TEXTE
71/2026

Abschlussbericht

SoftAWERE

Energieeffizienz-Kennwerte von Komponenten und Werkzeugen der Softwareentwicklung und Vorarbeiten zur Etablierung einer Kennzeichnung für energieeffiziente Software.

von:

Max Schulze

Sustainable Digital Infrastructure Alliance e.V., Hamburg

Jens Gröger, Felix Behrens

Öko Institut e.V., Berlin

Herausgeber:
Umweltbundesamt

Für Mensch & Umwelt

Umwelt Bundesamt

TEXTE 71/2026

EVUPLAN des Bundesministeriums für Wirtschaft und Klimaschutz

Forschungskennzahl 37EV 20 102 0

Abschlussbericht

SoftAWERE

Energieeffizienz-Kennwerte von Komponenten und Werkzeugen der Softwareentwicklung und Vorarbeiten zur Etablierung einer Kennzeichnung für energieeffiziente Software.

von

Max Schulze

Sustainable Digital Infrastructure Alliance e.V., Hamburg

Jens Gröger, Felix Behrens

Öko Institut e.V., Berlin

Im Auftrag des Umweltbundesamtes unter Fachaufsicht des BMWK

Impressum

Herausgeber
Umweltbundesamt
Wörlitzer Platz 1
06844 Dessau-Roßlau
Tel: +49 340-2103-0
Fax: +49 340-2103-2285
buergerservice@uba.de
Internet: www.umweltbundesamt.de

Durchführung der Studie:
Sustainable Digital Infrastructure Alliance e.V.
Colonnaden 51
20354 Hamburg

Abschlussdatum:
Mai 2023

Redaktion:
V 1.4 Energieeffizienz
Andreas Halatsch

Z 2.3 Digitale Transformation und Beratungsstelle Green IT
Anna Zagorski, Marina Köhn

DOI:
https://doi.org/10.60810/openumwelt-7617

ISSN 1862-4804

Dessau-Roßlau, Mai 2026

Die Verantwortung für den Inhalt dieser Veröffentlichung liegt bei den Autor*innen und Autoren.

Kurzbeschreibung: Forschungsvorhaben SoftAWERE Abschlussbericht

Software hat einen großen Einfluss auf den Energieverbrauch und die Leistungsanforderungen der Informationstechnik. Um die Energie- und Ressourceneffizienz bereits im Entwicklungsprozess der Software zu erhöhen, müssen die Verbräuche bereits während der Entwicklungsphase bekannt sein. Mit dem Ziel, Software-Entwickler*innen Werkzeuge an die Hand zu geben, um den Energieverbrauch bereits bei der Softwareentwicklung zu senken, hat das Umweltbundesamt (UBA), finanziert vom Bundesministerium für Wirtschaft und Klimaschutz (BMWK), das Forschungsvorhaben „SoftAWERE – Software Architektur-Werkzeuge für energieeffiziente und ressourcenschonende Entwicklung" durchgeführt.

Folgende Ergebnisse und Methodiken konnten bisher im Rahmen des SoftAWERE Forschungsvorhabens realisiert werden:

1. Eine Herangehensweise und Methodik für die Einbettung von Messungen zum Energieverbrauch und zu Umweltwirkungen (Abiotischer Rohstoffverbrauch - ADP , Treibhausgaspotenzial - GWP, Kumulierter Energieaufwand – KEA und Wasserverbrauch – Water Usage) in gängigen Entwicklungsprozessen.
2. Architektur und Aufbau eines digitalen Messlabors, mit dem die Methodik validiert werden kann und gängige quelloffene Bibliotheken hinsichtlich Energieverbrauch und Umweltwirkungen evaluiert werden können.
3. Erarbeitung einer konzeptionellen Herangehensweise für eine Transparenzkennzeichnung für Software.

Für die Messung von Softwarecode wurden im Forschungsvorhaben SoftAWERE zwei Methoden entwickelt:

- „SoftAWERE-Light": Eine vereinfachte Variante, die ohne Messungen auskommt und mit mathematischen Formeln eine Schätzung zur Energie- und Umweltwirkung errechnet. Diese Variante eignet sich für Cloud- und virtualisierte Umgebungen, in denen eine physische Messung mit Messgeräten oder eine Messung über Server-Schnittstellen nicht möglich ist.
- „SoftAWERE": Die vollständige Variante, welche ohne physische Messgeräte auskommt und auf den bestehenden Schnittstellen von Server-Herstellern und Betriebssystemen aufbaut.

Abstract: SoftAWERE

The influence of software on energy consumption and the performance requirements of information technology is a significant factor that must be considered. To enhance energy efficiency and reduce environmental impact throughout the software development process, it is essential to ascertain both energy consumption and environmental impact during the development phase. To equip software developers with the necessary tools to reduce energy consumption and environmental effects during the software development process, the Federal Environment Agency (UBA) has undertaken the research project "SoftAWERE - Software architecture tools for energy-efficient and resource-saving development"), funded by the Federal Ministry of Economics and Climate Protection (BMWK).

The following results and methodologies have been achieved thus far as part of the SoftAWERE research project:

1. An approach and methodology for integrating measurements of energy consumption and environmental impacts, such as Abiotic Depletion Potential (ADP), Global Warming Potential

(GWP), Cumulative Energy Expenditure (CED) and Water Usage, into standard development processes.
2. A digital measurement laboratory which enabled the methodology to be validated and common open-source libraries to be evaluated regarding their energy consumption and environmental impacts.
3. A conceptual approach to transparency labelling for software.

Two methods for measuring software code were developed as part of the SoftAWERE research project.

The "SoftAWERE-Light" variant is a simplified version that does not necessitate direct measurements and instead employs mathematical formulae to estimate the energy and environmental impact. This variant is well-suited for cloud and virtualised environments where physical measurement with measuring devices or measurement via server interfaces is not feasible.

In contrast, the "SoftAWERE" variant is a comprehensive solution that does not require the use of physical measuring devices and is based on the existing interfaces of server manufacturers and operating systems.

Inhaltsverzeichnis

Abbildungsverzeichnis ... 9
Tabellenverzeichnis ... 12
Abkürzungsverzeichnis ... 13
Zusammenfassung ... 15
Summary ... 27
1 Hintergrund & Zielsetzung ... 39
1.1 Zielsetzung ... 49
1.2 Untersuchungsrahmen ... 51
1.3 Voraussetzung und bestehende methodische Ansätze ... 53
2 Methodische Herausforderungen und Lösungsansätze ... 55
2.1 Methodische Herausforderungen ... 55
2.2 Methodenentwicklung ... 57
2.3 Auswertung Messergebnisse & Bewertung ... 75
2.3.1 Verständlichkeit ... 75
2.3.2 Plausibilität ... 76
2.3.3 Richtungssicherheit ... 79
2.3.4 Reproduzierbarkeit ... 80
2.3.5 Bewertung der Methodik insgesamt ... 86
2.4 Anwendung der Methode ... 86
3 Handlungsempfehlungen für Politik, Forschung und die Software-Community ... 90
3.1 Kennzeichnung ... 91
3.1.1 Energie-Kennzeichnung für Software ... 91
3.1.2 Das Label ... 92
3.2 Forschungs- und Standardisierungsbedarf ... 95
3.3 Verbrauchsbezogene Maßnahmen ... 98
3.4 Empfehlungen für die Software-Entwicklungsgemeinschaft ... 99
3.5 Aus- und Weiterbildung ... 100
4 Anwendung der Ergebnisse ... 101
4.1 Umweltwirkung von Webseiten ermitteln ... 101
4.1.1 Messung von CMS-basierten Webseiten ... 102
4.1.2 Messung von statisch generierten Webseiten ... 102
4.2 Messung von Anwendungen auf Cloud-Infrastruktur ... 103
4.3 Kompilierte Anwendungen & Batch Prozesse ... 103

4.4 Open-Source Komponenten....104
4.5 KI-Modelle – Training & Inferenz....104
5 Kommunikation der Forschungsergebnisse....105
5.1 Begleitformate....106
5.2 Kommunikationswerkzeuge....107
5.3 Wiki für Green Coding Wissen & Praktiken....107
6 Quellenverzeichnis....108
A Kommunikationsunterlagen....111
A.1 Logo....111
A.2 One-Pager....112
A.3 Internetauftritt....114
A.4 Projekt-Flyer....116
B Protokolle und Ergebnisse der Veranstaltungen....119
B.1 Erstes Begleitkreistreffen....119
B.2 Zweites Begleitkreistreffen....124
B.3 Experten Workshop....129
B.4 Stakeholder Analyse....134
B.5 Hackathon 1....134
B.6 Hackathon 2....135
C Ergebnisse....137
C.1 Leitfaden für Energieeffiziente Software-Entwicklung....137
C.2 Auswahl an Untersuchungsgegenständen....146
C.3 Umrechnungsformel Digital Ressource nach Energie....154
C.4 Ausgabe des Messskriptes get_metrics.py (Beispiel)....154
C.5 EEA-Tabelle zu den THG-Emissionen....156
D Messaufbau....158
D.1 Erfahrungen mit der Durchführung der Messung....158
D.2 Messaufbau....158
D.3 Vorbereitende Schritte zur Messung....159
D.4 Ausführung der zu messenden Software....160
D.5 Interpretation der Messergebnisse....162
D.6 Detaillierte Spezifikation des Testsystems....163
D.7 Mess-Skript aus dem Test-Labor....166
D.8 Docker Compose Instruktionen für den Aufbau des Testlabors....177

Abbildungsverzeichnis

Abbildung 1: Übersicht der untersuchten und entwickelten Methoden hinsichtlich Komplexität und Genauigkeit (schematische Darstellung) 18
Abbildung 2: Vergleich der Methoden in Bezug auf unterstützte Laufzeitumgebungen 21
Abbildung 3: Beispiel des Labels mit einem Energieverbrauch von 723 Ws .23
Abbildung 4: Übersicht Software-Herstellungsprozess und Qualitätssicherung 24
Abbildung 5: Übersicht der Methoden und ihrer Anwendbarkeit im Software-Lebenszyklus 26
Figure 6: Overview of the analysed and developed methods based on complexity and accuracy. 30
Figure 7: Comparison of methods in relation to supported runtime environments 33
Figure 8: Example of the label with an energy use of 723 Ws 35
Figure 9: Overview of the software development process and quality assurance 36
Figure 10: Overview of the methods and their applicability in the software life cycle 38
Abbildung 11: Übersicht des digitalen Ökosystems mit digitaler Infrastruktur als Grundlage 41
Abbildung 12: Schematische Darstellung der Rohstoffverbräuche und Umweltwirkungen von digitaler Infrastruktur 42
Abbildung 13: Darstellung der Ressourcenanforderung von Windows-Versionen zwischen Windows 95 und 10 43
Abbildung 14: Grafische Darstellung des Mooreschen Gesetzes 44
Abbildung 15: Übersicht Bedarf an Rechenleistung von KI-Modellen 46
Abbildung 16: Beispielrechnung Drupal CMS 48
Abbildung 17: Beispiel-Abbildung von Testabdeckung und Zustand 59
Abbildung 18: Beispiel eines IT Monitoring Systems 62
Abbildung 19: Beispiel Abfrage der Boavizta API 63
Abbildung 20: Ergebnis einer Beispielabfrage an die Boavizta API 64
Abbildung 21: Vergleich der Formeln mit dem Spec Power Modell 66
Abbildung 22: Strommessung Testlabor unter Voll-Last 67
Abbildung 23: Architektur und Aufbau des Testlabors 70
Abbildung 24: Messoberfläche im Testlabor 72
Abbildung 25: Instruktionsdatei für Test-Subjekte im Labor 73
Abbildung 26: Python-Code zur stufenweisen Auslastung der CPU-Kerne 76
Abbildung 27: CPU-Auslastung bei steigendem "CPUStress" von 1 bis 60 Kernen 77
Abbildung 28: Energieverbrauch der CPU bei steigendem "CPUStress" von 1 bis 60 Kernen 78

Abbildung 29: Zusammenhang von Messdauer und Leistungsaufnahme - thermische Effekte 82
Abbildung 30: Rekursive Berechnung der Fibonacci-Zahl mit der Programmiersprache *Python*....83
Abbildung 31: Relative Standardabweichung der Mittelwerte der Emissionen als Funktion der Länge der Messreihe 84
Abbildung 32: Drift der Leistungsaufnahme....85
Abbildung 33: Treibhauspotenzial zur Berechnung der Fibonacci-Zahl Fib (47) mit unterschiedlichen Programmiersprachen....88
Abbildung 34: Korrelation zwischen Emissionen und Größe von Bibliotheken 89
Abbildung 35: Beispiele für Kennzeichnungen bei GitHub 93
Abbildung 36: Beispiel des Labels mit einem Energieverbrauch von 723 Ws .93
Abbildung 37: Logo Standard Variante....111
Abbildung 38: Logo Invert Variante....111
Abbildung 39: Logo Farb-Variante als Beispiel für Anpassung für Kennzeichnung 112
Abbildung 40: Zusammenfassung des Vorhabens in Englisch 113
Abbildung 41: Webseite des Vorhabens beim Umweltbundesamt....114
Abbildung 42: Webseite des Vorhabens auf dem Internetauftritt der SDIA .115
Abbildung 43: Der SDIA Knowledge Hub mit gesondertem Bereich für SoftAWERE 116
Abbildung 44: Flyer Vorderseite....117
Abbildung 45: Flyer Rückseite 118
Abbildung 46: Übersicht Präsentation 1. Begleitkreis 120
Abbildung 47: Diskussionsergebnisse - 1. Fragestellung 121
Abbildung 48: Diskussionsergebnisse - 2. Fragestellung 122
Abbildung 49: Diskussionsergebnisse - 3. Fragestellung 123
Abbildung 50: Auszug Präsentation – 2. Begleitkreistreffen 125
Abbildung 51: Auszug Präsentation – 2. Begleitkreistreffen 125
Abbildung 52: Auszug Präsentation – 2. Begleitkreistreffen 126
Abbildung 53: Auszug Präsentation – 2. Begleitkreistreffen 126
Abbildung 54: Impuls #1 & Kommentare der Begleitkreismitglieder....127
Abbildung 55: Impuls #2 & Kommentare der Begleitkreismitglieder....127
Abbildung 56: Impuls #3 & Kommentare der Begleitkreismitglieder....128
Abbildung 57: Impuls #4 & Kommentare der Begleitkreismitglieder....129
Abbildung 58: Ergebnisse Arbeitsgruppe „Methodology“ 131
Abbildung 59: Ergebnisse Arbeitsgruppe „Application“ 132
Abbildung 60: Ergebnisse Arbeitsgruppe „Outlook“ 133
Abbildung 61: Test-Skript faculty.py....158
Abbildung 62: Schematischer Aufbau des Messlabors....159
Abbildung 63: Instruktionen der Messung „.gitlab-ci.yml“ 161
Abbildung 64: Inhalt von „requirements-metrics.txt“....161

Abbildung 65: Umrechnungsfaktoren von Stromverbrauch in ökologische Kennzahlen 162
Abbildung 66: Ergebnisausgabe in der CI/CD Konsole 163
Abbildung 67: Detaillierte CPU-Spezifikationen 164
Abbildung 68: Rechenzentrum des Testlabors 165
Abbildung 69: Python Skript für die Sammlung der Messergebnisse im Test-Labor 166
Abbildung 70: docker-compose.yml Datei 177

Tabellenverzeichnis

Tabelle 1: Übersicht der ausgewählten Umweltindikatoren 21
Table 2: Overview of selected environmental factors 33
Tabelle 3: Windows Versionen - Systemanforderungen und Programmcode...42
Tabelle 4: Messreihe Mehrfachaufruf von faculty.py...............................79
Tabelle 5: Ausgewählte Bibliotheken..80
Tabelle 6: Gemessene Mittelwerte für verschiedene Bibliotheken..........81
Tabelle 7: Treibhauspotenzial zur Berechnung der Fibonacci-Zahl Fib (47) mit unterschiedlichen Programmiersprachen..........................87
Tabelle 8: Agenda 119
Tabelle 9: Übersicht Untersuchungsgestände – Python 148
Tabelle 10: Übersicht Untersuchungsgestände – Java..............................149
Tabelle 11: Übersicht Untersuchungsgestände – C 150
Tabelle 12: Übersicht Untersuchungsgestände – C++ 151
Tabelle 13: Übersicht Untersuchungsgestände – JavaScript 152
Tabelle 14: Übersicht Untersuchungsgestände – Go Lang 153

Abkürzungsverzeichnis

AG	Auftraggeber, Umweltbundesamt
AN	Auftragnehmer, SDIA und Öko Institut
API	Application Programming Interface
BMC	Baseboard Management Controller
BMWK	Bundesministerium für Wirtschaft und Klimaschutz
CD	Continuous Delivery
CDN	Content Delivery Network
CI	Continuous Integration
CMS	Content Management System
CMS	Content Management System
CPU	Core Processing Unit, Prozessor
CSS	Cascading Style Sheets
EEA	Europäische Energie Agentur
FOSS	Free & Open-Source Software (kostenlose und quelloffene Software)
FOSS	Free and Open-Source Software, zu deutsch: freie und quelloffene Software
GPU	Graphics Processing Unit
GUI	Graphical User Interface, dt. grafische Benutzeroberfläche
HDD	Hard Disk Drive
HTML	Hypertext Markup Language
IDE	Integrated Development Environment, dt. Integrierte Entwicklungsumgebung
IPMI	Intelligent Platform Management Interface
JSON	JavaScript Object Notation
JSON	JavaScript Object Notation
KVM	Kernel-based Virtual Machine (KVM) ist eine quelloffene Virtualisierungstechnologie, die das Betriebssystem Linux bereitstellt
kWh	Kilowattstunde
NIC	Network Interface Connector
OCP	Open Compute Project der Open Compute Foundation
POP	Point of Presence
PUE	Power Usage Effectiveness
RAM	Random Access Memory (‚Arbeitsspeicher‘)
RAPL	Running Average Power Limit
S12Y	Von I18N. „S, 12 Buchstaben und Y“ steht für Sustainability.
SDIA	Sustainable Digital Infrastructure Alliance e.V.
SSD	Solid State Drive
TDP	Thermal Design Power
THG	Treibhausgasemissionen

TWh	Terrawattstunden (Strom)
UBA	Umweltbundesamt, Dessau

Zusammenfassung

Der Satz "Software is eating the world" (deutsch: "Software frisst die Welt") wurde von Marc Andreessen geprägt, einem bekannten Silicon Valley-Investor und Mitbegründer von Netscape (Andreessen 2011). Er drückt die zunehmende Durchdringung und Einflussnahme von Software in nahezu allen Lebensbereichen aus.

Die Aussage bedeutet, dass Softwareanwendungen und digitale Produkte in immer mehr Branchen und Aspekten des täglichen Lebens eine zentrale Rolle spielen. Im heutigen Kontext der rasant ansteigenden Digitalisierung bedeutet dieser Satz aber auch, dass Software für zunehmend große Ressourcen- und Energieverbräuche verantwortlich ist (siehe Kapitel 1).

Die Effizienzpotenziale bezogen auf das einzelne Softwareprodukt scheinen oftmals gering. Durch die enorme Verbreitung, die große Skalierungsfähigkeit von Software und Häufigkeit der Nutzung der Software lohnen sich allerdings auch kleinste Einsparpotenziale (siehe Seite 49, „Beispielrechnung Drupal CMS“). Im Digitalsektor und in den öffentlichen Debatten hat sich das Bewusstsein, dass nicht nur physische Hardware, sondern auch immaterielle Software für Umweltwirkungen verantwortlich ist, noch nicht hinreichend durchgesetzt. Ein Grund dafür ist, dass für die Messung dieser Umweltwirkungen noch keine validierten Mess-Methodiken und Werkzeuge existieren, die im Entwicklungsprozess integriert und etabliert sind.

Für eine Software-Anwendung sind nutzbare Ressourcen Arbeitsspeicherkapazität, CPU-Zeit, Langzeitspeicherkapazität und Netzwerkübertragungskapazität. Diese Ressourcen werden von der Software-Anwendung verbraucht, um Funktionen bereitzustellen. Programmierung, Programmiersprache und Funktionsumfang der Anwendung bestimmen den Ressourcenverbrauch. Im Rahmen des SoftAWERE Vorhabens sind diese Ressourcen, die von Software-Anwendungen verbraucht werden, als digitale Ressourcen definiert.

Digitale Ressourcen werden, im Modell von SoftAWERE, durch IT-Hardware - Computer und Netzwerk-Equipment – bereitgestellt. Die Bereitstellung der digitalen Ressourcen verbraucht physische Ressourcen, Strom für den Betrieb der IT-Hardware und Rohstoffe und Energie für die Herstellung der Geräte.

Mit diesem theoretischen Modell können die folgenden Handlungsfelder zur Reduktion der Umweltwirkungen untersucht werden:

1. Den Effizienzgrad mit der eine Software-Anwendung digitale Ressourcen verbraucht („Verbrauch“), mit dem Ziel einen gleichbleibendem Funktionsumfang mit weniger digitalen Ressourcen zu bereitzustellen, zum Beispiel eine Video-Konferenz-Anwendung, bei gleichbleibender Qualität.
2. Die Menge von digitalen Ressourcen, die von einer Software-Anwendung für den Betrieb benötigt werden („Reservierung“): Benötigt eine Anwendung z.B. bei jeder neuen Version mehr digitale Ressourcen, das heißt höhere Hardware-Anforderungen für den Betrieb, so wird eine zunehmend leistungsfähigere IT-Hardware benötigt, um die Ressourcen bereitzustellen.
3. Der Nutzungsgrad der IT-Hardware („Auslastung“), welcher durch die Software-Anwendung und die Art und Weise wie diese digitalen Ressourcen angefordert und verbraucht werden, beeinflusst wird. Ein hoher Nutzungsgrad bedeutet, dass die IT-Hardware so kontinuierlich wie möglich bei einer optimalen Auslastung genutzt wird und so wenig Leerlauf wie möglich entsteht.

Mit SoftAWERE sind Methoden („Light“ und „Full“) geschaffen worden, die den Zusammenhang zwischen dem digitalen Ressourcenverbrauch von Software-Anwendungen und der damit verbundenen Umweltwirkung, herstellen. So wird für jede digitale Ressource, z.B. 1 GB Arbeitsspeicher, die Umweltwirkung aus der Herstellung und dem Betrieb berechnet und für Software-Entwickelnde sichtbar gemacht. Die Transparenz der Umweltwirkung der digitalen Ressourcen, die für die Anwendung bereitgestellt werden müssen, schafft einen Anreiz und eine Entscheidungshilfe für Software-Entwickelnde, um die beschriebenen Maßnahmen zu evaluieren und zu implementieren.

Das Umweltbundesamt hat mit der Einführung des Blauen Engels erstmalig auf die Umweltrelevanz von Software aufmerksam gemacht. Durch die Einführung des staatlichen Umweltsiegels wurden der Energieverbrauch und die Ressourceninanspruchnahmen zum ersten Mal sichtbar gemacht und Eigenschaften identifiziert, die eine wesentliche Rolle für eine gute Umweltperformance von Software spielen. Unter Ressourceninanspruchnahmen von Software wird verstanden, dass Software auf die Hardware zugreift und während der Laufzeit Energieverbräuche erzeugt. Die Hardware besteht aus einer Vielzahl von Rohstoffen, beispielsweise seltene Erden. Die Herstellung der Hardware-Komponenten ist zudem sehr energieintensiv und benötigt hochreine Chemikalien, die ein hohes Treibhausgaspotential haben. Je größer die Inanspruchnahme der Software auf der Hardware ist, desto leistungsstärkere Hardware wird gebraucht, was dazu führt, dass Hardware erneuert oder erweitert werden muss.

Im vorliegenden Vorhaben wurde das SoftAWERE-Werkzeug (im Folgenden „SoftAWERE“) entwickelt, das Software-Entwickelnden während des Entwicklungsprozesses Rückmeldungen zu den Energieverbräuchen und Umweltwirkungen gibt, die durch diese Software verursacht werden. Das Werkzeug lässt sich als digitale Messumgebung für die Energieverbräuche und Umweltwirkungen von Software beschreiben.

Es ermöglicht:

- die Umweltbelastungen während des Entwicklungsprozesses von neuer und überarbeiteter Software zu erkennen sowie Veränderungen des Energieverbrauchs und der Hardwareinanspruchnahmen zu überprüfen,
- den Prozess der Messung von Energieverbrauch und Umweltwirkung zu automatisieren und damit zu einem bedienerfreundlichen Bestandteil des Arbeitsprozesses während der Softwareentwicklung zu machen.

SoftAWERE und SoftAWERE Light eignen sich nicht nur für Laufzeitumgebungen, die auf einer lokalen Instanz betrieben werden, z.B. auf dem Endgerät des Software-Entwickelnden.

SoftAWERE Light wurde primär für den Einsatz in Server-Umgebungen entwickelt, in denen keine Mess-Schnittstellen vom Betriebssystem oder der zugrundeliegenden Hardware zur Verfügung stehen. Diese erweiterte Version von SoftAWERE eignet sich für den Einsatz auf dedizierten und privaten Server-Umgebungen (z.B. in gesicherten Unternehmensnetzwerken).

Beschreibung der Methode

Methodisch setzt das Vorhaben auf den bisherigen Forschungsarbeiten des Umweltbundesamtes auf. Der methodische Ansatz für die Anwendung von standardisierten Lebenszyklusanalyse-Methoden (LCA) zur Messung der Energie- und Hardwareflüsse von Software und die ganzheitliche Betrachtung von Umweltwirkungen wurde vom Umweltcampus Birkenfeld und dem Öko-Institut erarbeitet (Kern et al. 2018) und bildet die Grundlage für SoftAWERE. Bei

dieser Methode wurde ein physisches Messlabor aufgebaut, mit dem die Energie- und Hardwareinanspruchnahmen gemessen werden können (im Folgenden „OSCAR-Methode“).

In der Kriterienentwicklung für das Umweltsiegel „Blauer Engel“ wurde als Forschungsmethode ein Standardnutzungsszenario entwickelt (Hilty et al. 2015) um Software in einer reproduzierbaren Art und Weise hinsichtlich ihrer Energieverbräuche der darunterliegenden Hardware messen zu können. Das Standardnutzungsszenario beschreibt dabei einen sich wiederholenden Ablauf an automatisierten Nutzerinteraktionen mit der Software und simuliert dabei ein solches typisches Nutzungsverhalten mit der Software. Diese Vorgehensweise wurde in SoftAWERE weiterentwickelt (siehe Kapitel 2.2).

Statt der Entwicklung der Nutzungsszenarien, die einer typischen Nutzung der Software entspricht, werden bestehende automatisierte Integrationstests als Nutzungsszenarien verwendet, um die Messung der Software zu ermöglichen. Integrationstests beschreiben in der Softwareentwicklung wiederholende Tests, die die Interaktion von einzelnen Komponenten prüfen. Es ist gängige Praxis, die Qualität von Software-Projekten durch die Implementierung von Integrationstests zu überprüfen und Fehlern in der Programmierung vorzubeugen. Die Integrationstests simulieren die Nutzung der Software und liefern ein reproduzierbares Messergebnis. Dabei wird unterstellt, dass der Integrationstest alle relevanten Funktionen der Software beinhaltet (es besteht eine hohe Testabdeckung). Dieses Vorgehen verringert den Arbeitsaufwand für viele Software-Projekte, insbesondere wenn diese bereits über eine ausreichende Testabdeckung verfügen (z.B. 80% der Funktionen über die Tests angesprochen und ausgeführt werden). Es schränkt jedoch auch gleichzeitig die Vergleichbarkeit ein, da die Nutzungsszenarien bzw. Integrationstests von Software-Anwendungen nicht identisch sind.

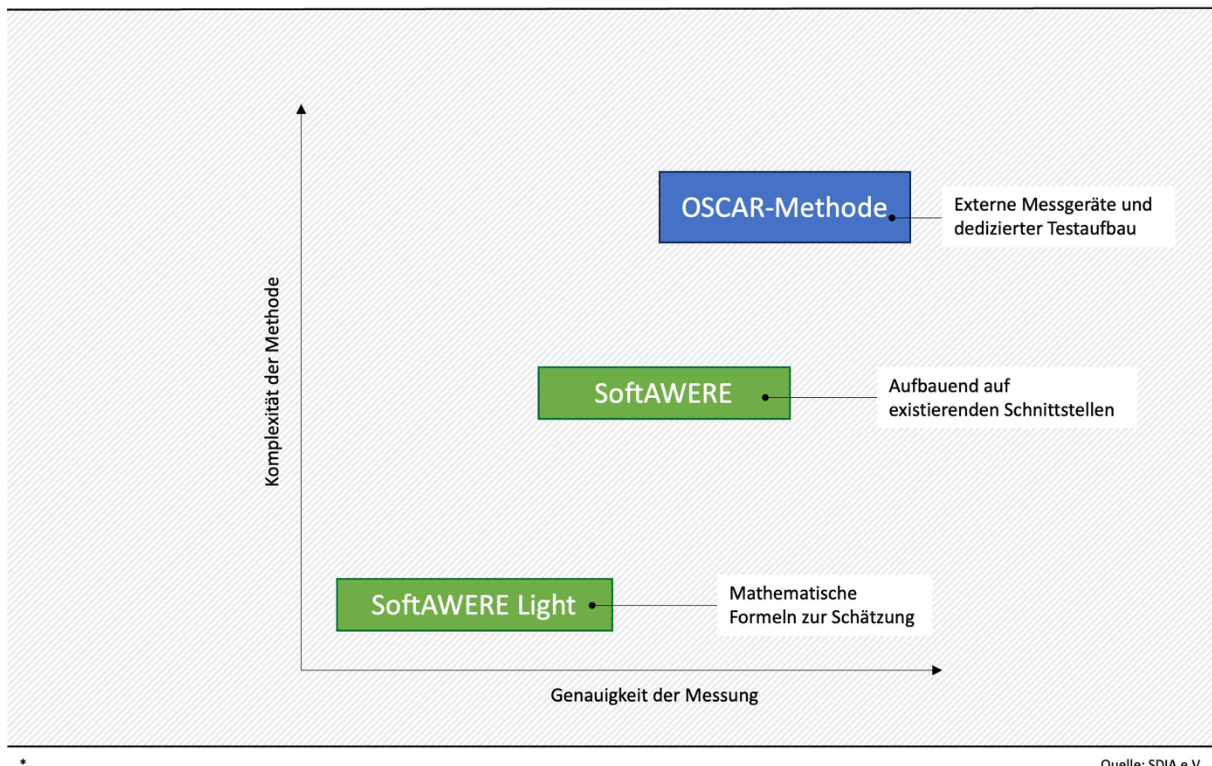
Eine Besonderheit von SoftAWERE ist der Verzicht auf externe Strommessgeräte und besondere Anforderungen an die Technik und den Messaufbau. Ziel des Forschungsteams war es, eine hohe Anwendbarkeit im Software-Markt sicherzustellen, weshalb sich der Messstand auch auf gängiger Server-Hardware ohne den Einsatz von Spezialequipment und Zugang zu Rechenzentren realisieren lässt. Somit ermöglicht das Vorhaben Unternehmen und Software-Entwickelnden SoftAWERE in ihren eigenen Umgebungen auszuführen (siehe Anhang D.8). Zu beachten ist, dass das Vorhaben sich auf Linux-basierte Laufzeit-Umgebungen beschränkt. Linux ist das weltweit-gängigste Server-Betriebssystem (W3Techs, n.d.).

Im Rahmen des Vorhabens wurden zwei Varianten von SoftAWERE entwickelt.

1. „SoftAWERE-Light“: Eine vereinfachte Variante, die ohne Messungen auskommt und mit mathematischen Formeln eine Schätzung des Energieverbrauchs und der Umweltwirkungen errechnet. Diese Variante eignet sich für Cloud- und virtualisierte Umgebungen, in denen eine physische Messung mit Messgeräten oder eine Messung über Server-Schnittstellen nicht möglich ist.
2. „SoftAWERE“: Die vollständige Variante, welche ohne physische Messgeräte auskommt und auf den bestehenden Schnittstellen von Server-Herstellern und Betriebssystemen aufbaut.

Die folgende Abbildung 1 zeigt, wie die Methoden entsprechend ihrer Komplexität und Genauigkeit eingeschätzt werden können.

Abbildung 1: Übersicht der untersuchten und entwickelten Methoden hinsichtlich Komplexität und Genauigkeit (schematische Darstellung)



Quelle: SDIA e.V.

Beide Methoden wurden mit Hilfe des Testaufbaus verifiziert (siehe Kapitel 2.2, und Kapitel 2.3). SoftAWERE ist für Server-basierte Anwendungen entwickelt worden und beschränkt sich auf die Kalkulation der Rechenleistung und Speichernutzung. Die Erfassung der Umweltwirkungen vom Netzwerkverkehr wurde aufgrund der mangelnden Datenlage ausgenommen.

SoftAWERE-Light – mathematische Schätzung

In virtualisierten oder containerisierten Laufzeitumgebungen ist der Zugriff auf Schnittstellen eingeschränkt, die für die Messung des Energieverbrauchs nötig sind. Zudem lassen sich oft der Server-Hersteller und Modell-Kennzeichnung nicht ermitteln. Um in diesen Umgebungen dennoch Informationen zu Umweltwirkungen und dem Energieverbrauch der Software zu ermitteln, wurde zuerst ein Annäherungsansatz entwickelt, der auf Informationen, die vom Betriebssystem, der Firmware des Computers, aus dem Rechenzentrum oder von den Software-Entwickelnden selbst bereitgestellt werden, aufbaut und je nach Verfügbarkeit der Informationen eine höhere oder geringere Genauigkeit ergibt:

- Die Auslastung der Systemkomponenten (CPU-Last, Arbeitsspeicher-Nutzung, Festplatten-Nutzung, Netzwerkauslastung) – im Folgenden „digitale Ressourcen“ [1]

[1] Der Begriff wird auch im Glossar der SDIA definiert: siehe https://knowledge.sdialliance.org/glossary/digital-resource-primitives, abgerufen am 21.10.2023

- CPU-Typ des Servers (für den ein TDP-Wert[2] verfügbar sein muss)[3]
- PUE des zugrundeliegenden Rechenzentrums. Falls nicht verfügbar, wird ein Durchschnittswert für europäische Rechenzentren genutzt – PUE 1.8 (Bertoldi 2015).
- Physischer Standort des Rechenzentrums oder des Computers in/auf dem die Software ausgeführt wird (für die Ermittlung der THG-Emissionen aus dem Stromnetz).

Sind diese Informationen verfügbar, so lässt sich mit einer Formel der Energieaufwand pro Auslastungsstufe des Servers bestimmen. Diese Formel wurde im Rahmen des Vorhabens (siehe Kapitel 2.2) und durch die Software-Community mehrfach weiterentwickelt und verbessert (siehe Kapitel 2.2, Abbildung 21 für eine grafische Darstellung der Varianten) sowie in wissenschaftlichen Veröffentlichung publiziert (Kennes 2023).

SoftAWERE - Messung ohne Mess-Equipment

Die Methodik SoftAWERE setzt auf bestehende technische Grundlagen auf:

- IPMI & RAPL Schnittstellen: Um auf den Einsatz von externen Messinstrumenten zu verzichten, setzt SoftAWERE in gängigen Server-Modellen auf den verfügbaren Standard-Schnittstellen zur Strommessung auf.
- GitLab CI/CD Runner & Docker: Für die Orchestrierung und Isolierung der Messläufe wurde auf die CI/CD Pipelines von GitLab zurückgegriffen. Die Isolierung der Messungen vom Hauptsystem geschieht mithilfe von Docker Containern. (siehe Kapitel 2.2 und Anhang D.8)
- Docker Compose: Das Messlabor selbst besteht aus Open-Source Komponenten, die auf fast (nur Linuxbasierte Server) beliebiger Server-Hardware gestartet werden kann. Die Orchestrierung der Messstand-Komponenten geschieht mithilfe von Docker Compose.
- Prometheus, Thanos und Grafana: Für die Messung und visuelle Auswertung der Ergebnisse wurden bestehende Open-Source-Telemetrie Systeme[4] eingesetzt. Für die Langzeitspeicherung der aufgezeichneten Zeitreihen wird die Datenbank Thanos in einem skalierbaren Kubernetes Cluster verwendet.
- Weitere Nutzung von verschiedenen APIs: Scaphandre, FreeIPMI, Boavizta API, und Boagent: Für den Zugriff auf die IPMI und RAPL-Schnittstellen kommen zwei Bibliotheken zum Einsatz, FreeIPMI und Scaphandre. Für die Ermittlung der Umweltwirkung kommt die Boavizta API zum Einsatz (welche auf der Forschung des Umweltbundesamts aufbaut). Der Boagent wird für die Ermittlung der Hardware-Eigenschaften verwendet (Speicherplatz, CPU-Anzahl, Arbeitsspeicher, etc.).

Energiesparfunktionen von Server-Hardware und CPU: Durch die Aktivierung dieser Funktion sinkt der Energieverbrauch bei niedriger Auslastung erheblich. Hierdurch ist es möglich, die Auswirkung der erzeugten Rechenlast auf den Energieverbrauch der untersuchten Software zu

2 Der Thermal Design Power (TDP)-Wert einer CPU gibt die maximale Wärmemenge, gemessen in Watt (W), an, die der Prozessor bei typischer Arbeitsbelastung abgeben kann. Er ist eine wichtige Kennzahl für Systementwickler*innen und Nutzer*innen, um zu verstehen, wie viel Wärmemanagement erforderlich ist, damit die CPU innerhalb ihres sicheren Betriebstemperaturbereichs bleibt.

3 Für die gängigen Intel Server-Chips sind diese auf der Intel Webseite abrufbar: https://ark.intel.com/content/www/de/de/ark/products/series/125191/intel-xeon-scalable-processors.html

4 Telemetrie-Systeme werden für die Überwachung von Software-Anwendungen eingesetzt. Sie zeichnen unter anderem den Verbrauch von Rechenleistung, Speicherverbrauch und Netzwerkverkehr auf. Die Systeme werden genutzt, um die Auslastung von Infrastruktur und Performance von Software-Anwendungen im laufenden Betrieb zu beobachten. Im SoftAWERE-Kontext versteht man unter Telemetriedaten die Übertragung von Messwerten, die an einem Ort (Server) gemessen werden und an einen anderen Messort (...) übertragen werden.

ermitteln. SoftAWERE kann in der Vollversion in GitLab-basierten Entwicklungsumgebungen eingesetzt werden, die auf dedizierten und virtualisierten Systemen laufen. Voraussetzung ist die Verfügbarkeit von Telemetrie-Daten, die über IPMI bzw. RAPL erfasst werden.

Hintergrundinformationen zu IPMI und RAPL

Die IPMI und RAPL-Schnittstellen sind in virtualisierten und Cloud-Umgebungen nicht verfügbar. In diesen Umgebungen ist auch eine Messung mit externen Strommessgeräten nicht möglich, da die Rechenzentren nicht zugänglich sind und die Infrastruktur von vielen Kunden geteilt wird. Daher hat das Vorhaben auch eine mathematische Vorgehensweise entwickelt („SoftAWERE-Light"), welche eine Schätzung des Energieverbrauchs ermöglicht (siehe Kapitel 2.2).

Mit SoftAWERE lassen sich Anwendungen in verschiedenen Programmiersprachen testen. Es können zudem Nutzungsszenarien programmiert werden, um bestimmte Sachverhalte z.B. mit einer Beispiel-Software-Anwendung zu überprüfen. Ein Beispiel dafür ist die Berechnung von Fibonacci-Zahlen, um die Geschwindigkeit von Programmiersprachen zu ermitteln (siehe Kapitel 2.4).

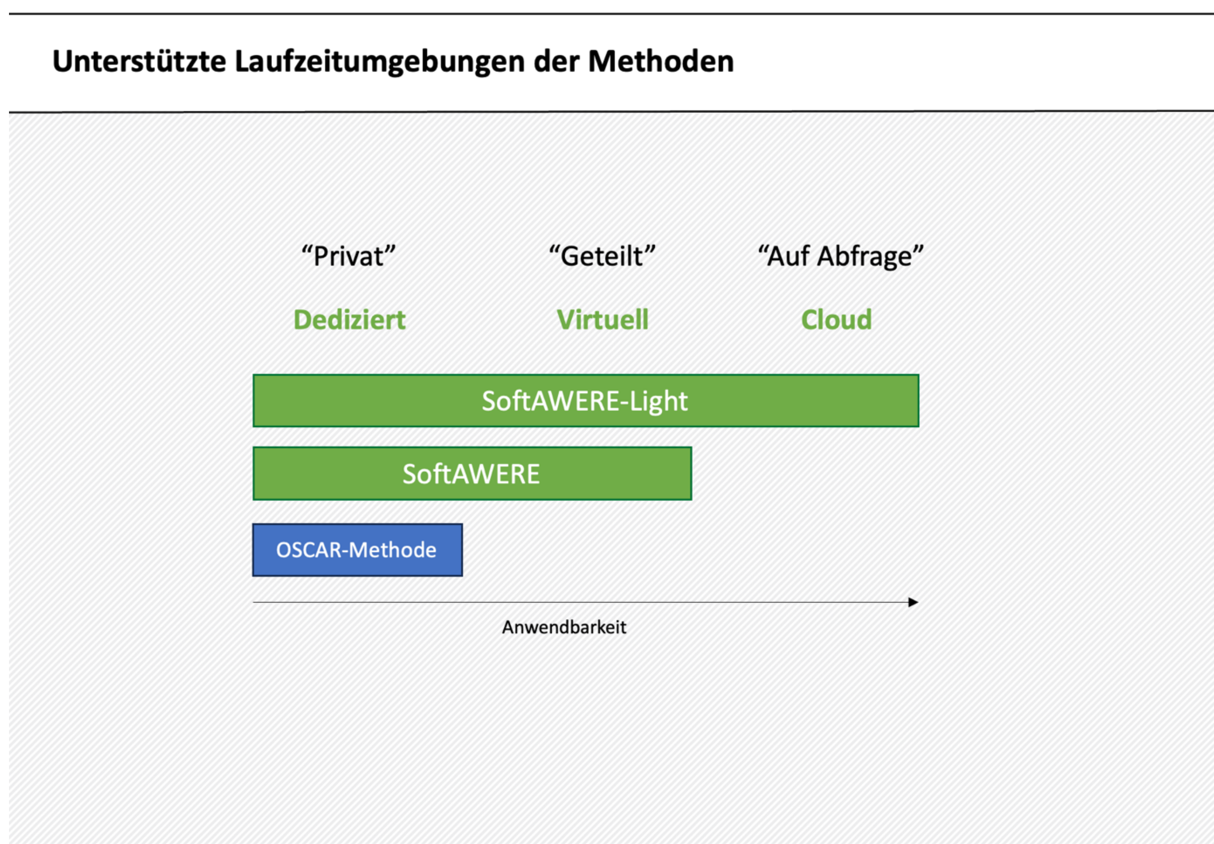
SoftAWERE ist besonders für Anwendungen geeignet, die kompiliert werden. Einige Anwendungsfälle werden im Folgenden aufgeführt (siehe auch Kapitel 4):

- das Training oder Feintuning von KI-Modellen (Daten werden in einem maschinellen Lernprozess in ein Modell eingearbeitet),
- Daten-Analyse-Software oder Anwendungsfälle, in denen eine Software-Anwendung oder ein anderes Softwarewerkzeug Daten verarbeitet, sammelt, formatiert oder in andere Operationen ausführt, die Zeit und Rechenleistung in Anspruch nehmen,
- Statische Webseiten, die vor der Auslieferung kompiliert[5] werden (siehe Beispiel der SDIA-Website im Kapitel 4.1.2),
- Klassische Unternehmensanwendungen, die entweder eine große Nutzungstestabdeckung haben oder vor der Auslieferung kompiliert werden (z. B. Java, C#-Anwendungen) und
- Open-Source Werkzeuge und Bibliotheken, die über Integrationstests verfügen (eine Liste, der im Vorhaben untersuchten Open-Source Projekte, siehe Anhang C.2).

Abbildung 2 stellt die beiden Methoden und ihre Anwendbarkeit in verschiedenen Laufzeitumgebungen dar.

[5] Werden auch als „statische Webseiten-Generatoren" bezeichnet („Static site generators – SSGs"). Beispiele sind Jekyll, Next.js, Hugo und Gatsby. Siehe auch https://en.wikipedia.org/wiki/Static_site_generator (abgerufen am, 09.01.2024)

Abbildung 2: Vergleich der Methoden in Bezug auf unterstützte Laufzeitumgebungen



Quelle: SDIA e.V.

Energieverbrauch und Umweltwirkungen

Beide SoftAWERE-Varianten machen sowohl den Energieverbrauch als auch die Umweltwirkungen für Software-Anwendungen messbar (siehe Kapitel 2.3). Insgesamt werden die in Tabelle 1 dargestellten Indikatoren mit SoftAWERE für Software-Entwickelnde messbar gemacht:

Tabelle 1: Übersicht der ausgewählten Umweltindikatoren

Umweltwirkungsindikator	Beschreibung	SoftAWERE-Variante
Stromverbrauch	Der Stromverbrauch, der während der Ausführung der Software im Messstand entstanden ist.	SoftAWERE, SoftAWERE-Light
Operative THG-Emissionen	Emissionen, die durch die nachgelagerte Infrastruktur für die Stromerzeugung und durch den Verbrauch von Energieträgern entstehen.	SoftAWERE, SoftAWERE-Light
THG-Emissionen der Herstellung	Eine Schätzung der Emissionen, die durch die Herstellung des IKT-Equipments entstanden sind, die für die Ausführung der Software notwendig ist.	SoftAWERE

Umweltwirkungsindikator	Beschreibung	SoftAWERE-Variante
Abbau abiotischer Ressourcen[6] bei der Herstellung	Schätzung der Menge an abiotischen Ressourcen, die in dem IKT-Equipment eingebettet sind, welches für die Ausführung der Software notwendig ist.	SoftAWERE

Dem Forschungsteam war es besonders wichtig, nicht nur den Energieverbrauch zu messen und darzustellen, sondern auch eine Schätzung der Herstellungs- und Ressourcenaufwände der IT-Hardware (bspw. Server) darzustellen, welche für die Ausführung der Software notwendig sind.

In vielen Fällen führt der Fokus auf Energieeffizienz zu frühem Austausch von Servern, ohne den damit verbundenen Ressourcenaufwand oder THG-Emissionen aus der Herstellung und dem Transport gegenüberzustellen (Prakash et al. 2012). Aufgrund der mangelnden Transparenz und öffentlichen, verifizierten Daten von IKT-Herstellern und Rechenzentrumsbetreibern beschränkt sich die Berechnung innerhalb von SoftAWERE auf eine Schätzung[7].

Anwendung von SoftAWERE

SoftAWERE kann von Unternehmen, Organisationen und Individuen auf eigener Infrastruktur als digitaler Messstand von Software eingesetzt werden. Dafür notwendig sind lediglich Server-Umgebungen, die über die entsprechenden IPMI und RAPL-Schnittstellen verfügen. Für den eigenen Aufbau von SoftAWERE ist im Anhang D.8 eine vollständige Anleitung[8] hinterlegt.

Die SoftAWERE-Light Variante lässt sich zusätzlich auch in Umgebungen nutzen, die nicht über die notwendigen Schnittstellen verfügen, wie z.B. Cloud Umgebungen, solange Telemetrie-Daten zur Auslastung der CPU, des Arbeitsspeichers und des Permanentspeicherplatzes verfügbar sind. Die entsprechenden Formeln sind in Kapitel 2.3 dokumentiert. Durch die mathematische Herangehensweise eignet sich SoftAWERE-Light für den Einsatz in einem sehr breiten Spektrum von Anwendungen, insbesondere zur Laufzeit der Anwendungen (im Betrieb):

- Cloud-basierte Anwendungen, die auf einer oder mehreren virtuellen Maschinen oder Cloud-Instanzen laufen. Mithilfe der Formeln lassen sich digitale Ressourcenverbräuche (CPU-Auslastung, Arbeitsspeicher, Speicherauslastung) in Energieverbrauch umrechnen.
- Anwendungen in privaten virtualisierten Umgebungen (z. B. Xen, OpenStack, VMware), in denen ein Zugriff auf die zugrundeliegenden Server-Systeme und das Rechenzentrum aus Sicherheits- oder technischen Gründen nicht möglich ist.
- Desktop-Anwendungen, bei denen ein Zugriff auf die zugrundeliegenden Energie-Messschnittstellen nicht möglich ist (z. B. aufgrund von Sicherheitseinstellungen oder älteren Betriebssystemen, die noch nicht über die notwendigen Schnittstellen verfügen).

Die SoftAWERE-Methodik eignet sich insbesondere für die Anwendung im Software-Entwicklungsprozess und Anwendungsfälle auf dedizierter Hardware:

[6] Der Begriff "eingebettete abiotische Ressourcenerschöpfung" bezieht sich auf die Erschöpfung nicht lebender (abiotischer) natürlicher Ressourcen wie Mineralien und fossile Brennstoffe, die während der Produktion und des Lebenszyklus eines Produkts verbraucht oder beeinträchtigt werden. Dieses Konzept verdeutlicht die versteckten Umweltkosten, die in Waren und Dienstleistungen enthalten sind, indem es den Abbau und die schwindende Verfügbarkeit dieser wichtigen, nicht erneuerbaren Ressourcen berücksichtigt. Es ist ein wichtiger Aspekt in der Nachhaltigkeitsforschung und unterstreicht die Notwendigkeit eines verantwortungsvollen Ressourcenmanagements und der Entwicklung nachhaltiger Alternativen.

[7] Die Schätzung baut auf bestehender Forschung des Umweltbundesamts auf (Gröger et al. 2021).

