 14 August 2023).

[44] EnergyStar, Energy star computers version 8.0 final specification – Rev. July 2022, 2022, URL https://www.energystar.gov/sites/default/files/asset/document/ENERGY%20STAR%20Computers%20Version%208.0%20Final%20Specification%20Rev.%20July%202022.pdf. (Accessed 14 August 2023).

[45] S. Naumann, A. Guldner, E. Kern, The eco-label blue angel for software - development and components, in: Advances and New Trends in Environmental Informatics: Digital Twins for Sustainability, Springer, 2021, pp. 79–89, http://dx.doi.org/10.1007/978-3-030-61969-5_6.

[46] E. Kern, L.M. Hilty, A. Guldner, Y.V. Maksimov, A. Filler, J. Gröger, S. Naumann, Sustainable software products — towards assessment criteria for resource and energy efficiency, Future Gener. Comput. Syst. 86 (2018) 199–210, http://dx.doi.org/10.1016/j.future.2018.02.044.

[47] J. Mancebo, F. García, C. Calero, A process for analysing the energy efficiency of software, Inf. Softw. Technol. 134 (2021) http://dx.doi.org/10.1016/j.infsof.2021.106560.

[48] P. Lago, N. Condori-Fernandez, The Sustainability Assessment Framework (SAF) Toolkit: Instruments to Help Sustainability-Driven Software Architecture Design Decision Making, S2 Group, Vrije Universiteit Amsterdam, 2022, URL https://github.com/S2-group/SAF-Toolkit.

[49] C. Wohlin, P. Runeson, M. Hst, M.C. Ohlsson, B. Regnell, A. Wessln, Experimentation in Software Engineering, Springer, 2012, p. 68, http://dx.doi.org/10.1007/978-3-642-29044-2.

[50] F. Shull, J. Singer, D.I. Sjøberg, Guide to Advanced Empirical Software Engineering, Springer, 2007.

[51] N. Rteil, R. Bashroush, R. Kenny, A. Wynne, Interact: IT infrastructure energy and cost analyzer tool for data centers, Sustain. Comput.: Inform. Syst. 33 (2022) http://dx.doi.org/10.1016/j.suscom.2021.100618.

[52] T. Kennes, Measuring IT carbon footprint: What is the current status actually?, 2023, arXiv:2306.10049.

[53] S. Kreten, Modellbildung und Umsetzung von Methoden zur energieeffizienten Nutzung von Containertechnologien (Ph.D. thesis), Trier University, 2022, p. 162, http://dx.doi.org/10.25353/ubtr-xxxx-099b-6fe5.

[54] W. Oliveira, R. Oliveira, F. Castor, A study on the energy consumption of android app development approaches, in: 14th International Conference on Mining Software Repositories, MSR, IEEE, 2017, pp. 42–52, http://dx.doi.org/10.1109/msr.2017.66.

[55] T. Johann, M. Dick, S. Naumann, E. Kern, How to measure energy-efficiency of software: Metrics and measurement results, in: 2012 1st International Workshop on Green and Sustainable Software, GREENS 2012 - Proceedings, 2012, pp. 51–54, http://dx.doi.org/10.1109/GREENS.2012.6224256.

[56] R. Verdecchia, R. Saez, G. Procaccianti, P. Lago, Empirical evaluation of the energy impact of refactoring code smells, in: 5th International Conference on Information and Communication Technology for Sustainability, (ICT4S), vol. 52, 2018, pp. 365–383, http://dx.doi.org/10.29007/dz83.

[57] A. Guldner, M. Garling, M. Morgen, S. Naumann, E. Kern, L.M. Hilty, Energy consumption and hardware utilization of standard software: Methods and measurements for software sustainability, in: From Science to Society, Springer, Cham, 2018, pp. 251–261.

[58] L. Ardito, R. Coppola, M. Morisio, M. Torchiano, Methodological guidelines for measuring energy consumption of software applications, in: M. Risi (Ed.), Sci. Program. (2019) 1–16, http://dx.doi.org/10.1155/2019/5284645.