[8] Eine tagesaktuelle Version der Dokumentation findet sich unter: https://sdia.io/sawe-docs

- HPC & KI-Anwendungen/Training, die auf dedizierten Server-Systemen ausgeführt werden, auf denen die IPMI oder RAPL-Messwerte in einem IT-Monitoring/Telemetrie System (wie Prometheus) verfügbar sind.
- CI/CD Entwicklungsprozesse, bei denen Software-Anwendungen mit Hilfe von automatisierten Tests ausgeführt werden oder in Kompilierungsprozessen für die Auslieferung aufbereitet werden.
- Lokale Software-Entwicklung auf Endgeräten, die über RAPL oder IPMI-Schnittstellen verfügen.

Insbesondere SoftAWERE-Light ermöglicht eine einfache Schätzung von Umweltwirkungen für digitale Produkte auf Basis von bestehenden IT-Monitoring und Telemetrie-Daten. So ist es möglich, von digitalen Produkten die Umweltwirkung im Betrieb der Anwendungen (z. B. für Software-as-Service-Angebote oder Endkunden-Plattformen) für Nutzer transparent zu machen.

Im Rahmen des Vorhabens hat das Forschungsteam eine mögliche Kennzeichnung (siehe Abbildung 3) auf Basis der Messdaten für Open-Source-Software und Bibliotheken erarbeitet (siehe Kapitel 3.1).

Abbildung 3: Beispiel des Labels mit einem Energieverbrauch von 723 Ws

Kennzeichnung und Normung

Das Vorhaben liefert die Grundlage, um die beiden Herangehensweisen in den Prozess der Normung einzubringen.

Insbesondere die mathematische Herangehensweise eignet sich für die Normung als Standard-Methode für die Ermittlung von Näherungswerten von Energieverbräuchen in Cloud- und virtualisierten Umgebungen. Ein Standard in Form einer Norm erscheint hier besonders interessant, da aktuell alle Cloud-Anbieter entweder keine Berichte zur Umweltwirkung oder nicht vergleichbare Berichte mit nicht standardisierten (und nicht öffentlich zugänglichen) Methoden für Kunden bereitstellen.

Beide Herangehensweisen lassen sich hinsichtlich der ganzheitlichen Betrachtung von Umweltwirkungen (Energie- und Ressourcenaufwand) und Indikatoren in die Normung einbringen. SoftAWERE-Light sollte als Standard-Methode für die Erfassung von Umweltwirkungen von Software im Entwicklungsprozess und als Grundlage für eine Transparenz-Kennzeichnung etabliert werden. Zum Zeitpunkt des Vorhabens gab es keine relevanten Standards, die sich mit der Messung der Umweltwirkung von Software auseinandersetzen. Hierzu wird empfohlen, einen neuen Normungsvorschlag in entsprechende Gremien einzubringen.

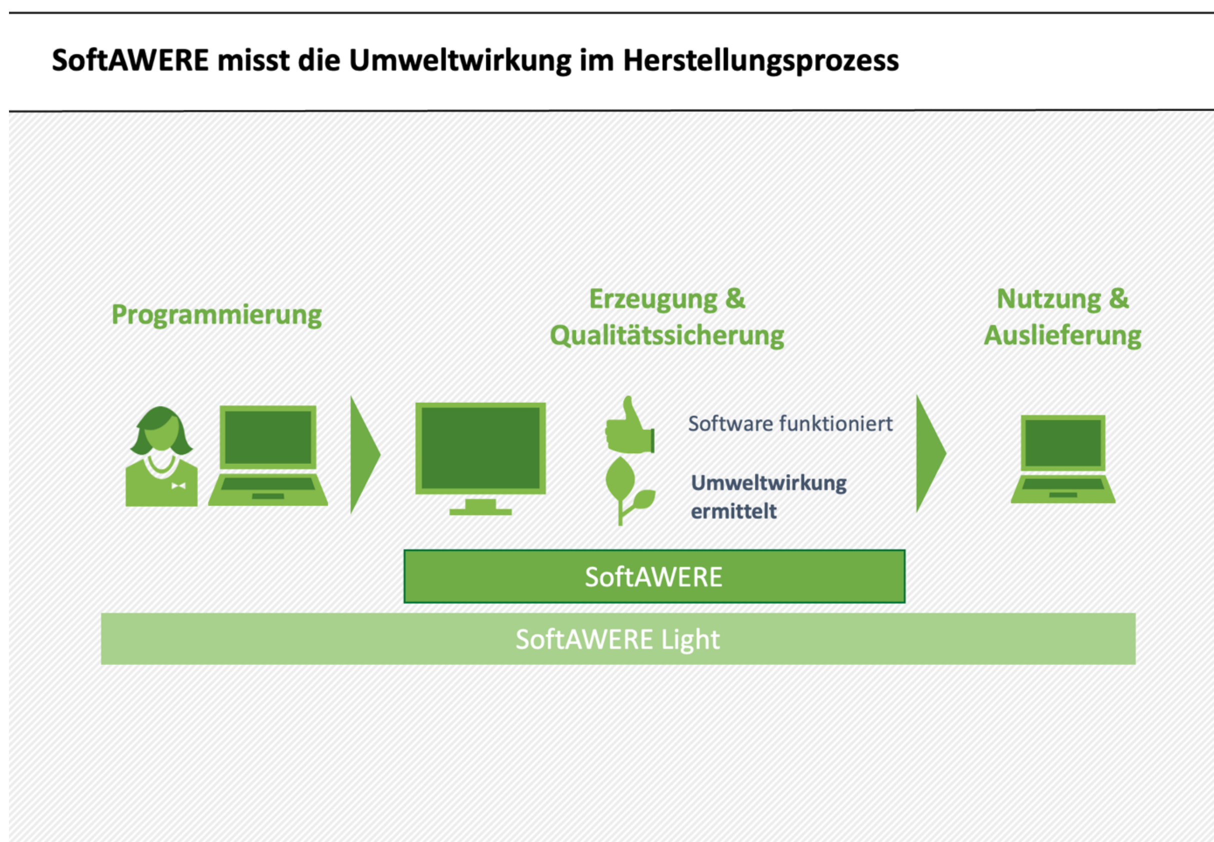
Im Rahmen der ISO-Normung wurde ein Standardisierungsvorgang für EcoDesign-Richtlinien für digitale Dienste ins Leben gerufen (ISO/IEC CD TS 20125).

Mit dem Messaufbau aus dem Vorhaben und den durchgeführten Tests (siehe Kapitel 2.3), liefert das Vorhaben eine plausible, richtungssichere und reproduzierbare Methode für die Erhebung von Umweltwirkungen von Software-Anwendungen im Rahmen des Software-Qualitätssicherungsprozesses. SoftAWERE kann als Teil des Test- und

Qualitätssicherungsprozesses in die Normen ISO/IEC TS 30103 und 29119 eingebracht werden[9]. Die ISO-Norm 30103 („Software and Systems Engineering — Lifecycle Processes — Framework for Product Quality Achievement“) beschäftigt sich mit Software-Qualität in den verschiedenen Lebenszyklusphasen von Software, z.B. die Anforderungsdefinition, Architektur, und Entwicklung. Die ISO-Norm 29119 („Software and systems engineering — Software testing“) beschäftigt sich mit dem Aufbau von Test-Umgebungen zur Qualitätssicherung von Software-Anwendungen. Beide Normen lassen sich möglicherweise um Aspekte zur Umweltwirkung erweitern, entweder als Betrachtung im Lebenszyklus von Software (ISO 30103) oder als Aspekt der Qualitätssicherung (ISO 29119).

Abbildung 4 veranschaulicht die relevanten Phasen aus dem Software-Entwicklungsprozess und die mögliche von der SoftAWERE bzw. Soft-AWERE-Light-Methode.

Abbildung 4: Übersicht Software-Herstellungsprozess und Qualitätssicherung



Quelle: SDIA e.V.

Zusätzlich wurde auf Basis von SoftAWERE im Rahmen des Vorhabens das Konzept einer Transparenz-Kennzeichnung entwickelt, welche aus den Messdaten automatisch generiert werden kann. Diese Kennzeichnung macht die Messdaten sichtbar und dient als Indikator, dass eine Software-Anwendung vermessen worden ist und die Messdaten offen verfügbar sind. Für eine Beurteilung der Nachhaltigkeit einer Software-Anwendung sollte weiterhin auf die Kriterien des Blauen Engels zurückgegriffen werden.

Diese Transparenz-Kennzeichnung bzw. Indikator zur korrekten Messung auf Basis der SoftAWERE-Methode und offenen Messdaten sollte in Normungsprozesse eingebracht werden.

[9] Im Rahmen des Vorhabens wurden nicht gesondert relevante Standards untersucht, die Empfehlungen haben keinen Anspruch an vollständig.

Schlussfolgerungen und Fazit

Das Vorhaben hat zwei Herangehensweisen entwickelt, die einen wichtigen Schritt in Richtung Transparenz hinsichtlich Energie- und Umweltwirkungen für Software ermöglichen.

SoftAWERE-Light, die mathematische Herangehensweise, die auf qualifizierter Schätzung beruht, ist einfach anzuwenden und kann in bisherige IT-Monitoring-Systeme eingebaut werden. Jedoch ist die Betrachtung auf den Energieverbrauch limitiert und die Methode sollte durch weitere Forschung auch um die anderen Umweltindikatoren (siehe Tabelle 1) erweitert werden.

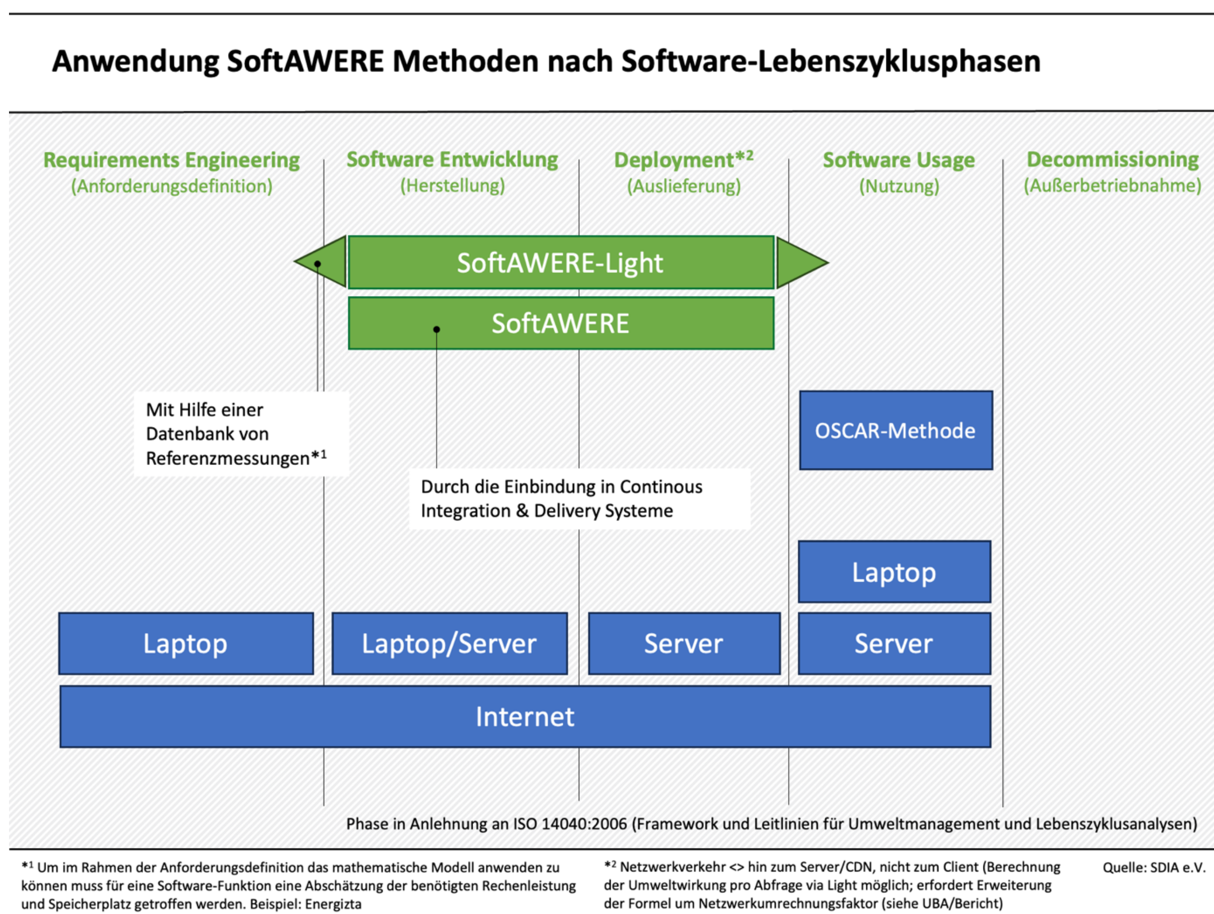
Mit SoftAWERE lassen sich viele verschiedene Software-Anwendungen messen und die Umweltwirkungen ermitteln. SoftAWERE betrachtet jedoch nicht den gesamten Lebenszyklus einer Software-Anwendung und sollte in weiteren Forschungsprojekten noch um Messdaten aus den lokalen Entwicklungsumgebungen (Desktop- und Laptop-Computer von Software-Entwickelnden) erweitert werden und auch die Betriebsmessungen (z. B. in Kombination mit SoftAWERE-Light) in einer einfachen Darstellung vereinen. Vor allem die Nutzungsphase wird bei SoftAWERE bzw. bei SoftAWERE-Light ausgeklammert, da der Fokus dieses Vorhabens primär auf der Herstellungs- und Auslieferungsphase von Softwarecode liegt. So lassen sich die Umweltlasten für eine Softwareanwendung über den gesamten Lebenszyklus abbilden.

Damit wird ein weiterer wichtiger Baustein für die Normung und Differenzierung von digitalen Produkten (transparent und umweltfreundlich) geliefert.

Eine Darstellung der Methoden und Ihrer Anwendbarkeit auf die Lebenszyklusphasen findet sich in Abbildung 5. Die Abbildung 5 zeigt einen möglichen, vereinfachten, typischen Softwarelebenszyklus. Zunächst einmal erfolgt die Anforderungsdefinitionsphase (Requirements Engineering). Daraufhin erfolgt die Herstellungs- bzw. Programmierphase (kann auch Testing miteinschließen). Desweiteren wird die Software an den Kunden ausgeliefert und in einem nächsten Schritt in der eigenen technischen Umgebung genutzt. Als letzten Schritt in einem Software-Lebenszyklus ist die Löschung/Deinstallation von Software. Normalerweise erfolgen diese Phasen in einer regelmäßigen Iteration, d.h. der Programmiercode wird regelmäßig, beispielsweise in Integrationstests oder Funktionstests auf verschiedene Kriterien hin geprüft, bevor bzw. nachdem die Software ausgeliefert wird. Die beiden Methoden SoftAWERE-Light und SoftAWERE beschränken sich nur auf die Programmierungs- und Deployment-Phase (Auslieferung) der Software.

Die SoftAWERE-Light-Methode lässt sich auch auf die Nutzungsphase übertragen, denn die Formeln lassen sich auch auf Endgeräten für eine Schätzung der Umweltwirkungen anwenden. Gleiches gilt für die Anforderungsdefinition auf den Endgeräten von Software-Entwickelnden. Auch hier kann SoftAWERE-Light möglicherweise adaptiert werden. Da die Übertragbarkeit in diese Phasen plausibel erscheint, jedoch im Vorhaben nicht weiter untersucht wurde, sind in der Abbildung 5, Pfeile in beide Richtungen dargestellt, die diese Übertragbarkeit symbolisieren.

Abbildung 5: Übersicht der Methoden und ihrer Anwendbarkeit im Software-Lebenszyklus



Eine weitere Schwäche sowohl von SoftAWERE als auch SoftAWERE-Light ist die erschwerte Vergleichbarkeit von Messergebnissen verschiedener Software-Anwendungen (z.B. Open-Source Bibliothek mit einer Desktop-Anwendung). Da die Softwarelandschaft sehr divers ist, ist es sehr schwierig, eine Vergleichbarkeit zwischen Softwareanwendungen überhaupt herstellen zu können. Daher wurde insbesondere bei der Kennzeichnung auf eine Transparenzkennzeichnung zurückgegriffen, die auszeichnet, dass die Software hinsichtlich ihrer Energie- und Umweltwirklungen vermessen wurde. Hier ist weitere Forschung notwendig, um eine Klassifizierung von Software-Anwendungen zu entwickeln. Für jede Art von Software können dann Standard-Nutzungsszenarien entwickelt werden, deren Ausführung und Messung zu vergleichbaren Ergebnissen hinsichtlich der Umweltwirkungen von Software-Produkten führt. Diese Klassifizierung und Definition von Standard-Funktionen für verschiede Arten von Software (z. B., Kollaborationssoftware, Video-Konferenz-Software) eignetet sich ebenfalls für die internationale Normung.

Zuletzt sollte weitere Forschung hinsichtlich Best-Practice-Methoden von ressourceneffizienter Programmierung („Green Coding“) betrieben werden. Es fehlt an konkreten und verifizierten Handlungsempfehlungen, um die Umweltwirkungen von Software-Anwendungen zu reduzieren. Bestehende Kataloge fokussieren sich auf die individuellen Handlungen von Entwickelnden und bieten keine konkreten Empfehlungen zur Wahl von Komponenten oder Architekturansätzen. Die Handlungsempfehlungen können im Messstand des SoftAWERE-Vorhabens hinsichtlich Ihrer Effektivität überprüft werden. Weitere Forschung könnte somit den ersten Katalog mit verifizierten Handlungsempfehlungen veröffentlichen.

Summary

The phrase "Software is eating the world" was first used by Marc Andreessen, a prominent Silicon Valley investor and co-founder of Netscape (Andreessen 2011). The phrase "software is eating the world" is an expression that conveys the growing penetration and influence of software in nearly all aspects of modern life.

The statement indicates that software applications and digital products are assuming a pivotal role in a growing number of industries and aspects of daily life. In the context of the accelerated digitalisation of contemporary society, however, the aforementioned assertion also implies that software is accountable for an escalating consumption of resources and energy (for further details, please refer to Chapter 1).

The potential for efficiency gains in relation to individual software products is frequently perceived to be limited. Nevertheless, given the vast distribution, remarkable scalability, and high frequency of use of software, even the smallest potential savings are worthwhile (see page 49). In the digital sector and in public discourse, there is a lack of awareness that software, in addition to its energy consumption, is also responsible for the environmental impacts stemming from the physical hardware that is required to run the software – both the end-user devices and the increasing number of servers, storage and network systems within data centres. This lack of awareness is partly due to the absence of validated measurement methods and tools for assessing these environmental impacts in the development process of software applications.

In the context of software applications, the term "usable resources" encompasses a range of capabilities, including working memory capacity, CPU time, long-term storage capacity, and network transmission capacity. These 'resources' are employed by the software application in order to facilitate the provision of functions. The resource consumption is contingent upon the programming language employed, the specific functionality of the application, and the overall scope and scale of the software. In the context of the SoftAWERE project, the term "digital resources" has been introduced to encompass the aforementioned resources consumed by software applications.

In the SoftAWERE model, digital resources are produced by information technology (IT) equipment, specifically computing, storage and network systems. To provision digital resources for a software application, IT equipment must be manufactured and run using electricity. This results in the consumption of physical resources, from energy and water used during the manufacturing, to raw materials, as well as emissions, pollution and other environmental impacts.

The theoretical model (using digital resources as a proxy) can be employed to analyse the following measures for the reduction of environmental impact:

1. The efficiency of resource and energy utilisation in the manufacture of IT hardware. Higher efficiency implies that the same IT hardware, exhibiting the same performance, can be manufactured using a reduced quantity of raw materials and less energy, yet produce an equal number of digital resources.
2. The extent to which a software application is able to consume digital resources in an efficient manner, with the objective of providing the same range of functions and scale with a reduced number of digital resources.

3. The degree of utilisation of the IT hardware is contingent upon the software application and the way it requests and consumes digital resources. A high degree of utilisation signifies the continuous and optimal utilisation of IT hardware, with minimal idle time.
4. The quantity of digital resources necessitated by a software application for operational purposes. For instance, if an application requires an increasing number of digital resources for operation with each new version, the requisite IT hardware will also increase.

The SoftAWERE methodology has yielded the creation of two distinct methods, designated "Light" and "Full," which serve to establish a connection between the digital resource consumption of software applications and the associated environmental impact.

For each digital resource, for example 1 GB of RAM (‘working memory’), the environmental impact of production and operation is calculated and visualised for software developers. The transparency of the environmental impact of the digital resources that are required for the application provides software developers with an incentive and a decision-making aid for the evaluation and implementation of the measures. Furthermore, it enables the attribution of accountability. Given that an application utilises digital resources, it is thus accountable for the environmental impact resulting from their provision.

The Federal Environment Agency was among the first to highlight the environmental significance of software with the introduction of the Blue Angel. The introduction of the environmental label, ‘Blue Angel’, made energy and resource consumption visible for the first time, thereby identifying characteristics that play a pivotal role in the optimal environmental performance of software.

In the context of the Blue Angel, the term 'resource consumption' is used to describe the process by which software accesses hardware and is the driver of energy consumption during its runtime. The hardware itself consists of a variety of raw materials, some of which are classified as rare earth elements. The production of these components is also a highly energy-intensive process, requiring the use of high-purity chemicals that have a significant greenhouse gas potential. As the utilisation of the software on the hardware increases, so too does the necessity for more powerful hardware, which in turn requires either replacement or expansion.

As part of this project, the SoftAWERE tool (hereinafter referred to as "SoftAWERE") was developed. During the development process, it provides software developers with feedback on the energy consumption and environmental impact caused by the software which is being produced. The tool may be defined as a digital measurement environment for the energy consumption and environmental impact of software.

SoftAWERE enables the following:

- ► to recognise the environmental impact of the development process of new and revised software, as well as to check changes in energy consumption and hardware usage.
- ► the automation of the process of measuring energy consumption and environmental impact, thus integrating it into the user's work process during software development in a straightforward manner.

It should be noted that both SoftAWERE and SoftAWERE Light are not suitable for runtime environments that are operated on a local instance, for example on the software developer's end device. SoftAWERE Light was primarily developed for use in server environments in which no measurement interfaces are available from the operating system or the underlying hardware. The extended version of SoftAWERE is suitable for use on dedicated and private server environments (e.g. in secure company networks).

Methodology

In terms of methodology, the project is informed by the previous research work of the Federal Environment Agency. The methodological approach for the application of standardised life cycle analysis (LCA) methods to measure the energy and hardware flows of software and the holistic consideration of environmental impacts was developed by the Environmental Campus Birkenfeld and the Öko-Institut (Kern et al. 2018) and forms the basis for SoftAWERE.

In the development of criteria for the "Blue Angel" environmental label, a standard usage scenario was developed as a research method (Hilty et al. 2015) with the objective of enabling the reproducible measurement of software in relation to the energy consumption of the underlying hardware. The standard usage scenario describes a repetitive sequence of automated user interactions with the software, thereby simulating typical usage behaviour. This approach was further developed in SoftAWERE (see Chapter 2.2).

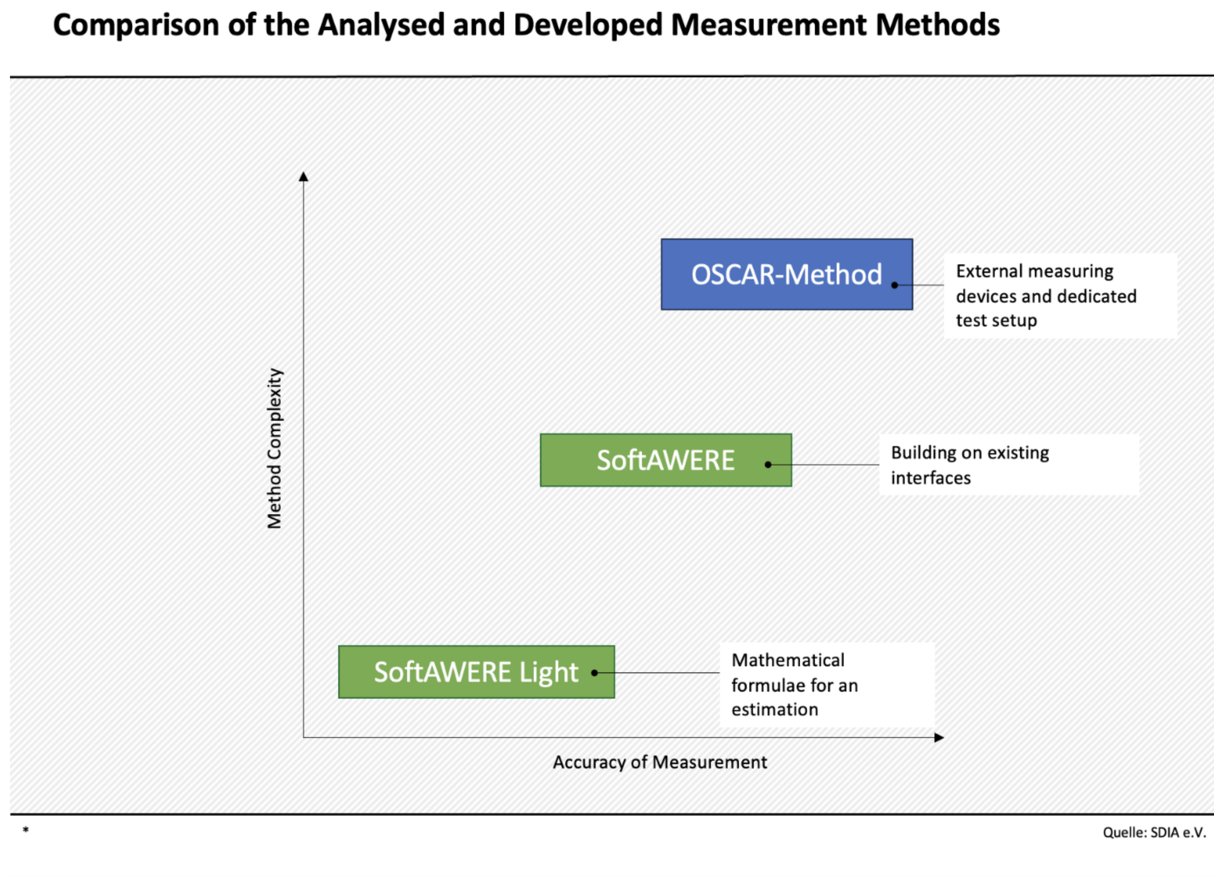
In lieu of developing usage scenarios that align with a typical utilisation of the software, the existing automated integration tests are employed as usage scenarios to facilitate the assessment of the software. In the context of software development, integration tests are defined as repetitive tests that are designed to assess the interaction between individual components. It is a common practice in software development to implement integration tests as a quality assurance measure to identify and prevent errors in the programming code. Integration tests simulate the utilisation of the software, thereby ensuring the reproducibility of test results. It is assumed that the integration test encompasses all pertinent functions of the software, thereby ensuring a high level of test coverage. This approach has the effect of reducing the workload for a significant proportion of software projects, particularly those that have already achieved a high level of test coverage (for example, 80% of the functions have been addressed and executed via the tests). However, this approach also has the limitation of not allowing for direct comparability, as the specific usage scenarios and integration tests of individual software applications are not identical.

A distinctive feature of SoftAWERE is the absence of external current measuring devices and the absence of specific technological and measurement setup requirements. The objective of the research team was to guarantee a high level of applicability in the software market. To this end, the measurement stand can also be implemented on standard server hardware without the use of specialised equipment or access to data centres. The project thus enables companies and software developers to run SoftAWERE in their own environments (see Appendix D.8). It should be noted that the project is limited to Linux-based runtime environments. Linux is the world's most popular server operating system *(W3Techs, n.d.)*.

Two distinct variants of SoftAWERE were developed as part of the project:

1. The "SoftAWERE-Light" system is a simplified version of the original SoftAWERE system that does not require measurements and instead calculates an estimate of the energy and environmental impact using mathematical formulas. A simplified version that obviates the necessity for measurements and instead calculates an estimate of the energy and environmental impact through the application of mathematical formulas. This variant is appropriate for cloud and virtualised environments in which direct physical measurement with measuring devices or measurement via server interfaces is not feasible.
2. "SoftAWERE": The comprehensive variant, which does not necessitate the utilisation of physical measuring devices and is predicated upon the pre-existing interfaces of server manufacturers and operating systems.

Figure 6: Overview of the analysed and developed methods based on complexity and accuracy.



The efficacy of both methodologies was validated through the utilisation of the test setup (for further details, please refer to Chapters 2.2 and 2.3). SoftAWERE was designed for deployment in server-based applications and is constrained to the estimation of computational capacity and memory utilisation and storage utilisation. The assessment of the environmental impact of network traffic was not included in the scope of this study due to the unavailability of pertinent data.

SoftAWERE-Light - mathematical estimation

In virtualised or containerised runtime environments, access to the interfaces required for measuring energy consumption is limited. Furthermore, it is frequently not feasible to ascertain the manufacturer and model of the server. To ascertain the environmental impact and energy consumption of the software in these environments, an approximation approach was initially developed based on information provided by the operating system, computer firmware, data centre, or software developers. The resulting level of accuracy is contingent upon the availability of the aforementioned information.

- The utilization of the system components (CPU load, RAM usage, hard disk usage, network utilization) - hereinafter referred to as "digital resources"[10]

[10] The term is also defined in the SDIA glossary: see https://knowledge.sdialliance.org/glossary/digital-resource-primitives, retrieved on 10/21/2023

- CPU type of the server (for which a TDP value[11] must be available)[12]
- PUE of the underlying data centre. If not available, an average value for European data centres is used - PUE 1.8 (Bertoldi 2015)
- Physical location of the data centre or computer in/on which the software is running (for determining GHG emissions from the electricity grid)

If this information is available, a formula can be used to determine the energy consumption per server utilization level. This formula was further developed and improved several times as part of the project (see Chapter 2.2, "Development of an approximation method that works without IPMI & RAPL") and by the software community (see Chapter 2.2 and Figure Abbildung 20 for a graphical representation of the variants) and was also published as a scientific paper (Kennes 2023).

SoftAWERE interface-based measurement

The full version of SoftAWERE is based on existing technical foundations:

- IPMI & RAPL interfaces: To avoid the use of external measuring instruments, SoftAWERE uses the available standard interfaces for current measurement in common server models.
- GitLab CI/CD Runner & Docker: We used the CI/CD pipelines from GitLab to orchestrate and isolate the measurement runs. Docker containers are used to isolate the measurements from the main system (see Chapter 2.2 and Appendix D.7).
- Docker Compose: The measurement lab itself consists of open-source components that can be started on almost any server hardware (Linux-based servers only). Docker Compose is used to orchestrate the test bench components.
- Prometheus, Thanos and Grafana: Existing open-source telemetry systems[13] were used for the measurement and visual evaluation of the results. The Thanos database in a scalable Kubernetes cluster is used for the long-term storage of the recorded time series.
- Further use of various APIs: Scaphandre, FreeIPMI, Boavizta API, and Boagent: Two libraries are used to access the IPMI and RAPL interfaces, FreeIPMI and Scaphandre. The Boavizta API (which is based on research by the German Federal Environment Agency) is used to determine the environmental impact of the underlying servers. The Boagent is used to determine the hardware properties (storage space, number of CPUs, RAM, etc.).

Energy-saving functions of server hardware and CPU: Activating this function significantly reduces energy consumption at low workloads. This makes it possible to determine the effect of the generated computing load on the energy consumption of the software being analysed. The full version of SoftAWERE can be used in GitLab-based development environments that run on

[11] The Thermal Design Power (TDP) value of a CPU indicates the maximum amount of heat, measured in watts (W), that the processor can dissipate under a typical workload. It is an important metric for system designers and users to understand how much thermal management is required to keep the CPU within its safe operating temperature range.

[12] For the common Intel server chips, these are available on the Intel website: https://ark.intel.com/content/www/de/de/ark/products/series/125191/intel-xeon-scalable-processors.html

[13] Telemetry systems are used to monitor software applications. Among other things, they record the consumption of computing power, memory usage and network traffic. The systems are used to monitor the utilization of infrastructure and performance of software applications during operation. In the SoftAWERE context, telemetry data refers to the transmission of measured values that are measured at one location (server) and transmitted to another measurement location (...).

dedicated and virtualized systems. A prerequisite is the availability of telemetry data collected via IPMI or RAPL.

Background information on IPMI and RAPL

The IPMI and RAPL interfaces are not available in virtualised and cloud environments. In these environments, measurement with external power meters is also not possible, as the data centres are not accessible, and the infrastructure is shared by many customers. The project has therefore also developed a mathematical approach that makes it possible to estimate energy consumption (see Chapter 2.2).

SoftAWERE can be used to test applications in almost any programming language. Usage scenarios can also be programmed to check certain facts, e.g. with a sample software application. One example of this is the calculation of Fibonacci numbers to determine the speed of programming languages (see Figure Abbildung 30 in Chapter 2.4).

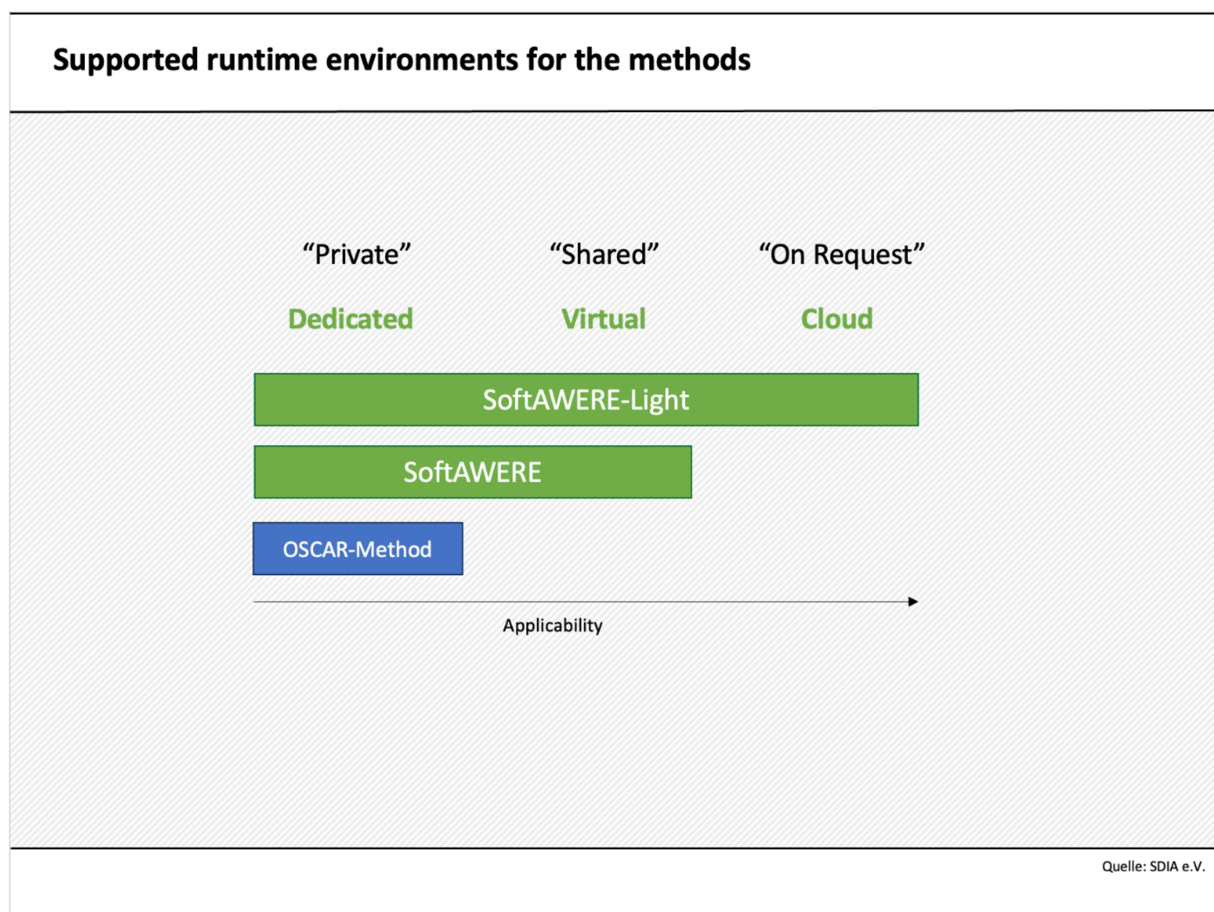
SoftAWERE is particularly suitable for applications which are compiled. Some use cases are listed below (see also Chapter 4):

- the training or fine-tuning of AI models (data is incorporated into a model in a machine learning process),
- Data analysis software or use cases in which a software application or other software tool processes, collects, formats, or performs other operations on data that require time and computing power,
- Static websites that are compiled before delivery[14] (see example of the SDIA website, in Chapter 4.1.2),
- Classic enterprise applications that either have a large usage test coverage or are compiled before delivery (e.g. Java, C# applications).
- Open-source tools and libraries that have integration tests (for a list of the open-source projects examined in the project, see Appendix C.2).

Figure 7 provides a visual representation of the two methods and their applicability in different runtime environments.

[14] Also known as "static site generators" ("SSGs"). Examples are Jekyll, Next.js, Hugo and Gatsby. See also https://en.wikipedia.org/wiki/Static_site_generator (retrieved on, 09.01.2024)

Figure 7: Comparison of methods in relation to supported runtime environments



Energy consumption & environmental impact

Both SoftAWERE variants make both energy consumption and the environmental impact of software applications measurable (see Chapter 2.3). Overall, SoftAWERE makes the following indicators measurable for software developers:

Table 2: Overview of selected environmental factors

Environmental impact indicator	Description	SoftAWERE variant
Power consumption	The power consumption that occurred during the execution of the software in the measuring station.	SoftAWERE, SoftAWERE-Light
Operational GHG emissions	Emissions caused by the downstream infrastructure, such as power plants, for electricity generation based on the consumption of energy carriers.	SoftAWERE, SoftAWERE-Light
GHG emissions from production	An estimate of the emissions generated by the production of the ICT equipment necessary to run the software.	SoftAWERE

Environmental impact indicator	Description	SoftAWERE variant
Depletion of abiotic resources[15] of production	Estimation of the quantity of abiotic resources embedded in the ICT equipment required to run the software.	SoftAWERE

It was particularly important to the research team not only to measure and present energy consumption, but also to provide an estimate of the manufacturing and resource costs incurred by the ICT equipment and data centre.

In many cases, the focus on energy efficiency leads to early replacement of servers without comparing the associated resource expenditure or GHG emissions from manufacturing and transportation (Prakash et al. 2012). Due to the lack of transparency and public, verified data from ICT manufacturers and data centre operators, the calculation within SoftAWERE is limited to an estimate[16].

Application of SoftAWERE

SoftAWERE can be used by companies, organizations, and individuals on their own infrastructure as a digital measuring station for the environmental impact of software. All that is required are server environments that have the corresponding IPMI and RAPL interfaces. Complete instructions for setting up SoftAWERE yourself can be found in Appendix D.8[17].

The SoftAWERE-Light variant can also be used in environments that do not have the necessary interfaces, such as cloud environments, if telemetry data on CPU utilization, memory and storage space is available. The corresponding formulas are documented in Chapter 2.2, "Development of an approximation method that works without IPMI & RAPL", the mathematical approach makes SoftAWERE-Light suitable for use in a very wide range of applications, especially during application runtime (operation):

- Cloud-based applications that run on one or more virtual machines or cloud instances. The formulas can be used to convert digital resource consumption (CPU utilization, memory, storage utilization) into energy consumption.
- Applications in private virtualized environments (e.g. Xen, OpenStack, VMware) in which access to the underlying server systems and the data centre is not possible for security or technical reasons.
- Desktop applications where access to the underlying energy measurement interfaces is not possible (e.g. due to security settings or older operating systems that do not yet have the necessary interfaces).

The full version of SoftAWERE is particularly suitable for use in the software development process and use cases on dedicated hardware:

- HPC & AI applications/training running on dedicated server systems where the IPMI or RAPL readings are available in an IT monitoring/telemetry system (such as Prometheus).

[15] The term "embedded abiotic resource depletion" refers to the depletion of non-living (abiotic) natural resources such as minerals and fossil fuels that are consumed or degraded during the production and life cycle of a product. This concept highlights the hidden environmental costs embedded in goods and services by taking into account the depletion and dwindling availability of these important non-renewable resources. It is an important aspect of sustainability research and highlights the need for responsible resource management and the development of sustainable alternatives.

[16] The estimate is based on existing research by the Federal Environment Agency, see (Gröger et al. 2021).

[17] A daily updated version of the documentation can be found at: https://sdia.io/sawe-docs.

- CI/CD Development processes in which software applications are executed with the help of automated tests or prepared for delivery in compilation processes.
- Local software development on end devices that have RAPL or IPMI interfaces.

SoftAWERE-Light makes it easy to estimate the environmental impact of digital products based on existing IT monitoring and telemetry data. This makes it possible to make the environmental impact of digital products during application operation (e.g. for software-as-a-service offerings or end customer platforms) transparent for users.

As part of the project, the research team developed a possible label based on the measurement data for open-source software and libraries (see Chapter 3.1) which can be seen on Figure 8.

Figure 8: Example of the label with an energy use of 723 Ws

Labelling and standardization

The project provides the foundation for integrating both methodologies into the standardisation process.

The mathematical approach is particularly well-suited to standardisation as a standard method for determining approximate values of energy consumption in cloud and virtualised environments. It is evident that the establishment of a standard would be particularly beneficial in this context, given that many cloud providers currently lack the provision of any reports on environmental impact, or alternatively, offer reports that are not comparable due to the use of non-standardised methods that are not publicly accessible to customers.

Both approaches can be incorporated into standardisation with regard to the holistic consideration of environmental impacts (energy and resource consumption) and indicators. The SoftAWERE-Light method was developed as a standard approach for recording the environmental impact of software in the development process and as a basis for transparency labelling. At the time of the project, there were no relevant standards dealing with the measurement of the environmental impact of software. It is recommended that a new standardisation proposal be submitted to the relevant committees.

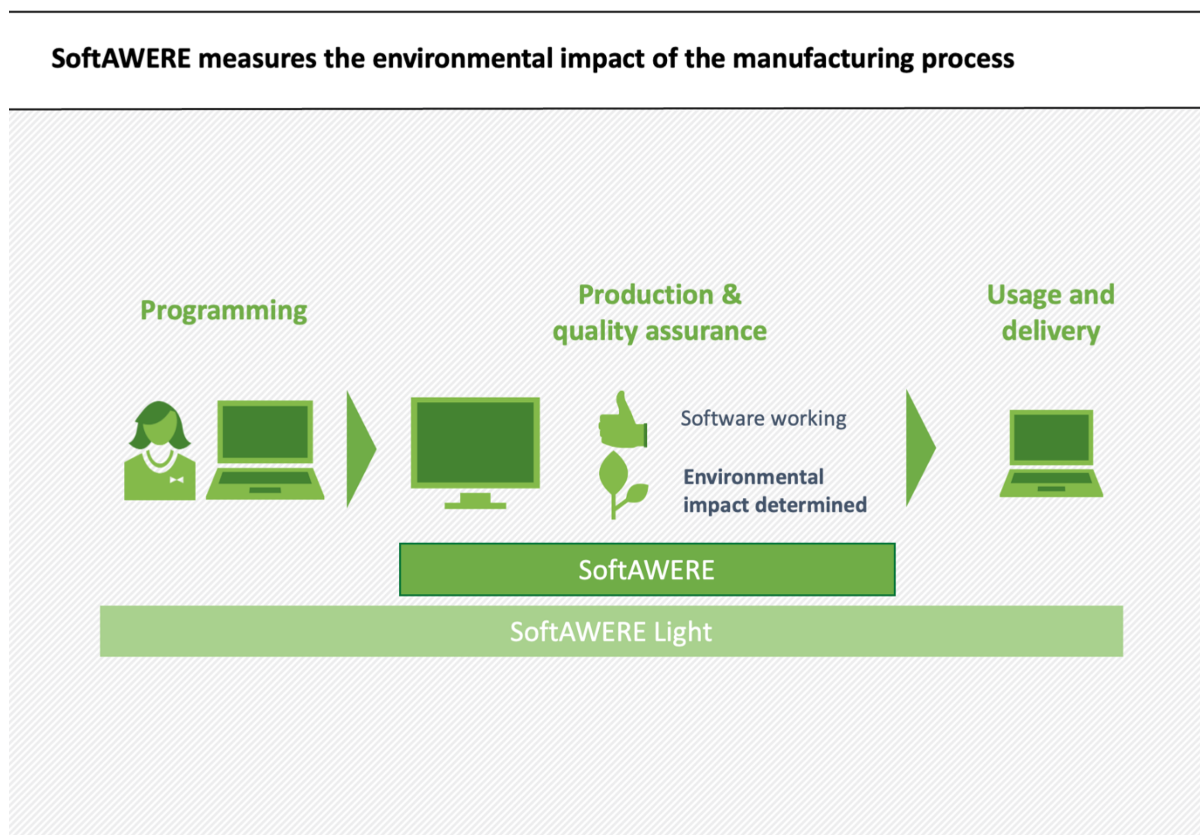
As part of the ISO standardisation process, a standardisation process for EcoDesign guidelines for digital services has been initiated (ISO/IEC CD TS 20125). The research team is actively engaged in this process and is striving to incorporate SoftAWERE as a recommendation or requirement for assessing the environmental impact of digital services within this standard.

The measurement setup from the project and the tests carried out (see Chapter 2.3) provide a plausible, reliable and reproducible method for determining the environmental impact of software applications as part of the software quality assurance process.

It is recommended that SoftAWERE be incorporated into the ISO/IEC TS 30103 and 29119 standards as part of the testing and quality assurance process. The ISO standard 30103 ("Software and Systems Engineering - Lifecycle Processes - Framework for Product Quality Achievement") addresses the topic of software quality across the various stages of the software development lifecycle, including aspects such as requirements definition, architecture, and development. ISO standard 29119 ("Software and systems engineering - Software testing") addresses the development of test environments for the quality assurance of software

applications. Both standards have the potential to be expanded to encompass aspects of environmental impact, either as a consideration within the software life cycle (ISO 30103) or as an aspect of quality assurance (ISO 29119).

Figure 9: Overview of the software development process and quality assurance



Quelle: SDIA e.V.

Furthermore, the concept of transparency labelling, which can be automatically generated from the measurement data, was developed on the basis of SoftAWERE as part of the project. The aforementioned labelling renders the measurement data visible and functions as an indicator that a software application has been measured and that the measurement data is openly available. It is recommended that the Blue Angel criteria be employed to assess the sustainability of a software application.

This transparency labelling or indicator for correct measurement, based on the SoftAWERE method and open measurement data, should be incorporated into standardisation processes.

Conclusion and summary

The project has developed two novel approaches that facilitate a significant advancement in the field transparency of environmental impact of software applications.

The SoftAWERE-Light approach, which is based on qualified estimation within a mathematical framework, is straightforward to implement and can be integrated into existing IT monitoring systems. Nevertheless, the analysis is constrained to energy consumption, and the methodology should be expanded through further investigation to encompass the remaining environmental indicators (see Table 2).

SoftAWERE is a versatile tool that can be employed to assess the environmental impact of a diverse range of software applications. However, SoftAWERE does not consider the entire life

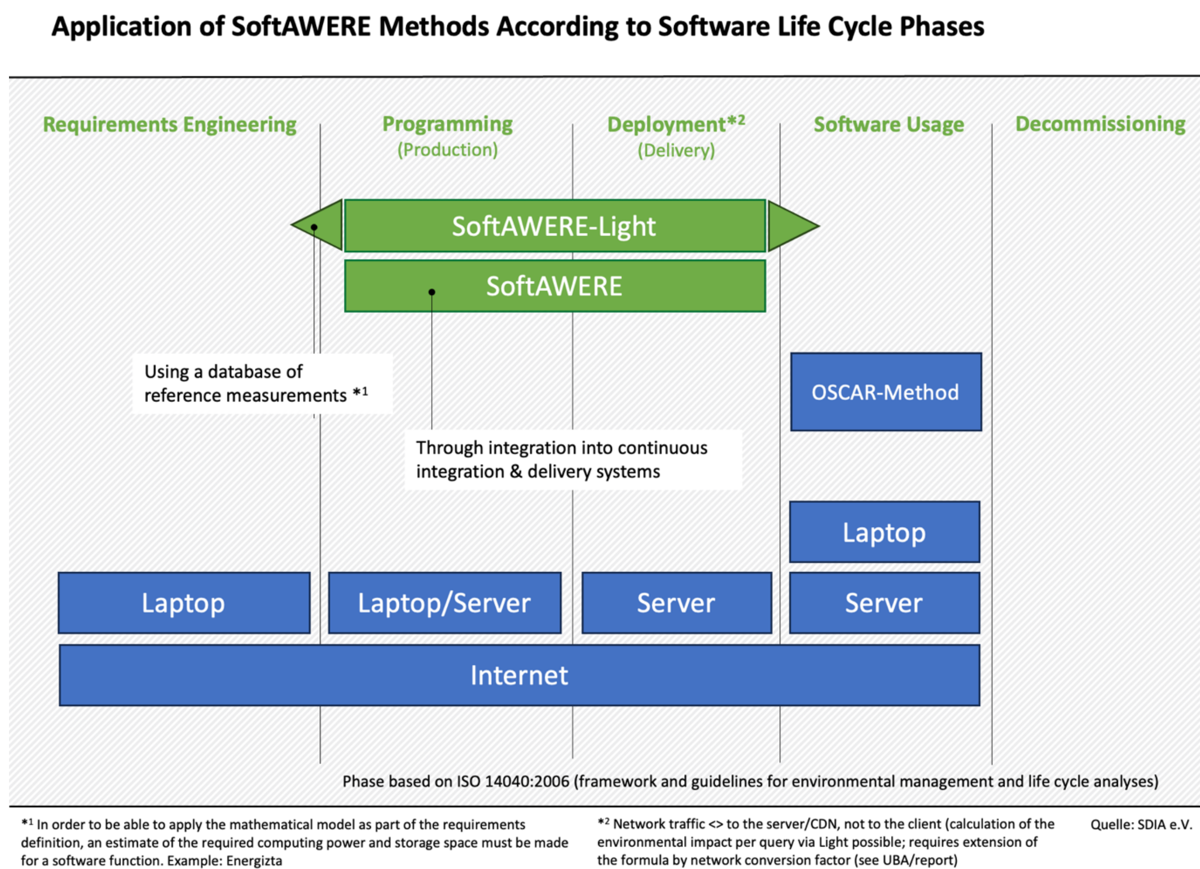
cycle of a software application. Consequently, further research is required to expand the approach to include measurement data from the local development environments (desktop and laptop computers of software developers) and to combine the operational measurements (e.g. in combination with SoftAWERE-Light) in a simple visualisation. The utilisation phase is not included in SoftAWERE and SoftAWERE-Light, as the focus of this project is on the creation and delivery of software code. The further development would allow the environmental impact of a software application to be mapped over its entire life cycle, providing another important building block for the standardisation and differentiation of digital products (transparent and environmentally friendly).

An illustration of the SoftAWERE methods and their applicability to the life cycle phases can be found in Figure 10. Figure 10 depicts a potential, streamlined, archetypal software life cycle. The initial phase is that of requirements definition, which is also referred to as requirements engineering. Subsequently, the production or programming phase (which may also encompass testing) ensues. Moreover, the software is delivered to the customer (deployed), where it is subsequently utilised within their own technical environment. The final stage of the software life cycle is the removal or uninstallation of the software (decommissioning). These phases typically occur in a regular iteration, whereby the programming code is subjected to periodic assessment for a range of criteria, such as in integration or functional tests, prior to or subsequent to the delivery of the software. The SoftAWERE-Light and SoftAWERE methods are applicable solely to the programming and deployment phase (delivery) of the software.

As SoftAWERE-Light can be transferred to the utilisation phase, the formulas can also be applied to end devices to estimate the environmental impact. Similarly, the definition of requirements on the end devices of software developers can be adapted using SoftAWERE-Light. As the transferability to these phases appears plausible, but was not investigated further in the project, arrows symbolising this transferability are shown in both directions in Figure 10.

Figure 10 illustrates the pertinent phases of the software development process.

Figure 10: Overview of the methods and their applicability in the software life cycle



A further shortcoming of both SoftAWERE and SoftAWERE-Light is the challenge of comparing measurement outcomes from disparate software applications (for example, an open-source library with a desktop application). Given the considerable diversity of software applications, it is inherently challenging to establish comparability between them. Considering these challenges, transparency labelling was employed as a means of indicating that the software has been evaluated in terms of its energy and environmental impact. Further research is required to develop a categorisation of software applications.

Subsequently, standard usage scenarios can be developed for each type of software, with the execution and measurement of which will yield comparable results with regard to the environmental impact of software products. The proposed classification and definition of standard functions for different types of software (e.g. collaboration software, video conferencing software) is suitable for international standardisation.

Ultimately, further research is required to identify optimal practices. Currently, there is a dearth of concrete and verified recommendations for reducing the environmental impact of software applications. Existing catalogues concentrate on the individual actions of developers and do not offer specific guidance on the selection of components or architectural approaches. The efficacy of these recommendations can be evaluated in the SoftAWERE project's measurement laboratory. Consequently, further research could potentially publish the inaugural catalogue comprising verified recommendations for action.

1 Hintergrund & Zielsetzung

Die Digitalisierung und die neu entstehende Digitalwirtschaft erschaffen neue digitale Prozesse, Dienstleistungen und Produkte. Diese bestehen aus Software: Instruktionen für Computer, die diese ausführen. Für die Ausführung dieser Instruktionen wird Energie für den Betrieb des Computers benötigt. Für die Herstellung der benötigten Computer werden kritische Rohstoffe verbraucht. Für viele Software-Anwendungen werden Hochleistungscomputer- und Speichersysteme, sogenannte „Server" gebraucht. Diese erzeugen, wie fast alle Computer, Abwärme und erfordern häufig besondere Gebäude-Infrastruktur, zum Beispiel Kühlung und redundante elektrische Infrastruktur für ihren Betrieb. Wie effizient Software mit den Server-Ressourcen umgeht, ist entscheidend. Je mehr Ressourcen Software verbraucht, desto höher ist der Bedarf an Rechenzentrumsflächen, Gebäude-Infrastruktur und Server-Equipment und damit an Energie und Rohstoffen.

Das Forschungsvorhaben beschäftigte sich mit dem Herstellungsprozess von Software, der „Software-Entwicklung". In diesem Kontext beschäftigte sich das Vorhaben mit der Bereitstellung von Indikatoren zur Energie- und Ressourceneffizienz für die prozessbeteiligten Fachleute. Das Vorhaben hatte das Ziel, Methoden und Werkzeuge zu schaffen, die im Prozess der Software-Entwicklungen eingesetzt werden können, um bessere Entscheidungen hinsichtlich Effizienz und Ressourcenaufwand zu treffen. Als Forschungsmethode wurden Nutzungsszenarien von Open-Source-Bibliotheken, die als Integrationstests definiert wurden, benutzt, um die Energie- und Ressourcenaufwände zu ermitteln. Zudem wurde eine Transparenz-Kennzeichnung für Open-Source Bibliotheken und Werkzeuge evaluiert (siehe Kapitel 3.1).

Schätzungen (Nagle et al. 2022) zu Folge sind in 98% aller Software-Anwendungen freie und quelloffene Software-Komponenten [18] (FOSS, „free and open source software") enthalten. Dies gilt gleichermaßen für proprietäre und kommerzielle Anwendungen, die auf kostenlose, quelloffene Bausteine zurückgreifen. Diese Bausteine erlauben es, Funktionalitäten wiederzuverwenden und somit schneller funktionsfähige Software-Anwendungen herzustellen, was im Falle einer kommerziellen Anwendung Kosten reduziert und somit die Rentabilität erhöht. Im Entwicklungsprozess werden diese Komponenten durch Software-Entwickler*innen verbaut. Durch die in Kapitel 3.1 evaluierte Kennzeichnung für diese Komponenten sollen Software-Entwickelnde befähigt werden die Ressourceneffizienz in den Auswahlprozess mit einzubeziehen. Die Kennzeichnung stellt den Energie- und Ressourcenverbrauch dar und zeigt, wie effizient Ressourcen bei der Ausführung von Nutzungsszenarien eingesetzt werden. Damit können Software-Entwickelnde den Ressourcenaufwand der Komponente vor dem Einbau bereits abschätzen.

Um die Software-Gemeinschaft, insbesondere die FOSS-Gemeinschaft[19,] auf die Relevanz von Energie- und Ressourceneffizienz aufmerksam zu machen, notwendiges Wissen zu vermitteln und die im Vorhaben entstehenden Werkzeuge und Methoden in die Praxis zu bringen, wurden im Rahmen des Vorhabens Veranstaltungen durchgeführt (siehe Anhang B). Ein Begleitkreis aus Wissenschaft und Wirtschaft begleitete das Vorhaben (siehe Anhang B für Protokolle).

[18] Komponente: eine Softwareeinheit, die von einer anderen Software aufgerufen werden kann oder als Eingabe für eine andere Software dient und über eine Paketverwaltung installiert werden kann." (Nagle, et al., 2022, S. 13)

[19] Die FOSS-Gemeinschaft ist für einen Großteil der Herstellung von Software-Komponenten verantwortlich: https://www.wired.com/2016/08/open-source-won-now/, abgerufen am 18. September 2023

Software als treibende Kraft der Digitalisierung & Digitalwirtschaft

In der "Digitalstrategie Deutschland" (Bundesministerium für Digitales und Verkehr 2022) werden der Digitalisierung viele wichtige Handlungsfelder zugeschrieben, von vernetzter Gesundheit, nachhaltigen Gebäuden, Mobilität, Wirtschaftswachstum bis hin zur Bewältigung der Klimakrise. Digitalisierung bedeutet analoge Vorgänge ins digitale zu wandeln – durch den Einsatz von Informationstechnik (IT), deren Instruktionen in Programmcode geschrieben werden - Software.

Neben der Digitalisierung von analogen Prozessen und Wirtschaftsfeldern gibt es mittlerweile einen globalen Wirtschaftsraum, die Digitalwirtschaft, in dem Unternehmen ausschließlich digitale Produkte und Dienstleistungen über das Internet einer globalen Kundschaft anbieten.

Grundlage von sowohl Digitalisierung als auch Digitalwirtschaft ist die digitale Infrastruktur. Das Umweltbundesamt definiert digitale Infrastruktur als technische Basis der Informationsgesellschaft bestehend aus Rechenzentren und Telekommunikationsnetzwerken (Köhn, Gröger, and Stobbe 2020).

In Abbildung 11 ist eine Übersicht des digitalen Ökosystems dargestellt, welches die Unterschiede zwischen der virtuellen Manifestierung von digitalen Produkten und der physischen Manifestierung in digitaler Infrastruktur aufzeigt. Digitale Produkte und Dienstleistungen bestehen aus Software-Technologien[20], Geschäftsmodellen und digitalen Ressourcen (siehe Kapitel 4). Digitale Infrastruktur ist die physische Manifestierung, hier entstehen direkte Umweltwirkungen und Ressourcenverbräuche, durch die die Erzeugung von digitalen Ressourcen, die wiederum für den Betrieb von Software notwendig sind.

Die Bausteine der digitalen Infrastruktur

Die digitale Infrastruktur setzt sich primär aus 3 Komponenten zusammen (siehe auch Abbildung 11), welche im Zusammenspiel die Produktion von digitalen Ressourcen ermöglichen:

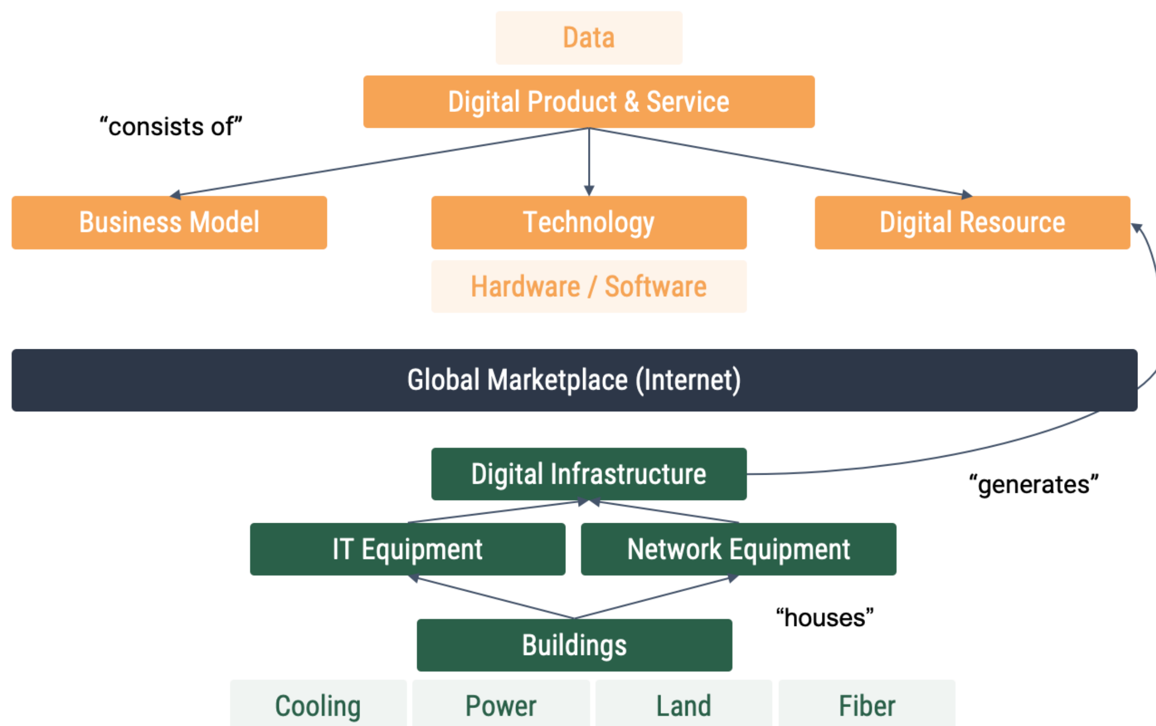
- IKT-Equipment, wie z.B. Server, Speichersysteme, Netzwerktechnik, Verkabelung, Server Schränken, Netzteilen, Stromverteilern, et cetera.
- Gebäude, die entsprechende Fläche für das IKT-Equipment bereitstellen und über redundante Stromversorgungs- und Kühlsysteme verfügen (Rechenzentren). Diese bieten zudem einen Anschluss an das Internet mit einer hohen Bandbreite bzw. mit einer niedrigen Latenz.
- Land & Infrastruktur für die Anlieferung von Elektrizität (Stromnetz), Wasser (Kühlung), und Internet/Netzwerkverbindung (Glasfaser)

Jeder dieser Bausteine verbraucht Rohstoffe und Energie und diese können in einer Lebenszyklusanalyse betrachtet werden. (Whitehead, Andrews, and Shah 2015).

Als Beispiel kann ein digitales Produkt für Unternehmen oder Privatpersonen herangezogen werden, beispielsweise eine Chat-Anwendung oder Video-Konferenz-Anwendung. Dieses Produkt setzt Software-Technologien ein, welche digitale Ressourcen verbrauchen und damit Infrastruktur beanspruchen. Die digitale Infrastruktur wiederum ist für die physischen Ressourcenverbräuche und Umweltwirkungen verantwortlich. Diese werden in Abbildung 12 schematisch dargestellt (Gröger et al. 2021).

[20] wie zu Beginn des Kapitels beschrieben wurde, z.B. fertige Software-Komponenten, aus denen eine neue Anwendung zusammengebaut werden kann. Diese Anwendung wird mit einem Geschäftsmodell versehen und durch digitale Ressourcen angetrieben.

Abbildung 11: Übersicht des digitalen Ökosystems mit digitaler Infrastruktur als Grundlage



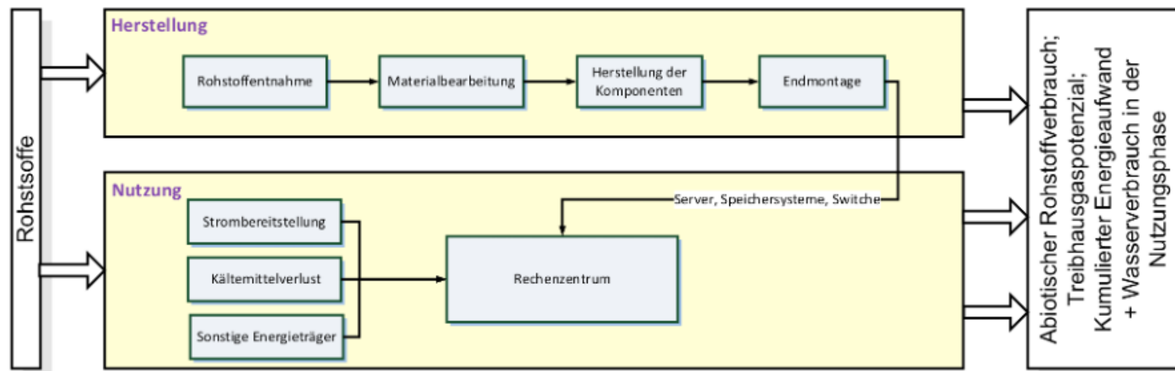
License: CC BY-NC-SA – SDIA, Commonwealth

Übersicht des digitalen Ökosystems welches als Rückgrat der Digitalisierung und Digitalwirtschaft agiert.

Hier wird der Zusammenhang aus digitalen Produkten, Software und der zugrundeliegenden Technologien dargestellt. Ein Beispiel dafür ist die Open-Source Technologie „Tensorflow“, welches die Entwicklung des digitalen Produkts „ChatGPT“ ermöglicht. Tensorflow ist dabei eine Technologie, die frei- und offen verfügbar ist, und aus der ein Unternehmen ein Produkt erschaffen kann. Dafür braucht es zusätzlich ein Geschäftsmodell, im Falle von ChatGPT, ein Abo-Modell und digitale Ressourcen. Diese Ressourcen sind das Produkt von digitaler Infrastruktur wo aus Energie und ICT Equipment Leistung entsteht, die zur Speicherung, Verarbeitung und Transport von Daten genutzt werden kann. Im Schaubild ist diese Leistung als „digitale Ressource(n)“ zusammengefasst. Digitale Infrastruktur wiederrum ist die physische Infrastruktur der Digitalwirtschaft. Sie besteht aus Gebäuden, Server-, Netzwerk- und Speicherequipment und benötigt Land, Kühlung, Glasfasernetze und Strom. Das Internet fungiert im Schaubild sowohl als Marktplatz für digitale Produkte als auch als Transportnetzwerk für digitale Ressourcen.

Auszug aus dem „Trade Competitiveness Briefing Paper Taxonomy Guide: Infrastructure in the Digital Economy“
Quelle: Commonwealth Library (Schulze, Kumar, and Oghia 2021)

Abbildung 12: Schematische Darstellung der Rohstoffverbräuche und Umweltwirkungen von digitaler Infrastruktur



Quelle: Green Cloud Computing(Gröger et al. 2021), Umweltbundesamt, Juni 2021

Energie- und Ressourceneffizienz rücken in den Mittelpunkt

Jedes Problem, jede Dienstleistung, jeder Industrieprozess, jedes Produkt, welches digitalisiert werden soll, wird durch Software gesteuert oder durch Software ersetzt. Die Qualität, Komplexität und Architektur des Programmcodes der Software hat direkte Auswirkungen auf den Energie- und Ressourcenverbrauch der Software (siehe Anhang C.1 für einen Leitfaden).

Das Umweltbundesamt hat in einer Studie bereits untersucht, wie sich die Qualität von Software auf die Lebensdauer von Hardware, z.B. Smart-Home-Geräten, auswirken kann (Jaeger-Erben et al. 2023). Anhand der steigenden Hardware-Anforderungen[21] und der Menge des Programmcodes von Microsoft Windows-Generationen lässt sich dieses Phänomen beispielhaft veranschaulichen. Dabei sagt die Menge an Programmcode nichts über den Energie- und Ressourcenverbrauch aus, lässt aber teilweise auf die Komplexität der Software-Anwendung schließen, was wiederum zu einem höheren Verbrauch an Rechenleistung und Arbeitsspeicher führt.

Tabelle 3: Windows Versionen - Systemanforderungen und Programmcode

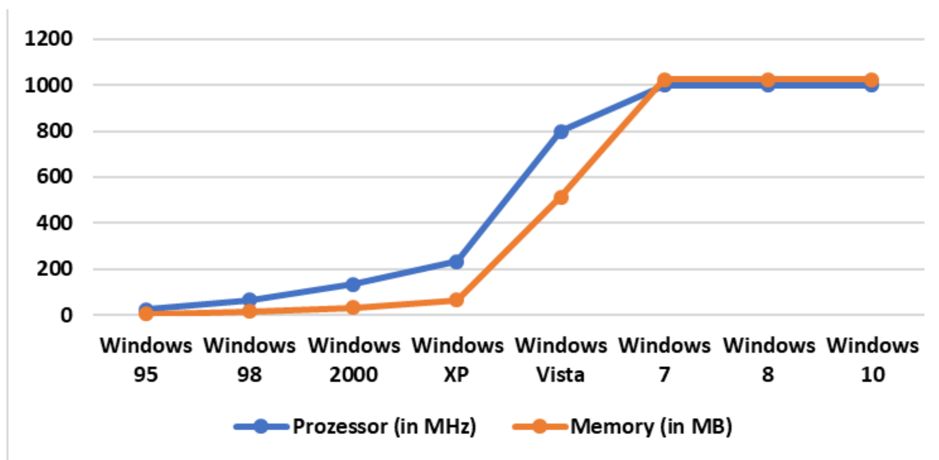
Windows Generation	Jahr der Veröffentlichung	Geschätzte Anzahl an Programmcode Zeilen	Hardware-Anforderungen
Windows 1.0	1985	100.000	
Windows 3.1	1992	3 bis 4 Millionen	x286 oder x386 CPU, 2-4 MB Arbeitsspeicher, 10-15 MB Festplattenspeicher
Windows 95	1995	11 Millionen	386DX CPU, 4 MB Arbeitsspeicher (8 MB empfohlen), 50-55 MB Festplattenspeicher

[21] Mit den steigenden Hardware-Anforderungen geht ein höher Ressourcenverbrauch, sowohl für die Herstellung von neuen Computern als auch für die benötigte Energie, einher.

Windows Generation	Jahr der Veröffentlichung	Geschätzte Anzahl an Programmcode Zeilen	Hardware-Anforderungen
Windows XP	2001	45 Millionen	Pentium 233 MHz CPU (300 MHz empfohlen), 64 MB Arbeitsspeicher (128 MB empfohlen), 1.5 GB Festplattenspeicher
Windows 7	2009	40-50 Millionen	1 GHz CPU, 1 GB Arbeitsspeicher (2 GB for 64-bit), 16-20 GB Festplattenspeicher
Windows 10	2015	Über 50 Millionen	1 GHz CPU, 1 GB Arbeitsspeicher (2 GB for 64-bit), 16-20 GB Festplattenspeicher
Windows 11	2021	Keine Quelle	64-bit CPU, 4 GB RAM or more, 64 GB Festplattenspeicher

Quellen: "Windows Secrets: Undocumented Features, Programming Secrets, and Troubleshooting Tips", Paul Thurrott, Windows 1.0, XP und 7; New York Times für Windows 3.1 und Windows 95[22], Windows Report für Windows 10[23]

Abbildung 13: Darstellung der Ressourcenanforderung von Windows-Versionen zwischen Windows 95 und 10



Greenpeace-Studie 2013: Wäre das Vista-Betriebssystem 2007 flächendeckend eingesetzt worden, hätten 50 % der Computer ausgetauscht werden müssen. Quelle: wikipedia.org/software_bloat; eigene Darstellung

Durch die Kopplung von Digitalisierung als Vorbedingung bzw. als Notwendigkeit für eine nachhaltige Wirtschaft, wie zum Beispiel die „Twin Transition“ Strategie der EU nahelegt[24], und die Bekämpfung der Klimakrise entsteht zunehmender Zeitdruck - es muss noch schneller, noch mehr Software hergestellt werden. Hinzu kommen wirtschaftliche Faktoren, wie zum Beispiel

[22] Artikel: https://www.nytimes.com/1995/07/31/business/microsoft-s-mobilization-overview-windows-of-opportunity-for-microsoft.html, abgerufen am 18. September 2023

[23] Artikel: https://windowsreport.com/windows-11-how-many-lines-of-code/, abgerufen am 18. September 2023

[24] siehe: https://ec.europa.eu/commission/presscorner/detail/en/ip_22_1467, abgerufen am 14. Februar 2024

die steigende Nachfrage nach Informatik-Experten und -Expertinnen und die steigenden Kosten für die Entwicklung und den Betrieb von komplexeren Software-Anwendungen.

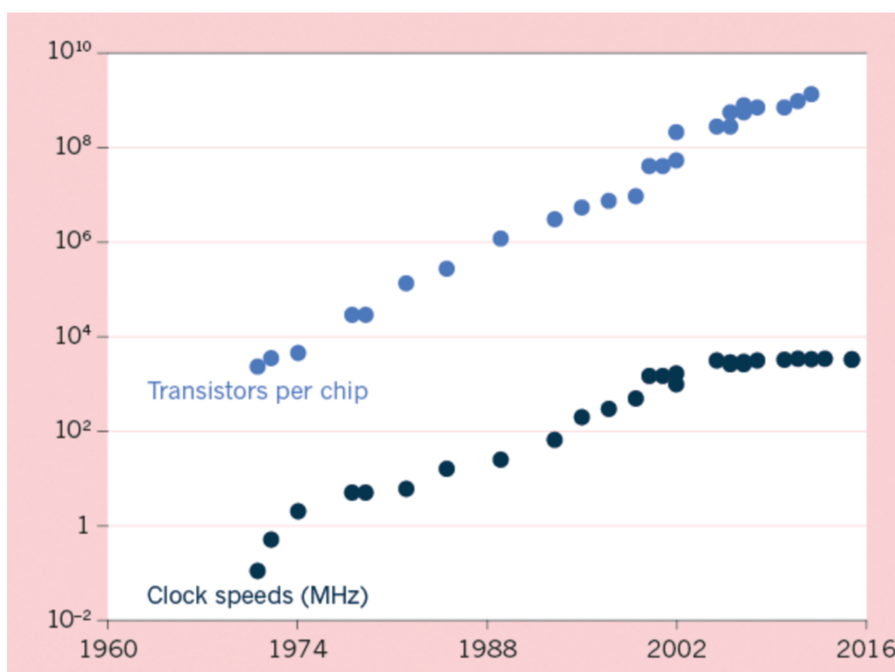
Auch die wachsende Digitalwirtschaft erhöht den Bedarf an zusätzlichen Software-Anwendungen. Durch das hohe Wirtschaftswachstum und eine hohe Rentabilität drängen mehr Unternehmen mit digitalen Produkten und Dienstleistungen in die Digitalwirtschaft. Dieses Wachstum führt zu einem größeren Softwarebedarf und einer stärkeren Nachfrage an digitalen Ressourcen und damit zu einem Ausbau von digitaler Infrastruktur. Der Branchenverband Bitkom prognostiziert für 2024 eine Verdopplung des Umsatzwachstums um 4,7% auf 223,2 Milliarden Euro (Müller 2023).

Das Paradigma lautet dabei nicht "Qualität", sondern "schnell auf den Markt", was sich beispielsweise durch das Mantra vom Unternehmen Meta (u.a. Betreiber von Facebook), einem der größten Akteure in der Digitalwirtschaft erkennen lässt: "move fast and break things"[25] [z. dt. "agiere schnell und mache Dinge kaputt"].

Sowohl die Digitalisierung als auch die Digitalwirtschaft sorgen dafür, dass die Menge an programmierter Software drastisch steigt. Die Software sorgt die für die Bereitstellung von Produkten und Dienstleistungen. Dabei steigt die Komplexität der Software - von einfacher Textverarbeitung bis hin zu komplexen Programmen, die fertige literarische Texte schreiben (Lex/GPT-3[26]).

Seit 1970 hat der zunehmende Ressourcenbedarf komplexer Software und das wachsende Datenvolumen eine Lösung in immer schnelleren Chips gefunden. Dieses Phänomen ist als „Moore's Law" bekannt, eine Vorhersage von Gordon Moore, einem Gründer von Intel. Er sagte voraus, dass sich die Anzahl der Transistoren in einem integrierten Schaltkreis fester Größe etwa alle 2 Jahre verdoppelt (Moore 1965).

Abbildung 14: Grafische Darstellung des Mooreschen Gesetzes



Quelle: Magazin „Nature" (Waldrop 2016).

[25] Wikipedia, Meta Platforms: https://en.wikipedia.org/wiki/Meta_Platforms#History, abgerufen am 18. November 2022

[26] OpenAI Foundation: https://openai.com/blog/gpt-3-apps/, abgerufen am 18. November 2022

Das Mooresche Gesetz hat laut Nvidia CEO Jensen Huang (Witkowski, n.d.) 2022 sein Ende gefunden, da es physikalisch nicht mehr möglich ist, die Anzahl an Transistoren die auf einem Schaltkreis platziert werden, zu erhöhen. Andere Vorhersagen (Kumar 2015; Theis and Wong 2017) schätzen ein Ende im Jahr 2025.

Das Mooresche Gesetz ist nicht als ein Naturgesetz zu verstehen, sondern eher als eine Faustregel, nach der sich die Digitalwirtschaft über Jahrzehnte hinweg entwickelt hat. Das Ende des Gesetzes bedeutet nicht, dass nicht weitere Effizienzgewinne realisierbar sind, sondern bedeutet lediglich, dass diese nicht durch Miniaturisierung erzielt werden können. Generelle Computer-Chips, wie sie heute in gängigen Laptops, Desktop-Computern, Telefonen, et cetera, eingesetzt werden, werden nicht schneller. Gleichzeitig steigt der Bedarf an Rechenleistung zunehmend, was zur Folge hat, dass der Bedarf an Infrastruktur steigt, was wiederum zu einem erhöhten Energie- und Rohstoffbedarf führt.

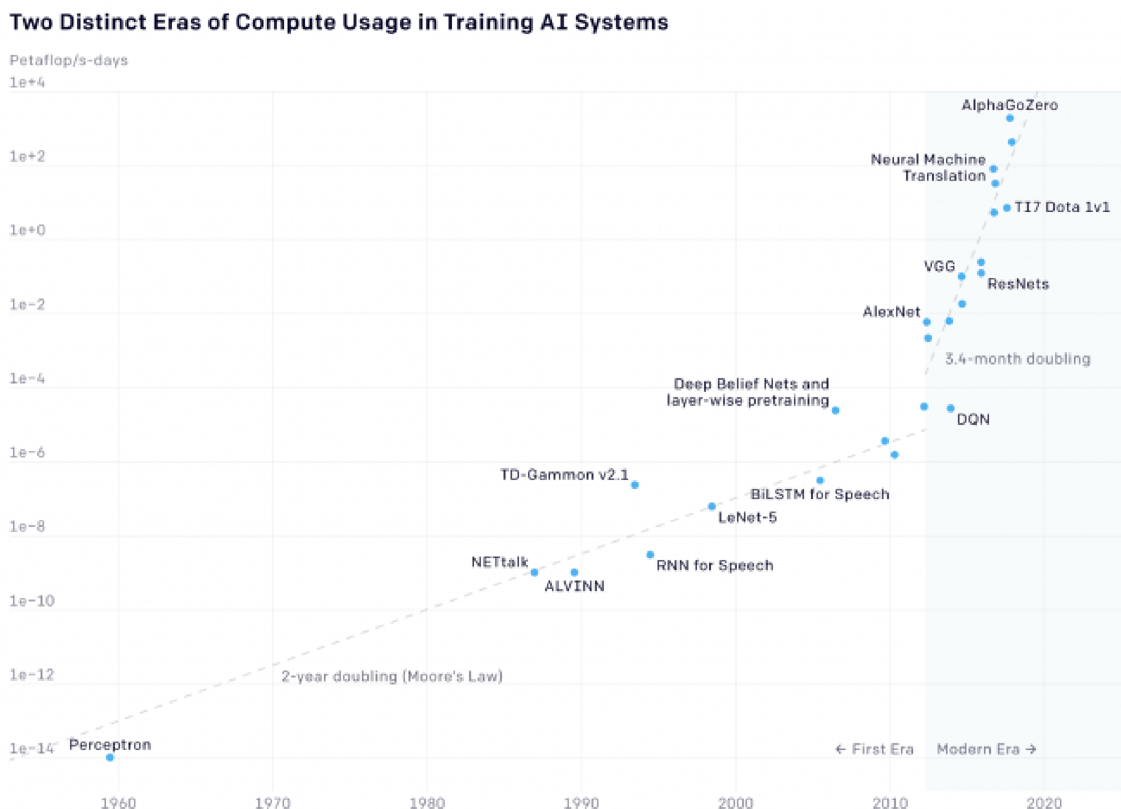
Chips haben nahezu das Maximum an Effizienz erreicht. Daher steigt die Notwendigkeit, Software selbst effizienter zu programmieren. Weltweit haben Forscher (Xu et al. 2010) bereits Aspekte identifiziert, die zur Reduzierung von Ressourcenanforderungen von Software führen können, zum Beispiel „Software Debloating“. Leiserson et al haben dazu aktuelle Ansätze in einer Studie zusammengetragen und argumentieren, dass diese insbesondere in Zusammenhang mit dem Ende vom Mooreschen Gesetz von zunehmender Relevanz sein werden (Leiserson et al. 2020).

Neue Technologien wie das maschinelle Lernen, Künstliche Intelligenz und Blockchain werden als Hoffnungsträger für neue Anwendungsfelder und Geschäftsmodelle angepriesen. Blockchain Technologien sollen bestehende komplexe Prozesse, wie zum Beispiel das Double Spending Problem (Chohan 2021) lösen und somit das Bank-System aufbrechen und schlussendlich ersetzen. Sie schaffen auch Möglichkeiten für eine Differenzierung von digitalen Produkten und Dienstleistungen durch neue Funktionalitäten, wie zum Beispiel der KI-gestützte Microsoft „Copilot“ für Microsoft Word[27].

Im Jahr 2018 hat die Organisation OpenAI, die im Jahr 2022 einen herausragenden wirtschaftlichen Erfolg mit dem Large-Language-Modell „ChatGPT“ verzeichnen konnte, bereits festgestellt, dass der Bedarf an Rechenleistung der größten KI-Modelle sich alle 3,4 Monate verdoppelt. Diese Entwicklung ist bereits seit dem Jahr 2012 festzustellen (Hao 2019) (siehe Abbildung 15). KI-Modelle können grundliegend auch als Softwareanwendungen angesehen werden, deren Ineffizienzen und riesigen Hardwareinanspruchnahmen dieselben erhöhten Energie- und Ressourcenverbräuche nach sich ziehen.

[27] siehe https://support.microsoft.com/en-us/copilot-word, abgerufen am 14.02.2024

Abbildung 15: Übersicht Bedarf an Rechenleistung von KI-Modellen



Quelle: MIT Technology Review, 2019 (Hao 2019)

Dieser stetige Anstieg in der Nachfrage nach Rechenleistung, Speicher- und Netzwerkkapazität kann langfristig nicht nur durch den Ausbau von digitaler Infrastruktur bedient werden. Laut der International Energy Agency[28] ist der weltweite jährliche Stromverbrauch der digitalen Infrastruktur von 424 TWh im Jahr 2015 auf 580-800 TWh im Jahr 2021 gestiegen. Das entspricht 1-1.5% des weltweiten Stromverbrauchs.

Bisherige Maßnahmen zur Steigerung der Energieeffizienz fokussieren sich auf Effizienzsteigerung der elektrischen und mechanischen Infrastruktur von Rechenzentren und der Software-Infrastruktur (Programmiersprachen, Virtualisierung, Betriebssysteme). Um weitere Effizienzpotenziale zu erschließen, muss auch ein Fokus auf die Effizienz von Software-Anwendungen, Werkzeugen und Komponenten gesetzt werden.

Experten der IT sind sich jedoch einig: „Software is eating the world“ [z. Dt. “Software frisst die Welt”]. Der Fokus muss auf die Qualität und Effizienz von Software gelegt werden (Conner-Simons 2020).

Der Herstellungsprozess von Software

Der typische Entwicklungsprozess von Software unterscheidet sich in drei Schritte, die für das Verständnis des Vorhabens von Relevanz sind:

- die lokale, individuelle Programmierung,

[28] IEA – International Energy Agency - Data Centres and Data Transmission Networks https://www.iea.org/reports/data-centres-and-data-transmission-networks, abgerufen am 18. September 2023

- die kollaborative Überprüfung der Anwendung und Zusammenführung der Arbeitsergebnisse („Integration“) (Soares et al. 2022),
- das Veröffentlichen der Ergebnisse an Nutzer und die Community der Entwickelnden (“Release”).

In der lokalen Entwicklung treffen Software Entwickelnde häufig unbewusst Entscheidungen, die einen direkten Einfluss auf den Ressourcenverbrauch der Anwendung haben. Eine dieser Entscheidungen ist die Auswahl von Bibliotheken die notwendig sind, um die funktionalen Anforderungen zu erfüllen. Des Weiteren entscheiden die Entwickelnden wie komplex bzw. fehlertolerant der Programmcode für die gewünschte Funktionalität sein muss (“Qualität”).

Entwickelnde entscheiden zudem ob im Rahmen des Projekts eine testgetriebene Entwicklung praktiziert wird (häufig handelt es sich um automatisierte Tests, die sicherstellen, dass die funktionalen Anforderungen erfüllt sind). Wird der Programmcode später einmal in einer Art verändert, welche indirekt zu einer Nichterfüllung der Anforderungen führt, würden die Tests in den meisten Fällen Alarm schlagen. Insbesondere in kollaborativen Arbeitsumgebungen, bei denen mehrere Entwickelnde am selben Programmcode arbeiten, sind solche Tests in der Praxis meist unverzichtbar.

In dieser Phase der Entwicklung bekommen die Software-Entwickelnden bereits verschiedene Signale zur Qualität, Komplexität und Geschwindigkeit des Programmcodes - je nachdem, wie die lokale Entwicklungsumgebung konfiguriert ist. Repräsentative Kennzahlen zum Energie- und Ressourcenverbrauch haben Entwickelnde typischerweise nicht, insbesondere nicht für die späteren Ressourcenverbräuche in der produktiven, realen, Laufzeitumgebung der Software.

Relevanz von Open-Source Bibliotheken

Ein Großteil von digitalen Produkten wird auf Basis von freien und quelloffenen (FOSS, free and open-source) Komponenten aufgebaut. Laut einer Umfrage von Black Duck Software, eine international renommierte Organisation, die Open-Source Systeme und Bibliotheken auf Sicherheitslücken überprüft, (aus dem Jahr 2016[29]) geben die befragten IT-Experten an, dass in ihrem Unternehmen mehr als 65% Open-Source Software in der Entwicklung der eigenen Produkte und Anwendungen zum Einsatz kommen.

Die potenziellen systemischen Verbesserungseffekte durch Effizienzverbesserung der Komponenten auf die gesamte Software-Landschaft sind enorm. Gängige Bibliotheken wie ReactJS werden 19 Millionen Mal[30] pro Woche heruntergeladen. In der öffentlichen Komponenten Datenbank von Java und Apache Maven wurden im Jahr 2021 mehr als 2 Millionen neue Pakete veröffentlicht. Insgesamt befinden sich in der Maven Datenbank 28 Terabyte[31] an Komponenten- und Bibliotheken-Software, allein für die Programmiersprache Java. Es wird deutlich, dass auch nur eine kleine Verbesserung des Energieverbrauchs sich auf Millionen von Anwendungen auswirken kann (siehe Abbildung 16).

[29] 2016 Future of Open-Source Survey Results, SlideShare Präsentation, https://www.slideshare.net/blackducksoftware/2016-future-of-open-source-survey-results, abgerufen am 22. November 2022

[30] Profil Seite für die Komponente „ReactJS“ im öffentlichen Paketregister NPM für die Programmiersprache JavaScript: https://www.npmjs.com/package/react, aberufen am 22. November 2022

[31] Maven Paketregister für die Programmiersprache Java: https://mvnrepository.com/repos/central, abgerufen am 21. November 2022

Abbildung 16: Beispielrechnung Drupal CMS

Um die möglichen Effekte einer Verbesserung des Programmcodes einer Komponente mit einer hohen weltweiten Verbreitung aufzuzeigen, wurde im Folgenden eine Beispielrechnung für das Content Management System (CMS) Drupal aufgestellt.

- Drupal ist eines der am Meisten eingesetzten CMS-Systeme für Internetseiten weltweit. Laut Statista[32] setzen rund 937.000 Webseiten das CMS ein.
- Im Jahr 2008 lag der durchschnittliche Energieverbrauch eines Servers bei 250 Watt.
- Man kann davon ausgehen, dass eine typische mit Drupal realisierte Internetseite im Schnitt einen Zehntel eines typischen Servers nutzt. Der Gesamtenergieverbrauch beträgt daher in etwa 205,2 Millionen kWh pro Jahr.
- Wird durch die Verbesserung des Programmcodes eine Reduktion des digitalen Ressourcenverbrauchs um 1% realisiert und an alle Drupal-basierten Webseiten aufgespielt, würde das einer Reduktion des Energieverbrauchs von ca. 2 Millionen kWh pro Jahr entsprechen.
- Das entspricht dem Jahresstromverbrauch von 1380 1-Personen-Haushalten in Deutschland.

Verantwortliche im Prozess der Software-Entwicklung

Wer ist verantwortlich für die Ressourceneffizienz einer Software-Anwendung, eines digitalen Produkts oder einer digitalen Dienstleistung? Es wäre einfach die Verantwortung auf die Software-Entwickelenden zu reduzieren.

Doch wie in den meisten Wertschöpfungsketten befindet sich die Umsetzungsrolle am Ende eines längeren Entscheidungsprozesses, in den viele andere Akteur*innen und Entscheider*innen involviert sind. Insbesondere die funktionalen und nicht-funktionalen Anforderungen werden nicht von den Entwickelnden selbst bestimmt. In der Digitalwirtschaft sind es oft Produkt Manager*innen, Analyst*innen, Projekt Manager*innen, oder Prozess-Ingenieur*innen.

In dem Zusammenspiel aus Anforderungsdefinition und Umsetzung fehlt jedoch die Einschätzung von Umweltwirkung und Ressourcenaufwand (abseits von zeitlichen Aufwänden der Mitarbeitenden und den damit verbundenen Kosten). So entsteht der Eindruck, dass einer Software unbeschränkt Funktionen hinzufügt werden können, ohne dass ein Anstieg von Energieverbrauch oder Umweltwirkungen entsteht. Nicht nur fehlt es an Werkzeugen zur Quantifizierung von Energie- und materiellen Ressourcenaufwänden, sondern auch eine fundamentale Veränderung der Kernprinzipien der Software-Gemeinschaft, angefangen bei der Definition der Anforderungen an die Software, welche um eine Spezifikation der Ressourcenaufwände (Energie- und Material) erweitert werden sollte.

Ausschlaggebend bei der Definition von Anforderung ist der Arbeitszeitaufwand der Software-Entwickelnden, da damit typischerweise der Großteil der Kosten verbunden ist. Diese einseitige Betrachtung vom Arbeitsaufwand der Software-Fachkraft im Vergleich zum Energie- und materiellen Ressourcenaufwand lässt sich z.B. auf die Unix Philosophie von Eric Raymond (Raymond 2008) und anderen zurückführen: "Value developer time over machine time" [Werte die Zeit der Fachkraft größer als die Zeit der Maschine]. Spart der Einsatz eines ineffizienten

[32] Umfrageergebnis auf Statista, https://de.statista.com/statistik/daten/studie/320912/umfrage/weltweite-anzahl-der-mit-drupal-erstellten-internetseiten/, abgerufen am 18. September 2023

Programmcodes also der Fachkraft Zeit, sorgt aber dafür, dass mehr Rechenleistung (Maschinen-Zeit) verbraucht wird, ist das laut dieser Philosophie akzeptabel und richtig. Übertragt man diesen Grundsatz auf den Energie- und Ressourcenverbrauch, so ist es akzeptabel, dass mehr Energie- und Server-Ressourcen verbraucht werden, solange es Entwicklungszeit einspart.

Wo dieses Prinzip im Jahr 2003, bei der Erscheinung von Eric Raymonds Buch, noch als sinnvoll erscheint, muss das Prinzip im Kontext von modernen Software-Anwendungen und digitalen Produkten überdacht werden. Das Unternehmen Meta, das Plattformen wie Facebook, Instagram und WhatsApp betreibt, hat allein im Jahr 2021 $19 Milliarden in Server und Rechenzentren investiert (Moss 2021), welche für den Betrieb von vier digitalen Produkten notwendig sind: Instagram, WhatsApp, Facebook, Oculus/Metaverse, im Vergleich zu ca. $9 Milliarden in Personalkosten (Eira 2020).

Es wird deutlich, dass die materiellen Ressourcenaufwände und Energieverbräuche in die Definition von Anforderungen einfließen und als Feedback-Schleife zwischen den Verantwortlichen in der Anforderungsdefinition und der Ausführung bzw. Umsetzung als Teil der Wertschöpfung verstanden werden sollten.

Ein zweiter großer Einfluss, insbesondere bei großen und komplexen Software-Applikationen oder einem Gesamtsystem, kommt aus dem Bereich der Software-Architektur. Die Rolle des*der Architekt*innen werden in der Digitalwirtschaft oft durch die Rolle des "Chief Technology Officers" [z. dt. Technischer Geschäftsführer] ausgeführt, im Umfeld der Digitalisierung und traditionellen Unternehmen gibt es ganze Abteilungen für "Enterprise Architecture" [z. dt. IT-Unternehmensarchitektur].

Die Rolle der Architektur ist insbesondere wichtig, da viele moderne Software-Anwendungen nicht von Grund auf neu erschaffen werden, sondern aus vielen, oft tausenden vorgefertigten Bauteilen (z.B. Bibliotheken für Standard-Funktionen wie Textmanipulation oder mathematische Funktionen, oder ganze Funktionsblöcke z.B. für die Verkleinerung von Bildern) bestehen. Softwarearchitektur beschreibt aber auch ganze Konzepte der Organisation von Systemkomponenten sowie der Beschreibung ihrer Beziehungen zueinander. Bei deren Auswahl dieser Bausteine ist die Architektur federführend.

Bei der Auswahl der Komponenten und Bibliotheken fließen viele Faktoren ein - von nicht-funktionalen Anforderungen wie zum Beispiel Modifizierbarkeit, Wartbarkeit, Geschwindigkeit oder Einfachheit der Implementierung (folgend dem Prinzip "Entwicklerzeit ist wertvoll"). Geschwindigkeit spielt eine wichtige Rolle, aber nicht auf die Wertschöpfung pro Ressourceneinheit bezogen (Effizienz), sondern auf die Geschwindigkeit der Bereitstellung einer Funktionalität. Wenn damit ein hoher Ressourceneinsatz verbunden ist, wird das oft durch neue Infrastruktur gelöst (folgend dem Prinzip "[Server-]Blech ist billig").

1.1 Zielsetzung

Die grundsätzliche Zielsetzung des Vorhabens ist es, Werkzeuge für den Prozess der Software-Entwicklung zu schaffen, die es Software-Entwickler*innen ermöglicht, die Energie- und Ressourcenverbräuche zu messen. Die Ergebnisse dieser Messungen zeigen mögliche Schwerpunkte innerhalb einer Anwendung, deren Anpassung zu einer Senkung der Umweltwirkung führt. Das Vorhaben stellt mit einem Leitfaden (siehe Anhang C.1) Ansätze und Vorgehensweise für solche Anpassungen zur Verfügung.

Der Fokus des SoftAWERE Vorhabens liegt in der Herstellungsphase von Software und dem Prozess einer Software-Anwendung – dem Feld der „Software-Entwicklung".

Für die Bereitstellung des Werkzeugs hat das Vorhaben das Ziel, eine Methodik zur Messung von Energie- und Ressourcenverbräuchen (siehe Kapitel 4) zu entwickeln, welche in modernen Entwicklungsumgebungen einsetzbar sein soll. Diese Umgebungen sind oft virtualisiert oder basieren auf Cloud-Infrastruktur und finden auf kollaborativen, von mehreren Entwickler*innen genutzten Arbeitsumgebungen statt. Das Werkzeug soll in zwei Bereichen des Entwicklungsprozesses zum Einsatz kommen:

- Im Entwicklungsprozess selbst, insbesondere im Schritt der Integration und Bereitstellung („Release") einer Software-Anwendung
- Im Zusammenspiel mit einer möglichen Kennzeichnung (siehe Kapitel 3.1) auch in der Entwicklung von quelloffenen Software-Komponenten, und deren Integration und Bereitstellung (Napoleão, Petrillo, and Hallé 2020), zum Beispiel um den durchschnittlichen Energieverbrauch der Komponente als Entscheidungskriterium für Entwickelnde sichtbar zu machen.

Um die im Rahmen des Vorhabens entstehenden Werkzeuge, Methoden und Handlungsempfehlungen der Software-Gemeinschaft zu vermitteln, wurden zudem virtuelle und hybride Veranstaltungsformate wie Workshops, Webinare und Hackathons durchgeführt (siehe Anhang B).

Werkzeuge zur Messung der Verbräuche in der Software-Entwicklung

Zusätzlich zur lokalen Entwicklungsumgebung, ist es für das SoftAWERE Vorhaben wichtig, die Indikatoren zum Energie- und Ressourcenverbrauch auch in der kollaborativen Phase der Entwicklung - der Zusammenführung der Code-Veränderungen und der Ausführung der Integrationstests auf einem gemeinsamen System - darzustellen. In dieser Phase haben auch die zwei anderen verantwortlichen Gruppen, die für die Architektur und die Anforderungsdefinition zuständig sind, Zugriff auf die Ergebnisse und können so den möglicherweise erhöhten oder gesunkenen Energie- und Ressourcenverbrauch überprüfen. Ziel des Vorhabens ist es, die Indikatoren in mindestens einer kollaborativen Arbeitsumgebung (z.B. Gitlab) zu integrieren und für Entwicklungsteams in der Open-Source Gemeinschaft zur Verfügung zu stellen.

Das Vorhaben liefert fehlende Werkzeuge und Integration in Entwicklungsprozess

Zusammenfassend liefert das SoftAWERE Vorhaben Werkzeuge und Indikatoren für die Messung von Energie- und Ressourcenverbräuchen für Open-Source Bibliotheken und Komponenten, Leitfäden und Handlungsoptionen für mögliche Verbesserungen sowie ein Konzept für eine mögliche Kennzeichnung von energie- und ressourceneffizienten Bausteinen.

Das Vorhaben überprüft und validiert die Genauigkeit der Messmethodik, indem es die entwickelten Werkzeuge in die Integrationstests von jeweils zehn populären Bibliotheken und Komponenten aus den sechs gängigsten Programmiersprachen implementiert. Die Implementierung des Testaufbaus wird im Anschluss des Vorhabens den Verwaltern der Bibliotheken und Komponenten übergeben und dienen als Demonstrationen für andere Entwickler*innen.