[59] L. Cruz, Green software engineering done right: A scientific guide to set up energy efficiency experiments, 2021, http://dx.doi.org/10.6084/m9.figshare.22067846.v1, Blog post.

[60] A. Guldner, J. Murach, Measuring and assessing the resource and energy efficiency of artificial intelligence of things devices and algorithms, in: Advances and New Trends in Environmental Informatics, Springer, Cham, 2022, pp. 185–199, http://dx.doi.org/10.1007/978-3-031-18311-9_11.

[61] P. Lago, S.A. Koçak, I. Crnkovic, B. Penzenstadler, Framing sustainability as a property of software quality, Commun. ACM 58 (10) (2015) 70–78.

[62] P. Lago, Architecture design decision maps for software sustainability, in: 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Society (ICSE-SEIS), IEEE, Montreal, QC, Canada, 2019, pp. 61–64.

[63] N. Condori-Fernandez, P. Lago, M.R. Luaces, Á.S. Places, An action research for improving the sustainability assessment framework instruments, Sustainability 12 (4) (2020) http://dx.doi.org/10.3390/su12041682.

[64] M. Funke, P. Lago, R. Verdecchia, Variability features: extending sustainability decision maps via an industrial case study, in: 20th International Conference on Software Architecture Companion, ICSA-C, IEEE, 2023, pp. 1–7, http://dx.doi.org/10.1109/ICSA-C57050.2023.00024.

[65] G. Procaccianti, P. Lago, A. Vetrò, D. Méndez Fernández, R. Wieringa, The green lab: Experimentation in software energy efficiency, in: 37th International Conference on Software Engineering, ICSE, vol. 2, ACM/IEEE, 2015, pp. 941–942, http://dx.doi.org/10.1109/ICSE.2015.297.

[66] I. Malavolta, E.M. Grua, C.-Y. Lam, R. de Vries, F. Tan, E. Zielinski, M. Peters, L. Kaandorp, A framework for the automatic execution of measurement-based experiments on android devices, in: 35th IEEE/ACM International Conference on Automated Software Engineering, ASE '20, ACM, 2021, pp. 61–66, http://dx.doi.org/10.1145/3417113.3422184, AndroidRunner: https://github.com/S2-group/android-runner.

[67] S. Swanborn, I. Malavolta, Robot runner, 2021, URL https://github.com/S2-group/robot-runner. Accessed 21 November 2023.

[68] S.G., Github - S2-group/experiment-runner: Tool for the automatic orchestration of experiments targeting software systems, 2022, URL https://github.com/S2-group/experiment-runner.

[69] I. Malavolta, K. Nirghin, G. Scoccia, S. Romano, S. Lombardi, G. Scanniello, P. Lago, Javascript dead code identification, elimination, and empirical assessment, IEEE Trans. Softw. Eng. 49 (7) (2023) 3692–3714, http://dx.doi.org/10.1109/TSE.2023.3267848, URL http://www.ivanomalavolta.com/files/papers/TSE_2023.pdf.

[70] M. Dordevic, M. Albonico, G. Lewis, I. Malavolta, P. Lago, Computation offloading for ground robotic systems communicating over WiFi - An empirical exploration on performance and energy trade-offs, Empir. Softw. Eng. 28 (140) (2023) 1573–7616, http://dx.doi.org/10.1007/s10664-023-10351-6.

[71] M. Dinga, I. Malavolta, L. Giammattei, A. Guerriero, R. Pietrantuono, An empirical evaluation of the energy and performance overhead of monitoring tools on docker-based systems, in: Service-Oriented Computing, ICSOC '23, Springer, Cham, 2023, pp. 181–196, http://dx.doi.org/10.1007/978-3-031-48421-6_13.