Das SoftAWERE Projekt liefert der Open-Source Gemeinschaft drei konkrete Werkzeuge, die in gängige Arbeitsabläufe integriert werden können. Diese Werkzeuge sind selbst auch als Open-Source verfügbar, können also fernab des Vorhabens weiterentwickelt und verbessert werden:

- Ein lokales Arbeitswerkzeug, mit dem der Energie- und Ressourcenverbrauch von Tests gemessen und angezeigt werden kann.

- Eine Integration in die gängige Release-Prozess-Automatisierungsplattform "Gitlab" und "GitHub", um die Messung der Energieverbräuche der Integrationstests auch dort durchzuführen und sichtbar zu machen.
- Eine grafische Darstellung für die Software-Anwendung, die die Ergebnisse der Messung der Integrationstests als historische Entwicklung darstellt. Die grafische Oberfläche kann auch für das Energie-Monitoring im Betrieb der Software eingesetzt werden, ist aber nicht Teil des eigentlichen SoftAWERE Vorhabens.

Eine Transparenz-Kennzeichnung für Open-Source Projekte

Gemeinsam mit der Open-Source Gemeinschaft und dem Projekt-Partner Öko Institut e.V., eruiert das SoftAWERE Team die Möglichkeit einer Kennzeichnung für Open-Source Bibliotheken und Komponenten hinsichtlich des Energieverbrauchs. Ziel dabei ist es, der Entwickler*Innen-Community eine Möglichkeit zu geben, die Effizienz von Bibliotheken, Werkzeugen und Bausteinen gegenüber der Entwickler-Gemeinschaft transparent zu machen.

Zusätzlich schafft eine Kennzeichnung möglicherweise Anreize für die Hersteller der Bibliotheken und Komponenten die Energie- und Ressourceneffizienz zu erhöhen. Ein Großteil der weltweiten Software-Bibliotheken wird von Freiwilligen als quelloffene (open-source) Software hergestellt und gewartet (die FOSS bzw. Open-Source Gemeinschaft). Im Zusammenspiel mit den im Vorhaben entwickelten Mess- und Arbeitswerkzeugen, Handlungsempfehlungen und einer Kennzeichnung wurde der FOSS Gemeinschaft Möglichkeiten und Anreize bieten, den Energie- und Ressourcenverbrauch zu senken.

Mit dem Blauen Engel für Software (Gröger et al. 2018), gibt es bereits eine Kennzeichnung, die von der FOSS-Community, z.B. für KDEs Okular[33], angenommen und implementiert wurde.

1.2 Untersuchungsrahmen

Das primäre Ziel des Vorhabens ist es, ein Werkzeug für Software-Entwickelnde zu entwickeln, welches den Energie-Verbrauch und die Umweltwirkungen einer typischen Software-Anwendung schon im Entstehungsprozess messen kann. Die Informationen, die durch das Werkzeug bereitgestellt werden, können durch Entwickelnde in den Entscheidungsprozess der Programmierung miteinbezogen werden und können im Zusammenspiel mit Handlungsempfehlungen aus diesem Vorhaben zu ressourcenschonenderer Software führen.

Das Vorhaben versucht nicht eine Vergleichbarkeit zwischen Software-Anwendungen zu ermöglichen. Nur wenige Software-Anwendungen sind funktional identisch und können verglichen werden. Ein Vergleich auf Funktionsebene (z.B. Speichern eines Text-Dokuments in einer Textverarbeitungssoftware) ist sinnvoll und denkbar, jedoch nicht Teil des Vorhabens.

Stattdessen wird der Schwerpunkt auf die schnelle Wiederholbarkeit und einfache Anwendbarkeit des Werkzeugs und der Messung gelegt, auch wenn dabei Einbußen hinsichtlich der Genauigkeit in Kauf genommen werden müssen. Da der Energie- und Ressourcenverbrauch in gängigen Software-Prozessen nicht als Qualitätskriterium erfasst wird, ist das aus Sicht des Forschungsteams wichtiger, entsprechende Metriken in Entwicklungsprozesse zu integrieren und sichtbar zu machen und erst in einem zweiten Schritt an Vergleichbarkeit von Software-Anwendungen zu forschen.

[33] KDE-Okular wurde mit dem Blauen Engel für Software ausgezeichnet: https://www.blauer-engel.de/de/produkte/kde-okular

Für die Entwicklung und Implementierung der Werkzeuge werden freie und quelloffene Software-Komponenten als Test-Subjekte genutzt. Diese Komponenten haben nützliche Eigenschaften für die Erprobung von Messmethoden und Werkzeugen:

- **Systemrelevant:** Populäre Komponenten und Bibliotheken finden sich in den meisten modernen Software-Anwendungen (siehe 1.1 Hintergrund), eine 1%-ige Senkung des Energieverbrauchs kann bereits weitreichende Effekte haben.
- **Kollaborativ:** Alle quelloffenen Komponenten werden in kollaborativen, offenen Prozessen weiterentwickelt, in denen alle Entwickelnden neue Verbesserungen und Änderungen beitragen können. Zur Qualitätssicherung verfügt ein Großteil der offenen Komponenten über automatisierte Tests. Diese Tests führen die Funktionalitäten der Komponente in isolierten Umgebungen aus.
- **Offen und veränderbar:** Durch die Open-Source-Lizenzen der Komponenten ist es für das Forschungsvorhaben möglich, Änderungen vorzunehmen und Experimente durchzuführen, ohne dabei etwaige Rechte zu verletzen.
- **Kontinuierliche Integration und Tests:** Viele Komponenten verfügen über automatisierte Prozesse, um bei jeder Veränderung die Tests zur Prüfung der Funktionalitäten auszuführen und neue Versionen zu kompilieren bzw. bereitzustellen.

Eine vollständige Übersicht der Auswahlkriterien für die Testsubjekte und eine Liste der ausgewählten Testsubjekte sind im Anhang C.2 aufgeführt.

Lebenszyklusperspektive: Herstellung und Auslieferung

Die Lebenszyklusanalyse ist eine Analysetechnik um die Umweltauswirkungen, die mit allen Lebensphasen eines Produkts verbunden sind, d. h. von der Rohstoffgewinnung über die Materialverarbeitung, die Herstellung, den Vertrieb und die Nutzung, zu bewerten (Muralikrishna and Manickam 2017).

Software-Anwendungen können gleichermaßen mit den Phasen aus der Lebenszyklusanalyse betrachtet werden, jedoch ist eine andere Benennung der Phasen sinnvoll, um den Zusammenhang mit dem Prozess der Software-Herstellung herzustellen.

- **Software-Entwicklung (LCA-Phase: „Herstellung“):** Der typische Prozess der Software-Entwicklung, die Programmierung und die dazugehörigen Werkzeuge (Entwicklungsumgebungen) und Geräte (Laptop oder Desktop Computer, Internetanschluss).
- **Deployment (LCA-Phase: „Auslieferung“):** Eine typische Software-Anwendung wird entweder für Desktop-Geräte ausgeliefert, als installierbares Paket, oder als Online-Anwendung, die in Echtzeit über eine Internetverbindung bereitgestellt wird.
- **Software Usage (LCA-Phase: „Nutzung“):** Eine anwendende Person oder eine Maschine nutzt die Software-Anwendung als Produkt oder Dienstleistung auf einem eigenen Gerät. Dabei entsteht Energie- und Ressourcenverbrauch auf der Nutzungsseite.
- **Decommissioning (LCA-Phase: „Entsorgung“):** Die Software wird vom Anwender entfernt. Daten, die durch die Anwendung gespeichert, wurden und Programmcode wird sowohl von Servern als auch Speichersystemen entfernt.

Für das Forschungsvorhaben wurde sich nur auf die Betrachtung der Herstellungsphase, den Prozess der Software-Entwicklung und eine simulierte Nutzungsphase beschränkt. Um die

Nutzungsphase zu simulieren, werden die Funktionstests der untersuchten Software-Komponenten genutzt.

Der geografische Fokus liegt auf Europa

Für die Berechnung von Umweltwirkungen wird im Forschungsvorhaben der geografische Fokus auf Europa gelegt. Als Rechenzentrumsstandort für das Test-Labor wurde die Niederlande gewählt. Dieser Standort wurde gewählt, da dort sowohl wiederaufbereitete Server-Hardware zum Einsatz kommt als auch die Abwärme in einem Gewächshaus genutzt wird.

Für die Berechnung der CO2-Emissionen für den gemessenen Energieverbrauch wird der entsprechende Wert der CO2-Emissionen pro kWh der European Energy Agency (EEA) Datenbank (‘Greenhouse Gas Emission Intensity of Electricity Generation in Europe’, n.d.) genutzt. Die Umrechnungsfaktoren für die EU-Mitgliedstaaten sind im Anhang C.10 aufgeführt.

1.3 Voraussetzung und bestehende methodische Ansätze

Methodisch setzt das Vorhaben auf den bisherigen Arbeiten des Umweltbundesamtes auf. Insbesondere auf den Arbeiten des Umweltcampus Birkenfeld und dem Öko-Institut im Bereich der Kriterienentwicklung für nachhaltige Softwareprodukte. In einem Papier aus dem Jahr 2018 (Kern et al. 2018) haben die Forscher*innen einen methodischen Ansatz für die Anwendung von standardisierten Lebenszyklusanalyse-Methoden (LCA) zur Messung der Energie- und Hardwareflüsse von Software erarbeitet.

Die Autor*innen haben für jeden Indikator ein Messverfahren festgelegt, um die Praktikabilität der Kriterien zu bewerten. Anschließend führten sie Fallstudien mit 11 Softwareprodukten aus den Produktgruppen "Textverarbeitungsprogramme", "Webbrowser", "Content-Management-Systeme" und "Datenbanksysteme" durch. Diese Softwaretypen repräsentieren drei verschiedene Software-Architekturmuster: lokale Anwendungen, Anwendungen mit entfernter Datenverarbeitung und Serveranwendungen. Die Auswahl spezifischer Produkte für die Fallstudien basierte auf Faktoren wie hoher Installations- oder Benutzeranzahl und langer Nutzungsdauer (Gröger et al. 2018).

Das Vorhaben baut auf dieser Forschungsarbeit auf, vereinfacht aber den Ansatz durch einige maßgebliche Weiterentwicklungen mit der Zielsetzung, eine Messung während des Entwicklungsprozesses zu ermöglichen:

- Für die Messung der Energieflüsse wird kein externes Strommessgerät benötigt, sondern es werden Hardware-Schnittstellen wie IPMI und RAPL verwendet (siehe Kapitel 2)
- Das Nutzungsszenario muss nicht manuell definiert und konfiguriert werden, sondern kann in Form von gängigen automatisierbaren Test-Umgebungen definiert und ausgeführt werden. Dazu zählen unter anderem Selenium[34] für Web-Browser-basierte Test-Abläufe oder Junit[35] für einfache funktionale Tests.

[34] Selenium ist ein Werkzeug, um die Ausführung von Nutzungsszenarien in Browser-Anwendungen zu programmatisch zu automatisieren: https://www.selenium.dev/

[35] JUnit ist ein Test-Baukasten mit denen Software-Funktionen programmatisch ausgeführt und die Ergebnisse überprüft werden können. Es existieren entsprechende äquivalente Werkzeuge in allen gängigen Programmiersprachen, siehe: https://junit.org/junit5/

- Die Bestimmung der Hardwareflüsse und damit verbundenen Umweltwirkungen ist automatisiert und muss aufgrund der benutzten Hardwareschnittstellen nicht manuell für jeden Server-Typ ermittelt werden.
- Die Ausführung der Energie- und Ressourcenmessungen ist als Teil eines typischen Software-Entwicklungsprozesses automatisiert und wird bei jeder signifikanten Veränderung des Programmcodes ausgeführt[36].
- Die Auswertung der Umweltwirkungen und Ressourcenverbräuche erfolgt automatisch und jede Ausführung wird aufgezeichnet. Das ermöglicht Entwickelnden die Veränderung der Umweltwirkungen über den Entwicklungszeitraum zu analysieren.

Voraussetzungen

Um den methodischen Ansatz des Vorhabens zu erproben und die entsprechenden Werkzeuge zu entwickeln, wurde ein Labor bzw. Testaufbau realisiert. Das Labor wurde als Open-Source Software veröffentlicht[37].

- Das Server-System, auf dem die Messung aufbaut, muss über eine IPMI-Schnittstelle verfügen, die über nicht-proprietäre Schnittstellen angesprochen werden kann. Ein gängiges Werkzeug aus der UNIX-Umgebung ist das Werkzeug „GNU Free IPMI“[38].
- Das Server-System muss im „Power Saving Mode“ konfiguriert werden, damit die CPU bei geringer Auslastung einen messbaren, geringeren Stromverbrauch hat[39]. Server, welche im „High Performance Mode“ konfiguriert sind, können unter Umständen (‘Energy-Efficient Design of Data Centres Using Power Management and Virtualisation (LEAP)’, n.d.) keine messbaren Änderungen im Stromverbrauch bei Veränderung der CPU-Auslastung darstellen.
- Im Server-System muss zwingend eine Intel CPU verbaut werden, die über die Intel RAPL-Schnittstelle verfügt (Khan et al. 2018).
- Das Server-System muss mit einer Linux Distribution betrieben werden, die Container-Technologie unterstützt, im Falle des Vorhabens Docker & Docker Compose[40].
- Die Architektur des Labors nutzt eine quelloffene CI/CD Ausführungsumgebung, den „GitLab Runner“[41]. Dieser wird als Teil der Docker-Compose Architektur gestartet. Um dem GitLab Runner Software-Projekte zur Ausführung (und Messung) zu geben, muss er an eine entweder private[42] oder öffentliche GitLab Instanz[43] angeschlossen werden.

[36] In typischen Software-Entwicklungsumgebungen wird eine Versionskontrolle wie Git oder SVN eingesetzt – laut einer Umfrage von Stack Overflow (‘Stack Overflow Developer Survey 2021’, n.d.) verwenden 90% aller Befragten Git als Versionskontrolle. In diesen Versionskontrollsystemen ist es typisch, lokale Änderungen durchzuführen und diese dann bei einem Zwischenergebnis in die Versionskontrolle einzuchecken. Dieser Vorgang wird als „Push“ bezeichnet. In gängigen Entwicklungsprozessen führt ein „Push“ von einem Entwickler zum Start von CI/CD Prozessen, welche wiederrum die funktionalen Tests ausführen, und damit sicherstellen das die Veränderungen der entwickelnden Person nicht zu neuen Fehlern in der Anwendung führen.

[37] Siehe https://gitlab.opencode.de/OC00004041236/softawere

[38] siehe https://www.gnu.org/software/freeipmi/

[39] Ein Beispiel für so eine Spezifikation findet sich in der „OCP Leopard v3 Spec“ im Abschnitt 4.3.1 in der explizit gefordert wird das, dass BIOS von Herstellern so konfiguriert werden muss das der Energieverbrauch minimiert wird. Siehe https://www.opencompute.org/documents/facebook-server-intel-xeon-mb-v30, abgerufen am 17. Oktober 2023

[40] Siehe https://docs.docker.com/compose/install/, abgerufen am 17. Oktober 2023

[41] Siehe https://docs.gitlab.com/runner/

[42] Wie eine private GitLab Umgebung installiert werden kann, findet sich in der Dokumentation von GitLab: https://docs.gitlab.com/ee/install/install_methods.html

[43] Ein Beispiel ist GitLab.com oder OpenCode.de welches von Komm.ONE bereitgestellt wird.

2 Methodische Herausforderungen und Lösungsansätze

Den Energieverbrauch der funktionalen Tests während der Ausführung zu messen und daraus repräsentative Indikatoren für den Energieaufwand einer Funktion der Software-Anwendung zu liefern, ist eines der Kernansätze des Vorhabens. Dieser Ansatz basiert auf dem vorhergehenden Projekt des Umweltbundesamts in Zusammenarbeit mit dem Öko-Institut und dem Umwelt-Campus Birkenfeld (Gröger et al. 2018).

Wenn dieselbe Herangehensweise für z.B. Open-Source Bibliotheken für Java angewendet werden soll, so ist die Menge an verfügbaren, quelloffenen Bibliotheken bereits ein Hindernis. Allein in der zentralen Bibliothek für Java Komponenten liegen 9 Millionen Softwarepakete[44], für die jeweils ein Nutzungsszenario zu definieren wäre.

Für jedes funktionsgleiche Paket ein manuelles Nutzungsszenario zu definieren, ist ein langwieriger Prozess. Jedoch verfügt ein Großteil der Pakete über automatisierte Nutzungsszenarien in der Form von Unit- und Integrationstests, die zwar nicht miteinander vergleichbar sind, jedoch eine adäquate Möglichkeit bieten, den Energie- und Ressourcenaufwand pro Funktionalität indikativ zu messen und darzustellen.

2.1 Methodische Herausforderungen

Um die Energie- und Ressourcenaufwände von Software in gängigen Software-Entwicklungsprozessen, die mit Container- und Virtualisierungsumgebungen arbeiten, zu ermitteln und darzustellen, müssen einige Herausforderungen adressiert werden:

- Alle Schnittstellen zur Messung der Energie von Server-Systemen sind nur auf der „root“ (Administrator) Ebene zugänglich. Werden, wie üblich, Container- oder Virtualisierungstechnologien eingesetzt, ist ein Zugriff auf die Schnittstellen nicht möglich. Jeder Container bzw. virtueller Server wird vom Hauptsystem aus Sicherheitsgründen isoliert.
- Damit die Ressourcenaufwände des Server-Systems, z.B. aus der Herstellung des Servers, bestimmt werden können, muss sowohl der Herstellername als auch die Systemkonfiguration (Arbeitsspeicher, CPU-Typ, Festplattenspeicher- und Typ) bekannt sein. In virtualisierten Umgebungen sind diese Informationen durch die Sicherheitsisolierung nicht verfügbar.
- Die Vergleichbarkeit der Ergebnisse kann nicht gegeben sein, da die Funktionalitäten und Test-Abdeckung der untersuchten Software-Bibliotheken nicht identisch sind.
- CI/CD-Vorgänge werden bei jeder signifikanten Veränderung des Programmcodes ausgeführt, bei großen Projekten bis zu mehreren hundert Durchläufen pro Tag. Die Methode selbst sollte performant sein und nicht zu einer signifikanten Verlängerung der Ausführungszeit führen.

Typische Software-Entwicklungsprozesse, insbesondere von Open-Source Bibliotheken nutzen Container, um die Entwicklungs- und Ausführungsumgebung zu isolieren und zu verpacken. Die Vorteile der Containerisierung hat Donald Firesmith, Mitarbeiter des „Software Engineering Institute“ an der Carnegie Mellon Universität, in einem Artikel beschrieben (Firesmith 2017). Dazu gehören unter anderem der reduzierte digitale Ressourcenverbrauch, der durch die

[44] Maven Repository, eine von mehreren zentralen Paketverzeichnissen für Java Software: https://mvnrepository.com/repos/central (Abruf: 22 November 2022)

weniger komplexe Virtualisierungstechnologie zu Stande kommt und die Vermeidung von doppelten Betriebssystemen wie bei vielen vollisolierten Virtualisierungstechnologien der Fall ist. Insbesondere die Vorteile der Containerisierung für Software-Tests beschreibt Merkel (Merkel 2014). Es wird deutlich, dass Container-basierte Test- und Entwicklungsumgebungen insbesondere für die Isolierung (Ammann and Offutt 2016) einer Applikation wichtig sind.

Davon lassen sich folgende relevante Anwendungsfälle ableiten, die für das Vorhaben von besonderer Bedeutung sind:

- **Virtualisierte Entwicklungs- bzw. Testumgebungen**, sowohl in modernen Cloud-Infrastrukturen als auch in privaten Unternehmensumgebungen (z.B. VMware aber auch KVM, oder OpenStack-basiert).
- **Containerbasierte Umgebungen** die ähnlich zu virtualisierten Umgebungen keinen Zugriff auf das darunterliegende Computer System haben, um entsprechende Schnittstellen zum Energieverbrauch abzufragen. Die containerbasierte Ausführung von Tests ist insbesondere in modernen kollaborativen Test- und Arbeitsumgebungen von Software-Entwicklungsteams üblich (Merkel 2014).

Für die Messung von Energieverbräuchen zur Laufzeit einer Software-Entwicklung sind insbesondere die folgenden Schnittstellen relevant:

- **Intel RAPL -** eine Schnittstelle, die von Intel für die Messung von Energieverbräuchen der CPU für Forschungszwecke bereitgestellt wird und auf den üblichen Linux-Server Systemen verfügbar ist. Nachteil dieser Schnittstelle ist, dass sie nur die Stromverbräuche der CPU liefert und nicht die Verbräuche des gesamten Systems[45] ermittelt.
- **Intelligent Platform Management Interface (IPMI)[46] -** eine Schnittstelle (Kavanagh, Armstrong, and Djemame 2016; Minyard 2006), die von einigen Server-Systemen, die über einen speziellen Baseboard Management Controller[47] (BMC) verfügen, bereitgestellt wird. Über die Schnittstelle können Stromverbräuche des Gesamtsystems ermittelt werden. Der BMC läuft "out-of-band"[48], misst also das Gesamtsystem, ohne sich selbst zu messen. Die Verfügbarkeit der IPMI-Schnittstelle variiert je nach Hersteller. Auch diese Schnittstelle erfordert Systemzugriff.

Keine der untersuchten Schnittstellen ist konsistent in allen Laufzeitumgebungen verfügbar. Hinzu kommt die Beschränkung auf Umgebungen, in denen ein Systemzugriff möglich ist. In vielen modernen Entwicklungs- und Testumgebungen ist dies nicht der Fall. Diese sind entweder virtualisiert (Microsoft Azure CI, Amazon AWS Amplify) oder nutzen eine containerbasierte Isolierung der Tests bei der Ausführung (bspw. GitLab, GitHub, CircleCI).

Durch diese Ausgangslage hat sich das Forscherteam für die Entwicklung einer Annährungsmethodik entschieden (siehe Kapitel 2.2). Mit dieser Methodik wird der Energie- und Ressourcenverbrauch anhand der Auslastung des Gesamtsystems und der eingesetzten Komponenten mathematisch errechnet. Dieser Ansatz ist hilfreich, wenn kein Systemzugriff auf

[45] Es fehlt der Energieverbrauch von Arbeitsspeicher, Festplatten Lese- und Schreibvorgängen, Netzwerkverkehr und etwaiger Zusatzkarten wie Grafikkarten.

[46] Die IPMI-Spezifikation ('Intelligent Platform Management Interface Specification Second Generation' 2015) wurde zuletzt in 2015 aktualisiert. Mittlerweile wird für IPMI ein Nachfolger (RedFish) entwickelt, siehe https://www.intel.com/content/www/us/en/products/docs/servers/ipmi/ipmi-home.html

[47] Für eine visuelle Darstellung des BMC, siehe https://en.wikipedia.org/wiki/Intelligent_Platform_Management_Interface#IPMI_components

[48] Siehe https://en.wikipedia.org/wiki/Out-of-band_management#Implementation

die IPMI oder RAPL-Schnittelle möglich ist. Diese Methodik ist als Sekundärlösung zu verstehen – das entwickelte Werkzeug wird, falls möglich, immer zuerst auf IPMI oder RAPL-Schnittstellen zurückgreifen. Ist dies nicht möglich wird die Annäherungsmethodik, auf Basis der mathematischen Berechnungen von Tom Kennes, verwendet.

2.2 Methodenentwicklung

Wie in Kapitel 1.3 beschrieben, wird der methodische Ansatz für die Anwendung von standardisierten Lebenszyklusanalyse-Methoden (LCA) zur Messung der Energie- und Hardwareflüssen von Software vom Öko-Institut (Gröger et al. 2018; 2021) als Grundlage verwendet.

Anstatt manuell definierte Nutzungsszenarien, wie in dem von Jens Gröger und Kollegen entwickelten methodische Ansatz, für die Messung zu nutzen, werden in diesem Vorhaben automatisierte, wiederholbare Szenarien verwendet - die Unit- und Integrationstests („Tests“) der untersuchten Software-Komponenten. Während der Ausführung dieser Tests wird der Energieverbrauch gemessen.

Die These dieses Ansatzes ist, dass die Ausführung der Tests äquivalent zu der Nutzung der Bibliothek oder Komponente von einem Nutzer im Kontext einer Software-Anwendung ist. Die Tests werden geschrieben, um gleichbleibende Funktionalität[49] bei veränderten Implementierungen sicherzustellen („Integrität“). Davon lässt sich schlussfolgern, dass sich die Ergebnisse der Messung für ein Test-Szenario prinzipiell extrapolieren lassen und als Nutzungseinheit (bzw. Standardnutzungsszenario) betrachtet werden können. Wird für das Test-Szenario der Energie- und Ressourcenaufwand bestimmt, so lässt sich dieser pro Nutzungseinheit hochrechnen. Hat eine Anwendung z.B. 100 Nutzer pro Tag und eine Nutzungseinheit hat einen Energieaufwand von 1 kWh, so kann geschlussfolgert werden, dass 100 kWh (100 Nutzer multipliziert mit 1 kWh/Nutzungseinheit) an Energieaufwand pro Tag entstehen. Somit ist es möglich, für die Nutzung einer Software eine indikative Aussage zum Energie- und Ressourcenaufwand, der durch Nutzung entsteht, zu treffen.

Beispiel für die Anwendung der These

Wird eine Komponente in eine Software verbaut, so nutzt diese Software in einem idealisierten Szenario 100% der Funktionen der Komponente. Das bedeutet, dass die Software auch 100% der verfügbaren Schnittstellen („APIs“ – Application Programming Interfaces) der Komponente anspricht. Dadurch wird ein Großteil des Programmcodes der Komponente ausgeführt.

Durch die Tests wird der Einbau der Komponente simuliert, in dem alle Schnittstellen künstlich (in Integritäts-Tests) angesprochen werden. Hat die Komponente eine 100%ige Testabdeckung[50], so lässt sich ableiten, dass die Tests einer realen Implementierung der Komponente in Software nahekommen. In beiden Fällen wird nahezu jede Zeile Programmcode ausgeführt.

Wird während der Ausführung der Tests nun sowohl der Verbrauch von digitalen Ressourcen als auch der Energieverbrauch gemessen, lässt sich daraus der Ressourcenaufwand pro Nutzungseinheit ableiten. Die Nutzungseinheit ist hier: 1 Implementierung der Komponente in einer Anwendung und Nutzung von einem Großteil der bereitgestellten Funktionalitäten oder Schnittstellen.

[49] Im Falle von Software-Komponenten insbesondere eine konsistente API bzw. Konformität mit einer bestehenden Spezifikation.

[50] wie z.B. die Express Komponente für JavaScript, welche auch in dem Vorhaben untersucht worden ist: https://coveralls.io/github/strongloop/express

Die Ergebnisse können bei einer hohen Testabdeckung als zuverlässiger Indikator für die Nutzung der Komponente in einer Software betrachtet werden. Die Ergebnisse sind nicht untereinander vergleichbar, geben jedoch einen Hinweis zur Energie- und Ressourceneffizienz der Komponente.

Komponenten sind in sich selbst keine Software-Anwendung, sondern Bausteine, aus denen Anwendung erschaffen werden können und die Programmierung durch wiederverwendbare Komponenten vereinfachen. Um Rückschlüsse auf den Energie- und Ressourcenverbrauch der Software-Anwendung treffen zu können, bietet diese im Beispiel beschriebene indikative Herangehensweise über die Integritätstests der Komponenten die beste Möglichkeit einer Annäherung.

Eine Bestätigung der Praxistauglichkeit dieser These findet sich schnell, wenn man die populärsten Komponenten untersucht. Auf der populären Plattform GitHub findet man bei einer manuellen Überprüfung, dass mehr als 80% der Top 20 Projekte über Funktions- oder Integrationstests verfügen[51]. Ein Beispiel ist das populäre Content Management System Drupal, welches in der Version 9.5 mehr als 28.000 funktionale Tests verfügt[52].

Testgetriebene Entwicklung (TDD) (Janzen and Saiedian 2005) ist eine entscheidende Praxis in der Softwareentwicklung und beinhaltet das Schreiben von Tests, bevor der eigentliche Code geschrieben wird. Zwei grundlegende Arten von Tests, die in TDD verwendet werden, sind Unit-Tests und Integrationstests, die jeweils unterschiedliche Zwecke erfüllen.

Unit-Tests sind darauf ausgelegt, die kleinsten Codeeinheiten wie Funktionen, Methoden oder Klassen isoliert zu überprüfen. Diese Tests werden konstruiert, um einzelne Komponenten unabhängig von ihren Abhängigkeiten zu bewerten (Khorikov 2020). Während Unittests werden häufig simulierte Objekte („Mocks“, „Stubs“) verwendet, um diese Abhängigkeiten zu ersetzen und sicherzustellen, dass der Fokus ausschließlich auf der zu prüfenden Einheit liegt. Das Hauptziel von Unit-Tests besteht darin, zu überprüfen, ob jedes kleine Codefragment sich korrekt verhält, wenn es bestimmte Eingaben erhält. Auf diese Weise tragen sie dazu bei, Probleme und Regressionen auf niedrigster Ebene der Software zu identifizieren und zu beheben. Unit-Tests zeichnen sich durch ihre schnelle Ausführung aus, da sie keine komplexen Interaktionen mit externen Systemen beinhalten.

Im Gegensatz dazu haben Integrationstests einen breiteren Anwendungsbereich und bewerten die Interaktionen zwischen verschiedenen Komponenten oder Modulen, wenn sie in das System als Ganzes integriert sind (Alsaqqa, Sawalha, and Abdel-Nabi 2020). Anders als Unittests involvieren Integrationstests echte Abhängigkeiten wie Datenbanken oder Netzwerkdienste, was ihnen ermöglicht, zu bewerten, wie diese Komponenten innerhalb des Systems zusammenarbeiten. Integrationstests zielen darauf ab, sicherzustellen, dass verschiedene Teile der Software harmonisch zusammenarbeiten, wenn sie integriert werden. Sie dienen als wertvolles Werkzeug zur Identifizierung von Problemen im Zusammenhang mit Datenfluss, Kommunikation zwischen Komponenten und dem Gesamtverhalten des Systems. Aufgrund ihrer Einbindung in reale Abhängigkeiten und komplexeren Interaktionen können Integrationstests jedoch längere Ausführungszeiten als Unittests haben.

Die untersuchten Software-Komponenten zeichnen sich durch eine hohe Qualität der Testaufbauten aus. Die gängigen Prinzipien in der aus der agilen Software-Entwicklung für Tests (Steinfeld 2020; Alsaqqa, Sawalha, and Abdel-Nabi 2020) sind die vollständige Isolierung des Testaufbaus, der Einsatz von Standard-Testdaten und die Wiederholbarkeit des Tests sowohl in

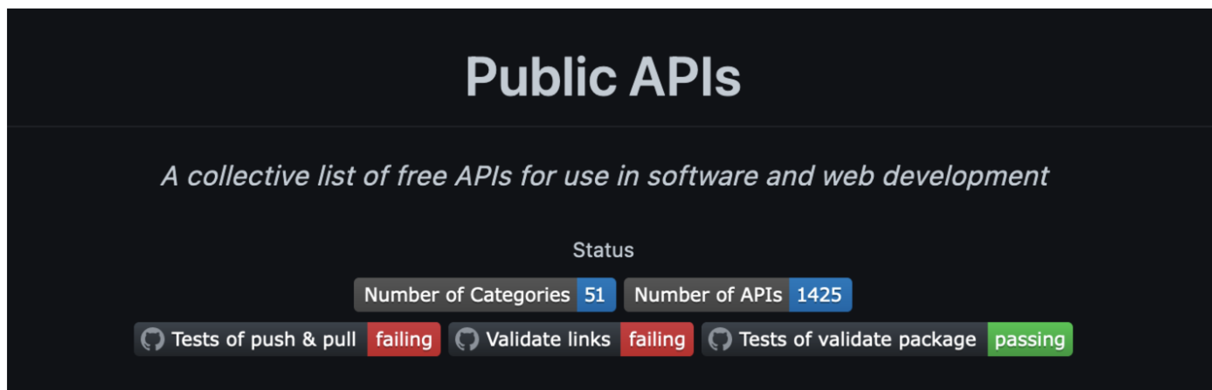
[51] siehe https://github.com/topics, abgerufen am 21.10.2023

[52] siehe https://git.drupalcode.org/project/drupal/-/tree/9.5.x/core/tests, abgerufen am 21.10.2023

der lokalen Entwicklungsumgebung als auch auf den Testsystemen. Die Isolierung des Testaufbaus kann durch den Einsatz von containerbasierten Entwicklungsumgebungen erfolgen, aber auch durch klassische Unit-Test Frameworks[53] die vor Beginn jedes Tests, alle Abhängigkeiten auf den Ursprungsstatus zurücksetzen (z.B. eine Datenbank leeren).

Zuletzt werden die Tests in der kollaborativen Entwicklungsphase meistens in zentralen "Continous Integration" Systemen ausgeführt[54] (Cano et al. 2021). Die Ergebnisse sind sowohl für eine nicht technische Zielgruppe als auch für die technischen Kollaborateure des Projekts, sichtbar. Ein Beispiel dieser sog. Badges auf einem GitHub-Repository findet sich in Abbildung 17.

Abbildung 17: Beispiel-Abbildung von Testabdeckung und Zustand



Beispiel der Anzeige von Status-Bildern für die verschiedenen Zustände innerhalb des Programmcodes & Tests
Quelle: GitHub.com, Public APIs Projekt[55]

Da die Methodik auf der Ausführung der Tests aufbaut, wurde bei der Auswahl der untersuchten Software-Komponenten ein besonderes Augenmerk auf eine hohe Testabdeckung und einen funktionstüchtigen Testaufbau gelegt. Als weiteres Auswahlkriterium wurde die Popularität der Software-Komponenten, gemessen an der Anzahl der Lesezeichen oder Downloads der Bibliothek miteinbezogen. Zuletzt wurde für jede Komponente geprüft, ob diese sich noch in aktiver Entwicklung befindet. Diese Überprüfung wurde über das Datum der letzten Veränderung im Programmcode durchgeführt. Die Komponenten wurden für die Programmiersprachen Java, Python, C, C++, Javascript und Golang ausgewählt. Für jede Sprache sollen mindestens 5 Test-Subjekte, d.h. die Ausführung von Integrations- und Unittests vorhanden sein. Die Übersicht der ausgewählten Open Source Projekte befindet sich im Anhang A.3.

Für die ausgewählten Test-Subjekte wurden im Rahmen des Vorhabens die entwickelte Methodik und die CI/CD Werkzeuge eingebaut und verwendet, um den Energie- und Ressourcenverbrauch für jede Komponente zu messen und sichtbar zu machen.

Entwicklung einer Annäherungsmethode die ohne IPMI & RAPL auskommt

In virtualisierten oder containerisierten Laufzeitumgebungen ist der Systemzugriff auf die IPMI und RAPL-Schnittstelle eingeschränkt. Zudem lässt sich oft der Server-Hersteller bzw. die genaue Produktkennzeichnung (z.B. der Herstellername oder die Modell-Kennzeichnung für den

[53] Ein sehr populäres Unit-Testing-Framework aus der Java Welt ist JUnit. Die Dokumentation zur Test Isolierung findet sich hier: https://junit.org/junit5/docs/current/user-guide/#writing-tests-test-instance-lifecycle

[54] Beispiele dafür sind Jenkins, CircleCI, GitLab, GitHub Actions, etc.

[55] siehe https://github.com/public-apis/public-apis/actions/workflows/test_of_validate_package.yml, abgerufen am 21.10.2023

Server) nicht ermitteln. Um in diesen Umgebungen dennoch Informationen zu Umweltwirkungen und Energieverbrauch der Software bereitzustellen, wurde zuerst ein Annäherungsansatz entwickelt, der auf verfügbaren Informationen aufbaut:

- Die Auslastung der Systemkomponenten (CPU-Last, Arbeitsspeicher-Nutzung, Festplatten-Nutzung, Netzwerkauslastung) – im Folgenden „digitale Ressourcen“ [56]
- CPU-Typ des Servers (für den ein TDP-Wert[57] verfügbar sein muss)[58]
- PUE des zugrundeliegenden Rechenzentrums, falls nicht verfügbar, wird ein Durchschnittswert für europäische Rechenzentren genutzt – PUE 1.8 (Bertoldi 2015)
- Physischer Standort des Rechenzentrums oder des Computers in/auf dem die Software ausgeführt wird.

Sind diese Informationen verfügbar, so lässt sich mit einer Formel der Energieaufwand pro Auslastungsstufe des Servers bestimmen.

Im ersten Schritt wird ein Server in 4 digitalen Ressourcen zerlegt, die durch die Server-Systeme und Chips produziert werden: CPU Zeit, Arbeitsspeicher, Festplattenspeicher und Netzwerktransfer (Kennes 2023).

Da sich der Gesamtenergieverbrauch E_{tot} eines Servers aus der Summe der von seinen Komponenten verbrauchten Energie und dem Basis-Energieverbrauch im Ruhezustand zusammensetzt, kann folgende Formel definiert werden:

$$E_{tot} = E_{cpu} + E_{mem} + E_{IO} + E_{net} + \beta_{idle}$$

Die Variablen sind wie folgt definiert:

- E_{tot}: Der Gesamtenergieverbrauch des Servers
- E_{cpu}: Der Energieverbrauch der CPU
- E_{mem}: Der Energieverbrauch des Arbeitsspeichers
- E_{IO}: Der Energieverbrauch der Schreib- und Lesevorgänge auf dem Permanentspeicher
- E_{net}: Der Energieverbrauch des Netzwerkverkehrs
- β_{idle}: Energy consumption of the server when idling

Für die mathematischen Formeln wird davon ausgegangen, dass der Energieverbrauch für keine dieser Komponenten direkt gemessen werden kann. Man kann sie jedoch in einen Faktor aufteilen, der gemessen werden kann, nämlich den Verbrauch der Komponenten (U), und einen Faktor, der den Verbrauch in Energie umwandelt (f). Dabei ist zu beachten, dass die Komplexität,

[56] Der Begriff wird auch im Glossar der SDIA definiert: siehe https://knowledge.sdialliance.org/glossary/digital-resource-primitives, abgerufen am 21.10.2023

[57] Der Thermal Design Power (TDP)-Wert einer CPU gibt die maximale Wärmemenge, gemessen in Watt (W), an, die der Prozessor bei typischer Arbeitsbelastung abgeben kann. Er ist eine wichtige Kennzahl für Systementwickler*innen und Nutzer*innen, um zu verstehen, wie viel Wärmemanagement erforderlich ist, damit die CPU innerhalb ihres sicheren Betriebstemperaturbereichs bleibt.

[58] Für die gängigen Intel Server-Chips sind diese auf der Intel Webseite abrufbar: https://ark.intel.com/content/www/de/de/ark/products/series/125191/intel-xeon-scalable-processors.html

die der Umwandlung von Verbrauch in Energie zugrunde liegt, in den zweiten Faktor (*f*) einfließt.

Jede Verbesserung, den Energieverbrauch von Software genau zu messen, würde durch die Verbesserung dieses zweiten Faktors (*f*) erfolgen.

Der Gesamtenergieverbrauch eines Servers ist also definiert als:

$$E_{tot} = U_{cpu} f_{cpu}(U_{cpu}) + U_{mem} f_{mem}(U_{mem}) + U_{IO} f_{IO}(U_{IO}) + U_{net} f_{net}(U_{net}) + \beta_{idle}$$

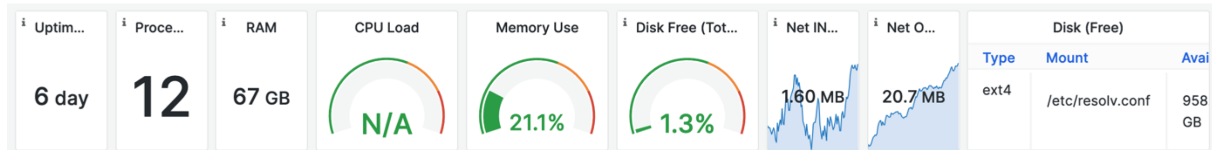
- E_{tot}: Die gesamte von einem Server verbrauchte Energie
- *PUE (Power Usage Effectiveness):* Der PUE-Wert gibt an, wie effektiv die zugeführte Energie in einem Rechenzentrum verbraucht wird. Je näher sich der Wert an 1,0 annähert, desto energieeffizienter arbeitet das Rechenzentrum und desto besser ist seine Energiebilanz. Der PUE-Wert ist eine dimensionslose Kennzahl. PUE ist im Standard ISO/IEC 30134-2 und EN 50600-4-2 definiert. Die PUE teilt die gesamte RZ-Leistungsaufnahme inklusive Kühlung und Stromversorgung, etc. durch die Leistungsaufnahme der IT-Systeme. Der PUE-Wert ist somit ein Maß für die Energieeffizienz der Rechenzentrumsinfrastruktur.
- U_{cpu}: Aktuelle CPU-Auslastung, gemessen in Anzahl der CPU-Kerne
- $f_{cpu}(U_{cpu})$: Funktion des CPU-Energieverbrauchs pro CPU-Nutzung in Anzahl der CPU-Kerne.
- U_{mem}: Aktuelle Arbeitsspeichernutzung, gemessen in Bits
- $f_{mem}(U_{mem})$: Funktion des Energieverbrauchs des Arbeitsspeichers pro Arbeitsspeichernutzung in Bits
- U_{IO}: Aktuelle Speichernutzung, gemessen in Bits
- $f_{IO}(U_{IO})$: Funktion des Speicherenergieverbrauchs pro Speichernutzung in Bits
- U_{net}: Aktuelle Netzwerknutzung, gemessen in Bits
- $f_{net}(U_{net})$: Funktion des Netzwerkenergieverbrauchs pro Netzwerknutzung in Bits
- β_{idle}: Energieverbrauch des Servers im Leerlauf

Zu beachten ist, dass die Funktionen, die mit $f_x(U_x)$ bezeichnet werden, die Energieverteilung pro Nutzungseinheit darstellen. Ihre Einheit ist also E/U_x. Diese Funktionen sind auch nicht unbedingt linear. Die funktionale Darstellung sagt nichts über die Art der Beziehung zwischen Komponentenverwendung und Energieverbrauch aus. Es kann andere Faktoren geben, die ebenfalls berücksichtigt werden sollten, z. B. der Servertyp oder das Alter des Servers.

Die Formel kann mit den Messungen der digitalen Ressourcen Verbräuche einer Software-Anwendung aufgelöst werden und ermöglicht es, rein mathematisch die Energieverbräuche eines Servers zu bestimmen. In Abbildung 18 ist ein typisches IT Monitoring System dargestellt, in dem die digitalen Ressourcenverbräuche für dedizierte und virtualisierte Server aufgezeichnet werden. Mit diesen Werten kann die Formel genutzt werden, um den Energieverbrauch des Servers, oder Containers auszurechnen. In typischen Container-

Orchestrierungsplattformen wie Kubernetes können die digitalen Ressourcenverbräuche mit einer einfachen Erweiterung ebenso aufgezeichnet werden[59].

Abbildung 18: Beispiel eines IT Monitoring Systems



Quelle: eigene Darstellung, Sustainable Digital Infrastructure Alliance e.V.

Das Beispiel zeigt die übliche Messung von digitalen Ressourcenverbräuchen in IT Monitoring Systemen.

Mit den gesammelten Informationen lassen sich zudem die Umweltwirkungen des zugrundeliegenden Servers über die Boavizta-API bestimmen. Diese API baut auf den Modellen des Green Cloud Computing Projekts (Gröger et al. 2021) auf und stellt das darin enthaltende Tabellen-Kalkulationsmodell als JSON-basierte API-Schnittstelle bereit. In Abbildung 19 ist eine Abfrage an die API beispielhaft dargestellt.

[59] Erweiterung für Kubernetes: https://github.com/prometheus-operator/kube-prometheus

Abbildung 19: Beispiel Abfrage der Boavizta API

```
curl -X 'POST' \
  'https://api.boavizta.org/v1/server/?verbose=false&archetype=compute_medium' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "model": {
    "type": "rack"
  },
  "configuration": {
    "cpu": {
      "units": 2,
      "core_units": 12,
      "die_size_per_core": 245
    },
    "ram": [
      {
        "units": 12,
        "capacity": 64,
        "density": 1.79
      }
    ],
    "disk": [
      {
        "units": 4,
        "type": "ssd",
        "capacity": 400,
        "density": 50.6
      }
    ],
    "power_supply": {
      "units": 2,
      "unit_weight": 2.99
    }
  }
}'
```

Quelle: eigene Darstellung, Sustainable Digital Infrastructure Alliance e.V.

Die Abfrage kann wie folgt parametrisiert werden:

- „configuration“, „cpu“ – hier können die Anzahl der physisch verbauten CPUs angegeben werden und die Anzahl der CPU-Kerne pro CPU.
- „configuration“, „ram“ – hier kann die Menge der Arbeitsspeichermodule und deren Speicherkapazität angegeben werden.
- „configuration“, „disk“ – hier kann angegeben werden, wie viele Festplatten verbaut sind, welche Art von Festplatte (HDD oder SSD) und mit welcher Speicherkapazität diese ausgestattet sind.

- „configuration“, „power_supply“ – zuletzt lässt sich spezifizieren wie viele Netzteile im Server verbaut sind (üblicherweise 2 Stück für die Fähigkeit Netzteile im laufenden Betrieb zu tauschen)

Eine Abfrage an die API liefert die geschätzten Umweltwirkungen des Servers, wie in Abbildung 20 dargestellt, im JSON-Format zurück.

Abbildung 20: Ergebnis einer Beispielabfrage an die Boavizta API


```
{
  "gwp": {
    "embedded": {
      "value": 1501.4,
      "significant_figures": 5,
      "min": 1501.4,
      "max": 1501.4,
      "warnings": [
        "End of life is not included in the calculation"
      ]
    },
    "use": {
      "value": 10317,
      "significant_figures": 5,
      "min": 563.41,
      "max": 36960
    },
    "unit": "kgCO2eq",
    "description": "Total climate change"
  },
  "adp": {
    "embedded": {
      "value": 0.16588,
      "significant_figures": 5,
      "min": 0.16588,
      "max": 0.16588,
      "warnings": [
        "End of life is not included in the calculation"
      ]
    },
    "use": {
      "value": 0.00174387,
      "significant_figures": 6,
      "min": 0.000324327,
      "max": 0.00867403
    },
    "unit": "kgSbeq",
    "description": "Use of minerals and fossil ressources"
  },
  "pe": {
    "embedded": {
      "value": 19475,
      "significant_figures": 5,
      "min": 19475,
      "max": 19475,
      "warnings": [
        "End of life is not included in the calculation"
      ]
    },
    "use": {
      "value": 349500,
      "significant_figures": 5,
      "min": 318.45,
      "max": 15290000
    },
    "unit": "MJ",
    "description": "Consumption of primary energy"
  }
}
```


Quelle: eigene Darstellung, Sustainable Digital Infrastructure Alliance e.V.

Diese Informationen werden auch bei der Entwicklung des CI/CD Werkzeugs im späteren Verlauf des Vorhabens miteinbezogen. Die Erhebung der materiellen Ressourcenverbräuche und damit verbundener Umweltwirkungen ist zwar nicht Teil des Vorhabens, wurde aber dennoch mit Hilfe der Open-Source Schnittstelle von Boavizta realisiert, um zusätzlich zum

Energieverbrauch indikativ auch Ressourcenverbräuche für die Komponenten sichtbar zu machen und eine lebenszyklusbasierte Betrachtung[60] zu ermöglichen.

Den operativen Stromverbrauch der IT-Hardware und die Umweltwirkung aus der Herstellung gemeinsam zu betrachten ist wichtig, denn nur so lässt sich ein möglicher Rebound-Effekt eingrenzen. Wird allein der Strom, der für die Erzeugung von digitalen Ressourcen von der IT-Hardware verbraucht wird, betrachtet, so entsteht ein Fokus auf die Effizienz, mit der der Strom in digitale Ressourcen gewandelt wird, ohne dabei den materiellen Aufwand des Effizienzgewinns gegenüberzustellen. So macht es möglicherweise Sinn einen Server jedes Jahr auszutauschen, weil ein neuer Server ggf. effizienter Strom in digitale Ressourcen wandelt. Jedoch entsteht so ein zusätzlicher Materialaufwand und Rohstoffverbrauch, mit den damit verbundenen Umweltwirkungen. Um diese Schlussfolgerung zu vermeiden, hat das Forschungsteam sich entschlossen, den Stromverbrauch der gemessenen Software immer im Zusammenhang mit dem Ressourcenverbrauch der IT-Hardware darzustellen, um so ein Bewusstsein für die Wechselwirkung von operativem Stromverbrauch und Umweltwirkungen aus der Herstellung für Software-Entwickelnde zu schaffen.

Weiterentwicklung der Formeln durch die Software-Community

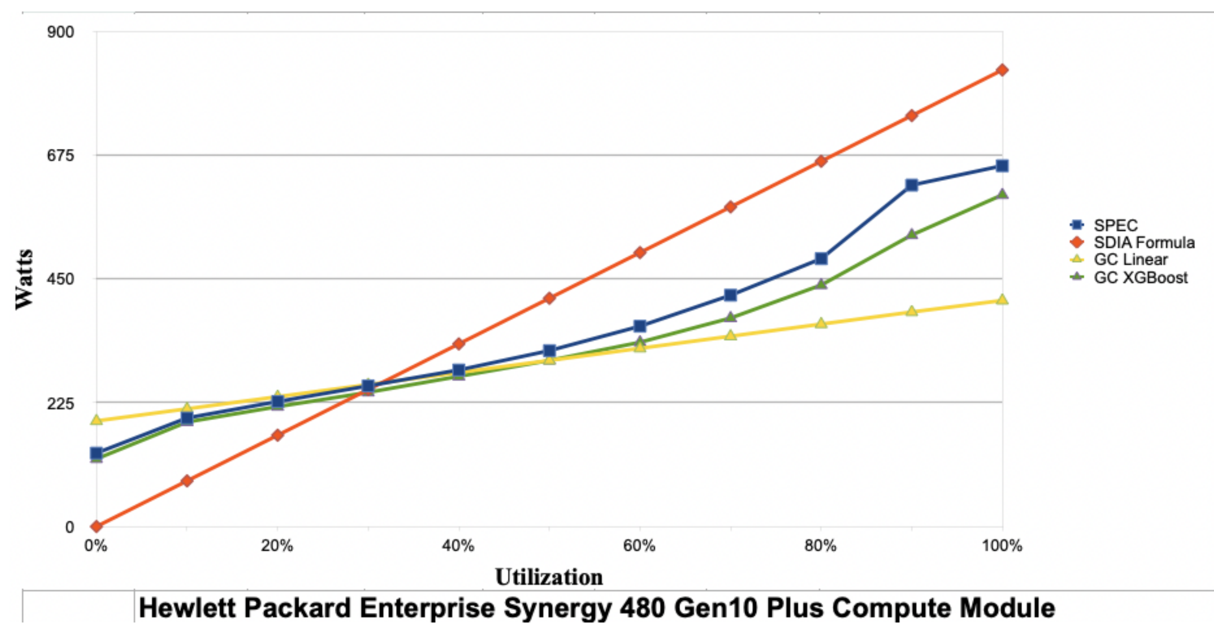
Durch die Einbindung der Software-Community in das Vorhaben (siehe Kapitel 4) wurden bereits Projekte zur Verbesserung der Formeln ins Leben gerufen. Ein Projekt, initiiert von „Green Coding Berlin“, nutzt die offene Datenbank mit Messungen, die mit dem Spec Power[61] Lasttest durchgeführt worden sind. Das „Spec Power Model“ ist ein Modell, welches maschinelles Lernen[62] verwendet, um den Stromverbrauch einer CPU in verschiedenen Auslastungen (0-100%) zu bestimmen. Ein Vergleich der im Vorhaben entwickelten Formel mit dem Modell von Green Coding Berlin findet sich in Abbildung 21.

[60] siehe https://softawere-hackathon.gitlab.io/documentation/explanations/lca_based_environmental_criterias.html für eine Liste von Umweltindikatoren, die durch das Werkzeug dargestellt wird, abgerufen am 13.10.2023

[61] SPEC Power (https://spec.org/power_ssj2008/) ist eine Benchmarking-Tool welches entwickelt wurde um den Stromverbrauch von verschiedenen Computern (Server, Desktop-Computer, etc.) miteinander zu vergleichen. Dafür wird ein Standardnutzungsszenario auf den Computern ausgeführt und die Ergebnisse in einer offenen Datenbank (siehe https://spec.org/power_ssj2008/results/res2023q3/) zum Vergleich zur Verfügung gestellt. Die Durchführung der SPEC Power Messung ist Teil der Energy Star Zertifizierung (https://www.energystar.gov/), abgerufen am 13.10.2023

[62] Für das Training des Modells wurden die offenen SPEC Power Daten verwendet (https://spec.org/power_ssj2008/results/res2023q3/), abgerufen am 13.10.2023

Abbildung 21: Vergleich der Formeln mit dem Spec Power Modell



Es werden die Originalwerte der SPEC Power Datenbank (blaue Linie) mit den Schätzwerten aus dem Modell (gelbe und grüne Linie, sind zwei Konfigurationen des Modells) mit den Formeln aus dem Vorhaben (rote Linie) verglichen.
Quelle: Green Coding Berlin GmbH, Github.com[63]

Außerdem wurde eine Datenbank ins Leben gerufen, die die Stromverbräuche verschiedener CPUs unter verschiedenen Lasten katalogisiert und der Community zur Verfügung stellt[64].

Aufbau des Testlabors zur Messung der Energieverbräuche

Für das Vorhaben wurde ein Referenzsystem aufgebaut, auf dem die Messungen der Test-Subjekte durchgeführt werden können. Auf diesem Test-System ist ein Systemzugriff möglich, weshalb ein Zugriff auf alle Schnittstellen zur Messung des Energieverbrauchs möglich ist. Die Annäherungsformeln aus dem vorherigen Abschnitt werden nicht verwendet.

Die Anforderungen an das Testsystem wurden wie folgt definiert:

- Es sollten die IPMI und RAPL Messchnittstellen zur Verfügung stehen, um parallel und kontinuierlich aufzeichnen zu können.
- Es sollte Server-Hardware zum Einsatz kommen, die sich ähnlich zu modernen Cloudumgebungen verhält, damit die SoftAWERE-Light Methodik auf die Richtigkeit in virtualisierten Cloud-Umgebungen überprüft werden kann.
- Es sollte auf quelloffenen Komponenten aufbauen und keine Abhängigkeiten zu proprietären Standards oder Hersteller-Software für die Messungen haben.
- Die Aufbewahrung der Messungen und der Betrieb der Monitoringsysteme sollte außerdem außerhalb des Testsystems erfolgen, um Nebeneffekte, wie z.B. eine erhöhte Auslastung durch die Datenbank des Monitoringsystems zu vermeiden.

[63] siehe https://github.com/green-coding-berlin/spec-power-model, abgerufen am 13.10.2023

[64] siehe Energizta, https://github.com/Boavizta/Energizta, abgerufen am 13.10.2023

- Die Umweltwirkung des Testsystems selbst sollte minimal sein - der Einsatz von wiederaufbereiteter Hardware, erneuerbarer Energie und Abwärmenutzung sollte priorisiert werden.

Um die Anforderungen zu erfüllen, hat die SDIA-Community weitere Unterstützer angesprochen, um mögliche Infrastruktur-Unterstützer zu finden, die einen Teil von bestehender Infrastruktur für das Projekt bereitstellen können. Blockheating[65] aus den Niederlanden hat sich bereit erklärt, Server-Kapazität für das Projekt bereitzustellen. Diese Kapazität wird aus wiederaufbereiteten Open Compute Server-Systemen[66] (OCP) bereitgestellt. Diese Server zeichnen sich durch eine sehr hohe Differenz im Stromverbrauch zwischen Leerlauf und Volllast aus. In Last-Tests lag der Stromverbrauch im Leerlauf bei ca. 60 W für das Gesamtsystem und ca. 300 W unter Volllast (siehe Abbildung 22), eine Differenz von 240 W. Diese technischen Rahmenbedingungen erschienen für das SoftAWERE-Vorhaben als ideal.

Auf den OCP-Systemen ist der Messunterschied zwischen RAPL und IPMI besonders gut zu beobachten (siehe Abbildung 22). Die RAPL-Schnittstelle (Graph mit Überschrift „Scaph Host Power“) benötigt unter Volllast ca. 80 W weniger Strom im Vergleich zur IPMI-Schnittstelle (unter „IPMI Power Watts“). Dieser Unterschied lässt sich auch bei geringer Last feststellen, wie man zu Beginn der Messung sieht. Die RAPL-Schnittstelle benötigt ca. 20 W im Vergleich zur IPMI-Schnittstelle, die ca. 70 W aufweist (bei einer CPU-Auslastung von 0-10%).

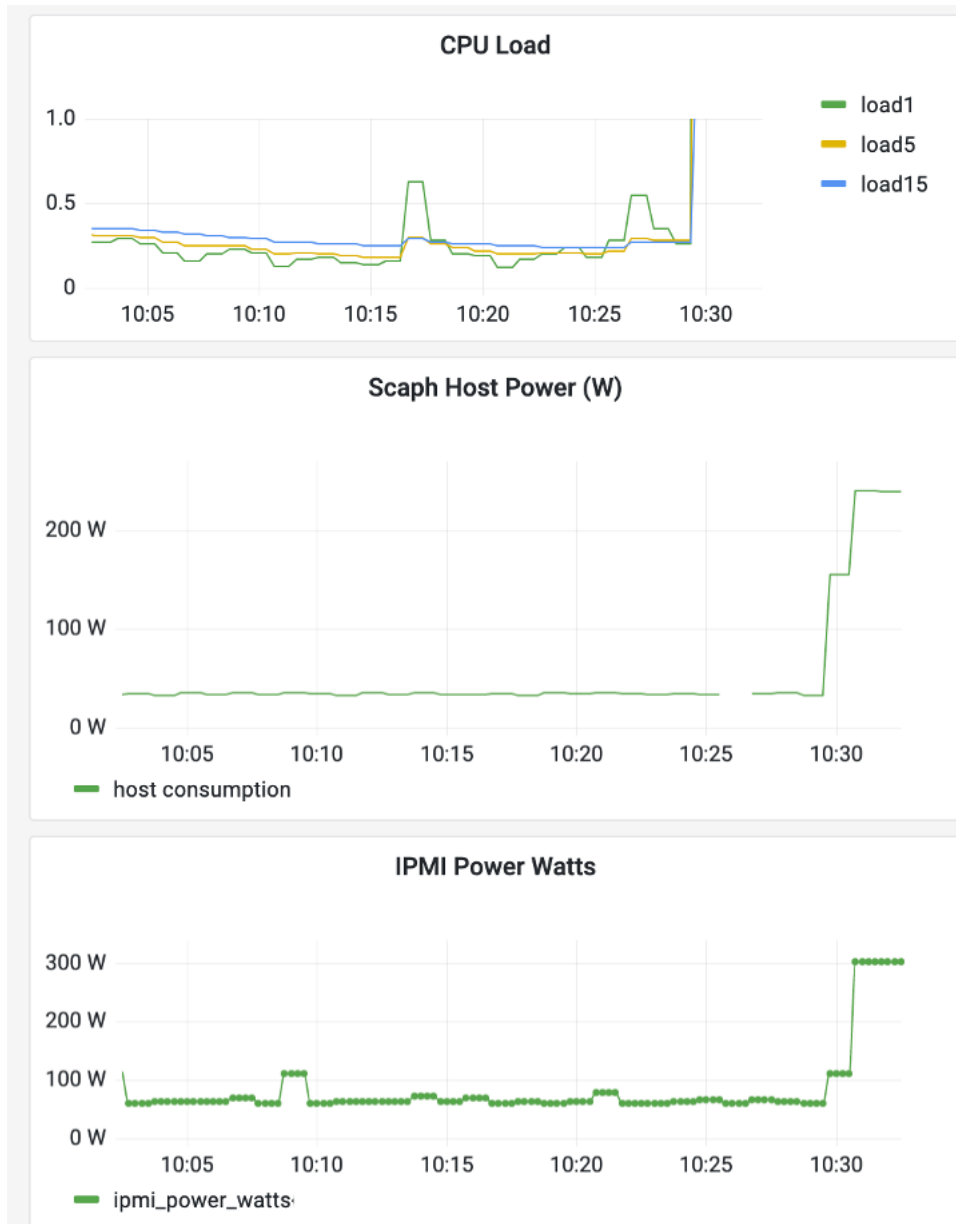
Die Abkürzung „Scaph“ in der Abbildung steht für „Scaphandre“, eine quelloffene Bibliothek, die vom Projekt eingesetzt wurde, um die RAPL-Schnittstelle anzusprechen.

[65] Blockheating: https://blockheating.com

[66] Open Compute Foundation: https://opencompute.org

Abbildung 22: Strommessung Testlabor unter Voll-Last

CPU-Auslastung, Stromverbrauch gemessen mit Intel RAPL und IPMI im direkten Vergleich. Die „CPU-Load" ist in Prozent dargestellt (1.0 = 100%, 0.5 = 50%, 0 = 0%). Die Linux-CPU-Last (load1, load5, load15) – entspricht durchschnittlichen Arbeitslast des Systems über verschiedene Zeitintervalle (1 Minute, 5 Minuten, 15 Minuten). Der Graph „Scaph Host Power" zeigt den Stromverbrauch des Systems gemessen mit der RAPL-Schnittstelle. Der Graph „IPMI Power Watts" zeigt den Stromverbrauch gemessen mit der IPMI-Schnittstelle.



Das Testsystem wurde mit Hilfe von „stress-ng"[67] unter Voll-Last gesetzt. Stress-ng ist ein Werkzeug aus der Linux-Community welches genutzt wird, um die Performance verschiedener Betriebssystemkonfigurationen und Hardwareausrüstung zu evaluieren.
Quelle: eigene Darstellung, Sustainable Digital Infrastructure Alliance e.V.

Die detaillierte Spezifikation des Test-Systems findet sich in Anhang D.6 und Abbildung 67.

67 stress-ng: https://wiki.ubuntu.com/Kernel/Reference/stress-ng, abgerufen am 13.10.2023

Für das Monitoring des Test-Systems und die Datenerhebung der Messungen wurde ein bestehendes Forschungssystem aus dem EU-Projekt "ECO-Qube"[68] wiederverwendet. Es handelt sich dabei um ein auf quelloffenes Datenbank- und Visualisierungssystem, bestehend aus Thanos[69] als Datenbank und Grafana[70] als visuelle Oberfläche.

Thanos ist ein Open-Source-Projekt, welches auf Prometheus[71] aufbaut, einem beliebten Monitoring- und Alarmierungswerkzeug, das in Cloud-Umgebungen eingesetzt wird. Es bietet eine hoch skalierbare und fehlertolerante Lösung für die Speicherung und Abfrage von Prometheus-Metrikdaten über längere Zeiträume. Thanos ermöglicht die langfristige Speicherung von Überwachungsdaten, effiziente clusterübergreifende Abfragen und hohe Verfügbarkeit durch die nahtlose Integration in bestehende Prometheus-Setups und Cloud-Speicherlösungen wie AWS S3 oder Google Cloud Storage.

Die gesamte Monitoring-Infrastruktur ist auf Kubernetes[72] aufgebaut und kann je nach Bedarf verkleinert oder vergrößert werden, z.B. wenn Tests ausgeführt werden. Kubernetes ist ein Open-Source-System zur Automatisierung der Bereitstellung, Skalierung und Verwaltung von containerisierten Anwendungen. Der Aufbau des Labors auf der Kubernetes-Plattform macht das Labor übertragbar auf andere Umgebungen (es kann fast überall installiert werden und ist portabel) und flexibel skalierbar.

In der Abbildung 23 ist die Software-Architektur des Labors dargestellt. Ein „CI job", welcher erstellt wird, um ein Test-Subjekt zu untersuchen wurde um ein „get_metrics"-Skript erweitert. Dieses Skript ruft nach Abschluss der Ausführung den „boagent" auf (Pfeil nach linksunten). Dieser wiederum liest die Messdaten der verschiedenen Schnittstellen aus, z.B. „Scaphandre" um die RAPL-Messdaten auszulesen oder IPMI („os"). Zudem sammelt der „boagent" die Systemmerkmale ein (Arbeitsspeicher, CPU Typ, etc.) und nutzt diese um bei der „boaviztapi" die Umweltwirkung der Server-Herstellung abzufragen. Nachdem diese Daten zusammengestellt worden sind, stellt der „boagent" sie dem „CI job" über das „get_metrics"-Skript bereit. Das Skript speichert die Daten zuletzt als Zeitreihe in Thanos bzw. Prometheus für die Analyse in Grafana. Dabei werden die Zeitreihen-Daten noch um die Laufzeitinformationen des Docker-Containers in dem der „CI job" läuft, angereichert (über den „docker-watcher" und „exporter"). Die Laufzeit-Daten sind der eigentliche Verbrauch von Arbeitsspeicher und CPU Zeit.

68 ECO-Qube: https://ecoqube.org, abgerufen am 13.10.2023

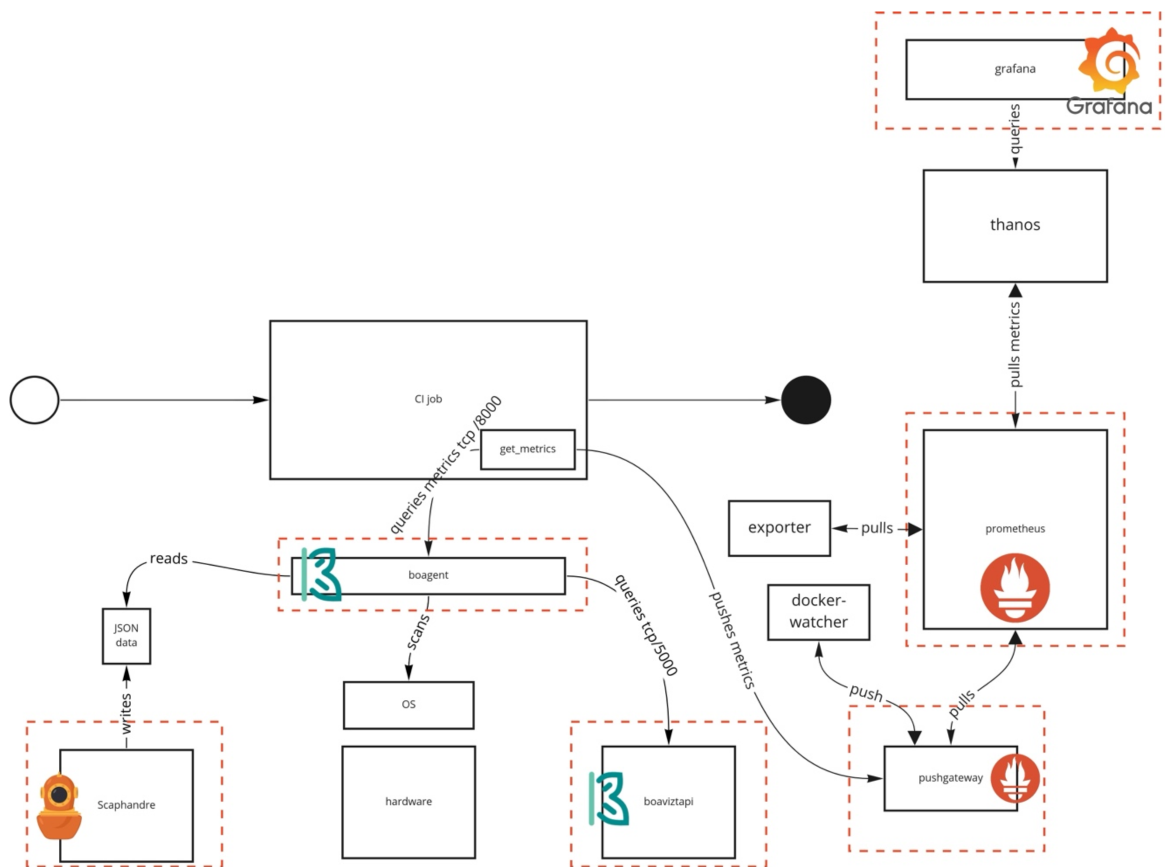
69 Thanos; https://thanos.io/, abgerufen am 13.10.2023

70 Grafana: https://grafana.com/, abgerufen am 13.10.2023

71 Prometheus ist ein weit verbreitetes Open-Source Monitoring- und Alarmierungs-Werkzeug, das für moderne, Cloud-native Umgebungen entwickelt wurde. Es zeichnet sich durch die Erfassung und Speicherung von Zeitreihendaten aus und ist daher von unschätzbarem Wert für die Überwachung der Leistung und des Zustands von Anwendungen und Infrastruktur. Mit seiner robusten Abfragesprache und seinen leistungsstarken Warnfunktionen ist Prometheus ein wichtiges Werkzeug, um die Zuverlässigkeit und Verfügbarkeit von Systemen in der heutigen komplexen Server-Landschaft sicherzustellen. Siehe https://prometheus.io/docs/introduction/overview/, abgerufen am 13.10.2023

72 Kubernetes ist eine Open-Source-Plattform für die Container-Orchestrierung, die die Bereitstellung, Skalierung und Verwaltung von containerisierten Anwendungen automatisiert. Sie bietet eine leistungsstarke und flexible Möglichkeit zur Orchestrierung von Container-Workloads und erleichtert so die Verwaltung und Skalierung von Anwendungen in einer dynamischen und Cloud-nativen Umgebung. Kubernetes hat sich zum De-facto-Standard für die Container-Orchestrierung entwickelt und ermöglicht es Organisationen, ihre Ressourcen effizient zu nutzen und die Ausfallsicherheit ihrer Anwendungen zu gewährleisten. Siehe https://kubernetes.io, abgerufen am 13.10.2023

Abbildung 23: Architektur und Aufbau des Testlabors



Quelle: eigene Darstellung, Sustainable Digital Infrastructure Alliance e.V.

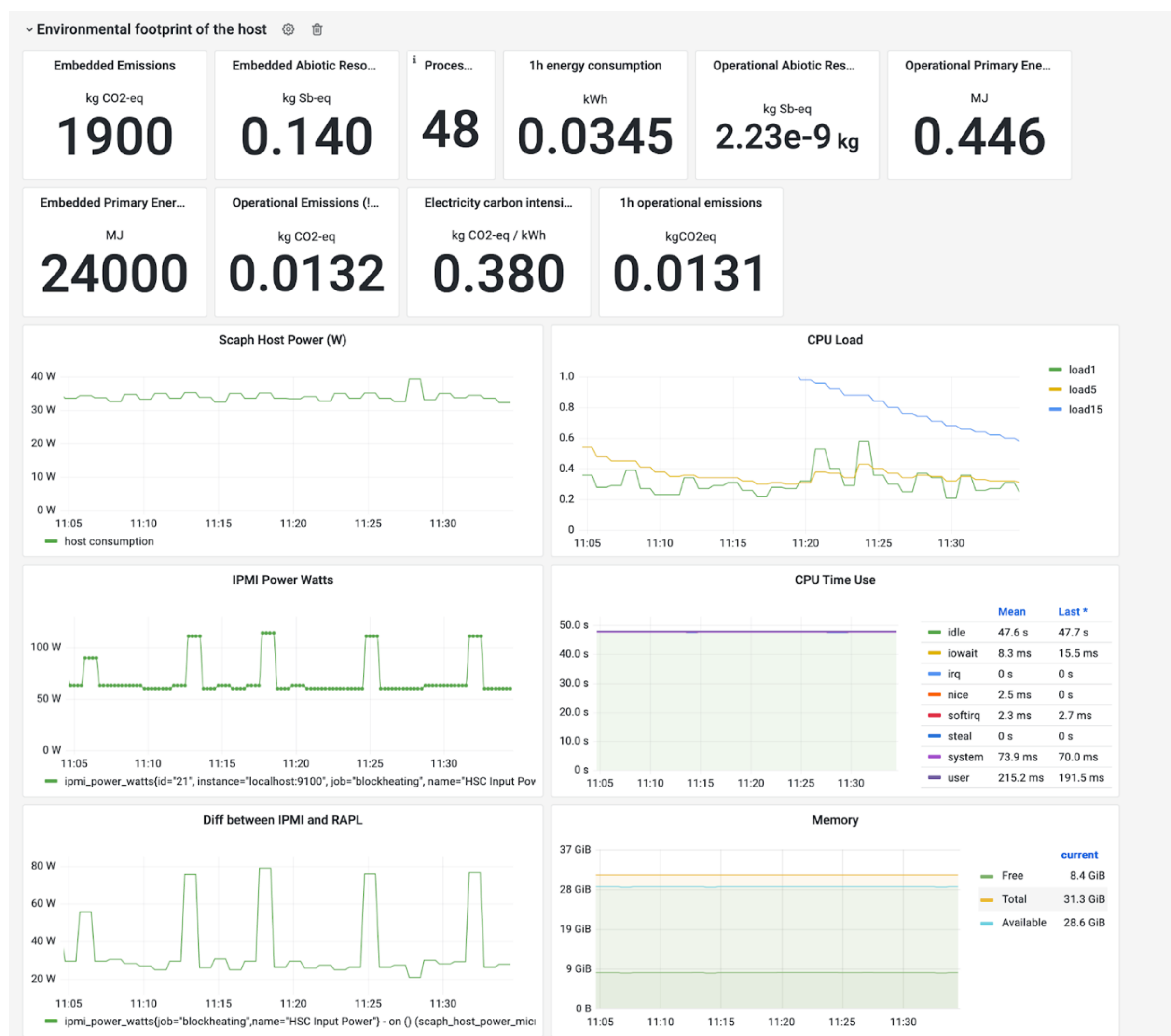
Eine Darstellung des Grafana Dashboards für das Testsystem findet sich in der Abbildung 24. Es zeigt die gemessenen Stromverbräuche (mit IMPI und RAPL), die digitalen Ressourcenverbräuche und die von der Boavizta API ermittelten Indikatoren zur Umweltwirkung des Server-Systems. Im Folgenden ist eine Übersicht der ermittelten Indikatoren dargestellt:

- Indikator„Embedded Emissions“ – Eine Kennzahl, die Auskunft über die Treibhausgasemissionen der eingesetzten Server gibt, die aus der Produktion des Servers stammen (Herstellerangabe).
- Indikator „Embedded Abiotic Resource Depletion Potential” - Eine Kennzahl zur Bestimmung des abiotischen Rohstoffverbrauchs, entstanden durch die Herstellung des Servers.
- Kennung „Processors“: Anzahl der CPU-Kerne im Server.
- Messung „1h Energy Consumption“: Der durchschnittliche Stromverbrauch des Servers in den letzten 60 Minuten.
- Indikator„Operational Abiotic Resource Depletion Potential”- Eine Kennzahl zur Bestimmung des abiotischen Rohstoffverbrauchs während des Betriebs des Servers.

- Indikator„Operational Primary Energy Consumption“, Kennzahl zum geschätzten Energieverbrauchs des Servers über 1 Stunde anhand von Herstellerangaben (keine Messung).
- Indikator „Embedded Primary Energy Consumption“, Kennzahl zum Energieverbrauch des Servers der durch die Herstellung entstanden ist (Herstellerangabe).
- Kennung „Operational Emissions“, Kennzahl zur geschätzten CO2-Intensität der Stromproduktion am Standort des Rechenzentrums (Niederlande) auf Basis von Umrechnungsfaktoren der Umweltagentur der Europäischen Union (siehe Anhang C.10).
- Kennung „Electricity carbon intensity“, der Umrechnungsfaktor der für die Berechnung der „Operational Emissions“ verwendet wurde.
- Kennung „1h operational emissions“, zusätzliche Kennzahl der Emissionen über einen 1 Stunden vorrausschauenden Horizont.

Zusätzlichen werden die realen Stromverbräuche via RAPL (Kennung „Scaph Host Power“) und IPMI (Kennung „IPMI Power Watts“) dargestellt als auch die Differenz in Wattsekunden der beiden Messchnittstellen (Kennung „Diff between IPMI and RAPL“). Auf der rechten Seite sind die Ressourcenverbräuche des Hauptsystems erfasst mit einem Fokus auf die CPU (Kennung „CPU Load“ und „CPU Time Use“) und Arbeitsspeicherauslastung (Kennung „Memory“) als primäre Treiber des Stromverbrauchs.

Abbildung 24: Messoberfläche im Testlabor



Quelle: eigene Darstellung, Sustainable Digital Infrastructure Alliance e.V.

Um die Tests für die Open-Source Projekte nun auf diesem Testsystem auszuführen, wurde auf jedem Server im Testsystem ein GitLab Runner[73] installiert. Dieser Runner startet auf dem System eine isolierte, containerbasierte Testumgebung, die wiederum die Ressourcenverbräuche der Tests isoliert und sie von Nebeneffekten des Hauptsystems isoliert.

Dieser Runner ist mit einem Team auf GitLab.com verbunden[74], welches den Runner für beliebige Testabläufe ansteuern kann. Der Runner wurde konfiguriert, um keine parallelen Tests auszuführen, sondern maximal einen Test gleichzeitig durchzuführen, da ansonsten möglicherweise zwei Testausführungen gleichzeitig laufen und damit die Messungen verfälscht würden.

[73] Die Dokumentation für den Gitlab Runner findet sich hier: https://docs.gitlab.com/runner/, abgerufen am 13.10.2023

[74] Das Team befindet sicher hier: https://gitlab.com/softawere-hackathon, abgerufen am 13.10.2023

Der gesamte Aufbau sowohl des Monitoring Systems und der Ausführungssysteme ist quelloffen und auf GitHub[75] und GitLab[76] verfügbar.

Analyse der ausgewählten Testsubjekte im Testlabor

Die ausgewählten Open Source Projekte werden manuell im GitLab-basierten Testlabor konfiguriert - dafür werden die Projekte kopiert ("Fork") und eine Instruktionsdatei hinzugefügt, die dem SoftAWERE Werkzeug Anweisungen gibt, wie die Tests ausgeführt werden sollen. Ein vollständiges Beispiel für eine solche Instruktionsdatei findet sich in der Abbildung 25.

[75] Die Kubernetes Konfiguration befindet sich hier: https://github.com/SDIAlliance/ecoqube-prometheus, abgerufen am 13.10.2023

[76] SoftAWERE GitLab Repository mit der Docker Compose Konfiguration für das Starten des Labors auf einem beliebigen Test-System findet sich hier https://gitlab.opencode.de/OC00004041236/softawere). Es enthält den GitLab Runner als Abhängigkeit.

Abbildung 25: Instruktionsdatei für Test-Subjekte im Labor

Die Instruktionsdatei („.gitlab-ci.yml“) spezifiziert sowohl die Ausführungsumgebung „image“ als auch die Schritte, die zur Ausführung der Tests notwendig sind „test – script“. Teil des Scripts ist der Start und das Ende der Messung.

```
# To contribute improvements to CI/CD templates, please follow the Development guide at:
# https://docs.gitlab.com/ee/development/cicd/templates.html
# This specific template is located at:
# https://gitlab.com/gitlab-org/gitlab/-/blob/master/lib/gitlab/ci/templates/Python.gitlab-ci.yml

# Official language image. Look for the different tagged releases at:
# https://hub.docker.com/r/library/python/tags/
image: python:latest

# Change pip's cache directory to be inside the project directory since we can
# only cache local items.
variables:
  PIP_CACHE_DIR: "$CI_PROJECT_DIR/.cache/pip"

before_script:
  - apt update
  - python --version  # For debugging
  - pip install virtualenv
  - virtualenv venv
  - source venv/bin/activate

  # --- Prepare the measurement itself --- #
  - apt install jq procps wget git-core -y
  - curl -O https://gitlab.com/softawere-hackathon/softawere/-/raw/main/gitlab-metric-script
/requirements-metrics.txt
  - pip install -r requirements-metrics.txt
  # --- Done preparing the measurement itself --- #

test:
  script:
    - pip install -r requirements/dev.txt && pip install -e .
    - pip install pytest-cov

    # --- Pass information to the measurement tool via environment variables --- #
    - export CI_JOB_SIZE=$(du -sk . | sed  's/\([0-9]*\).*$/\1/g')
    - export CI_JOB_AFTER_REQUIREMENTS_STARTED_AT=$(date +%s)

    # --- Everything after STARTED_AT is measured --- #
    - coverage run --module pytest --verbose tests | tee test_output.txt
    - coverage report --show-missing | tee coverage.txt

    # --- Pass information to the measurement tool via environment variables --- #
    - export CI_JOB_COVERAGE=$(sed -n --regexp-extended --expression='s/TOTAL.*\s([0-9\%]{2,4})/\1/p'
coverage.txt)
    - export CI_JOB_TEST_COUNT=$(sed -n --regexp-extended --expression='s/\=+\s([0-9]+[^\s]).*/\1/p'
test_output.txt)

    # --- Trigger the collection of measurement data and upload to NocoDB --- #
    - curl -s https://gitlab.com/softawere-hackathon/softawere/-/raw/main/gitlab-metric-script
/get_metrics.py | python
```

Die detaillierte Beschreibung der Konfigurationsschritte für die Testsubjekte im Labor findet sich im Anhang D.

Die Instruktionen (für Docker Compose) für den Aufbau eines Labors finden sich in Anhang D.6.

Zusätzlich wird ein Skript (siehe Anhang D.7) hinzugefügt, welches die Energieverbräuche für jede Ausführung abruft und die Ergebnisse in den Protokollen darstellt. In Abbildung 65 aus dem Anhang D.5 ist ein Beispiel für die Ausgabe in den Protokollen der Testausführung dargestellt.

Das Labor zeichnet konstant die Ressourcenverbräuche des Systems auf

Um die Messungen der Testsubjekte im Labor möglich zu machen, werden die Ressourcenverbräuche des Servers im 15 Sekunden-Takt aufgezeichnet. Dazu zählt sowohl der Energieverbrauch (gemessen über IMPI und RAPL) als auch die digitalen Ressourcenverbräuche.

Die Aufzeichnungen werden als Zeitreihe an die Thanos-Datenbank geschickt, welche die Zeitserien komprimiert und für 30 Tage aufbewahrt.

Wird eine Testserie aus einem Testsubjekt ausgeführt, so wird zu Beginn der Messung der aktuelle Zeitpunkt sekundengenau (UNIX Timestamp[77]) gespeichert. Nachdem die Tests durchgeführt worden sind, wird erneut der aktuelle Zeitpunkt gespeichert. Mit diesen beiden Zeitpunkten „Start" und „Ende" wird eine Anfrage an die Thanos-Datenbank formuliert, welche die Zeitserien-Daten in diesem Zeitraum zurückliefert, unter anderem den Stromverbrauch und die während der Testausführung genutzten digitalen Ressourcen.

Für den gesamten Testaufbau wurde eine quelloffene Dokumentation[78] erstellt, um es der Open-Source Gemeinschaft leicht zu machen, die Konzepte und Herangehensweise technisch nachzuvollziehen.

2.3 Auswertung Messergebnisse & Bewertung

In diesem Kapitel werden die Ergebnisse des Praxistest der entwickelten Messmethode beschrieben sowie die technische Umsetzung des Praxistests (Skripte, die die Last erzeugen). Anhand folgender vier Kriterien werden die Eignung des Messaufbaus und der Methode und die Messergebnisse überprüft:

- Verständlichkeit,
- Plausibilität,
- Richtungssicherheit,
- Reproduzierbarkeit.

Die Ergebnisse der Überprüfung der vier Kriterien werden in den nachfolgenden Kapiteln einzeln beschrieben.

2.3.1 Verständlichkeit

Verständlichkeit: Ein Anwender oder eine Anwenderin des Messaufbaus soll verstehen, welche Berechnungslogik hinter den Messungen liegt, wie die Messung durchgeführt wird und wie die gewünschten Messgrößen ermittelt werden können.

Bewertung der Verständlichkeit

Um mit dem Werkzeug arbeiten zu können, wird vorausgesetzt, dass man bei *Gitlab* registriert und mit der Funktionsweise von der freien, verteilten Versionsverwaltung *„Git"* vertraut ist. Versionskontrolle und kollaboratives Arbeiten gehören zum Alltag einer/eines Softwareentwicklerin/-entwicklers, sodass diese Fähigkeiten bei professionellen Anwender*innen vorausgesetzt werden können. Für interessierte Laien könnte dies eine Hürde darstellen.

[77] Ein Unix-Zeitstempel ist eine numerische Darstellung eines bestimmten Zeitpunkts und gibt die Anzahl der Sekunden an, die seit 00:00:00 Uhr koordinierter Weltzeit (UTC) am 1. Januar 1970 vergangen sind.

[78] siehe https://sdia.io/sawe-docs

Das Beispiel in Anhang D macht deutlich, dass der Messaufbau deutlich umfangreicher und stärker geschachtelt ist als eine einfache Energiemessung. Der komplexe Aufbau ermöglicht es, zusätzlich den anteiligen Ressourcenverbrauch und die Emissionen der Herstellung der Hardware der Software zuzuordnen. Außerdem kann der Messaufbau auf eine beliebige Hardware übertragen werden, die über einen *RAPL*-Chip zur Energiemessung der *CPU* verfügt.

Um im Detail nachzuvollziehen, was die genutzten Software-Module *Scaphandre* und *boaviztapi* tun, kann man den offen zugänglichen Programmcode einsehen. Um ihn verstehen zu können, sind Programmierkenntnisse erforderlich. Ohne diese Kenntnisse wird es kaum möglich sein, den Programmcode zu interpretieren, um bspw., die Annahmen über die Emissionsfaktoren zur Ermittlung der CO_2-Emissionen anhand des Energiebedarfs erkennen zu können.

2.3.2 Plausibilität

- Plausibilität: Die Ergebnisse und ihre Größenordnungen sollen logisch nachvollziehbar sein.

Die Plausibilität der Energiemessung kann über zwei Wege überprüft werden.

Zusätzlich zum *RAPL*-Chip der CPU berichtet auch der *IPMI*-Chip die Energieaufnahme des Servers (IPMI – Intelligent Platform Management Interface). Der *IPMI*-Chip ist nicht an der CPU, sondern als extra Chip auf dem Motherboard untergebracht. Er misst den Stromverbrauch des gesamten Servers, also auch inklusive aller Schnittstellen, Speicherbausteine, Festplatten und sonstigen Controllern. Dadurch liegt der gemessene Energieverbrauch des *IPMI*-Chips regelmäßig höher als der des *RAPL*-Chips. Beim vorliegenden Messystem zeigt er die gleichen Ausschläge wie der *RAPL*-Chip, jedoch mit einer höheren Leistung von ungefähr plus 20W. Allerdings ist der *IPMI*-Chip langsamer getaktet, weshalb nur Plausibilität und nicht exakte Übereinstimmung gezeigt werden kann.

Als zweite Möglichkeit kann die Messung der Energieaufnahme auch mit einem externen Energie- und Leistungsmessgerät an der Steckdose verifiziert werden. Das Vorgehen ist schon vielfältig untersucht worden und kann in der Literatur nachgelesen werden, beispielsweise bei Khan und Mazouz (Khan et al. 2018; Mazouz et al. 2014).

Plausibilitätsprüfung mit Belastung der Prozessorkerne

Um zu testen, ob der Wertebereich der Leistungsmessung plausibel ist, können die Prozessorkerne gezielt voll ausgelastet werden. Wird nacheinander jeweils ein Kern von insgesamt 48 Kernen im zu messenden Ecoqube-Server mehr ausgelastet, sollte eine gleichmäßige Treppe in der Kurve der Leistungsaufnahme entstehen bis hin zur Maximalleitung des Prozessors, die durch die maximale Arbeitstemperatur des Prozessors limitiert ist, die sogenannte *Thermal Design Power* (TDP). Wird versucht, weitere (virtuelle) Kerne auszulasten, so sollte die Leistungsaufnahme nicht weiter steigen.

Dieser Test wird mithilfe eines *Python*-Skripts durchgeführt, dessen Code in Abbildung 26 dokumentiert ist. In jedem Schleifendurchlauf wird 20 Sekunden lang ein weiterer CPU-Kern ausgelastet, von anfänglich 1 Kern bis zu 60 Kernen. Nach jedem Stress-Befehl wird eine Pause von 20 Sekunden eingelegt und die Schleife dann wiederholt.

Abbildung 26: Python-Code zur stufenweisen Auslastung der CPU-Kerne

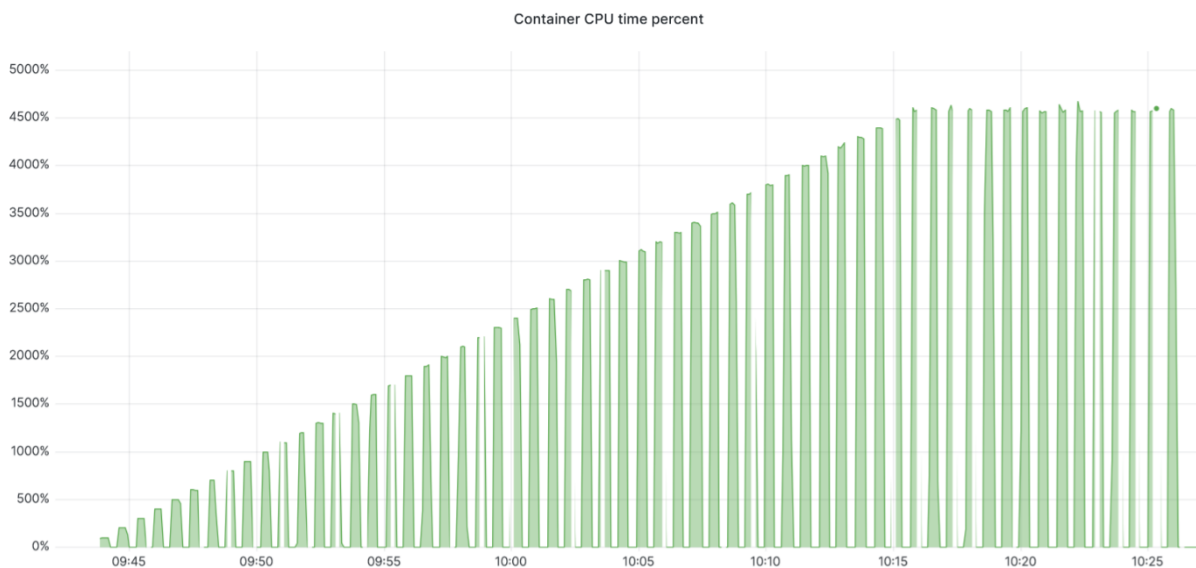
```
from time import sleep

import stressinjector as injector

from cpu import CPUStress
```

```
injector.CPUStress = CPUStress

if __name__ == '__main__':

  for i in range(60):

      CPUStress(seconds=20, cores=i+1)

      sleep(20)
```

Quelle: https://gitlab.com/softawere-hackathon/hackathon/fork-felix-oeko/-/blob/main/stress_test.py

Das Messergebnis dieses Testdurchlaufs für die prozentuale CPU-Auslastung ist in Abbildung 27 dokumentiert. Wie erwartet, steigt die prozentuale CPU-Auslastung von 100% für einen Kern jeweils mit steigender Kern-Auslastung in 100%-Schritten an. Der Mess-Server verfügt über insgesamt 48 Kerne. Die gemessene Auslastung, die durch den Mess-Container verursacht wird, erreicht jedoch nur 46 x 100% und stagniert dann bei diesem Wert, auch wenn das Mess-Skript bis zu 60 Kerne auslastet, also zu einer Überbelastung der CPU führen sollte (letzter Balken). Dass nicht alle 48 Kerne ausgelastet werden, ist dadurch erklärbar, dass das Mess-System selbst 2 Kerne für sich in Anspruch nimmt. Versucht der CPU-Stress-Befehl mehr Kerne in Anspruch zu nehmen, als verfügbar sind, so werden die verschiedenen Jobs auf die physisch verfügbaren Kerne aufgeteilt. Der CPU-Stress-Befehl wird dennoch nach 20 Sekunden beendet, auch wenn de facto weniger Rechenoperationen durchgeführt wurden. Bei „echten“ Rechenaufgaben, wie beispielsweise der Berechnung der Fakultät von 1 Million, hätte die Berechnung dann entsprechend länger gedauert.

Abbildung 27: CPU-Auslastung bei steigendem "CPUStress" von 1 bis 60 Kernen

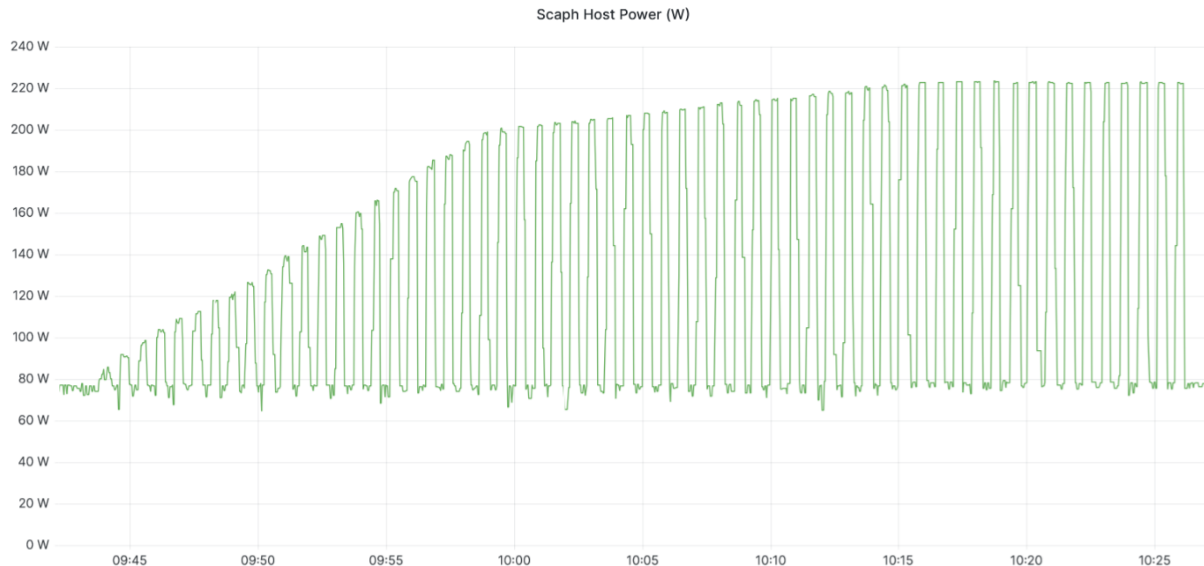


Quelle: Grafana Dashboard, SDIA, https://grafana.eco-qube.eu

Der Kurvenverlauf der CPU-Auslastung stellt erwartungsgemäß eine gleichmäßige Treppenstufe mit einer Stufenlänge von 20s und jeweils anschließenden 20s Lücken dar. Das Plateau ab dem 46. Kern zeigt die für das Test-Skript maximal verfügbare CPU-Auslastung.

Als nächstes wird der Energieverbrauch des Mess-Systems untersucht. In Abbildung 30 werden die Messwerte des *RAPL*-Chips dargestellt, die über das *Grafana-Dashboard* des Mess-Systems abgerufen werden können.

Abbildung 28: Energieverbrauch der CPU bei steigendem "CPUStress" von 1 bis 60 Kernen



Quelle: Grafana Dashboard, SDIA, https://grafana.eco-qube.eu

Der Energieverbrauch der CPU liegt im Idle-Zustand bei ca. 77 Watt. Ausgehend von dieser Baseline steigt der Energieverbrauch pro mehr belastetem Kern an. Von den Kernen 1 bis 23 steigt die Leistungsaufnahme in circa 5,3 Watt-Schritten auf 200 Watt an. Ab dem 24. Kern flacht sich die Treppenstufe ab und erreicht beim 46. Kern das Maximum von 223 Watt (1 Watt-Schritte). Es fällt auf, dass gerade nach der Hälfte der 46 verfügbaren Kerne, diese flachere Treppenstufe erreicht wird. Dies scheint eine Eigenart der CPU zu sein. Vermutlich ist dies in der Architektur der Kerne auf den CPU-Sockets und der Thermal Design Power begründet. Dies würde bedeuten, dass ab dem 23. Kern auch die Rechenleistung pro Kern geringer ist. Diese Hypothese kann hier nicht weitergehend untersucht werden. Ab dem 46. Kern bis zum 60. Kern nimmt die Leistungsaufnahme nicht weiter zu, sondern verbleibt auf dem Plateau von 223 Watt.

Was insgesamt auffällt, ist, dass die Leistungsaufnahme der Baseline von 77 Watt einige Schwankungen aufweist, die die Genauigkeit des Messergebnisses insbesondere bei kleinen CPU-Auslastungen negativ beeinflusst. Bei der Durchführung von Messungen über die maximale CPU-Auslastung sollte daher darauf geachtet werden, möglichst viele CPU-Kerne auszulasten (beispielsweise durch parallelen Aufruf des gleichen Tests in mehreren Instanzen) oder durch längere Messintervalle, bei denen sich die Schwankungen nivellieren.

Bewertung der Plausibilität

Die Messung der CPU-Auslastung und der CPU-Laufzeiten entspricht den Erwartungen, die aus dem Mess-Skript mit dem stufenweisen „*CPUStress*" abgeleitet wurden. Es stellt sich in Abbildung 27 mit der Treppenform und dem anschließenden Plateau ein plausibler Kurvenverlauf ein.

Aus der grafischen Darstellung in Abbildung 28 kann man zusammen mit den 20s-Pausen zwischen den einzelnen CPU-Auslastungen eine durchschnittliche Leistungsaufnahme von überschlägig 130 Watt berechnen. Dieser Wert stimmt gut mit dem durch das Messprotokoll

ausgegebenen Messergebnis von host_avg_consumption = 124.32 überein. Sowohl der Kurvenverlauf als auch das Messergebnis sind daher als plausibel zu bewerten.

2.3.3 Richtungssicherheit

- Richtungssicherheit: Es wird erwartet, dass eine stärkere Beanspruchung der Hardware durch mehr Rechenarbeit zu einem steigenden Energie- und Ressourcenbedarf führt, der sich in den Messergebnissen niederschlägt.

Messreihe zur Überprüfung der Skalierbarkeit

Das Skript `faculty.py` aus Abbildung 26 das die Fakultät der Zahl 1 Million berechnet, soll genutzt werden, um die Skalierbarkeit und Richtungssicherheit der Messung zu testen. Im Anhang D.4 findet sich die Instruktionsdatei für die Ausführung der Tests. Wie hinter dem einleitenden Kommentar „`## Your test starts here !`“ zu sehen ist, kann der Aufruf des zu untersuchenden Programms mit einer Schleife k-mal ausgeführt werden. Hierzu wird die Ausführbedingung der for-Schleife „`k<=1`“ entsprechend angepasst. Das Messergebnis des Energieverbrauchs der k-vielfachen Last sollte dem k-vielfachen Messergebnis einer einfachen Last entsprechen.

Es werden 10 unterschiedliche Messungen durchgeführt, bei denen die Anzahl der Aufrufe von `faculty.py` von eins („`k<=1`“) bis zehn („`k<=10`“) variiert wird. Es werden jeweils folgende Daten gemessen: Dauer des Prozesses zur Einrichtung und Installation aller Abhängigkeiten[79] [s], Dauer des Tests [s], Stromverbrauch des Tests [kWs]. Die Ergebnisse sind in Tabelle 4 aufgelistet.