[72] R. Horn, A. Lahnaoui, E. Reinoso, S. Peng, V. Isakov, T. Islam, I. Malavolta, Native vs web apps: Comparing the energy consumption and performance of android apps and their web counterparts, in: 10th IEEE/ACM International Conference on Mobile Software Engineering and Systems, 2023, pp. 44–54, URL http://www.ivanomalavolta.com/files/papers/MOBILESoft_2023.pdf.

[73] L. Wagner, M. Mayer, A. Marino, A. Nezhad, H. Zwaan, I. Malavolta, On the energy consumption and performance of WebAssembly binaries across programming languages and runtimes in IoT, in: 9th International Conference on Evaluation and Assessment on Software Engineering, EASE, IEEE, 2023, pp. 72–82, URL http://www.ivanomalavolta.com/files/papers/EASE_2023.pdf.

[74] S. Kreten, A. Guldner, S. Naumann, An analysis of the energy consumption behavior of scaled, containerized web apps, Sustainability 10 (2018) 2710, http://dx.doi.org/10.3390/su10082710.

[75] D. Junger, V. Wohlgemuth, E. Kammer, Conception and test of a measuring station for the analysis of the resource and energy consumption of material flow-oriented environmental management information systems (EMIS), in: EnviroInfo 2022, Gesellschaft für Informatik e.V., Bonn, 2022, p. 211.

**Achim Guldner**, M.Sc. is a researcher and PhD student at Trier University of Applied Sciences, Environmental Campus Birkenfeld, in conjunction with Trier University. He graduated in 2013 with a Master's degree in applied computer sciences. His research focuses on information systems for sustainable development. He worked in several R&D Projects, focusing on improving the resource and energy efficiency of and through ICT. Since 2016, he is involved in the research group "Green Software Engineering", where his focus is on measuring, analyzing, and optimizing the resource and energy efficiency of software systems. His Ph.D. thesis addresses, i. a., the transfer of the subject of Green Software to AI-based systems.

**Rabea Bender** is a researcher and Master's student at Trier University of Applied Sciences, Environmental Campus Birkenfeld. She holds a B.Sc. in Media Informatics.

**Coral Calero** is full professor and member of the Alarcos Research Group at UCLM, responsible for the Green and Sustainable Software line of research. She works on software sustainability and is an author of several journal publications about the topic. She is also editor of the Books "Green Software Engineering" and "Software Sustainability" edited by Springer.

**Giovanni S. Fernando** achieved his B.Sc. in Computer Science from Vrije Universiteit Amsterdam, while working as a research assistant for the S2 Group. He finished his degree with a thesis on researching the power consumption of docker-based micro-services. Following this he began working as a Software Engineer at KPN Amsterdam.

**Markus Funke** is a Ph.D. candidate at the Software and Sustainability group (S2) of the Vrije Universiteit Amsterdam, The Netherlands. He holds a joint Master's degree in Computer Science, appointed by the Vrije Universiteit Amsterdam and the University of Amsterdam, both in the Netherlands. His research interests focus on software architecture, architecture knowledge, and software sustainability, with particular interest in the relation to professional practice and the industry. Prior to joining the VU, he worked as a software developer and technical architect. As a researcher, he is involved in the VU Digital Sustainability Center (DiSC) and the EU research project "uDEVOPS".

**Jens Gröger** is senior researcher at the Oeko-Institut Berlin in the area of products and material flows. His research focuses on sustainable information and communications technology. He develops methods for assessing the energy efficiency of computers, software, telecommunications networks, and data centers. He advises German and European authorities on defining sustainability criteria for eco-labels, public procurement, and legislative projects. He has conducted sustainability analyses for a wide range of ICT products, such as computers, smartphones, televisions, video conferencing systems, servers, and storage products, and derived minimum requirements for the German Blue Angel eco-label. A key issue of his research are developing suitable methods for determining the environmental impact of complex hardware and software systems.

**Prof. Dr. Lorenz M. Hilty** is full Professor of Informatics and Sustainability at the University of Zurich (UZH), Sustainability Delegate of UZH, and Director of the Zurich Knowledge Center for Sustainable Development. His research explores the opportunities and risks of digitalization for sustainable development. He has published over 200 articles in his field. Lorenz initiated the ICT4S (Information and Communication Technologies for Sustainability) conference series with ICT4S 2013 in Zurich.

**Julian Hörnschemeyer** has a B.Sc. in Business Informatics and M.Sc. in Computer Science. During his Master's he started developing an interest in sustainable solutions and sustainable software in particular. In his Master's thesis he developed a model to evaluate the energy efficiency of decentralized communication protocols like Matrix and ActivityPub. Currently, he is the CTO of the startup RegioShopper, which develops software solutions for the marketing and distribution of organic and regional products.

**Geerd-Dietger Hoffmann** is Head of Engineering at Green Coding Berlin. Since 2008, his professional journey has seen pivotal roles at organizations worldwide, from eHealth Africa, CERN, IBM to ClimateFarmers. Mr. Hoffmann has leveraged his expertise in multiple programming languages and frameworks to address societal challenges such as health crisis management, sustainable building, and sustainable agriculture. He is a co-creator of various systems including publishing systems, health collaboration platforms, and a carbon capture monitoring system. Mr. Hoffmann has been instrumental in the development of impactful solutions like an online tool revolutionizing collaborative document creation. He is now focusing on Green Software and the impact software has on sustainability.

**Dennis M. Junger**, M.Sc. has a B.Sc. in Computational Engineering, an M.Sc. in Industrial Environmental Informatics, and is a scientific researcher in the Potentials of Green Coding project at HTW Berlin. He collaborates with the project partners Environmental Campus Birkenfeld and German Informatics Society (GI e.V.) to integrate contemporary green coding concepts into software development practices and education. With expertise in C++ and software development, he also lectures on these and various IT subjects.

**Tom Kennes** is a Technology Lead/Solution Architect at Nationale Nederlanden and provides advice with regard to using the Cloud in a sustainable manner. He is also an ambassador with the Sustainable Digital Infrastructure Alliance (SDIA), through which he is involved With various public and research activities around sustainability and IT. The SDIA is a non-profit organisation focused on improving and spreading knowledge around sustainable IT infrastructure, software, and practices.

**Dr. Sandro Kreten** is Vice President Technology of the early stage impact investor *capacura*, where he is helping startups to build energy efficient and scaling software. After completing his studies in mathematics and computer science at Trier University, Sandro finished his Ph.D. on "Modeling and implementation of methods for energy-efficient use of container technologies" at Trier University of Applied Sciences, Environmental Campus Birkenfeld and Trier University.

**Prof. Dr. Patricia Lago** is full professor in software engineering at Vrije Universiteit Amsterdam, The Netherlands, where she founded the Software and Sustainability research group in the Computer Science Department, and the VU DiSC - Digital Sustainability Center. Her passion in research is to create software engineering knowledge that makes software better, smarter, and more sustainable. Her research focuses primarily on software architecture design and decision making, software quality assessment, and software sustainability. She is the recipient of an Honorary Doctorate at NTNU, Norway, for her contribution to the field of software sustainability, and of the 2023 IEEE CS TCSE New Directions Award. She has a Ph.D. in Control and Computer Engineering from Politecnico di Torino and a Master in Computer Science from the University of Pisa, both in Italy. She has published over 250 articles in all major scientific conferences and journals of her field. She is a senior member of ACM, IEEE, and VERSEN. More info at: www.patricialago.nl.

**Franziska Mai** holds an M. Sc. from Trier University of Applied Sciences, Environmental Campus Birkenfeld. She wrote her bachelor thesis in the field of sustainable software and since 2020 she is involved in the further development of the Blue Angel for software products.