Tabelle 4: Messreihe Mehrfachaufruf von faculty.py

Anzahl an Aufrufen von `faculty.py`	Dauer insgesamt [s]	Dauer Ausführung der Schleife [s]	Dauer Einrichtung Container [s]	Energie verbrauch [kWs]
1	30	9	21	1,40
2	39	18	21	3,00
3	48	27	21	4,40
4	57	37	20	5,98
5	68	45	23	7,35
6	75	54	21	8,90
7	84	63	21	10,40
8	93	73	20	11,96
9	103	82	21	13,43
10	111	91	20	14,88
11	121	100	21	16,36

Quelle: eigene Darstellung, Öko-Institut e.V.

[79] in diesem Schritt werden notwendige Python-Pakete (Abhängigkeiten) installiert die für die Ausführung der Tests erforderlich sind.

Folgende Mittelwerte und Standardabweichungen wurden daraus berechnet, in Klammern ist die relative Standardabweichung angegeben:

- Dauer der Einrichtung des Containers: 20,9 ± 0,876 [s] (±4,2%)
- Dauer des Tests pro Aufruf: 9,068 ± 0,084 [s] (±0,1%)
- Stromverbrauch pro Aufruf: 1,486 ± 0,011 [kWs] (±0,7%)

Bewertung der Richtungssicherheit

Die relative Abweichung für die Dauer der Einrichtung des Containers ist kleiner als 5%, für die Dauer des Tests pro Aufruf und den Stromverbrauch pro Aufruf kleiner als 1%. Das Messergebnis des Stromverbrauchs für die k-fache Ausführung des Lasttreiber-Skriptes faculty.py entspricht sehr genau dem k-fachen Messergebnis einer einfachen Last. Damit ist die Richtungssicherheit mit einer Abweichung von weniger als 1% für eine simple Rechenaufgabe gezeigt. Außerdem kann damit gezeigt werden, dass die Energiemessung innerhalb des Messsystems tatsächlich zum richtigen Zeitpunkt (nach der Einrichtung des Containers) beginnt und aufhört und nicht etwa durch die Installation des Betriebssystems, des Software-Stacks oder der Datenauswertung mit get_metrics.py beeinflusst wird.

2.3.4 Reproduzierbarkeit

Reproduzierbarkeit: Die Messungen müssen zu einem späteren Zeitpunkt (und ggf. mit einem eigenen Messaufbau) wiederholt werden können, und die Streuung der Messergebnisse (Standardabweichung) soll unterhalb einer für die Anwendung akzeptablen Schwelle liegen.

Messreihe: Unit Tests von Bibliotheken

Das Testen von Software erzeugt eine Last, die sich eignet, den Stromverbrauch zu messen. Die Unit- und Integrationstests von Bibliotheken können somit als reproduzierbare Standardnutzungsszenarien genutzt werden (vergleiche Standardnutzugsszenario in (Gröger et al. 2018)). Der Continuous Integration and Continous Delivery (CI/CD) Test ermöglicht es, alle Funktionen einer Software einmal aufzurufen, um die Erfüllung der Aufgaben zu überprüfen und Softwarefehler erkennen zu können.

In folgender Tabelle 5 sind die vom Forschungsteam ausgewählten Bibliotheken aufgelistet, deren Energieverbrauch exemplarisch gemessen wurde. Sie sind in sechs verschiedenen Programmiersprachen geschrieben und weisen eine hohe Verbreitung bei GitHub auf. Für alle liegen automatisierte Tests (CI/CD) vor und alle sind open source. Sie werden mehrmals wöchentlich gepflegt und überarbeitet.

Tabelle 5: Ausgewählte Bibliotheken

Bibliothek	Sprache	Positive Bewertungen auf GitHub	Größe [MB]	Alle CI/CD Tests bestanden	Letztes Update vor
arrow[80]	C++	10.700	26,6	X	
express[81]	JavaScript	55.800	0,8	X	~ 12 h
Django[82]	Python	61.900	14,9	X	~ 24 h

[80] https://github.com/apache/arrow

[81] https://github.com/expressjs/express

[82] https://github.com/django/django

Bibliothek	Sprache	Positive Bewertungen auf GitHub	Größe [MB]	Alle CI/CD Tests bestanden	Letztes Update vor
Flask[83]	Python	57.700	3,1		~ 7 d
gin[84]	Go Lang	55.300	0,4	X	~ 19 h
harfbuss[85]	C++	2.300	97,6	X	~ 2 d
hugo[86]	Go Lang	56.900	27,6	X	~ 3 d
jadx[87]	Java	28.900	4,3	X	~ 2 d
material-ui[88]	JavaSkript	75.200	67,8	X	~ 12 h
radare2[89]	C	15.800	39,3		~ 16 h
vue[90]	JavaSkript	193.000	22,6	X	~ 21 d

Quelle: eigene Darstellung, Öko-Institut e.V.

Es wurden jeweils mehrere Testdurchläufe durchgeführt und dabei die Dauer [s] und die durchschnittliche Leistungsaufnahme [W] gemessen, nach der in Kapitel 2.2 beschriebenen Methode. Aus dem Produkt beider Werte ergibt sich der Energieverbrauch in Wattsekunden [Ws], der für die Darstellung in Wattstunden [Wh] umgerechnet wurde. In Tabelle 6 sind die gemessenen Mittelwerte nach Energieverbrauch aufsteigend aufgelistet. Die Durchläufe wurden händisch gestartet, während der Messung ist mehrfach die Ausgabe der Messergebnisse fehlgeschlagen, z.B. weil der Server abstürzte oder der Messaufbau weiterentwickelt wurde. Die Tests sind zu unterschiedlichen Tagen über mehrere Wochen hinweg gestartet worden. Dennoch weisen die Ergebnisse eine kleine Streuung auf, was auf eine Reproduzierbarkeit der Messung zumindest auf demselben Server hindeutet.

Tabelle 6: Gemessene Mittelwerte für verschiedene Bibliotheken

Bibliothek	Anzahl Messungen	Größe [MB]	Dauer [s]	Leistungsaufnahme [W]	Energie-verbrauch [Wh]	Standardabweichung Energieverbrauch [%]
arrow	6	26,6	121	70,743	2,378	0,4
express	1	0,8	5,8	53,447	0,086	-
flask	7	3,1	9,2	66,882	0,172	1,3
gin	2	0,4	13,2	72,953	0,267	7,1
guava	6	14,3	892	66,314	16,431	0,7

[83] https://github.com/pallets/flask

[84] https://github.com/gin-gonic/gin

[85] https://github.com/harfbuzz/harfbuzz

[86] https://github.com/gohugoio/hugo

[87] https://github.com/skylot/jadx

[88] https://github.com/mui-org/material-ui

[89] https://github.com/radareorg/radare2

[90] https://github.com/vuejs/vue

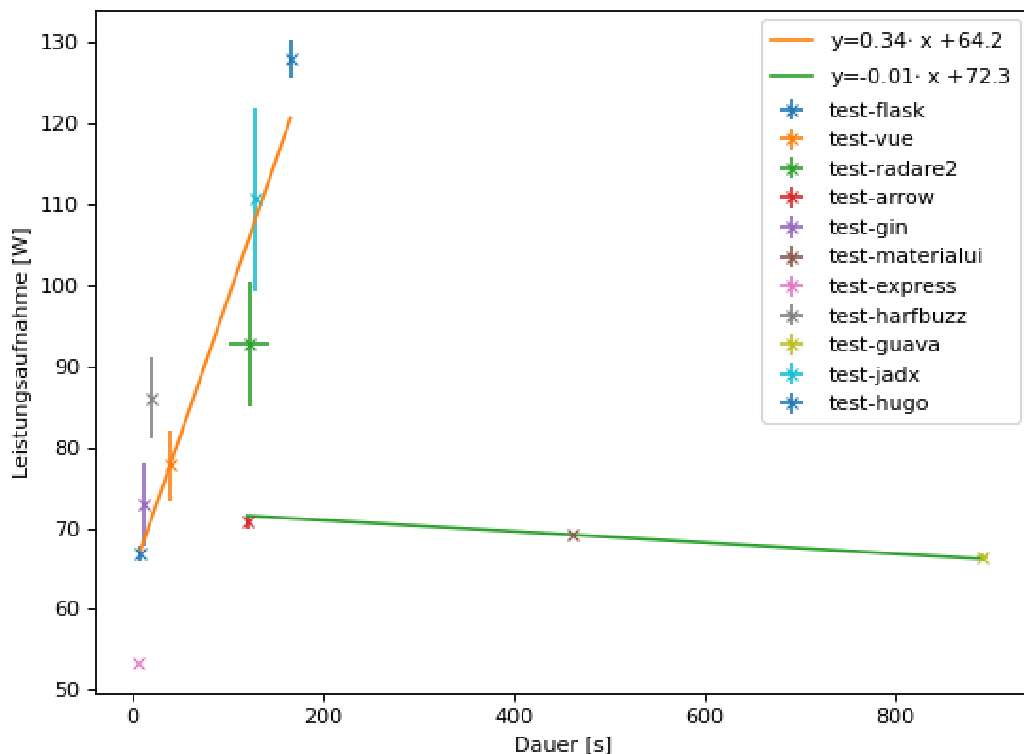
Bibliothek	Anzahl Messungen	Größe [MB]	Dauer [s]	Leistungsaufnahme [W]	Energie-verbrauch [Wh]	Standardabweichung Energieverbrauch [%]
harfbuzz	8	97,6	20,5	86,013	0,49	5,7
hugo	5	27,6	166,3	127,931	5,911	2,8
jadx	11	4,3	130	110,609	3,995	12,9
materialui	1	67,8	462,1	69,109	8,871	-
radare2	18	39,2	123,3	92,714	3,174	7,7
vue	20	22,6	41	77,782	0,885	4,5

Quelle: eigene Darstellung, Öko-Institut e.V.

Es ist zu erkennen, dass die Standardabweichung des Energieverbrauches bei den verschiedenen Bibliotheken sehr unterschiedlich groß ist, zwischen 0,4% und 12,9%. Sie hängt weder mit der Größe der Bibliothek noch mit der Anzahl an Messungen zusammen. Im Folgenden wurden daher für alle Bibliotheken die Streuung von Leistungsaufnahme und Dauer der Tests untersucht. Jede Bibliothek ist in einer anderen Farbe in Abbildung 29 dargestellt.

Man erkennt zwei verschiedene Phänomene in dieser Grafik. Zwischen 2s und 2min Durchführungsdauer nimmt die mittlere Leitungsaufnahme mit der Durchführungsdauer zu. Dieser Zusammenhang wird mit der orangenen Ausgleichslinie dargestellt. Davon weichen drei Bibliotheken ab: *arrow*, *materialui* und *guava*. Die grüne Ausgleichslinie zeigt, dass bei sehr großen Durchführungsdauern von mehr als 2min die Leistungsaufnahme mit der Durchführungsdauer leicht abnimmt.

Abbildung 29: Zusammenhang von Messdauer und Leistungsaufnahme - thermische Effekte



Quelle: Screenshot aus manueller Auswertung mit NocoDB Daten, Öko-Institut e.V.

Möglicherweise ist dies auf das Thermomanagement der CPU zurückzuführen. Damit die CPU nicht zu heiß wird und ihre „*Thermal Design Temperatur*“ nicht überschreitet, wird die Taktfrequenz der CPU automatisch abgesenkt. Vermutlich hat die CPU einen „Turbobereich“ mit maximaler Taktfrequenz, der bei kurzen Rechenaufgaben ausgeführt wird, der aber bei längeren Rechenaufgaben und der damit verbundenen Hitzeentwicklung wieder verlassen wird. Dies passiert auch bei der mehrfachen Ausführung der Tests unmittelbar hintereinander. Mit der Taktung nimmt auch die Leistungsaufnahme und damit die Wärmeproduktion ab. Bei Laufzeiten deutlich länger als 2min sind die Schwankungen der Messergebnisse deutlich kleiner als bei Laufzeiten kürzer als 2min. D.h. die Verzerrung durch das Aufwärmen der CPU fällt bei langen Laufzeiten nicht so sehr ins Gewicht.

Messreihe: Fibonacci-Reihe in verschiedenen Programmiersprachen

Die sogenannte *Fibonacci-Reihe*[91] lässt sich in jeder Programmiersprache mit wenigen Code-Zeilen berechnen. Die *Fibonacci-Reihe* ist eine Abfolge von Ganzzahlen, bei der die nächste Zahl die Summe der beiden vorherigen Zahlen ist. Nach der Definition des italienischen Mathematikers *Leonardo Fibonacci*[92] beschreibt das Populationswachstum eines gedachtes Kaninchenpärchen, das sich beginnend mit 1 und 1 sukzessive fortpflanzt (demnach: 1, 1, 2, 3, 5, 8, 13, ...). Die Ausführung eines rekursiv programmierten (sich selbst aufrufenden) Algorithmus wird im Rahmen dieses Projektes genutzt, um die Energieaufnahme und die CPU-Laufzeit bei verschiedenen Programmiersprachen zu messen.

[91] https://de.wikipedia.org/wiki/Fibonacci-Folge

[92] * ca. 1170, † 1240

In Abbildung 30 wird der Algorithmus exemplarisch in der Programmiersprache *Python* dargestellt. Die Ausführungsdauer steigt exponentiell mit der übergebenen Eingangsgröße `n`. Für die hier durchgeführten Messungen wurde die *Fibonacci-Zahl* für die Eingangsgröße 47 berechnet, was einen knapp 10 Milliarden-fachen rekursiven Aufruf der Funktion `fib(n)` zur Folge hat und dadurch eine deutliche CPU-Last erzeugt (fib(47) = 2.971.215.073).

Abbildung 30: Rekursive Berechnung der Fibonacci-Zahl mit der Programmiersprache *Python*

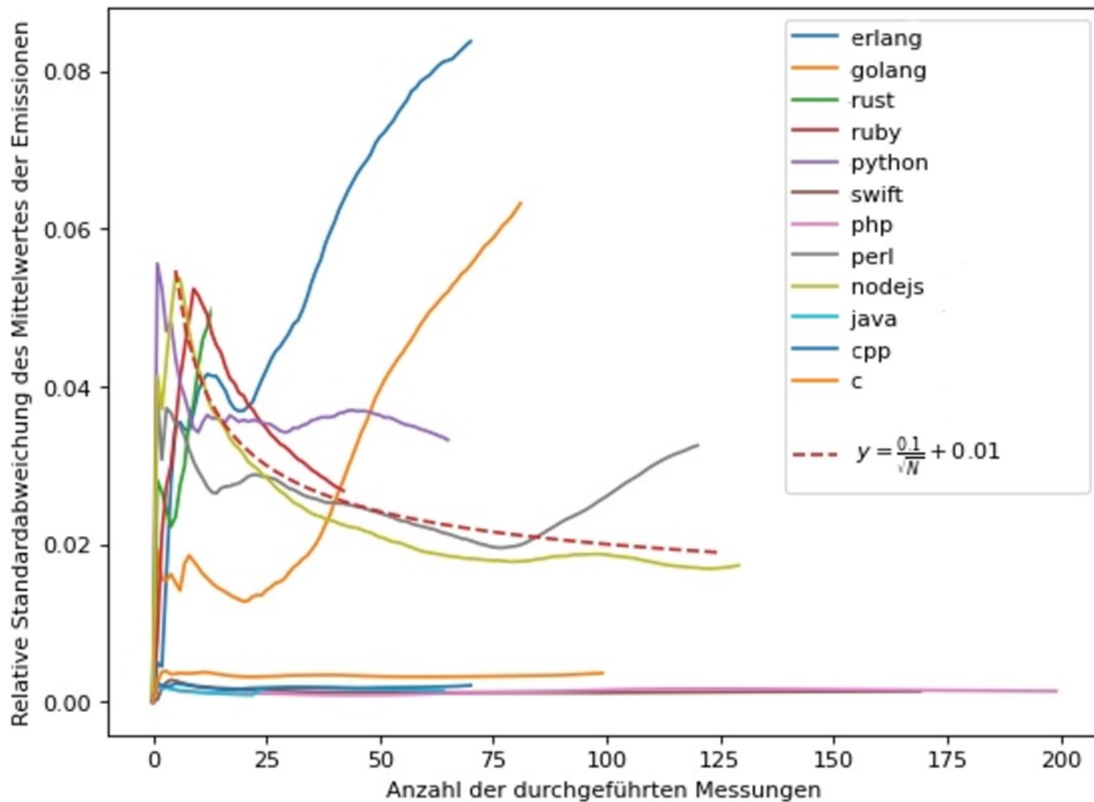
```
def fib(n):

if n <= 1: return n

return fib(n - 1) + fib(n - 2)
```

Quelle: SDIA 2023, https://gitlab.com/softawere-hackathon/language-tests/fibonacci-benchmark

Führt man mehrere Messungen hintereinander durch und bildet jeweils den Mittelwert aller bis dahin durchgeführten Einzelmessungen, ergibt sich ein Kurvenverlauf des Mittelwertes über die steigende Anzahl an Messungen. Aus diesem Kurvenverlauf kann wiederum die Standardabweichung der Mittelwerte berechnet werden. Alle Mittelwerte, die gemessen wurden, sind um den genauesten möglichen Wert verteilt. Bei statistisch unabhängigen Schwankungen ist die Standardabweichung des Mittelwertes genau die Standardabweichung der Messreihe geteilt durch die Wurzel der Anzahl der Messungen. Mit zunehmender Anzahl der Messungen sollte also die relative Abweichung des Mittelwertes proportional zu eins durch Wurzel N gegen Null gehen. Je länger die Messreihe, desto präziser wird der Mittelwert. Berechnet man für eine Messreihe in Abhängigkeit von der Anzahl der Messwerte den Mittelwert und dessen Standardabweichung, sollte es möglich sein, daraus eine Länge der Messreihe abzulesen, ab der der Mittelwert präzise genug ist.

In Abbildung 31 ist zu erkennen, dass der erwartete Kurvenverlauf von 1 durch Wurzel N (gestrichelte Linie) durch die hier durchgeführten Messungen nur teilweise bestätigt wird. Erst ab 10 Messungen tritt der erwartete Effekt von $1/\sqrt{N}$ auf. Je mehr Messwerte aufgenommen werden, desto präziser wird der Mittelwert. *Nodejs* und *Ruby* weisen einen ähnlichen Kurvenverlauf auf wie die gestrichelte Linie, welche die theoretische Erwartung widerspiegelt. Bei manchen Sprachen verlaufen die Kurven jedoch ganz anders als erwartet. Im folgenden Abschnitt lässt sich feststellen, dass dies an systematischen Fehlern liegt, u.a. an einer Drift der Leistungsaufnahme.

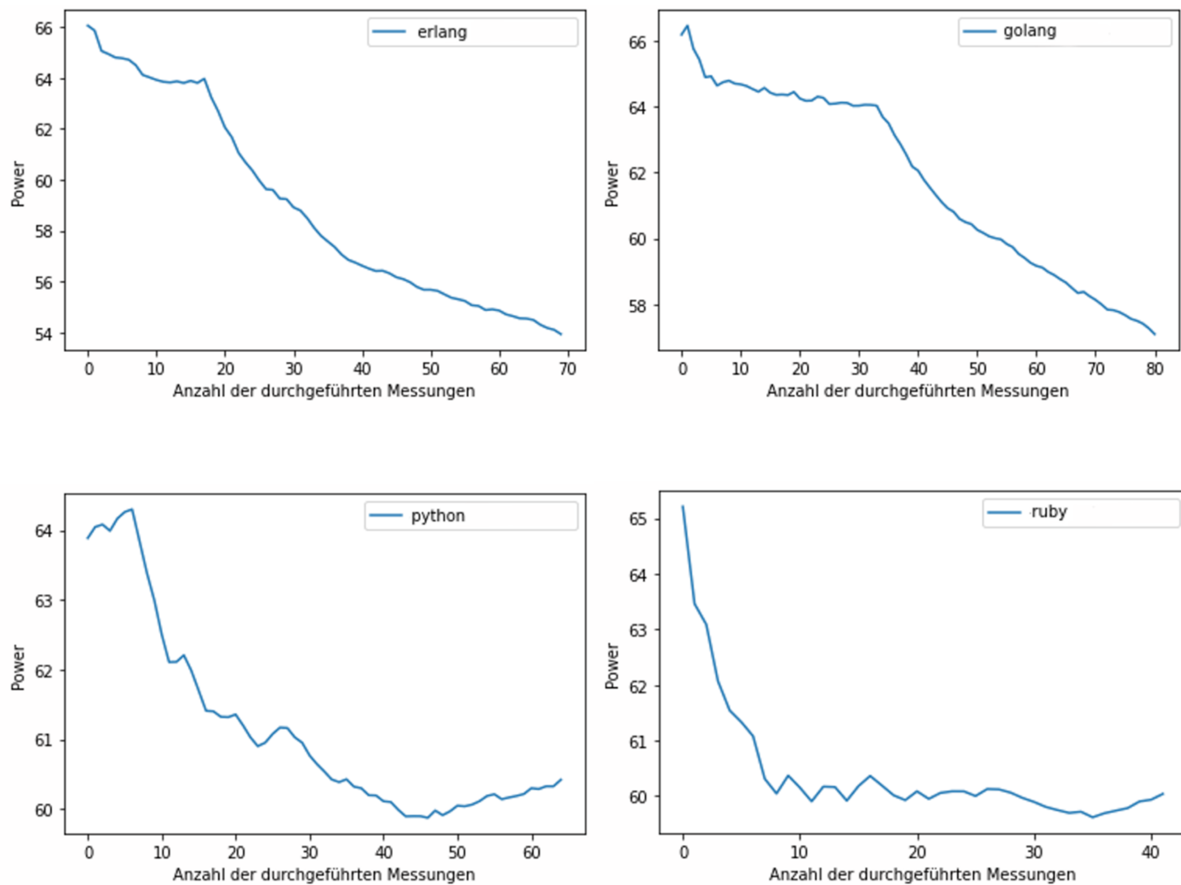
Abbildung 31: Relative Standardabweichung der Mittelwerte der Emissionen als Funktion der Länge der Messreihe



Quelle: Screenshot aus manueller Auswertung mit NocoDB Daten, Öko-Institut e.V.

Die starke Zunahme der relativen Standardabweichung des Mittelwertes bei *Erlang* (blau) und *Golang* (orange) ist in einem Drift des Mittelwertes begründet. Dies ist in Abbildung 32: Drift der Leistungsaufnahme zu erkennen. Die gemessene mittlere Leistungsaufnahme des Tests nimmt bei häufiger Wiederholung nahezu 10% ab, bei *Erlang* und *Golang* sogar fast 20%. Bei mehr Messungen wird der Mittelwert also weniger präzise, die Abweichung des Mittelwertes nimmt zu, wie in Abbildung 31 zu erkennen ist.

Abbildung 32: Drift der Leistungsaufnahme



Quelle: Screenshot aus manueller Auswertung mit NocoDB Daten, Öko-Institut e.V.

Da die Messungen automatisch direkt hintereinander ausgeführt wurden, kann angenommen werden, dass die CPU-Optimierung die Sprachen *GoLang* und *Erlang* bei wiederholter Kompilierung effizienter ausführt. Die Hardware ist bei mehrfacher Ausführung zunehmender Wärme und anderen externen Störfaktoren ausgesetzt, die nicht kontrolliert werden können.

In Fällen, in denen sich der Mittelwert bei mehrfacher Messung stabilisiert, kann davon ausgegangen werden, dass 20 Messungen ausreichen, um ein valides Ergebnis zu erhalten. Ein Vergleich der Tabelle 7 mit der Abbildung 31 zeigt, dass diejenigen Tests mit einer hohen Standardabweichung in der Gesamtverteilung der Emissionsmessung in Tabelle 7ebenfalls einen untypischen Kurvenverlauf der relativen Standardabweichung des Mittelwertes in Abbildung 31 aufweisen. Der Unterschied der relativen Standardabweichung in Tabelle 7 beträgt ungefähr einen Faktor 10: *java*, *swift*, *php*, *c++*, *c* weisen sehr geringe Standardabweichungen auf (1-2%), *erlang*, *golang*, *rust*, *ruby*, *perl*, *nodejs* und *python* weisen hingegen große Standardabweichungen auf (12-19%).

Bewertung der Reproduzierbarkeit und systematische Fehlerquellen

Aus den durchgeführten Tests kann man ableiten, dass der Messaufbau durchaus in der Lage ist, eine Präzision von 1-2% zu erreichen. Wenn dies deutlich verfehlt wird, können verschiedene systematische Fehlerquellen benannt werden:

- Die Messung war kürzer als 2s.
- Die CPU-Auslastung durch die Messung war so gering, dass die Schwankungen der Baseline (unbelastete CPU) das Messergebnis dominieren.

- Es wurden deutlich weniger als 20 Messungen durchgeführt.
- Das Messergebnis ist einem Drift aufgrund von Hardwareoptimierung und Optimierung des Compilers ausgesetzt (z.B., wenn der Test sehr einfach ist).
- die Taktung der CPU verändert sich aufgrund von thermischen Effekten, d.h. die Anfangstemperatur ist über die Messreihe nicht konstant.

Treten diese Effekte ein, werden Abweichungen von 10% und mehr erwartet.

Workarounds könnten folgende Vorgehensweisen sein.

- Zu geringe Messdauer: Mit einer *for*-Schleife kann die Messung mehrfach und damit länger ausgeführt werden.
- Zu geringe CPU-Auslastung: Durch parallele Ausführung desselben Messskriptes, beispielsweise parallel auf unterschiedlichen Kernen, kann die CPU-Auslastung so erhöht werden, dass der Einfluss der Baseline reduziert wird.
- Thermische Probleme: Bei der Wiederholung von Messungen 30s Pause zwischen den Messungen (oder weniger empfohlen, da energieaufwändiger: 30s am Anfang alle Kerne mit Stress auf eine Maximaltemperatur bringen und ohne Pause die Tests fortlaufend wiederholen).

2.3.5 Bewertung der Methodik insgesamt

Zusammenfassend können die vier Anforderungen an die Methode folgendermaßen bewertet werden.

- Verständlichkeit: Die Messmethode ist ein Werkzeug, das von Softwareentwickler*innen für Softwareentwickler*innen entwickelt wurde. Für ein volles Verständnis aller Softwaremodule sind Programmierkenntnisse notwendig. Die Anwendung ist jedoch auch interessierten Laien zugänglich, sofern diese mit den Grundfunktionen von *Git* vertraut sind.
- Plausibilität: Das Zustandekommen der Ergebnisse und ihrer Größenordnung können nachvollzogen werden.
- Richtungssicherheit: Das Messergebnis der k-vielfachen Last entspricht sehr genau dem k-vielfachen Messergebnis einer einfachen Last. (Das Ergebnis der Messung von 10 Ausführungen eines Tests innerhalb einer Messung ist das zehnfache Ergebnis der Messung einer Ausführung desselben Tests.)
- Reproduzierbarkeit: Messungen, die kürzer als 2s dauern, können nicht erfasst werden, weil die Taktung des RAPL-Chips dies nicht zulässt. Bei Messungen, die länger als 2min dauern, ist auf den verwendeten Servern mit thermischen Effekten zu rechnen, welche die Taktung der CPU reduzieren. Außerdem wurde gezeigt, dass die Energiemessung tatsächlich zum richtigen Zeitpunkt beginnt und aufhört und nicht etwa durch die Installation des Betriebssystems und Software-Stacks beeinflusst wird. Eine Messgenauigkeit von 1-2% ist möglich, große Abweichungen weisen auf oben genannte systemische Messfehler hin.

2.4 Anwendung der Methode

Während der Untersuchung des Messaufbaus sind zwei interessante Forschungsfragen aufgekommen, die mit dem Messaufbau beantwortet werden können:

- Welche Programmiersprachen brauchen am wenigsten Energie?

► Ist die Größe einer Bibliothek ausschlaggebend, ob sie viel oder wenig Energie verbraucht?

Im Folgenden wird mit dem Messaufbau untersucht, welche Programmiersprachen wie effizient die Fibonacci-Reihe berechnen können. Eine ähnliche Studie wurde von Pereira in 2021 durchgeführt (Pereira et al. 2021). Anders als beim SoftAWERE Versuch wurden nicht Fibonacci Zahlen berechnet, sondern es wurde das Computer Language Benchmarks Game[93] als Test-Software verwendet. Anders als Pereira, wurde in SoftAWERE nicht der Energieaufwand verglichen, sondern der Mittelwert des berechneten Treibhauspotenzials (in g CO_2e). Es ist zu beachten das die Studie von Pereira auf wesentlich komplexeren Tests und Vergleichen aufbaut, die nicht direkt mit der Vorgehensweise der SoftAWERE Tests vergleichbar sind.

Die Ergebnisse sind dennoch ähnlich, wobei Programmiersprachen wie C und C++ bei Pereira besser abschneiden. Das Mittelfeld in Pereiras Studie (Rust, Go, JavaScript) schneidet in den SoftAWERE-Tests besser ab. PHP, Ruby und Java sind in beiden Tests eher mit einem höheren Energie-Aufwand verbunden. Python schneidet in den SoftAWERE Tests wesentlich besser ab.

Messergebnisse der untersuchten Programmiersprachen

In der Tabelle 7 sind die Programmiersprachen mit den Ergebnissen der Energieverbrauchsmessungen aufgeführt. Die Messungen sind nach dem in Abschnitt 4.2.2 beschriebenen Fibonacci-Berechnungen durchgeführt worden. Die Tabelle 6 enthält die Anzahl der Testdurchläufe, den Mittelwert und die Standardabweichung. In der Tabelle 7 wurde eine Sortierung nach aufsteigendem Treibhauspotenzial vorgenommen.

Tabelle 7: Treibhauspotenzial zur Berechnung der Fibonacci-Zahl Fib (47) mit unterschiedlichen Programmiersprachen

Programmiersprache	Anzahl Testdurchläufe	Mittelwert des berechneten Treibhauspotenzials [g CO_2e]	Relative Standard-abweichung des Treibhauspotenzials
JavaScript (in NodeJS)	131	0,015	17%
Perl	122	0,017	17%
Erlang	72	0,018	17%
Golang	83	0,019	18%
Ruby	44	0,02	12%
Rust	18	0,021	14%
Python	67	0,021	12%
C	101	0,366	1%
C++	72	0,373	1%
Java (in JVM)	66	0,659	2%

[93] siehe https://benchmarksgame-team.pages.debian.net/benchmarksgame/index.html

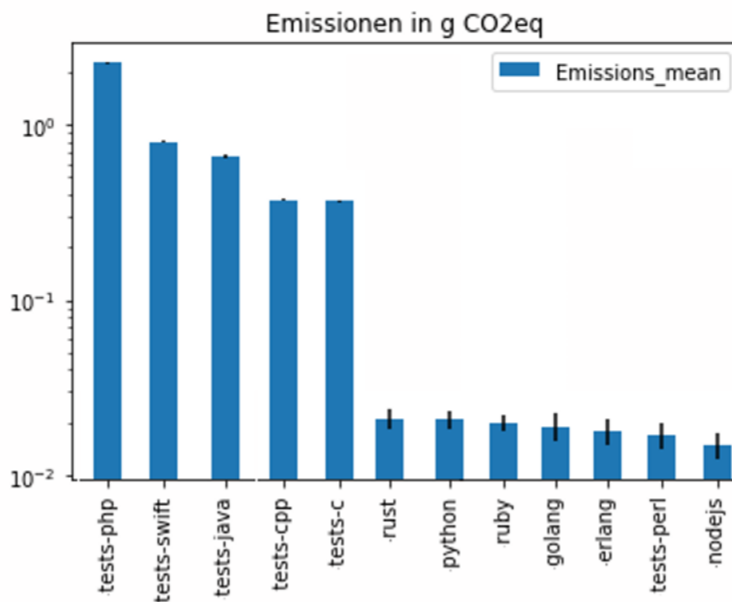
Programmiersprache	Anzahl Testdurchläufe	Mittelwert des berechneten Treibhauspotenzials [g CO_2e]	Relative Standard-abweichung des Treibhauspotenzials
Swift	171	0,802	1%
PHP	201	2,254	1%

Quelle: eigene Darstellung, Öko-Institut e.V.

In Tabelle 7 wird ersichtlich, dass die relative Standardabweichung bei geringen Treibhausgasemissionen mit 17% deutlich höher ist als bei den Programmiersprachen mit hohen Emissionen. Dies ist dadurch erklärbar, dass die Messungenauigkeit bei sinkenden Programmlaufzeiten abnimmt. Liegt die Programmlaufzeit unterhalb von 2 Sekunden, so sind die Messergebnisse gänzlich unbrauchbar.

Die Treibhausgasemissionen (blauer Balken) und ihre Standardabweichung (schwarzer Strich) werden zusätzlich in Abbildung 33 grafisch dargestellt. Die vertikale Skala der Treibhausgasemissionen ist logarithmisch, um die kleinen Ergebnisse und deren Fehlerbalken erkennen zu können.

Abbildung 33: Treibhauspotenzial zur Berechnung der Fibonacci-Zahl Fib (47) mit unterschiedlichen Programmiersprachen



Quelle: Screenshot aus manueller Auswertung mit NocoDB Daten, Öko-Institut e.V.

Der Unterschied bei den Treibhausgasemissionen zur Berechnung der Fibonacci-Zahl F(47) liegt zwischen den beiden Extremen 0,015 g CO_2e (*Nodejs*) und 2,254 g CO_2e (*Php*), die um einen Faktor von 150 auseinander liegen.

Die Ursachen für diese großen Unterschiede werden im Rahmen des vorliegenden Projektes nicht untersucht. Grundsätzlich lässt sich jedoch sagen, dass der rekursive Aufruf einer Funktion, in diesem Testfall ein rund 10-Milliarden-facher Selbstaufruf, keine typische Rechenaufgabe für eine Programmiersprache darstellt. Es hat den Anschein, dass einige Programmiersprachen damit besser zurechtkommen als andere. Diese Ergebnisse lassen sich

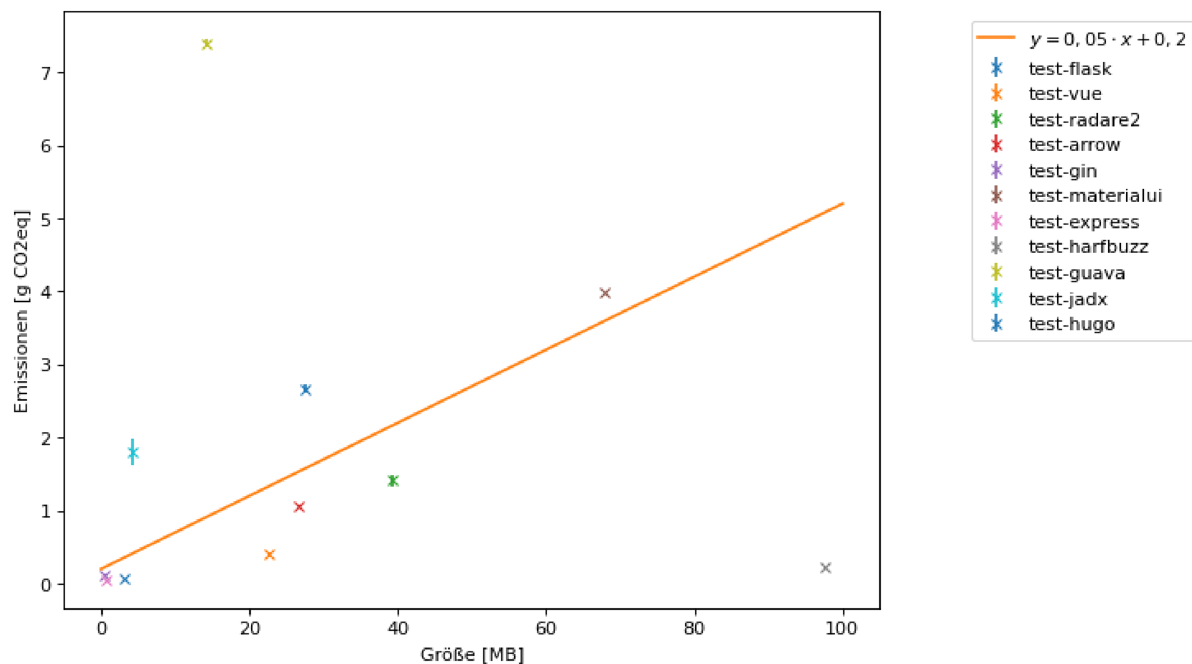
nicht auf die allgemeine Performanz der Programmiersprachen übertragen. Im Einzelfall sollte geprüft werden, welche Aufgaben durch welche Programmiersprache am effizientesten erfüllt werden können.

Die Forschungsfrage „Welche Programmiersprachen brauchen am wenigsten Energie?“ kann nicht alleine anhand der Berechnung der Fibonacci-Zahl beantwortet werden.

Korrelation zwischen Emissionen und Größe von Bibliotheken

Durch eine weitere Auswertung der Messreihe zur Durchführung von *Unit tests* verschiedener Bibliotheken wurde untersucht, ob die Größe der Bibliotheken mit deren Treibhauspotenzial in Beziehung stehen. Die Ergebnisse werden in Abbildung 34 visualisiert. Die Mittelwerte der verschiedenen Bibliotheken werden durch eine lineare Kurve interpoliert. Dabei gibt es mit den Bibliotheken *Guave* (deutlich höhere Emissionen) und *Harfbuzz* (deutlich geringere Emissionen) zwei Ausreißer von der Ausgleichskurve. Die Existenz dieser beiden Ausreißer zeigt, dass vereinfachende Annahme, dass die erzeugte Last proportional zur Größe der Bibliothek sei, nicht allgemeingültig ist. Für die Ausgleichskurve (orangene Linie) wurden die beiden Ausreißer ausgeschlossen. Dennoch beträgt die Standardabweichung der Steigung 0,033 und damit mehr als 60%. Die Korrelation zwischen Emissionen und der Größe ist also nur sehr schwach ausgeprägt.

Abbildung 34: Korrelation zwischen Emissionen und Größe von Bibliotheken



Quelle: Screenshot aus manueller Auswertung mit NocoDB Daten, Öko-Institut e.V.

Die Annahme, dass eine größere Bibliothek bei der Messung ihrer Tests mehr Treibhausgase verursacht, ist also nicht zulässig. Um die ökologische Wirkung von Software zu bestimmen, reicht es nicht aus, die Anzahl der Zeilen an Code als Kriterium heranzuziehen.

Die Forschungsfrage „Kann man an der Größe einer Bibliothek erkennen, ob sie viel oder wenig Energie verbraucht?“ kann daher weitgehend mit „nein“ beantworten. Jedoch gibt es einen gewissen Zusammenhang zwischen der Anzahl an Code-Zeilen und dem Energiebedarf zur Ausführung der Unit-Tests. Daher sollte weitergehend untersucht werden, welche einfach zählbaren Kenngrößen dazu geeignet sind, den Energiebedarf von Software vorherzusehen.

3 Handlungsempfehlungen für Politik, Forschung und die Software-Community

Ergebnisse und Methodik in Normungsprozess einbringen

Innerhalb des Vorhabens sind verschiedene neue Methoden entstanden (siehe Abschnitt 6) die eine wichtige Basis für die Lebenszyklusanalyse von digitalen Produkten schaffen. Das Vorhaben hat es ermöglicht, die digitalen Ressourcenverbräuche mit ihrer Umweltwirkung zu verbinden, insbesondere in Umgebungen, in denen eine Messung nicht möglich ist, wie zum Beispiel virtualisierte Cloud-Infrastruktur.

Diese Methodik sollte in Normungsprozesse und Gremien eingebracht werden, insbesondere im Kontext der Bewertung von Umweltwirkungen von digitalen Services & Produkten. SoftAWERE-Light wird mathematisch berechnet und ist eine Schätzung, entwickelt für Umgebungen in denen Messschnittstellen nicht verfügbar sind. Die volle Variante von SoftAWERE basiert auf echten Messungen, die durch Schnittstellen aus dem Server und Betriebssystem bereitgestellt werden. Beide würden sich für die Normung eignen, je nach Zweck und Zielsetzung.

Wissensplattform schaffen

Eine große Frage für Software-Betriebe, IT-Dienstleister und Informatiker, die sich stellt ist: „Was ist zu tun, damit die Umweltwirkungen meiner Software verringert werden können?". Im Rahmen des Vorhabens wurde daher ein erster Schritt zu einer offenen Wissensplattform geschaffen, in der das aktuelle Wissen zur Messung von Umweltwirkungen und Ressourcenverbräuchen konsolidiert wird. Unter dem Namen (S12Y.wiki[94], „Sustainability Wiki") steht eine Übersicht von Werkzeugen bereit und bietet Verweise auf offene Datensätze.

Es ist geplant, das Wiki über weitere Leitfäden und Handlungsempfehlungen sowie Benchmarking von Architekturen und Vorgehensweisen zu erweitern und Unternehmen und Bildungseinrichtungen Zugriff zu unabhängigen Lerninhalten zu Green-Coding und Transparenz in der digitalen Welt zu ermöglichen.

Europäische und internationale Zusammenarbeit fördern

Internationaler Wettbewerb soll nicht um die Messmethodik herum entstehen, sondern es sollten Ansätze zur Verbesserung und Reduzierung von Umweltwirkungen und Ressourcenverbrauch gefördert werden; für die Entwicklung der Methodik und die Normung von Kennzahlen und Vorgehensweisen lautet das Leitmotiv „Zusammenarbeit".

Europa oder Deutschland sollte hier mit der bestehenden Forschung und existierendem Know-How die Vorreiterrolle einnehmen und die europäische und internationale Gemeinschaft zur Zusammenarbeit auffordern und einladen.

Ganzheitliche Betrachtung priorisieren

Eine einseitige Betrachtung von Energieeffizienz oder von Umweltwirkungen ist nicht zielführend. Nur eine ganzheitliche Betrachtung von Ressourcenaufwänden (Energie und Rohstoffe) kann eine systematische Verbesserung der Umweltwirkung von Software und digitalen Produkten realisieren.

Das Ziel: Mit so wenig Ressourcen, Energie und Rohstoffen, eine Funktion, wie eine Videokonferenz bereitzustellen, ohne dabei kritische Grundeigenschaften wie Sicherheit und Zugänglichkeit negativ zu beeinflussen.

[94] Das S12Y.wiki ist unter der Domain: https://s12y.wiki verfügbar.

Am Beispiel von IT-Equipment lässt sich das Risiko einer einseitigen Betrachtung bereits darstellen: Neue Laptops oder Server sind zwar oft energieeffizienter, bezieht man aber den Rohstoffverbrauch mit ein, der durch das Ersetzen des Equipments alle 3–5 Jahre entsteht, ist die Gesamtumweltbilanz zunehmend negativ.

3.1 Kennzeichnung

Die Kennzeichnung von Software-Bibliotheken- und Komponenten hat zum Ziel, die Umwelteigenschaften von Software für Softwareentwickelnde und -nutzende sichtbar zu machen. Durch die Kennzeichnung kann zum einen signalisiert werden, dass die Bibliothek oder Komponente mit der SoftAWERE-Methode untersucht und optimiert wurde, und zum anderen können ausgewählte Messergebnisse dargestellt werden.

Zwei Arten von Softwareentwicklung, die am Anfang der digitalen Wertschöpfungskette stehen, sind für die SoftAWERE-Methoden besonders interessant:

- Art 1: Das Entwickeln von Bibliotheken, Compilern, Sprachen etc. und
- Art 2: das Einbinden von vorhandenen Bibliotheken, Funktionen und Modulen in Applikationen oder weitere Softwareprogramme.

Wenn im ersten Schritt die Energieeffizienz der Bibliotheken optimiert wird, skaliert die Einsparung im zweiten Schritt, da Bibliotheken z.T. millionenfach in unterschiedliche Softwareprodukte eingebunden werden. Es sollte erkennbar sein, ob eine Bibliothek unter Berücksichtigung der Energieeffizienz optimiert wurde und ob der Energiebedarf von Version zu Version eher zu- oder abnimmt.

Die Energieeffizienz einer Open-Source Bibliothek kann zunächst durch den Energieverbrauch des Servers während des Ausführens aller Unit- und Integrationstests beschrieben werden. Die Messmethode von SoftAWERE funktioniert genau nach diesem Prinzip und gibt eine grobe Orientierung für die größten Einflüsse und „Energie-bugs".

Das Messskript loggt die Inanspruchnahme digitaler Ressourcen der unterliegenden Hardware und die Energieaufnahme der CPU über die RAPL-Schnittstelle (running average power limit). Dieser ist in fast allen CPUs von Intel und AMD seit 2012 verbaut. In den meisten Fällen ist die CPU für den größten Teil der Energieaufnahme verantwortlich. Die Testumgebung des Vorhabens ist in eine eigenständige Gitlab-Runner-Instanz eingebettet, in der die zu untersuchende Bibliothek in einem Docker-Container einen CI/CD Built-Prozess durchläuft (siehe Kapitel 2.2). Das Messskript zeigt am Ende die Ergebnisse des Testlaufs (Energieverbrauch, Treibhausgasemissionen, Ressourcenverbrauch) in der Web-Konsole an (siehe Anhang D.5).

3.1.1 Energie-Kennzeichnung für Software

Um einen ersten Anreiz für Entwickler*innen zu schaffen, den Energieverbrauch ihrer Software zu messen und darüber zu informieren, würde eine *qualitative* Kennzeichnung genügen. Eine Energiekennzeichnung für Software sollte also im ersten Schritt darauf hinweisen, dass die gekennzeichnete Bibliothek unter dem Aspekt der Energieeffizienz geschrieben und optimiert wurde. Dafür sprechen mehrere Gründe:

Die Sensibilität für den Energieverbrauch von Software ist noch nicht im Mainstream angekommen. Die Sichtbarkeit zu stärken und eine schnelle und unkomplizierte Verbreitung des Labels ist zunächst stärker zu gewichten als quantitative Aspekte der Vergleichbarkeit und methodischer Präzision.

1. Die Messmethode zur Energiemessung ist Schwankungen von bis zu 10% ausgesetzt, da Compiler/Interpreter und CPU je nach Umgebungsrauschen verschiedene Optimierungsverfahren benutzen. Die Methode ist also zunächst nur dafür geeignet, die großen Energietreiber zu finden.
2. Quantitative Informationen verleiten zu vergleichenden Aussagen. Dies wäre aktuell nicht angemessen (siehe oben).
3. Open Source Bibliotheken werden in kooperativen Gemeinschaften programmiert und gepflegt. Eine direkte Konkurrenzsituation eines quantitativen Labels widerspricht den Werten dieser Communities.

Auf der anderen Seite gibt es jedoch auch gute Gründe für eine *quantitative* Kennzeichnung. Denn nur sie macht es möglich, bestehende Software-Produkte kontinuierlich zu verbessern und gegenüber sich selbst zu benchmarken. Dies erlaubt es, eine Aussage darüber zu treffen, ob der Energieverbrauch von Version zu Version zu- oder abnimmt. Dabei sollten folgende Punkte beachtet werden:

1. Als allgemein verständliche Mengenangabe eignet sich am besten der Energieverbrauch während der Ausführung der Tests auf der jeweiligen Testumgebung (`operational_electricity`). Diese Angabe beruht auf echten Energiemessungen und nicht auf Modellannahmen und berechneten Größen, wie der Herstellungsaufwand der Hardware, der Ressourcenverbrauch oder die Treibhausgasemissionen.
2. Es sollte in geeigneter Form dokumentiert werden, auf welcher Testumgebung die Messung ausgeführt wurde und welche Testläufe darin enthalten sind.
3. Bei jedem Versionsupdate sollte dokumentiert werden, wie sich der Energieverbrauch von einer Version zur nächsten verändert. Dabei muss sichergestellt werden, dass die Testumgebungen jeweils identisch sind, anderenfalls ist kein Vergleich zur vorigen Messung möglich.
4. Das Ergebnis soll von unabhängiger Seite überprüfbar oder mindestens nachvollziehbar sein.

Die Empfehlung ist daher, dass das Energielabel als Kommunikationsmittel dafür genutzt wird, um eine Aussage darüber zu treffen, dass:

1. die Software mit der SoftAWERE-Methode gemessen wurde,
2. sie bezüglich des Energie- und Ressourcenverbrauches optimiert wurde und
3. sie einen Energieverbrauch in einer bestimmten Höhe aufweist.

Da die Methode maßgeblich auf eine hohe Testabdeckung angewiesen ist und ein größerer Funktionsumfang zusammen mit mehr Tests einen größeren Energieverbrauch vermuten lässt, sollten die Information über die Testabdeckung neben dem Label gezeigt werden. Um glaubwürdig nachzuweisen, dass die Zahlen auf echten Messungen beruhen und möglicherweise eine Überprüfung zu ermöglichen, sollte eine eigene Datei zur Messung im Repository abgelegt werden. Diese Datei(en) enthält die Instruktionen für die Ausführung der Messung (siehe Anhang D.4 als Beispiel) und die Messergebnisse in einem maschinenlesbaren Format (z.B. JSON oder XML), siehe Kapitel 3.1.2 „Dokumentation der Messergebnisse mit Einheiten".

3.1.2 Das Label

Die Kennzeichnung kann z.B. auf GitHub, GitLab, Maven, NPM oder anderen öffentlichen Komponenten-Datenbanken eingebunden werden, ergänzend zu den typischen Kennzeichnungen wie in Abbildung 35.

Typische Kennzeichnungen in öffentlichen Datenbanken für Software-Komponenten umfassen unter anderem:

- Die aktuelle Version einer Software-Komponente und in welchem öffentlichen Register sie verfügbar ist (z.B. „npm“-Register, Version: 9.6.5)
- Die Lizenz unter der die Software-Komponente veröffentlicht wurde (z.B. Artistic 2.0, MIT oder Apache License)
- Der aktuelle Status der automatischen Test-Umgebung, laufen alle Tests ohne Fehler durch wird dies oft mit einer Kennzeichnung „passing“ oder „failed“ dargestellt.