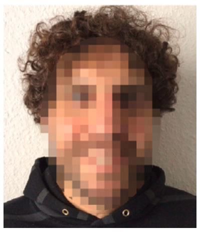
**Ivano Malavolta** is associate professor and Director of the Network Institute at the Vrije Universiteit Amsterdam, The Netherlands. His research focuses on software engineering, with a special emphasis on green software, software architecture, mobile software development, and robotics software. He applies empirical methods to assess practices and trends in the field of software engineering. He has authored more than 100 scientific articles in international journals and peer-reviewed international conference proceedings. He is a program committee member and reviewer of international conferences and journals in the software engineering field. He received a Ph.D. in computer science from the University of L'Aquila in 2012. He is a member of ACM, IEEE, VERSEN, and Amsterdam Data Science.

**Julien Murach** is a team leader in the special field of metering services/smart metering and is a former technical manager in the Distributed Systems and Artificial Intelligence research group. He holds a Bachelor's degree in Electrical Engineering from the University of Applied Sciences Berlin and a Master's degree in Computer Science from Trier University of Applied Sciences, Environmental Campus Birkenfeld.

**Kira Obergöker** is a researcher and Master's student at Trier University of Applied Sciences, Environmental Campus Birkenfeld, where she worked on developing criteria for evaluating the energy and resource efficiency of software and was involved in software efficiency measurements. She has a B.Sc. in Environmental Informatics and Business Information Systems.

**Prof. Dr. Benno Schmidt** holds a degree in computer science and worked as a professional software developer for 12+ years. He joined Bochum University of Applied Sciences in 2009 as a professor and teaches computer science with a focus on software engineering aspects and geospatial applications. In his research he deals with geovisualization topics and the development of green and socially sustainable software.

**Arne Tarara** is the CEO of Green Coding Berlin, a company that specializes in creating open source tools for transparency around software energy consumption. Being a professional software engineer for 16+ years he is active in the field of sustainable software engineering since 2021 and is an active contributor to other projects in the domain as well as a speaker at various conferences.



**Dr. Joseph P. De Veaugh-Geiss** is community and project manager for the KDE Eco initiative at the Free Software non-profit KDE e. V. In 2021 KDE's PDF reader and universal document viewer Okular was the first software product to be awarded the Blue Angel eco-label. Since 2022, the KDE community has adopted the goal of sustainable software design to create a better world for this and future generations. Dr. De Veaugh-Geiss also teaches a course on digitization at the Universität der Künste Berlin. Prior to working at KDE, Dr. De Veaugh-Geiss was a researcher in the field of experimental and theoretical linguistics at the Universität Potsdam, Germany.

**Sebastian Weber**, M.C.Sc. is a researcher at Environmental Campus Birkenfeld since 2023. His research focus is in the areas of Green Artificial Intelligence. Before that, he worked for 10 years in the field of data science/engineering as a consultant and permanent employee in various projects.

**Max Westing** is a researcher and Master's student at Trier University of Applied Sciences, Environmental Campus Birkenfeld, where he has worked on several projects dealing with Green Software, such as software efficiency measurements and green coding. He has a B.A. in Translation Studies for Information Technologies.

**Prof. Dr. Volker Wohlgemuth** has been a professor at HTW Berlin in material flow management, modeling and simulation, and the application and development of Environmental Management Information Systems (EMIS) since the winter semester of 2005/2006. His research focuses on all types of EMIS, Life cycle assessment (LCA), and Material Flow Cost Accounting (MFCA) as part of Green by IT and Software Sustainability as found in Green IT, especially Green Coding.

**Prof. Dr. Stefan Naumann** has taught computer science, mathematics, and environmental and sustainability informatics at Trier University of Applied Sciences, Environmental Campus Birkenfeld since 2008. His research focus is in the areas of Green IT, Green by IT, and Green Software, with a focus on Artificial Intelligence. He is co-initiator of various workshop series such as "Artificial Intelligence in Environmental Informatics" and "Energy Aware Software Engineering" and is a spokesperson for the Environmental Informatics Section of the German Informatics Society (GI e.V.). He has published over 100 relevant papers and has led numerous research projects in the area of Green Software and Green IT. Recently, he was significantly involved in the development of the Blue Angel for software products.