Abbildung 35: Beispiele für Kennzeichnungen bei GitHub

Das Standard-Design in Github nutzt dafür den Badge-Generator[95], ein Dienst welcher aus Messungen wie Tests („Tests funktionieren“, „Tests funktionieren nicht“) Grafiken generiert wie in Abbildung 35 dargestellt. Dies ist ein Online-Dienst, der einfach eingebunden werden kann, um Kennzeichnungen im Design obiger Abbildungen zu erstellen. Dieses Projekt stellt das Projekt-Label[96] öffentlich zugänglich zur Verfügung, sodass das Label in die README.md Datei der Bibliothek eingebunden werden kann.

Über eine URL mit der Angabe des Energieverbrauches kann automatisch das Label erzeugt werden: https://badgen.net/badge/Soft%20 Energy/723%20Ws/26674c?icon=https://www.digitalcarbonfootprint.eu/softawere.svg

Mit diesem Dienst kann die Kennzeichnung sich selbst erzeugen, unter Angabe der Messergebnisse („723 Ws“) und dem Icon („softawere.svg“).

Somit kann sich jede Person das Label selbst generieren und auf die Energieeffizienz der eigenen Software aufmerksam machen. Zu überprüfen ist das Messergebnis ausschließlich über die Plausibilität der Dokumentation und ggf. durch Nachmessen auf einer äquivalenten Hardware.

Die vorgeschlagene Kennzeichnung („Badge“) ist in Abbildung 36 dargestellt. Durch das SoftAWERE-Logo links in der Abbildung 36 neben dem Schriftzug „Soft Energy“ wird signalisiert, dass die SoftAWERE-Messmethode angewendet wurde, die Angabe „Soft Energy“ beschreibt den gemessenen Energieverbrauch, der im rechten Feld mit 723 Ws angegeben ist.

Abbildung 36: Beispiel des Labels mit einem Energieverbrauch von 723 Ws

Für den Energieverbrauch sollen zwei bis vier Stellen gerundet ohne Nachkommastellen inklusive des SI-Präfixes[97] und der Einheit Wattsekunden des Energiebedarfs elektrischer Energie während der Testlaufzeit dargestellt werden.

Nachfolgend sind Beispiele aufgelistet, wie die Zahlen angezeigt werden sollen:

0,03456789	Ws	35	mWS

[95] https://badgen.net/

[96] https://www.digitalcarbonfootprint.eu/softawere.svg

[97] https://de.wikipedia.org/wiki/Vors%C3%A4tze_f%C3%BCr_Ma%C3%9Feinheiten

0,3456789	Ws	346	mWs
3,456789	Ws	3457	mWs
34,567	Ws	35	Ws
345,6789	Ws	345	Ws
3456,789	Ws	3457	Ws
34567,89	Ws	35	kWs
345678,9	Ws	346	kWs
3456789	Ws	3457	kWs
34567890	Ws	35	MWs

Das Messskript wird so erweitert, dass die unterliegende Hard- und Software als Messreport zusammen mit den detaillierten Messergebnissen ausgegeben wird, bspw. als JSON oder Markup-Datei, die in dem Repository zu hinterlegen ist. Das Label soll ausschließlich benutzt werden, wenn eine entsprechende Dokumentation ebenfalls vorhanden ist.

Dokumentation der Messmethode

- Verweis auf SoftAWERE-Methode mit Versionsnummer

Dokumentation der Messumgebung

- Hardwareausstattung
  - Hersteller und Version von Server/ Computer, CPU, GPU/Grafikkarte
  - Anzahl von CPUs und Prozessorkernen, Taktfrequenz
  - Größe des Arbeitsspeichers
- Softwarestack (Name, Version)
  - Betriebssystem, Virtualisierung
- Testablauf
  - Auflistung aller durchgeführten Einzeltests
    (z.B. Nennung der Skriptnamen im Ordner main/tests)

Dokumentation der Messergebnisse mit Einheiten

Nachfolgend sind die englischen Bezeichnungen der Messergebnisse aufgelistet.

- Energieverbrauch
  - operational_primary_energy-[J] ("Primärenergieverbrauch")
  - operational_electricity [Ws] ("operativer Stromverbrauch")
  - embedded_primary_energy [J] („eingebetter Stromverbrauch aus Herstellung")

- Treibhausgasemissionen
  - operational_emissions [kg CO2eq] (“operative THG-Emissionen”)
  - embedded_emissions [kg CO2eq] (“eingebettete THG-Emissionen aus der Herstellung”)
- Ressourcenverbrauch
  - operational_abiotic_resources_depletion [kg Sbeq] (“operative Verbrauch von adiabatischen Ressourcen”)
  - embedded_abiotic_resources_depletion [kg Sbeq] (“eingebetteter Verbrauch von adiabatischen Ressourcen aus der Herstellung”)
- Messzeitraum
  - start_time („Startzeitpunkt der Messung“)
  - end_time („Endzeitpunkt der Messung“)

3.2 Forschungs- und Standardisierungsbedarf

Durch die Arbeit am Vorhaben wurden weitere wichtige und potenzielle Forschungsfelder deutlich, die im Folgenden zusammengefasst wurden.

Ressourcenverbrauch und Umweltwirkung von Netzwerk-Verkehr in Messungen miteinbeziehen

Die Effekte von Netzwerkverkehr sollten im Kontext von Software-Anwendungen weiter erforscht werden. So können weitere Entscheidungshilfen für Software-Entwickelnde für die Programmierung geschaffen werden, z.B. um zu entscheiden ob sich Datenkompression lohnt (höherer CPU-Aufwand, jedoch reduzierter Netzwerkverkehr).

Das Forschungsteam hat sich mangels Mess-Schnittstellen und Verfügbarkeit von öffentlichen Daten im Rahmen von SoftAWERE entschieden, die Netzwerk-Analyse nicht miteinzubeziehen. Hier empfiehlt sich weitere Forschung insbesondere hinsichtlich der folgenden Datenpunkte:

- Stromverbrauch und Herstellungsaufwand der Netzwerkschnittstelle („NIC“), die in Servern verbaut ist.
- Stromverbrauch und Herstellungsaufwand von „Top of the rack“ Netzwerk Equipment, z.B. Netzwerk-Switches die den Datenaustausch von Servern im selben Rack ermöglichen
- Stromverbrauch, Herstellungsaufwand und Zurechnung in „Leaf/Spine“ Netzwerken, die für die Vernetzung von Rechenzentren genutzt werden und in manchen Fällen von verschiedenen Software-Anwendungen geteilt werden.
- Stromverbrauch, Herstellungsaufwand und Zurechnung von Internet-Verkehr, z.B. über öffentliche Datensätze zur Umweltwirkung und Ressourcenverbrauch pro Datentransfer-Einheit (GB) an einem Internet-Exchange und Point-of-Presence (POP). So könnte der Internet-Verkehr einer Software-Anwendung je nach Route (z.B. über DE-CIX und 3 PoPs in Frankfurt, München und Berlin) berechnet werden.

Weiterentwicklung der Transparenzkennzeichnung zu einer Energieeffizienz-Kennzeichnung für Software-Anwendungen

Eine Kennzeichnung der Energieeffizienz, die auf der SoftAWERE-Methode basiert, ist mindestens den folgenden Einschränkungen unterworfen:

- Erstens kann nicht beantwortet werden, inwiefern die Ergebnisse dieser Methode mit denen anderer Methoden korrelieren, z.B. einer Messung in der realen Produktivumgebung der Software oder unter der Last eines Standardnutzungsszenarios. Es besteht weiterer Forschungsbedarf, um den Energieverbrauch von Software in unterschiedlichen Umgebungen zu quantifizieren und möglicherweise umzurechnen.
- Zweitens ist der Energiebedarf von Software stark abhängig von der Hardware, auf der sie ausgeführt wird, was die Reproduzierbarkeit einer quantitativen Messung erschwert. Die Ergebnisse sind derzeit nur dann reproduzierbar, wenn die Messungen auf der gleichen Testumgebung stattfinden. Die Kennzeichnung bedarf daher einer sehr detaillierten Dokumentation der unterliegenden Hardware.
- Drittens hat das Forschungsvorhaben den Fokus auf die Vergleichbarkeit von Open Source Bibliotheken gegenüber sich selbst, bzw. seinen vorherigen Versionen gelegt. Zielgruppe der SoftAWERE-Methode sind die Entwickler*innen von Software, die im kontinuierlichen Entwicklungsprozess eine Steigerung der Energie- und Ressourceneffizienz vorhandener Software realisieren möchten und weniger die Konsument*innen. Für die zweite Zielgruppe wäre eine Kennzeichnung von Interesse, welche eine quantitative Aussage über die Energieeffizienz unterschiedlicher Software-Produkte (mit der gleichen Funktionalität) macht, um besonders effiziente Software hervorzuheben. Dies ist mit dieser SoftAWERE-Methode nicht zulässig, da die Ausführung von Unit- und Integrationstests nur wenig mit der späteren Nutzung zu tun hat. Um diesen Vergleich zu ermöglichen, wäre die Ausführung eines Standardnutzungsszenarios erforderlich, wie es der Blaue Engel für Software vorsieht. Die hier zu entwickelnde Kennzeichnung kann daher ein Umweltkennzeichen für Konsument*innen, wie den Blauen Engel, nicht ersetzen.

Die SoftAWERE-Methoden lassen sich jedoch sowohl in den Blauen Engel für Software einbringen und durch weitere Forschung und Standardisierung zu einer Kennzeichnung entwickeln, die eine vertrauenswürde Aussage zur Umweltwirkung einer Software-Komponenten und Open-Source-Software trifft. Mit vertrauenswürdiger Mess-Methode und entsprechender Transparenz, kann damit eine konkrete, potentiell internationale Kennzeichnung als Entscheidungshilfe für Software-Entwickelende geschaffen werden.

Klassifizierung von Funktionen in digitalen Produkten und Software

Nachdem erste Messmethoden zur Erfassung von Ressourcenverbräuchen erprobt sind, sollte in den nächsten Schritten eine Klassifizierung der Hauptfunktion (vgl. Kühlschrank) für Software (sowohl digitale Produkte als auch Bibliotheken) erforscht werden, um eine Vergleichbarkeit zu gewährleisten.

Hier sollte die Open-Source Community und die Industrie selbst aktiv werden, um Definitionen analog zu den Produktkategorieregeln von Produkten zu finden. Durch Forschung können Kategorien definiert und entsprechende Rahmen für eine Normung geschaffen werden.

Zusätzlich sollte erforscht werden, wie digitale Produkte, die mehrere Hauptfunktionen haben können, Ihre Ressourcenaufwände und Umweltwirkung pro Funktion isoliert ermitteln können.

Vergleichbarkeit herstellen über Standardnutzungsszenarien für Hauptfunktionen einer Software oder einem digitalen Produkt

Es ist denkbar, für jede definierte Hauptfunktion (z. B. die Bearbeitung eines Textdokuments) ein Standardnutzungsszenario festzulegen, welches jede Software mit der Funktion anwenden kann, um eine vergleichbare Messung von Umweltwirkung und Ressourcenverbräuchen pro „Funktionsausführung“ zu berechnen.

Mit einer öffentlich zugänglichen Datenbank aus Funktionsdefinitionen und dazugehörigen Nutzungsszenarien sind übergreifende Benchmarks denkbar. Auch hier ist der Digitalsektor bzw. die Open Source Gemeinschaft gefragt, um eine solche Datenbank zu realisieren. Mögliche fehlende Benchmarking-Ansätze können durch die Forschung entwickelt werden.

Benchmarking von Software-Funktionen hinsichtlich Energie- und Ressourceneffizienz durchführen

Mit einer Datenbank von Hauptfunktionen von Software-Anwendungen (z.B. „Login Funktion mit E-Mail“, „PDF Export Funktion“), lassen sich Vergleiche durchführen („Benchmarking“). So kann der digitale Ressourcenverbrauch und Umweltwirkung pro Funktion gemessen werden und mit den Messungen einer anderen Anwendung direkt gegenübergestellt werden. So kann ein Qualitätsvergleich durchgeführt werden, was zu mehr Wettbewerb zwischen Software-Herstellern führen kann.

Werkzeuge für Informationsflüsse über den Lebenszyklus entwickeln

In den bestehenden Werkzeugen und Methoden gibt es weiterhin Lücken sowohl bei der Erfassung der digitalen Ressourcen in der Infrastruktur – ein Beispiel dafür ist Netzwerkverkehr – als auch im Lebenszyklus. Es sollte erforscht werden, wie die digitalen Ressourcenverbräuche während der Entwicklung/Herstellung einer Software erfasst werden können oder wie Verbräuche bei der Nutzung (z. B. auf dem entsprechenden Endgerät) entstehen. Zudem sollten Skalierungseffekte genauer betrachtet werden, z.B. die Menge der Nutzer, Aufrufe und bereitgestellte bzw. verarbeitete Daten. Mit der zunehmenden Größe von Software-Anwendungen, insbesondere im Kontext der Digitalwirtschaft sind diese Skalierungseffekte von zunehmender Relevanz.

Durch diese Erweiterung lässt sich die Transparenz zur Umweltwirkung und den Ressourcenverbräuchen von Software über den gesamten Lebenszyklus weiter präzisieren.

Normungsvorschlag für die Methodik zur Wandlung/Messung von digitalen Ressourcen in Kennzahlen zur Umweltwirkung und Ressourcenverbrauch

Auf Basis, von der im SoftAWERE Vorhaben entwickelten Methodik, welche es ermöglicht, den digitalen Ressourcenverbrauch von Software in den dafür notwendigen Ressourcenaufwand und Umweltwirkungen aus dem Betrieb und der Herstellung von IT-Hardware zu messen, lässt sich ein erster Normungsvorschlag entwickeln, der entweder mit Schätzungen (mathematisch arbeitet) oder eine vorgegebene Messmethodik auf dem Server-bzw. Client-System definiert.

Die SDIA verfolgt bereits aufbauend auf den Ergebnissen des Vorhabens die Integration der Methodik in die produktiven Umgebungen für Software-Anwendungen. Ein Beispiel für diese Integration ist die Einbettung in Container von Orchestrierungs- und Virtualisierungsplattformen, die für Entwickler*innen und den Betrieb verantwortliche handlungsfähige Indikatoren bereitstellt.

Wirkung von Architektur und Software-Patterns auf die Umweltwirkung und Ressourcenverbräuche erforschen

Mit dem realisierten SoftAWERE Test-Labor lassen sich verschiedene Vorgehensweisen, Architekturen und Standard-Patterns der Softwareentwicklung untersuchen und Empfehlungen hinsichtlich der Energie- und Ressourceneffizienz zur Verfügung stellen. Dies ist insbesondere für die Entwicklung eines Green-Coding-Curriculums relevant, denn die Empfehlungen und Lernmaterialien lassen sich mit SoftAWERE validieren.

Patterns und Architekturen zu katalogisieren und die Tests durchzuführen und zu analysieren und Verbesserungsmöglichkeiten abzuleiten, erfordert weitere Forschung. Die Ergebnisse

daraus sollten aus Sicht des Forschungsteams in einen Lernplan für Informatik übertragen werden („Green Coding Curriculum“).

3.3 Verbrauchsbezogene Maßnahmen

Aufmerksamkeit schaffen

Das Thema der Umweltwirkung von digitalen Produkten, von Software, der Digitalisierung und der Digitalwirtschaft als neuen Sektor, ist politisch auf der Agenda, erfordert aber weiterer Aufmerksamkeit.

Kommunikationsmaßnahmen sollten sich sowohl an die Software-Wertschöpfungskette (Entwickler, Architekten, Betriebsverantwortliche …) richten, als auch an Gesellschaft und Industrie, um das Verständnis für die ganzheitliche Umweltwirkung weiter zu fördern und zu sensibilisieren.

Des Weiteren sollten klare Fragestellungen eingesetzt werden, um ein politisches Momentum zum Thema Transparenz zur Umweltwirkung von Digitalisierung und Digitalwirtschaft aufzubauen, z. B. „Wie viel Umweltwirkung und Ressourcenverbrauch steckt in einer Minute Video-Streaming?“; „Wie viel Energie und Ressourcen verbraucht eine ChatGPT-Anfrage?“

Einheitliche Regulierung hinsichtlich Transparenz

Der Digitalsektor sollte in breitem Maße aufgefordert werden, Transparenz zur Umweltwirkung von digitalen Produkten, sowohl für Endverbraucher als auch für Industrie & Wirtschaft zu schaffen und Kennzahlen bereitzustellen.

Im ersten Schritt sollte das Ziel sein, Transparenz zu schaffen, analog dazu, dass der Energieverbrauch eines Kühlschranks oder eines Elektroautos klar erkennbar ist, und der Kunde somit die notwendigen Informationen für eine (nachhaltige) Kaufentscheidung hat.

Diese Transparenz und öffentliche Kennzahlen der digitalen Produkte bilden zudem eine Grundlage für weitere Forschung, z. B., um sinnvolle Grenzwerte zu identifizieren.

Strategie für nachhaltige Digitalisierung und Digitalwirtschaft etablieren

Die Regierung sollte eine Strategie entwickeln, um den Ressourcenaufwand der Digitalwirtschaft und der Digitalisierung gezielt zu steuern und einen Plan für einen nachhaltigen Wirtschaftssektor zu definieren.

Die Kernpfeiler der Strategie sind:

1. Transparenz von digitalen Produkten – für Monitoring, Forschung und Kundenentscheidungen
2. Transparenz der Infrastruktur – Hersteller von Equipment und Infrastrukturanbieter müssen aktiv aufgefordert werden, transparente Informationen zum Ressourceneinsatz und Umweltwirkungen an Kunden weiterzugeben
3. Aufmerksamkeit – Gesellschaft, Digitalsektor und Gesamtwirtschaft für Ressourcenaufwände sensibilisieren
4. Grenzwerte – Ressourcenaufwände begrenzen und Wettbewerb hinsichtlich „mehr Funktionalität mit gleichem Ressourceneinsatz“ ermöglichen
5. Internationale Zusammenarbeit – der Digitalsektor ist global, und Regulatorik bzgl. Transparenz und Grenzwerte sollten in der globalen Gemeinschaft beworben werden

Verfügbarkeit der Informationen zur Umweltwirkung und Ressourcenverbräuchen aus der digitalen Infrastruktur sicherstellen

Digitale Produkte benötigen digitale Ressourcen, die von digitaler Infrastruktur erzeugt werden. Die Anbieter von digitalen Ressourcen (z. B. Hosting Unternehmen, IT-Infrastruktur Anbieter, Cloud Infrastruktur Anbieter etc.) müssen aufgefordert oder incentiviert werden, Ihren Kunden Informationen zu Ressourcenverbräuchen und Umweltwirkung bereitzustellen.

Diese müssen auf Basis von standardisierten Kennzahlen z. B. aus bestehenden Indikatoren aus Standards für Ökobilanzen berechnet werden, um eine Vergleichbarkeit, und dadurch Wettbewerb, von Umweltwirkung der verkauften digitalen Ressourcen zu gewährleisten.

Liefern alle Anbieter von digitaler Infrastruktur gleichermaßen Informationen zur Umweltwirkung der bereitgestellten digitalen Ressourcen, wird die Transparenz von Herstellern digitaler Produkte vereinfacht, die auf die digitalen Ressourcen als Rohstoff angewiesen sind.

3.4 Empfehlungen für die Software-Entwicklungsgemeinschaft

Aktive Zusammenarbeit bei der Entwicklung von Messmethodik und Normung

Eine Mitwirkung des Digitalsektors an der Entwicklung von Messmethoden ist sinnvoll, um eine Anwendbarkeit zu gewährleisten. Eine aktive Zusammenarbeit mit Forschung und Gesetzgebung reduziert das Risiko von Überregulierung und von technischen Vorgehensmodellen, die in der Praxis nicht umsetzbar sind.

Kennzeichnung mit Fokus auf Messung & Optimierung von Energie- und Ressourcennutzung etablieren

Da eine Kennzeichnung, die eine Vergleichbarkeit von Open-Source Bibliotheken ermöglicht, ohne eine Abgrenzung der Hauptfunktion im Moment nicht möglich ist, sollte sich eine Kennzeichnung auf das Vorhandensein von Messungen im Entwicklungsprozess und vorgenommenen Optimierung beschränken.

Die Botschaft einer Kennzeichnung:

- „Das Projekt misst im Entwicklungsprozess Energie- und Ressourcenverbräuche und weisen die etwaige Veränderung (positiv oder negativ) bei jeder neuen Version aus.“
- „Das Projekt optimiert den Energie- und Ressourcenverbrauch der Bibliothek kontinuierlich auf Basis der Messungen.“

Fokus auf Veränderung von Energie- und Ressourcenverbrauch pro Version berechnen

Für jede neue Version einer (open-source) Bibliothek oder einer Software sollte die Differenz der Ressourcenverbräuche ausgewiesen werden. Dies sollte bei gleichbleibender Testabdeckung (prozentual) geschehen.

Werden also neue Funktionen hinzugefügt, sollten auch entsprechende Integrationstests hinzugefügt werden, sodass die Testabdeckung z. B. bei 90 % konstant bleibt. Ziel sollte es sein, dass eine neue Funktion nicht zu einem erhöhten Ressourcenaufwand führt.

Bau- und Integrationsprozesse auf unnötige Energie- und Ressourcennutzung überprüfen und optimieren

Automatisierte Prozesse zum Bau einer Software oder zur Ausführung der Tests sollten hinsichtlich ihrer Häufigkeit und Ausführungskriterien optimiert werden. Eine hohe Wiederholung, z. B. bei jeder Veränderung einer nicht funktionsrelevanten Datei wie einem

README-Dokument, sorgt für unnötige Energie- und Ressourcenverbräuche durch verbrauchte digitale Ressourcen (und damit Server Ressourcen, Energie etc.)

Auch hier können Messwerkzeuge helfen, den Ressourcenaufwand pro Integrations- oder Bauprozess zu ermitteln, und entsprechende Optimierung durchzuführen.

Transparenz für die Auswahl von Bibliotheken für Entwickler schaffen

Langfristiges Ziel sollte es sein, Entwicklern und Architekten bei der Auswahl von Bibliotheken oder Komponenten eine Entscheidungshilfe zu bieten, um die Ressourcenaufwände und Umweltwirkung in den Auswahlprozess einzubeziehen.

Um eine entsprechende Vergleichbarkeit zu schaffen, ist die Benennung und Abgrenzung einer Hauptfunktion notwendig (siehe 5.2 Forschungsbedarf)

3.5 Aus- und Weiterbildung

Verantwortungsgefühl an Informatiker*innen vermitteln

Verantwortung für den Ressourcenaufwand und daraus entstehende Umweltwirkung sollte allen Lernenden in der Informatik vermittelt werden. Jede neue Software-Anwendung, deren Architektur und Funktionalität hat eine reale Auswirkung – egal ob mehr Server-Ressourcen benötigt werden, der Anwender ein schnelleres Endgerät kaufen muss, Netzwerkverkehr entsteht oder mehr Energieverbrauch entsteht.

Eine Sensibilisierung zur realen und ganzheitlichen Wirkung von Software sollte in Lehrplänen verankert werden.

Praktische Beispiele für Messung von Umweltwirkung und Ressourcenverbräuche anwenden und vermitteln

Werkzeuge, die zur Messung von Umweltwirkung und Ressourcenverbrauch eingesetzt werden können, sollten praktisch eingesetzt und verschiedene Messmethoden und Vorgehensweisen angewandt werden.

Modelle der Wirkmechanismen, Architekturen und Optimierungspfade sollten vermittelt werden und in konkreten Beispielen an die Lernenden weitergeben werden.

Ganzheitliches Denken vermitteln und Zusammenhänge zwischen Informatik und realer Umweltwirkung & Auswirkung herstellen

Modelle zum ganzheitlichen Denken oder eine systematische Betrachtung sollten an Lernende und praktizierende Informatiker vermittelt werden. Dies sollte im Zusammenspiel mit grundsätzlichen politischen Kommunikationsstrategien umgesetzt werden, um eine Sensibilisierung auch in der Wirtschaft sicherzustellen.

4 Anwendung der Ergebnisse

Das SoftAWERE Vorhaben wurde durchgeführt, um für eine breite Schnittmenge von Anwendungen und Entwicklungsprozessen Energie- und Umweltwirkungsmessungen zu ermöglichen. Im Folgenden werden einige Anwendungsbeispiele für SoftAWERE vorgestellt. Einige davon wurden im Rahmen der Veranstaltungen, die im Rahmen des Vorhabens durchgeführt wurden, erprobt oder basieren auf bestehenden, im Markt üblichen, Software-Anwendungsprozessen.

Information zu den Referenz-Projekten auf GitLab

Im Folgenden werden verschiedene Anwendungsbeispiele für die Ergebnisse des SoftAWERE Vorhabens dargestellt. Für viele der Beispiele wurden im Rahmen des Vorhabens, Code-Beispiele erarbeitet. Diese Code-Beispiele wurden auf GitLab angelegt und werden mit dem Versionskontrollsystem „Git" verwaltet. Für Anwendungsbeispiele bei denen Beispiel-Programmcode vorliegt, wurde in den folgenden Unterkapiteln ein entsprechender Link in der Fußnote notiert.

Auch ohne Git-Versionskontrolle bzw. eine Entwicklungsumgebung auf dem eigenen Computer, lassen sich diese Beispiele in einem Web-Browser betrachten – der Programmcode kann durchgesehen werden und im Bereich "CI/CD" kann für jedes Beispiel die Messung mit SoftAWERE betrachtet werden (siehe Anhang D.2 für ein Beispielaufbau).

Insgesamt lässt sich die Anwendbarkeit der beiden SoftAWERE Methoden (siehe Kapitel 2.2) wie folgt zusammenfassen:

- SoftAWERE-Light: Mathematische Herangehensweise: Bestimmung von Annäherungswerten insbesondere im Hinblick auf Energieverbrauch in produktiven IT-Betriebsumgebungen
- SoftAWERE: Labor bzw. Integration in CI/CD Prozesse: Präzisere Messung vom Bau bzw. Kompilierung von Anwendungen oder die Durchführung spezifischer Prozesse (KI-Training, Batch-Prozesse)

Die entwickelten Methoden eignen sich nicht für Desktop-Anwendungen (es sei denn sie werden in einer CI/CD Umgebung kompiliert bzw. gebaut), Embedded Systeme, Anwendungen für Mobilgeräte und die Erfassung von Umweltwirkung von Netzwerkübertragung (unter anderem Streaming-Anwendungen oder die Auslieferung von Inhalten über CDN).

4.1 Umweltwirkung von Webseiten ermitteln

Moderne Webseiten werden über zwei gängige Mechanismen realisiert:

1. Traditionelle Webseiten mit Content Management Systemen (CMS) wie Drupal, Joomla, Typo3, Wordpress, oder kommerziellen Systemen. Diese erfordern die laufende Bereitstellung einer Server Infrastruktur auf dem das CMS läuft und auf Nachfrage eines Nutzers, z.B. durch Besuch der Webseite, die Inhalte aufbaut und ausliefert.
2. Moderne Webseiten, die statisch mit Werkzeugen wie Gatsby, NextJS, Hugo, und anderen entwickelt wurden. Hier wird die gesamte Webseite mitsamt den Inhalten in statische Textdateien gewandelt (HTML, CSS und JavaScript), welche direkt über Content-Delivery-Netzwerk (CDN) ausgeliefert werden können und dadurch wesentlich weniger Server-Infrastruktur benötigen. Für die Auslieferung der Webseiten wird nahezu keine serverseitige Berechnung benötigt, lediglich Festplattenspeicher und Netzwerkkapazität sind erforderlich (vergleichbar mit der Auslieferung von Bild- und Video Inhalten).

4.1.1 Messung von CMS-basierten Webseiten

Bei der Messung der Umweltwirkung von traditionellen Webseiten muss unterschieden werden, auf welcher Art von Server das CMS läuft. SoftAWERE ist nicht für diesen Anwendungsfall gedacht, jedoch lassen sich die methodischen Herangehensweisen übertragen. Für diese Art der Anwendung liegt der Fokus auf die Erweiterung der Monitoringfähigkeiten des IT-Betriebs um den Aspekt des Energieverbrauchs und der Umweltwirkung mit Hilfe von SoftAWERE-Light.

Läuft das CMS auf einem physischen Server, auf dem die verantwortlichen Administrator*innen Systemzugriff haben („root"-Berechtigung), so kann der Energieverbrauch direkt mit IPMI[98] oder Scaphandre[99] gemessen werden. Der Ressourcenverbrauch des Servers kann über die Boavizta API[100] ermittelt werden. Die Nutzung dieser Tools kann aus dem Messaufbau von SoftAWERE abgeleitet werden (Anhang D.2).

Gängiger ist der Betrieb von CMS-Systemen auf virtualisierten Systemen, wie OpenStack, VMWare oder gängigen Cloud-Infrastruktur-Umgebungen. In diesen Umgebungen lässt sich oft nur die CPU-Auslastung, Arbeitsspeicherverbrauch und Netzwerkverkehr beobachten. Hier kann SoftAWERE-Light eingesetzt werden (siehe Kapitel 2.2). Mit SoftAWERE-Light lässt sich der digitale Ressourcenverbrauch in Energieverbrauch umrechnen. Dafür müssen lediglich die Formeln im IT-Monitoring System hinterlegt werden, wie z.B. in Grafana. Bekannt sein muss zudem dabei der CPU-Typ, sodass der TDP-Wert ermittelt werden kann[101].

4.1.2 Messung von statisch generierten Webseiten

Im Rahmen des zweiten Hackathons wurde als Beispielprojekt die Webseite der SDIA[102] untersucht und mit SoftAWERE vermessen. Die SDIA-Webseite gehört zur zweiten Kategorie. Es ist eine statisch generierte Webseite. Das bedeutet, dass mit SoftAWERE der gesamte Aufbau der Webseite gemessen werden kann. Dazu gehören die Generierung aller Seiten, die Vorkomprimierung von Bildern und die Kompression & Optimierung von JavaScript und CSS-Dateien.

SoftAWERE ist ideal dafür geeignet, für jede Version einer statisch generierten Webseite die Umweltwirkung der Erstellung („Kompilierung") zu ermitteln und jeder Version einen Fußabdruck bzw. eine Deklaration der Umweltwirkung zu geben. Hierbei ist hervorzuheben, dass die Auslieferung der Webseite über ein CDN nicht erfasst wird und eine andere Methode erfordert, da es sich dabei primär um die Messung von Netzwerkverkehr handelt.

Für die Erfassung wird der Bau-Prozess der Webseite in einem SoftAWERE CI/CD Prozess konfiguriert[103], was für statische Webseiten üblich ist. Während der Bau-Prozess durchgeführt wird, wird der Energieverbrauch des Servers aufgezeichnet. Die eingebetteten Emissionen und

[98] Als Beispiel kann der Prometheus IPMI Exporter direkt genutzt werden, wenn das System mit Prometheus als Monitoring-Dienst überwacht wird: https://github.com/prometheus-community/ipmi_exporter

[99] Mit Scaphandre lässt sich die Intel RAPL Schnittstelle über einen Prometheus Exporter auslesen: https://github.com/hubblo-org/scaphandre

[100] Mit der Boavizta API lässt sich der Ressourcenverbrauch für den Server bestimmen, entweder direkt über den Modellnamen (https://doc.api.boavizta.org/getting_started/single_server/) oder über die Konfiguration des Servers (https://doc.api.boavizta.org/getting_started/single_server/#retrieve-the-impacts-of-a-custom-server)

[101] Als Beispiel findet man bei Intel eine Liste der aktuellen Intel Xeon Platinum Prozessoren mit den entsprechenden TDP Angaben: https://www.intel.com/content/www/us/en/products/details/processors/xeon/scalable/platinum/products.html

[102] Der Link zum Gitlab-Repository: https://gitlab.com/softawere-hackathon/hackathon/sdia-website2

[103] Die Konfigurationsdatei für Gitlab findet sich im Gitlab-Repository: https://gitlab.com/softawere-hackathon/hackathon/sdia-website2/-/blob/main/.gitlab-ci.yml?ref_type=heads

der Ressourcenverbrauch des Servers werden proportional zugewiesen, je nachdem wie viel Kapazität des Servers der Bau-Prozess beansprucht hat und für wie lange[104].

Für jede Änderung auf der Webseite wird eine neue statische Version erstellt. Durch SoftAWERE kann für jede Version eine Umweltbilanz als JSON- hinterlegt werden. Aus diesen Daten kann zum Beispiel eine visuelle Darstellung auf der Webseite realisiert werden und die Umweltwirkung transparent gemacht werden.

4.2 Messung von Anwendungen auf Cloud-Infrastruktur

Wie auch bei den CMS-basierten Webseiten im vorherigen Abschnitt kann bei der Messung auf Cloud-Infrastrukturen auf SoftAWERE-Light zurückgegriffen werden.

Dabei werden die IT-Monitoringsysteme, welche bereits die digitalen Ressourcenverbräuche erfassen (CPU Zeit, Arbeitsspeicher, Netzwerkverkehr und Speicherplatz) mit der SoftAWERE-Light-Methodik erweitert werden. So lässt sich eine Schätzung des Energieverbrauchs realisieren.

Für die gängigen Cloud-Infrastruktur-Instanzen, wie z.B. bei Amazon AWS oder OVH ist der CPU Typ bekannt[105], der TDP-Wert lässt sich recherchieren und ist für die SoftAWERE Formeln erforderlich.

Ein Beispiel für die kommerzielle Implementierung von SoftAWERE-Light Formeln in einem IT-Monitoring-System findet sich in der Dynatrace IT-Monitoring-Software[106]. Diese ermöglicht die Erfassung von Energieverbräuchen und THG-Emissionen in Cloud-Umgebungen wie Microsoft Azure, Amazon AWS und Google Cloud.

4.3 Kompilierte Anwendungen & Batch Prozesse

SoftAWERE eignet sich besonders für die Ausführung eines in sich geschlossenen Prozesses, wie z.B. die Kompilierung einer Java-Anwendung, oder die Ausführung eines Batch-Prozesses mit einem Python-Skript.

Diese Prozesse können in CI-Konfigurationsdateien ausgedrückt werden. Schritte werden nacheinander ausgeführt (z.B. Dateien kopiert, dann ausgelesen und dann verarbeitet) und alle relevanten Metriken zur Umweltwirkung können während der Ausführung aufgezeichnet werden.

Solche Prozesse können in offenen Labor-Versionen von SoftAWERE einfach ausprobiert werden. Zugang zum Gitlab-Team[107] kann vom Forschungsteam über die GitLab Team-Seite angefragt werden.

Ein Beispiel für einen Batch-Prozess ist die Ausführung von Fibonacci-Berechnungen in verschiedenen Programmiersprachen. Das Ausführungsskript befindet sich hier[108]. Zu beachten

[104] Für das gesamte Vorhaben und die SoftAWERE Werkzeuge wurde eine Server-Lebenszeit von 5 Jahren fest definiert. Dieser Wert kann in den SoftAWERE Messskripten manuell angepasst werden.

[105] siehe z.B. die Liste der Amazon Web Services EC2-Instanzen mit CPU Typen hier: https://aws.amazon.com/de/ec2/instance-types/

[106] Ein Link zur Darstellung des Dynatrace Carbon Impact Tools findet sich hier: https://www.dynatrace.com/news/blog/dynatrace-carbon-impact-app/, abgerufen am 30.11.2023

[107] oder einfach über das Gitlab-Team auf den Button "Request Access" klicken: https://gitlab.com/softawere-hackathon

[108] Die Ausführung von Fibonacci Tests in verschiedenen Programmiersprachen: https://gitlab.opencode.de/OC00004041236/ /softawere/fibonacci-benchmark, abgerufen am 30.11.2023

ist dass für jede Programmiersprache ein Branch angelegt wurde, in dem ein Skript hinterlegt ist (".gitlab-ci.yml") welches die Ausführung steuert.

Durch den Einsatz von SoftAWERE im CI/CD-Prozess lässt sich für diese Batch- bzw. Bauprozesse eine Umweltbilanz generieren und entweder technisch lesbar als JSON-Datei hinterlegen oder für die Generierung von eigenen Darstellungen oder Kennzeichnungen verwenden.

4.4 Open-Source Komponenten

Im Rahmen des Vorhabens wurden insbesondere Open-Source Komponenten betrachtet und vermessen. Für die Messung von Komponenten werden die integrierten Tests als Proxy einer Nutzung verwendet. Ansonsten müsste für jede Komponente eine eigene Test-Anwendung geschrieben werden, welche die Funktionen der Komponente implementiert und ausführt.

Ein einfaches Beispiel für einen Testaufbau, in dem die integrierten Tests einer Open-Source-Komponente vorbereitet und ausgeführt werden, ist das Web-Framework von NodeJS "Express"[109]. Insgesamt werden hier in ca. 1 Minute, 1257 Funktionstests ausgeführt.

Wird einer Open-Source Komponente eine Funktion hinzugefügt oder eine Änderung an bestehenden Funktionen eingebaut, so lässt sich über SoftAWERE im CI/CD-Prozess ermitteln, ob dadurch eine höhere oder niedrigere Umweltwirkung entstanden ist.

4.5 KI-Modelle – Training & Inferenz

Moderne KI-Anwendungsentwicklung wird üblicherweise in zwei Arbeitsblöcke aufgeteilt. Im ersten Teil werden mit Daten neue KI-Modelle trainiert bzw. bestehende Modelle weiter verbessert und kalibriert. Im zweiten Schritt wird das Modell auf IT-Infrastruktur aufgespielt und als Schnittstelle (API) bereitgestellt, von der Entscheidungen abgeleitet werden können ("Inferenz"). In beiden Schritten kommen GPUs zum Einsatz, wobei für das Training üblicherweise GPUs mit mehr Leistung als in der Inferenz zum Einsatz kommen.

SoftAWERE eignet sich hervorragend, um insbesondere in der Trainingsphase, welche oft in CI/CD-Prozessen ausgeführt werden, die Energieverbräuche und Umweltwirkungen zu erfassen. Mit SoftAWERE lässt sich eine Gesamt-Umweltbilanz für das Training eines Modells generieren und z.B. als JSON-Datei hinterlegen. Dafür muss lediglich ein GPU-Exporter[110] hinzugefügt und SoftAWERE um diese Metrik erweitert und in die Erfassung der Energieverbräuche mit einbezogen werden.

[109] Der Testaufbau befindet sich hier: https://gitlab.com/softawere-hackathon/test-express/-/blob/main/.gitlab-ci.yml?ref_type=heads, die Webseite von Express befindet sich hier: https://expressjs.com, beide abgerufen am 30.11.2023

[110] Der GPU Exporte für Nvidia Karten befindet sich hier: https://github.com/utkuozdemir/nvidia_gpu_exporter

5 Kommunikation der Forschungsergebnisse

Das Vorhaben hat eine enge Einbindung der Software-Gemeinschaft und eine offene Kommunikation der (Zwischen-)Ergebnisse verfolgt.

Green Coding Summit in Berlin

Im Rahmen des Vorhabens wurde der Green Coding Summit im November 2023 in Berlin organisiert. Der Summit brachte 200 Teilnehmende an zwei historischen Orten, dem Französischen Dom und der Malzfabrik, zusammen. Die ausverkaufte Veranstaltung gliederte sich in zwei Tage: Der erste Tag war geprägt von Vorträgen und dem Austausch innerhalb der Community, während der zweite Tag Workshops zur Wissensvermittlung im Bereich nachhaltiger Softwareentwicklung bot. Diese strukturierte Aufteilung förderte den Dialog und die praktische Auseinandersetzung mit den Herausforderungen und Lösungsansätzen im Bereich der grünen Software.

Die Vorträge des ersten Tages des Summits wurden auf YouTube[111] veröffentlicht, wodurch ein breiteres Publikum erreicht und die Diskussion über nachhaltige Praktiken in der Softwareentwicklung über die unmittelbaren Veranstaltungstage hinaus ausgedehnt wurde. Die Veranstaltung unterstrich die Notwendigkeit, den Energieverbrauch und die Umweltwirkung von Software systematisch zu adressieren, und leistete einen wichtigen Beitrag zur Förderung von Nachhaltigkeit innerhalb der Software-Gemeinschaft.

Diese Initiative spiegelt das Engagement der SDIA wider, nicht nur die Diskussion über nachhaltige Digitalisierung anzustoßen, sondern auch konkrete Lösungswege wie SoftAWERE und Best Practices zu verbreiten. Der Erfolg des Summits, sowohl in Bezug auf die Teilnehmendenzahl als auch auf die Qualität der Beiträge war ein erfolgreicher Abschluss des Vorhabens.

Hybride Zwischenbilanz in Berlin

Um den Zwischenstand des Vorhabens zu kommunizieren und Feedback von der Software-Gemeinschaft zu bekommen, wurde am 23. September 2022 in Berlin eine hybride Veranstaltung durchgeführt. Mit 70 Teilnehmern, sowohl vor Ort als auch per Video zugeschaltet, war die Veranstaltung ein Erfolg. Nicht nur wurde über das Vorhaben berichtet, sondern es wurde auch der Gemeinschaft rund um Werkzeuge und Messmethoden durch Vorträge eine Plattform gegeben, um die Zusammenarbeit zu fördern. In dedizierten Workshops haben die Teilnehmenden zudem lernen und diskutieren können, sowohl über die Messmethoden, das Test-Labor als auch zu konkreten Maßnahmen für IT-Verantwortliche und Betreiber von Hosting-Angeboten und Rechenzentren.

Alle Vorträge wurden aufgezeichnet und im Anschluss, zusammen mit den Präsentationsunterlagen, sowohl im Knowledge Hub der SDIA als auch auf sozialen Medien verbreitet[112], um die Wirkung weiter zu erhöhen und die Lerninhalte einer breiten Öffentlichkeit zur Verfügung zu stellen. Der Knowledge Hub der SDIA ist eine offene Wissensplattform („Wiki“) mit Inhalten, die unter einer Creative Commons Lizenz der Öffentlichkeit und Digitalwirtschaft zur Verfügung gestellt wird.

[111] siehe https://www.youtube.com/playlist?list=PLOL05v40mafYuYgMzz1cxE5Z8Xv0_MFOm

[112] Die Präsentation und Aufzeichnung der OOP Konferenz: https://knowledge.sdialliance.org/softawere/oop-conference-presentation, abgerufen am 13.10.2023; die Präsentation der Continous Lifecylce Konferenz ist hier: https://knowledge.sdialliance.org/public-speaking/2022-11-17-continuous-lifecycle-conference-(de), abgerufen am 13.10.2023; die Präsentation der Agile Testing Days ist hier: https://knowledge.sdialliance.org/public-speaking/2022-11-23-agile-testing-days, abgerufen am 13.10.2023

Im Anschluss der Veranstaltung wurde Teilnehmern die Möglichkeit gegeben, etwaige Verbesserungen in eigenen Open-Source-Projekten via E-Mail an das Vorhaben-Team zu übermitteln und einer Jury zu übergeben, die bis zum Januar 2023 aus den Einsendungen mögliche Gewinner auswählt und diese mit einem Preisgeld prämiert.

Internationale Treffen und Workshops

Im Vorfeld und Nachgang der Veranstaltung in Berlin wurden zusätzlich noch in Amsterdam und London **Satellitenveranstaltungen** durchgeführt. Ziel dieser Veranstaltungen war es, für das Thema des Vorhabens, den Energieverbrauch und die Umweltwirkung von Software, auch auf internationaler Bühne Sichtbarkeit zu schaffen. Zudem wurden die Arbeiten im Rahmen des Vorhabens vorgestellt. Bei beiden Veranstaltungen kamen zwischen 30-40 Teilnehmer in Präsenz zusammen und haben zusätzlich zu kurzen Vorträgen auch gemeinsam an technischen Herausforderungen gearbeitet oder in kleinen Gruppen diskutiert.

Insgesamt wurden mit den 3 Veranstaltungen ca. 150 Personen aus der der Software-Community erreicht und erfolgreich in das Vorhaben als Stakeholder eingebunden.

Steuerungsgruppe der SDIA

Zusätzlich zu den Veranstaltungen bietet die SDIA ihren Mitgliedern die Möglichkeit, sich in **Steuerungsgruppen** über das Projekt zu informieren, Feedback zu geben und das Vorhaben zu unterstützen. Beiträge können sowohl über die Community Plattform der SDIA als auch in monatlichen Meetings eingebracht werden.

Vorstellung des Vorhabens bei Veranstaltungen

Die Zwischenergebnisse des Vorhabens werden parallel auch auf verschiedenen Konferenzen vorgestellt. Dazu zählen bereits die OOP-Konferenz im Januar 2022 als auch die Continuous Life Cycle-Konferenz als auch die Agile Testing Days im November 2022.

Auch in der Presse wird das Projekt wahrgenommen[113] (IT Finanzmagazin) und über die verschiedenen Veranstaltungen berichtet[114] (siehe Data Center Dynamics und Heise). Besonders die Veranstaltungen wurden positiv wahrgenommen. Sie waren ein wichtiger Schritt, die Software-Gemeinschaft auf den eigenen Energie- und materiellen Ressourcenverbrauch aufmerksam zu machen.

Zuletzt ist das SoftAWERE Vorhaben in viele der Kommunikationsaktivitäten der SDIA fest eingebunden und erfreut sich hohem Interesse der ca. 700 Community-Mitglieder in der SDIA. Weiterhin wurde das Vorhaben im Rahmen der KDE-Nutzergruppe und den Workshops des Blauen Engels für Software vorgestellt, einem Workshop des Öko-Instituts, einer Green Software Foundation-Arbeitsgruppe und dem Technical Leadership Board der Cloud Native Foundation.

Das Vorhaben hat zudem Expert*innen und Wissenschaft über einen Begleitkreis und separate Workshops in das Vorhaben eingebunden. Die entsprechenden Protokolle und Ergebnisse befinden sich im Anhang B.

5.1 Begleitformate

Für das Vorhaben wurden diverse Begleitformate durchgeführt, welche im Anhang B ausführlich dokumentiert sind.

[113] Im IT Finanzmagazin, https://www.it-finanzmagazin.de/sustainable-finance-bleibt-ohne-green-software-unglaubwuerdig-141582/, abgerufen am 13.10.2023

[114] Data Center Dynamics: https://www.datacenterdynamics.com/en/news/sdia-holds-european-workshops-to-test-drive-software-sustainability-tools/ und Heise: https://www.heise.de/news/Hackathon-fuer-Energie-Sparfuechse-Oekobilanz-von-Open-Source-Projekten-verbessern-7259205.html, abgerufen am 13.10.2023

Dazu zählen:

- 2 Begleitkreistreffen (siehe Anhang B.1 und B.2)
- 1 Expertenworkshop (siehe Anhang B.1 und B.2)
- 2 Hackathon-Veranstaltungen (siehe Anhang B.1 und B.2)
- 1 Webinar (siehe Anhang B.1 und B.2)

5.2 Kommunikationswerkzeuge

Für das Vorhaben wurde Kommunikationsmaterial erstellt, welches im Anhang B zu finden ist. Dazu gehören:

- Ein Logo für das Vorhaben und die entwickelte Kennzeichnung (siehe Anhang A.1)
- Eine einseitige Zusammenfassung des Vorhabens (siehe Anhang A.2)
- Ein Internetauftritt für das Vorhaben (siehe Anhang A.3)
- Ein Projektflyer (siehe Anhang A.4)

5.3 Wiki für Green Coding Wissen & Praktiken

Im Rahmen des Vorhabens hat das Forschungsteam ein Wiki zum gesammelten Wissen rund um Green Coding Praktiken erstellt[115]. Dieses Wiki, konzipiert als unabhängige Wissensbasis, zielt darauf ab, die Software-Gemeinschaft durch die Bereitstellung verifizierter Informationen und Best Practices im Bereich der umweltfreundlichen Programmierung zu stärken.

Die Plattform wurde als Teil des Vorhabens mit dem Umweltbundesamt ins Leben gerufen, um einen kollektiven Beitrag zur Reduktion der Umweltauswirkungen digitaler Produkte zu leisten.

Der Inhalt des Wikis wird über das Jahr 2024 hinweg in einer gemeinschaftlichen Anstrengung entwickelt, wobei auf die aktive Teilnahme und den Austausch innerhalb der Community gesetzt wird. Ziel ist es, eine umfassende und zugängliche Wissensbasis zu schaffen, die die Entwicklung nachhaltiger Softwarelösungen weltweit unterstützt und fördert.

[115] Das Wiki ist unter https://s12y.wiki verfügbar und wird nach Abschluss des Vorhabens von der SDIA weiter betrieben.

6 Quellenverzeichnis

Alsaqqa, Samar, Samer Sawalha, and Heba Abdel-Nabi. 2020. 'Agile Software Development: Methodologies and Trends'. *International Journal of Interactive Mobile Technologies (iJIM)* 14 (11): 246. https://doi.org/10.3991/ijim.v14i11.13269.

Ammann, Paul, and Jeff Offutt. 2016. *Introduction to Software Testing*. Cambridge University Press.

Andreessen, Marc. 2011. 'Why Software Is Eating the World'. Andreessen Horowitz. 20 August 2011. https://a16z.com/why-software-is-eating-the-world/.

Bertoldi, Paolo. 2015. 'The European Programme for Energy Efficiency in Data Centre: The Code of Conduct'. January 11. https://www.ca-eed.eu/ia_document/energy-efficiency-in-data-centres-the-code-of-conduct/.

Bundesministerium für Digitales und Verkehr. 2022. 'Digitalstrategie Deutschland'. Bundesministerium für Digitales und Verkehr. https://www.digitalstrategie-deutschland.de/static/67803f22e4a62d19e9cf193c06999bcf/220830_Digitalstrategie_fin-barrierefrei.pdf.

Cano, Patricia Ortegon, Ayrton Mondragon Mejia, Silvana De Gyves Avila, Gloria Eva Zagal Dominguez, Ismael Solis Moreno, and Arianne Navarro Lepe. 2021. 'A Taxonomy on Continuous Integration and Deployment Tools and Frameworks'. In *New Perspectives in Software Engineering*, edited by Jezreel Mejia, Mirna Muñoz, Álvaro Rocha, and Yadira Quiñonez, 1297:323–36. Advances in Intelligent Systems and Computing. Cham: Springer International Publishing. https://doi.org/10.1007/978-3-030-63329-5_22.

Chohan, Usman W. 2021. 'The Double Spending Problem and Cryptocurrencies'. SSRN Scholarly Paper. Rochester, NY. https://doi.org/10.2139/ssrn.3090174.

Conner-Simons, Adam. 2020. 'If Transistors Can't Get Smaller, Then Coders Have to Get Smarter'. MIT News | Massachusetts Institute of Technology. 5 June 2020. https://news.mit.edu/2020/mit-csail-computing-technology-after-moores-law-0605.

Eira, Astrid. 2020. 'Number of Facebook Employees 2022/2023: Compensation, Tenure & Perks'. Financesonline.Com. 23 April 2020. https://financesonline.com/number-of-facebook-employees/.

'Energy-Efficient Design of Data Centres Using Power Management and Virtualisation (LEAP)'. n.d. Accessed 17 October 2023. https://amsterdameconomicboard.com/app/uploads/2022/02/Happy-Flow-Manual.pdf.

Firesmith, Donald. 2017. 'Virtualization via Containers'. 24 September 2017. https://insights.sei.cmu.edu/blog/virtualization-via-containers/.

'Greenhouse Gas Emission Intensity of Electricity Generation in Europe'. n.d. Accessed 27 September 2023. https://www.eea.europa.eu/ims/greenhouse-gas-emission-intensity-of-1.

Gröger, Jens, Andreas Köhler, Stefan Naumann, Andreas Filler, Achim Guldner, Eva Kern, Lorenz M. Hilty, and Yuliyan Maksimov. 2018. 'Entwicklung und Anwendung von Bewertungsgrundlagen für ressourceneffiziente Software unter Berücksichtigung bestehender Methodik'. *Umweltbundesamt*, Texte, 105 (2018). https://www.umweltbundesamt.de/sites/default/files/medien/1410/publikationen/2018-12-12_texte_105-2018_ressourceneffiziente-software_0.pdf.

Gröger, Jens, Ran Liu, Dr. Lutz Stobbe, Jan Druschke, and Nikolai Richter. 2021. 'Green Cloud Computing: Lebenszyklusbasierte Datenerhebung zu Umweltwirkungen des Cloud Computing'. *Umweltbundesamt*, Texte 94/2021, , no. 94 (June), 202.

Hao, Karen. 2019. 'The Computing Power Needed to Train AI Is Now Rising Seven Times Faster than Ever Before'. MIT Technology Review. 11 November 2019. https://www.technologyreview.com/2019/11/11/132004/the-computing-power-needed-to-train-ai-is-now-rising-seven-times-faster-than-ever-before/.

Hilty, Prof. Dr. Lorenz, Dr. Wolfgang Lohmann, Dr. Siegfried Behrendt, Michaela Evers-Wölk, Prof. Dr. Klaus Fichter, and Dr. Ralph Hintemann. 2015. 'Green Software - Establishing and Exploiting Potentials for Environmental Protection in Information and Communication Technology (Green IT)'. 23/2015. Texte. Dessau-Roßlau: Umweltbundesamt. https://www.umweltbundesamt.de/en/publikationen/green-software.

‘Intelligent Platform Management Interface Specification Second Generation’. 2015. Intel Corporation, Hewlett-Packard Company, NEC Corporation, Dell Inc. https://www.intel.com/content/dam/www/public/us/en/documents/specification-updates/ipmi-intelligent-platform-mgt-interface-spec-2nd-gen-v2-0-spec-update.pdf.

Jaeger-Erben, Melanie, Erik Poppe, Eduard Wagner, Anton Schaefer, Jan Druschke, Jens Gröger, and Felix Behrens. 2023. ‘Analyse der softwarebasierten Einflussnahme auf eine verkürzte Nutzungsdauer von Produkten’. *Umweltbundesamt*, 13/2023, , January. https://www.umweltbundesamt.de/en/publikationen/analyse-der-softwarebasierten-einflussnahme-auf.

Janzen, D., and H. Saiedian. 2005. ‘Test-Driven Development Concepts, Taxonomy, and Future Direction’. *Computer* 38 (9): 43–50. https://doi.org/10.1109/MC.2005.314.

Kavanagh, Richard, Django Armstrong, and Karim Djemame. 2016. ‘Accuracy of Energy Model Calibration with IPMI’. In *2016 IEEE 9th International Conference on Cloud Computing (CLOUD)*, 648–55. https://doi.org/10.1109/CLOUD.2016.0091.

Kennes, Tom. 2023. ‘Measuring IT Carbon Footprint: What Is the Current Status Actually?’ arXiv. http://arxiv.org/abs/2306.10049.

Kern, Eva, Lorenz M. Hilty, Achim Guldner, Yuliyan V. Maksimov, Andreas Filler, Jens Gröger, and Stefan Naumann. 2018. ‘Sustainable Software Products—Towards Assessment Criteria for Resource and Energy Efficiency’. *Future Generation Computer Systems* 86 (September):199–210. https://doi.org/10.1016/j.future.2018.02.044.

Khan, Kashif Nizam, Mikael Hirki, Tapio Niemi, Jukka K. Nurminen, and Zhonghong Ou. 2018. ‘RAPL in Action: Experiences in Using RAPL for Power Measurements’. *ACM Transactions on Modeling and Performance Evaluation of Computing Systems* 3 (2): 1–26. https://doi.org/10.1145/3177754.

Khorikov, Vladimir. 2020. *Unit Testing Principles, Practices, and Patterns*. Simon and Schuster.

Köhn, Marina, Jens Gröger, and Dr. Lutz Stobbe. 2020. ‘Energie- und Ressourceneffizienz digitaler Infrastrukturen: Ergebnisse des Forschungsprojektes „Green Cloud-Computing“’. Umweltbundesamt. https://www.umweltbundesamt.de/sites/default/files/medien/376/publikationen/politische-handlungsempfehlungen-green-cloud-computing_2020_09_07.pdf.

Kumar, Suhas. 2015. ‘Fundamental Limits to Moore’s Law’. arXiv. http://arxiv.org/abs/1511.05956.

Leiserson, Charles E., Neil C. Thompson, Joel S. Emer, Bradley C. Kuszmaul, Butler W. Lampson, Daniel Sanchez, and Tao B. Schardl. 2020. ‘There’s Plenty of Room at the Top: What Will Drive Computer Performance after Moore’s Law?’ *Science* 368 (6495): eaam9744. https://doi.org/10.1126/science.aam9744.

Mazouz, Abdelhafid, Alexandre Laurent, Benoît Pradelle, and William Jalby. 2014. ‘Evaluation of CPU Frequency Transition Latency’. *Computer Science - Research and Development* 29 (3–4): 187–95. https://doi.org/10.1007/s00450-013-0240-x.

Merkel, Dirk. 2014. ‘Docker: Lightweight Linux Containers for Consistent Development and Deployment’. *Linux j* 239 (2): 2.

Minyard, Corey. 2006. ‘IPMI - A Gentle Introduction with OpenIPMI’. *Software Montavista* XP055165227.

Moore, Gordon E. 1965. ‘Cramming More Components onto Integrated Circuits’. *Electronics* 38 (8).

Moss, Sebastian. 2021. ‘Facebook Plans Huge $29-34 Billion Capex Spending Spree in 2022, Will Invest in AI, Servers, and Data Centers’. Magazine. Data Center Dynamics. 1 November 2021. https://www.datacenterdynamics.com/en/news/facebook-plans-huge-29-34-billion-capex-spending-spree-in-2022-will-invest-in-ai-servers-and-data-centers/.

Müller, Marion. 2023. ‘Digitale Wirtschaft wächst’. *Die Aktiengesellschaft* 68 (17): r247–48. https://doi.org/10.9785/ag-2023-681711.

Muralikrishna, Iyyanki V., and Valli Manickam. 2017. ‘Chapter Five - Life Cycle Assessment’. In *Environmental Management*, edited by Iyyanki V. Muralikrishna and Valli Manickam, 57–75. Butterworth-Heinemann. https://doi.org/10.1016/B978-0-12-811989-1.00005-1.

Nagle, Frank, James Dana, Jennifer Hoffman, Steven Randazzo, and Yanuo Zhou. 2022. ‘Census II of Free and Open Source Software’. *The Linux Foundation and The Laboratory for Innovation Science at Harvard*, March 2022. https://8112310.fs1.hubspotusercontent-

na1.net/hubfs/8112310/LF%20Research/Harvard%20Census%20II%20of%20Free%20and%20Open%20Source%20Software%20-%20Report.pdf.

Napoleão, Bianca Minetto, Fabio Petrillo, and Sylvain Hallé. 2020. 'Open Source Software Development Process: A Systematic Review'. arXiv. http://arxiv.org/abs/2008.05015.

Pereira, Rui, Marco Couto, Francisco Ribeiro, Rui Rua, Jácome Cunha, João Paulo Fernandes, and João Saraiva. 2021. 'Ranking Programming Languages by Energy Efficiency'. *Science of Computer Programming* 205 (May):102609. https://doi.org/10.1016/j.scico.2021.102609.

Prakash, Siddharth, Ran Liu, Karsten Schischke, and Dr. Lutz Stobbe. 2012. *Zeitlich optimierter Ersatz eines Notebooks unter ökologischen Gesichtspunkten*. Texte 44/2012. Umweltbundesamt. https://www.umweltbundesamt.de/publikationen/zeitlich-optimierter-ersatz-eines-notebooks-unter.

Raymond, Eric S. 2008. *The Art of UNIX Programming: With Contributions from Thirteen UNIX Pioneers, Including Its Inventor, Ken Thompson*. Nachdr. Addison-Wesley Professional Computing Series. Boston: Addison-Wesley.

Schulze, Max, Radika Kumar, and Michael J Oghia. 2021. 'Trade Competitiveness Briefing Paper Taxonomy Guide: Infrastructure in the Digital Economy'. *Trade Competitiveness Briefing Paper* 2022/01 (April). https://doi.org/10.14217/ComSec.952.

Soares, Eliezio, Gustavo Sizilio, Jadson Santos, Daniel Alencar, and Uira Kulesza. 2022. 'The Effects of Continuous Integration on Software Development: A Systematic Literature Review'. arXiv. http://arxiv.org/abs/2103.05451.

'Stack Overflow Developer Survey 2021'. n.d. Stack Overflow. Accessed 17 October 2023. https://insights.stackoverflow.com/survey/2021/?utm_source=social-share&utm_medium=social&utm_campaign=dev-survey-2021.

Steinfeld, Grant. 2020. '5 Steps of Test-Driven Development - IBM Developer'. IBM Developer. 6 February 2020. https://developer.ibm.com/articles/5-steps-of-test-driven-development/.

Theis, Thomas N., and H.-S. Philip Wong. 2017. 'The End of Moore's Law: A New Beginning for Information Technology'. *Computing in Science & Engineering* 19 (2): 41–50. https://doi.org/10.1109/MCSE.2017.29.

W3Techs. n.d. 'Usage Statistics and Market Share of Operating Systems for Websites, February 2024'. W3Techs. Accessed 5 February 2024. https://w3techs.com/technologies/overview/operating_system.

Waldrop, M. Mitchell. 2016. 'The Chips Are down for Moore's Law'. *Nature News* 530 (7589): 144. https://doi.org/10.1038/530144a.

Whitehead, Beth, Deborah Andrews, and Amip Shah. 2015. 'The Life Cycle Assessment of a UK Data Centre'. *The International Journal of Life Cycle Assessment* 20 (3): 332–49. https://doi.org/10.1007/s11367-014-0838-7.

Witkowski, Wallace. n.d. '"Moore's Law's Dead," Nvidia CEO Jensen Huang Says in Justifying Gaming-Card Price Hike'. MarketWatch. Accessed 15 September 2023. https://www.marketwatch.com/story/moores-laws-dead-nvidia-ceo-jensen-says-in-justifying-gaming-card-price-hike-11663798618.

Xu, Guoqing, Nick Mitchell, Matthew Arnold, Atanas Rountev, and Gary Sevitsky. 2010. 'Software Bloat Analysis: Finding, Removing, and Preventing Performance Problems in Modern Large-Scale Object-Oriented Applications'. In *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research*, 421–26. FoSER '10. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/1882362.1882448.

A Kommunikationsunterlagen

A.1 Logo

Das Logo wurde mit Hinblick auf die Entwicklung einer möglichen Kennzeichnung so gestaltet, dass sich damit verschiedene Zustände über die Farbe der Flagge ausdrücken lassen. So ist es denkbar, dass eine Bibliothek mit einer ungenügenden Energie- und Ressourceneffizienz mit einer roten Flagge ausgezeichnet wird, eine Bibliothek mit einer ausreichenden bzw. befriedigenden Effizienz eine orange/gelb Flagge erhält und eine Bibliothek mit einer hohen Effizienz eine grüne Flagge bekommt.

Abbildung 37: Logo Standard Variante

Quelle: eigene Darstellung, Sustainable Digital Infrastructure Alliance e.V.

Abbildung 38: Logo Invert Variante

Quelle: eigene Darstellung, Sustainable Digital Infrastructure Alliance e.V.

Abbildung 39: Logo Farb-Variante als Beispiel für Anpassung für Kennzeichnung

Quelle: eigene Darstellung, Sustainable Digital Infrastructure Alliance e.V.

A.2 One-Pager

Für die Ansprache von interessierten Akteuren aus Wissenschaft, Wirtschaft und Politik wurde außerdem eine einseitige Zusammenfassung in Englisch erstellt ("One Pager"). Dieser ist in der Anlage beigefügt und auch im Knowledge Hub der SDIA veröffentlicht.

Abbildung 40: Zusammenfassung des Vorhabens in Englisch

SoftAWERE - Project Summary

Tools for energy- & resource-efficient software development.

Digitization is giving rise to new business models and services that are predominantly based on software-related innovations. For example, artificial intelligence (AI) is enabling information technology (IT) to solve problems and initiate actions on its own. On the basis of assistant-supported functionalities, greater automation is taking place in transportation systems and in the manufacturing industry, and algorithms are being used to identify patterns and regularities in large data sets that give rise to new business ideas.

The disciplines of software development and data processing have so far not been constrained by technology. This is because inefficient programing is often compensated for by ever faster processors and more main memory, and the ability of networks to transfer ever more data in ever-shorter time does not contribute to data economy. High hardware utilization, due to inefficient or bloated software, in turn, has a direct impact on energy consumption and the hardware renewal cycle.

To ensure transparency in the software development community and provide developers with tools to reduce energy consumption as a first step, the German Federal Environment Agency (Umweltbundesamt) has launched a research project funded by the German Federal Ministry of Economics and Technology (BMWi).

The project is to consider the following **focus tasks**:

- Tools that support software developers in programing energy-efficient and hardware-saving software.
- Investigate the feasibility of labelling energy-efficient software and develop a concept for evaluating energy efficiency.
- Establish awareness of the problem of energy and resource consumption of software through various communication channels to the developer community.
- Increase transparency of energy consumption of software toward third parties. For this purpose, the feasibility of a national rating system for energy-efficient software and the possibility of a graphical representation for energy-efficient software will be investigated and a concept for the evaluation of energy efficiency will be developed.

In addition, the project is intended to support the achievement of climate protection goals and serve as a basis for finding further solutions in the future.

Version 2022-11-16 – Copyright by the Sustainable Digital Infrastructure Alliance e.V. 1

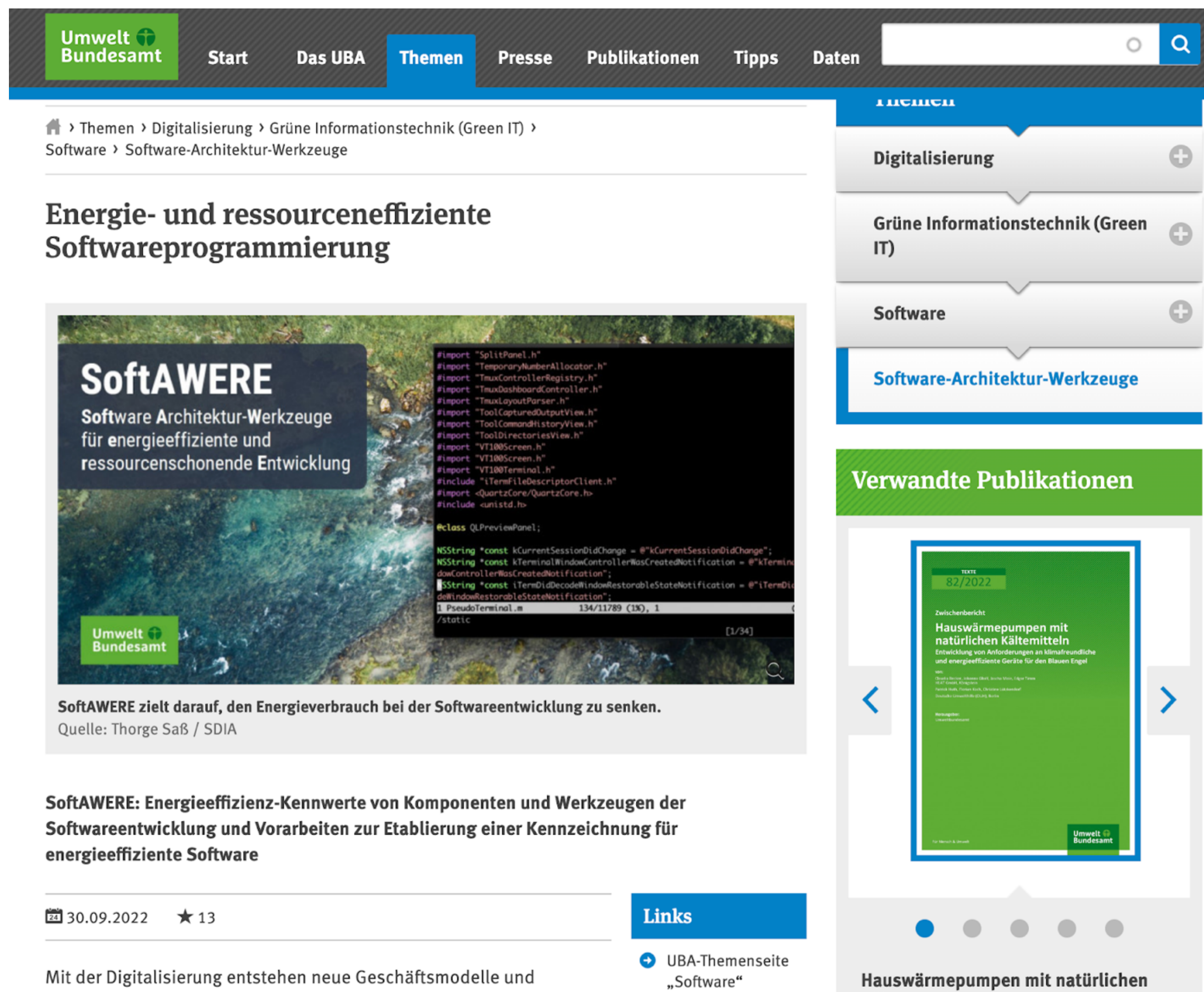
Quelle: Knowledge Hub[116] der Sustainable Digital Infrastructure Alliance e.V.

[116] https://knowledge.sdialliance.org

A.3 Internetauftritt

Auf der **Website des Umweltbundesamts**[117] wurde das Projekt auf einer dedizierten Themenseite im Bereich “Grüne Informationstechnik (Green IT)” veröffentlicht und darin auch auf die Themenseite der SDIA und Veranstaltungen verwiesen.

Abbildung 41: Webseite des Vorhabens beim Umweltbundesamt



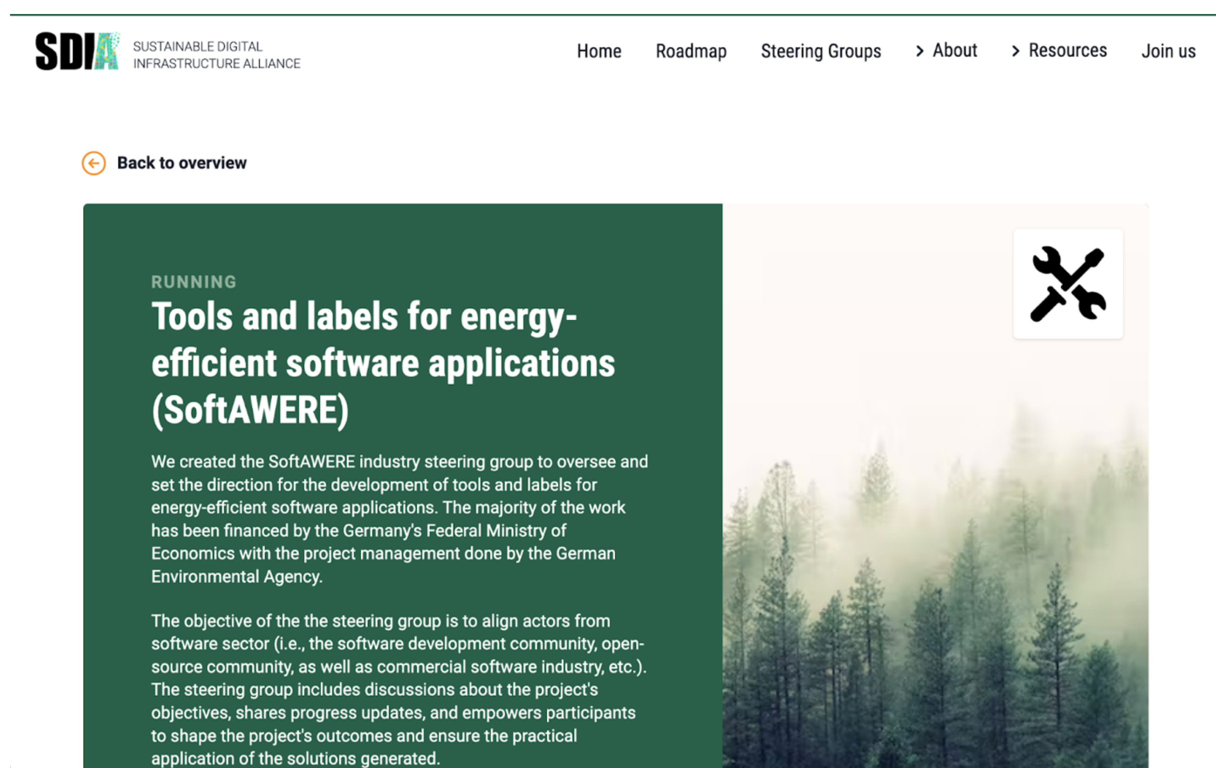
Quelle: eigene Darstellung, Sustainable Digital Infrastructure Alliance e.V.

Auf der **Webseite der SDIA**[118] wurde speziell für die Steuerungsgruppe eine Webseite eingerichtet, um es der Öffentlichkeit möglich zu machen, an den Ergebnissen und der Begleitung des Vorhabens teilzunehmen.

[117] Webseite des Vorhabens beim Umweltbundesamt: https://www.umweltbundesamt.de/themen/digitalisierung/gruene-informationstechnik-green-it/software/energie-ressourceneffiziente-softwareprogrammierung

[118] SDIA-Webseite: https://sdialliance.org/steering-groups/softawere

Abbildung 42: Webseite des Vorhabens auf dem Internetauftritt der SDIA

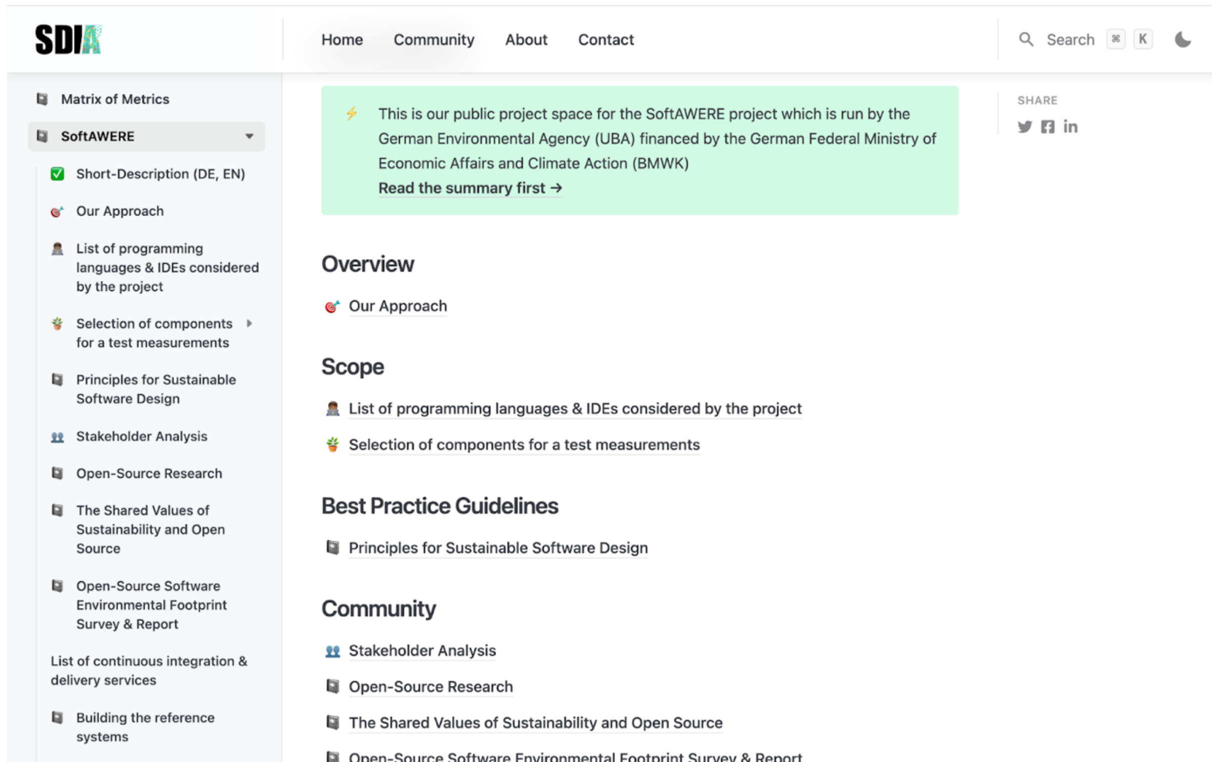


Quelle: eigene Darstellung, Sustainable Digital Infrastructure Alliance e.V.

Die Zwischenergebnisse und das Wissen, welches im Rahmen des Vorhabens erfasst wurde, werden außerdem **im Knowledge Hub der SDIA**[119] bereitgestellt um es Interessierten zu ermöglichen, die Inhalte zu nutzen.

[119] SDIA Knowledge Hub: https://knowledge.sdialliance.org/softawere

Abbildung 43: Der SDIA Knowledge Hub mit gesondertem Bereich für SoftAWERE



Quelle: eigene Darstellung, Sustainable Digital Infrastructure Alliance e.V.

Zuletzt wurde auf **GitLab** eine offene Gruppe angelegt,[120] auf der die Programmcodes und Blaupausen für das Test-Labor veröffentlicht sind. Auch hier schafft das Vorhaben Transparenz und gibt insbesondere der Open-Source-Gemeinschaft die Möglichkeit die Ergebnisse und Werkzeuge selbst auszuprobieren, Verbesserungen zu machen bzw. die Herangehensweisen zu überprüfen.

A.4 Projekt-Flyer

Der **Flyer für das Vorhaben** wurde als PDF auf Basis der Vorlage vom Umweltbundesamt erstellt. Der Flyer gibt eine Übersicht über das Vorhaben, zeigt Schaubilder vom Test-Labor und gibt Auskunft über die Vorhabenspartner*innen und Ansprechpartner*innen.

[120] GitLab: https://gitlab.com/softawere-hackathon

Abbildung 44: Flyer Vorderseite

Software-Effizienz ist immer bedeutsamer

Die Digitalisierung und die damit verbundene digitale Transformation von Industrie, Regierung und Gesellschaft befinden sich in vollem Gange. Und mit der verstärkten Nutzung von Hard- und Software steigt auch deren ökologischer Fußabdruck.

Der Softwareentwickler Marc Andreessen brachte dies bereits im Jahr 2011 in einem Essay auf den Punkt: „Software frisst die Welt auf." [Software is eating the World].

Dabei war die Effizienz der Software selbst lange Zeit kein Thema, es war wichtiger, schneller, mehr und größere Software-Anwendungen zu entwickeln. Dass dabei mehr Hardware-Ressourcen benötigt wurden, war bislang wirtschaftlicher als mehr Zeit in die Entwicklung und die Optimierung der Software zu investieren.

Sowohl aus technischer als auch aus ökologischer Perspektive hat dieses Vorgehen jetzt seine Grenzen erreicht. Die Aufmerksamkeit der Gesellschaft und Politik beschäftigt sich zunehmend mit den Umweltauswirkungen und dem Ressourcenverbrauch der Digitalwirtschaft. Für Software ist daher jetzt Zeit gekommen, effizienter und ressourcenschonender zu werden.

Soft AWERE

Schwerpunkte von SoftAWERE

Die hohe Inanspruchnahme von Hardware durch ineffiziente oder aufgeblähte Software hat unmittelbare Auswirkung auf den Energieverbrauch von Computern und den Bedarf nach immer leistungsfähigerer Informationstechnik.

Um ein Bewusstsein für die Umweltwirkungen von Software zu schaffen und geeignete Instrumente zu entwickeln, diesem Trend entgegenzuwirken, hat das Umweltbundesamt (UBA) das Projekt *„Software Architektur-Werkzeuge für energieeffiziente und ressourcenschonende Entwicklung"* – kurz *SoftAWERE* – ins Leben gerufen.

Das Vorhaben SoftAWERE hat folgende Schwerpunkte:

- Entwicklung von Werkzeugen, die Software-Entwickler*innen dabei unterstützen, energieeffiziente und hardwareschonende Software zu programmieren,
- Bewertung der Energie- und Ressourceneffizienz von Software,
- Prüfung der Machbarkeit einer Kennzeichnung von energie- und ressourceneffizienter Software,
- Bewusstseinsbildung innerhalb der Software-Entwickler*innen-Szene und
- Initiierung eines breiteren Diskurses zur Nachhaltigkeit von Software.

Methode

Im eigenen SoftAWERE-Testlabor können wir softwarebasierte Messungen durchführen (RAPL, IPMI). Durch diese Messungen können wir Umrechnungsformeln kalibrieren, die sich auch in Umgebungen einsetzen lassen, die keinen Zugriff auf softwarebasierte Messungen haben (z.B. in einer Virtualisierungsumgebung).

Das SoftAWERE-Testlabor ist offen für alle Open-Source-Projekte, die Messungen durchführen möchten.

48 0.380 0.0299 0.0114
1900 0.140 24000 0.0115 1.94e-9 0.389

Abb.1: Visualisierung der Messergebnisse

Abb.2: Berechnung der Umweltwirkung bei Ausführung von Softwaretests im SoftAWERE-Testlabor

Quelle: eigene Darstellung, Sustainable Digital Infrastructure Alliance e.V.

Abbildung 45: Flyer Rückseite

Projektdaten

Kurztitel: SoftAWERE

Projekttitel: Energieeffizienz-Kennwerte von Komponenten und Werkzeugen der Softwareentwicklung und Vorbereitung zur Etablierung einer Kennzeichnung für energieeffiziente Software

Auftraggeber: Umweltbundesamt
unter der Fachaufsicht des BMWK

Förderkennzeichen: UFOPLAN FKZ-Nr.: 37EV 20 102 0

Laufzeit: August 2021 bis Juni 2023

Projektdurchführung:

Wichtige Links

- **UBA-Themenseite Software und Umwelt**
  https://www.umweltbundesamt.de/themen/digitalisierung/gruene-informationstechnik-green-it/software-umwelt
- **SDIA SoftAWERE**
  https://sdialliance.org/steering-groups/softawere/
- **Software-Bewertungsmethodik**
  https://www.umweltbundesamt.de/publikationen/entwicklung-anwendung-von-bewertungsgrundlagen-fuer

Kontakt

Auftraggeber:
Umweltbundesamt
Andreas Halatsch
Fachgebiet V1.4 Energieeffizienz
Marina Köhn, Mathias Bornschein
Fachgebiet Z 2.3 Digitalisierung und Umweltschutz, E-Government

Projektteam:
Sustainable Digital Infrastructure Alliance e.V.
sdialliance.org
Colonnaden 5, 20354 Hamburg
Max Schulze (Projektleitung), Stephanie Leufgen

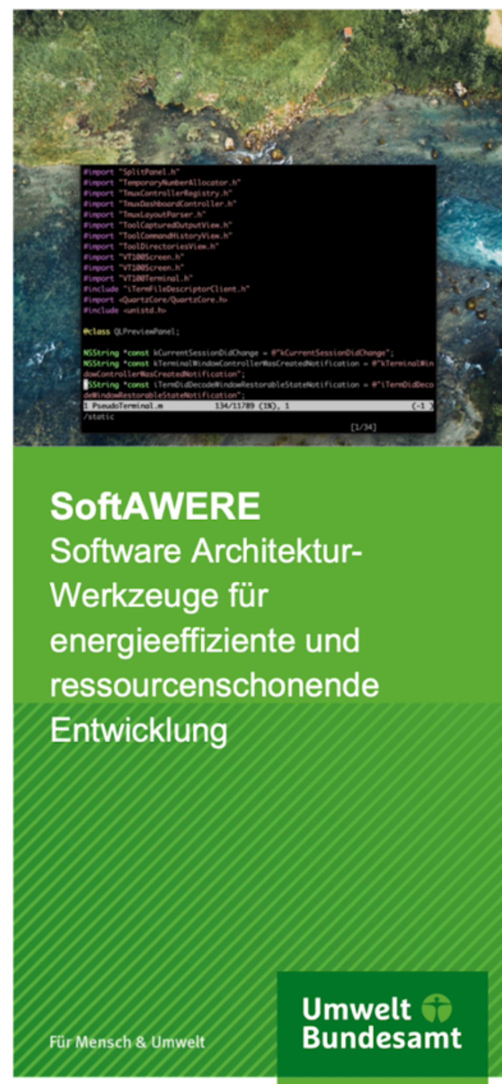
Öko-Institut e.V., Büro Berlin
www.oeko.de
Borkumstraße 2, 13189 Berlin
Jens Gröger, Felix Behrens

Herausgeber:
Umweltbundesamt
Postfach 14 06
06844 Dessau-Roßlau
Tel: +49 340-2103-0
info@umweltbundesamt.de
Internet: www.umweltbundesamt.de

Publikationen als pdf:
www.umweltbundesamt.de/publikationen

Bildquellen:
Sustainable Digital Infrastructure Alliance e.V.

Stand: 07.11.2022



Quelle: eigene Darstellung, Sustainable Digital Infrastructure Alliance e.V.

B Protokolle und Ergebnisse der Veranstaltungen

B.1 Erstes Begleitkreistreffen

Im Verlauf des Vorhabens wurden zwei Begleitkreistreffen durchgeführt. Das erste Begleitkreistreffen fand am 29.10.2021 virtuell statt. In den folgenden Tabellen und Abbildungen sind die Agenda des Begleitkreistreffens, sowie die Arbeitsergebnisse dokumentiert.

Tabelle 8: Agenda

Thema	Zeit
Begrüßung und Ziele des Vorhabens *Marina Köhn / Andreas Halatsch, UBA und Thomas Hinsch, BMWi*	10:00 – 10:15
Vorstellungsrunde *Begleitkreis*	10:15 – 10:30
Vorstellung vorangegangener Forschung *Jens Gröger, Öko Institut*	10:30 – 10:50
Vorstellung des Vorhabens SoftAWERE *Max Schulze, SDIA*	10:50 – 11:00
Herausforderung des Forschungsvorhabens *Max Schulze, SDIA*	11:00 – 11:10
Interaktiver Workshop *Begleitkreis / Max Schulze, SDIA*	11:25 – 12:30
Diskussion der Ergebnisse und weiterer Zeitplan *Thorge Saß, SDIA*	12:30 – 12:40
Abschluss *Andreas Halatsch, UBA und Thomas Hinsch, BMWi*	12:40 – 13:00

Übersicht Virtuelles Whiteboard mit Notizen aus dem Begleitkreis[121].

Abbildung 46: Übersicht Präsentation 1. Begleitkreis

Quelle: eigene Darstellung, Sustainable Digital Infrastructure Alliance e.V.

[121] SDIA Knowledge Hub enthält die Notizen zum Begleitkreis: https://knowledge.sdialliance.org/softawere/begleitkreistreffen-1

Abbildung 47: Diskussionsergebnisse - 1. Fragestellung

Quelle: eigene Darstellung, Sustainable Digital Infrastructure Alliance e.V.

Abbildung 48: Diskussionsergebnisse - 2. Fragestellung

HERAUSFORDERUNG

Wie können wir die Anwendbarkeit der Ergebnisse für die Entwicklergemeinschaft aber auch für die nächste Generation maximieren?

Entwicklergemeinschaft - Awareness schaffen

Abfallkette

Zukunftsmeile 2 einbringen

SSCP Angebot = Kooperationfläche, mit Studenten und Unternehmen

Symposiumtage, Studenten vorstellen

Zusammenfassung von Best Practice

In IDEs zu integrieren

Automatisieren, Aufwand reduzieren

Dekomplixierung der Methodik

Umfassende (crowd sourced) Plattform für Best Practices, Learning Content, Tools...

Einstellung != Verhalten

Was ist die Belohnung für den Entwickler?

An welche Plattform kann man sich anschließen?

Blog Beiträge, sehr verteilt

Verhaltensänderungen durch Habit Loops adressieren: cue - routine- reward

Lehren/Universitäten

Einfache Hebel und Konfigurationen zu finden

Schärfe - Genauigkeit zu reduzieren

Selbstpflege /Wiki Effekt

Integration in Curiculla

ohne Messeinrichtung

GAIA-X Anwendbarkeit

umweltverträglich/resourcen-schonend

"Nachhaltigkeit"

05 miro

Quelle: eigene Darstellung, Sustainable Digital Infrastructure Alliance e.V.

Abbildung 49: Diskussionsergebnisse - 3. Fragestellung

HERAUSFORDERUNG

Wie können Unternehmen gewonnen werden, die Einfluss auf „Softwarestandards“ und Softwarebibliotheken haben?

miro

Quelle: eigene Darstellung, Sustainable Digital Infrastructure Alliance e.V.

B.2 Zweites Begleitkreistreffen

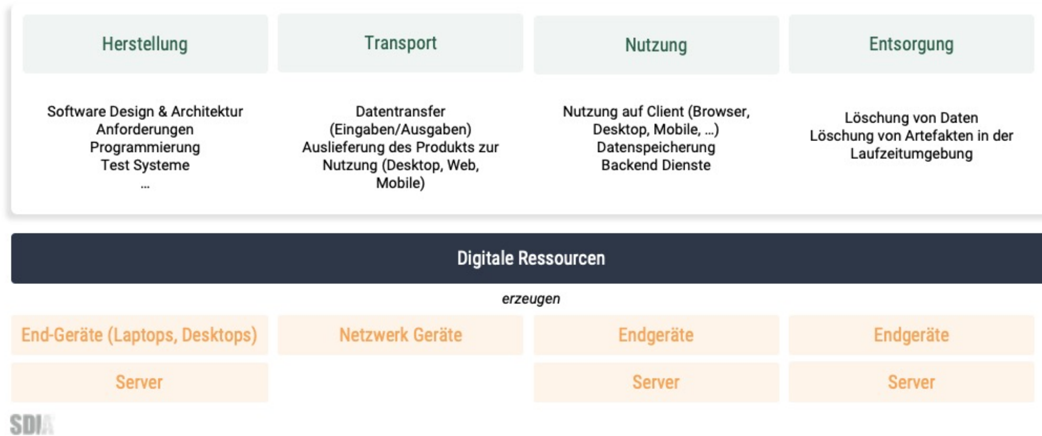
In einem zweiten Begleitkreistreffen, in den Räumen des Bundesministeriums für Wirtschaft und Klimaschutz, wurde wieder eine interessierte Gruppe von Stakeholdern aus der Politik, der Bildung, Forschung und der Industrie zusammengebracht. Das Treffen fand am 20.03.2023 statt. Es haben insgesamt 26 Teilnehmer sowohl vor Ort als auch per Videokonferenz teilgenommen.

Die Agenda des Begleitkreises war wie folgt aufgebaut:

- Begrüßung und Ablauf des Begleitkreistreffens
- Warum ist das Thema aus Sicht des UBAs relevant?
  - Relevanz in der Beschaffung
  - Digitalisierung - das ist auch Software; und die Umweltwirkung ist unbekannt
- Umweltpolitische Relevanz
- Zusammenfassung & Ergebnisse des Vorhabens SoftAWERE
- Ziele und Herausforderungen, die im Vorhaben aufgegriffen wurden
- Methodik zur Messung der Umweltwirkung von digitalen Produkten/Software
- Herausforderung des Forschungsvorhabens, sowie ein Blick in die Zukunft zum Thema digitale Produkte, Software und deren Umweltwirkung zu messen und transparent zu machen
  - Handlungsempfehlungen und Beispiele
  - Verantwortung übernehmen
  - Handlungs- und Forschungsbedarf - Deutschland als Vorreiter, europäisch, Umsetzung, Richtung Normung (Qualitätsstandard für die Art der Programmierung)

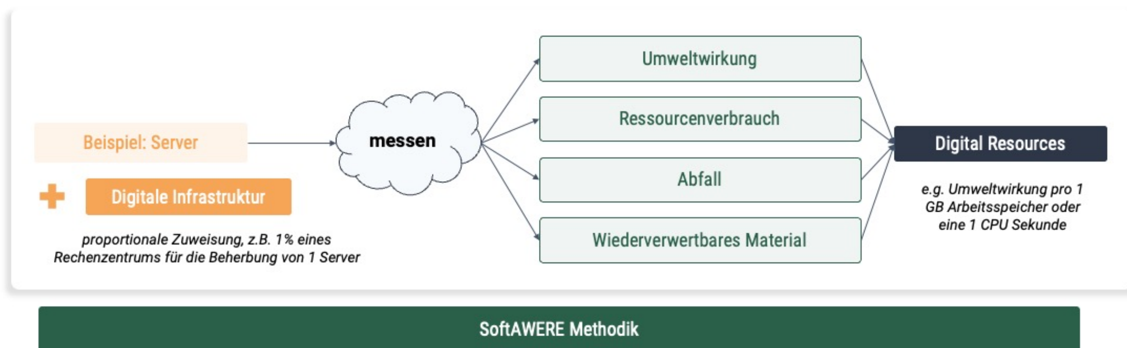
Im zweiten Teil des Treffens wurde gemeinsam mit den Teilnehmenden über mögliche Handlungsempfehlungen und nächste Schritte diskutiert. Die Ergebnisse aus der Diskussion sind in die Handlungsempfehlungen aus diesem Bericht mit eingeflossen und finden sich im Kapitel 3. Die Diskussion wurde mit Hilfe von Miro moderiert und strukturiert.

Abbildung 50: Auszug Präsentation – 2. Begleitkreistreffen



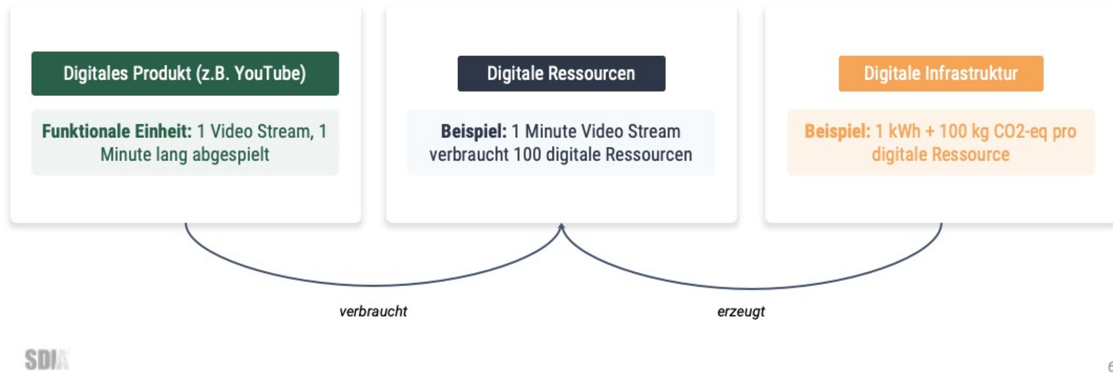
Quelle: eigene Darstellung, Sustainable Digital Infrastructure Alliance e.V.

Abbildung 51: Auszug Präsentation – 2. Begleitkreistreffen



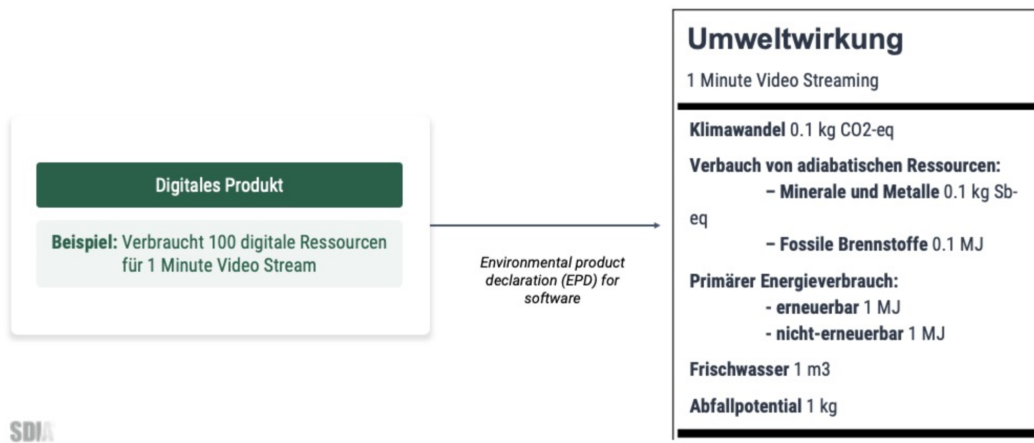
Quelle: eigene Darstellung, Sustainable Digital Infrastructure Alliance e.V.

Abbildung 52: Auszug Präsentation – 2. Begleitkreistreffen



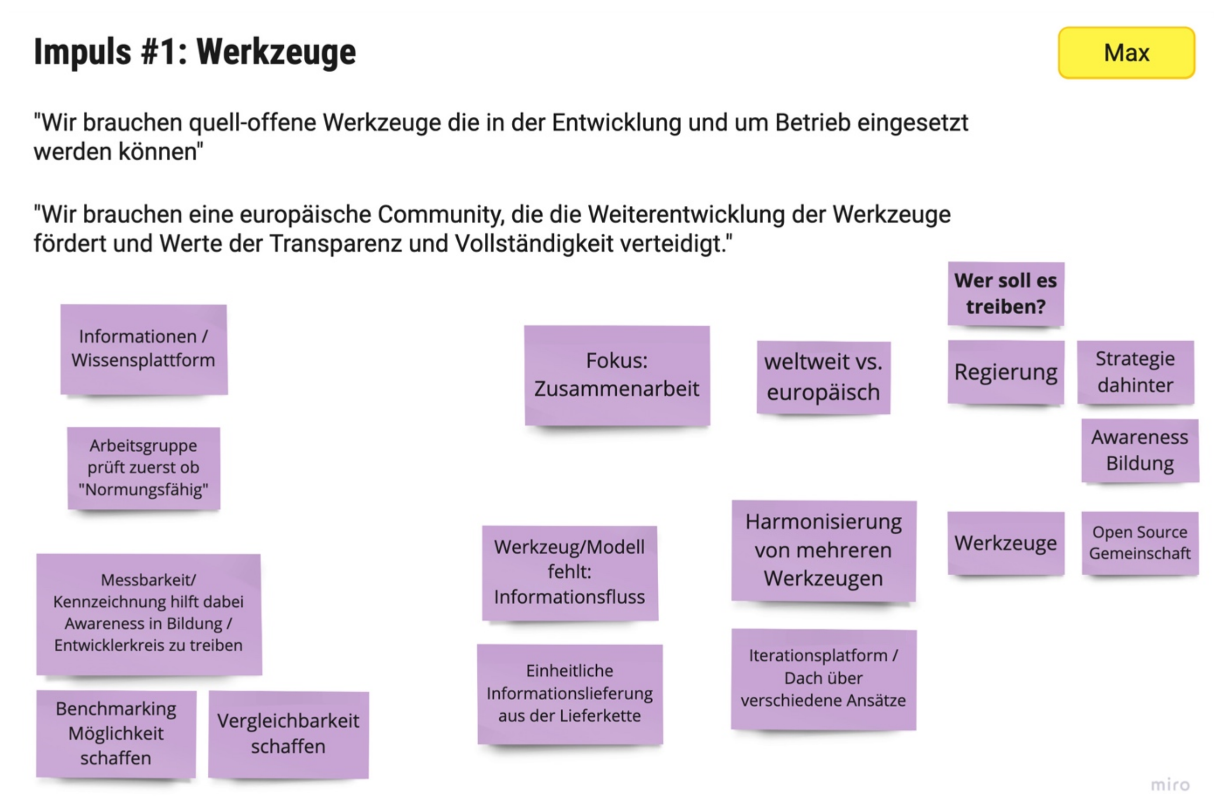
Quelle: eigene Darstellung, Sustainable Digital Infrastructure Alliance e.V.

Abbildung 53: Auszug Präsentation – 2. Begleitkreistreffen



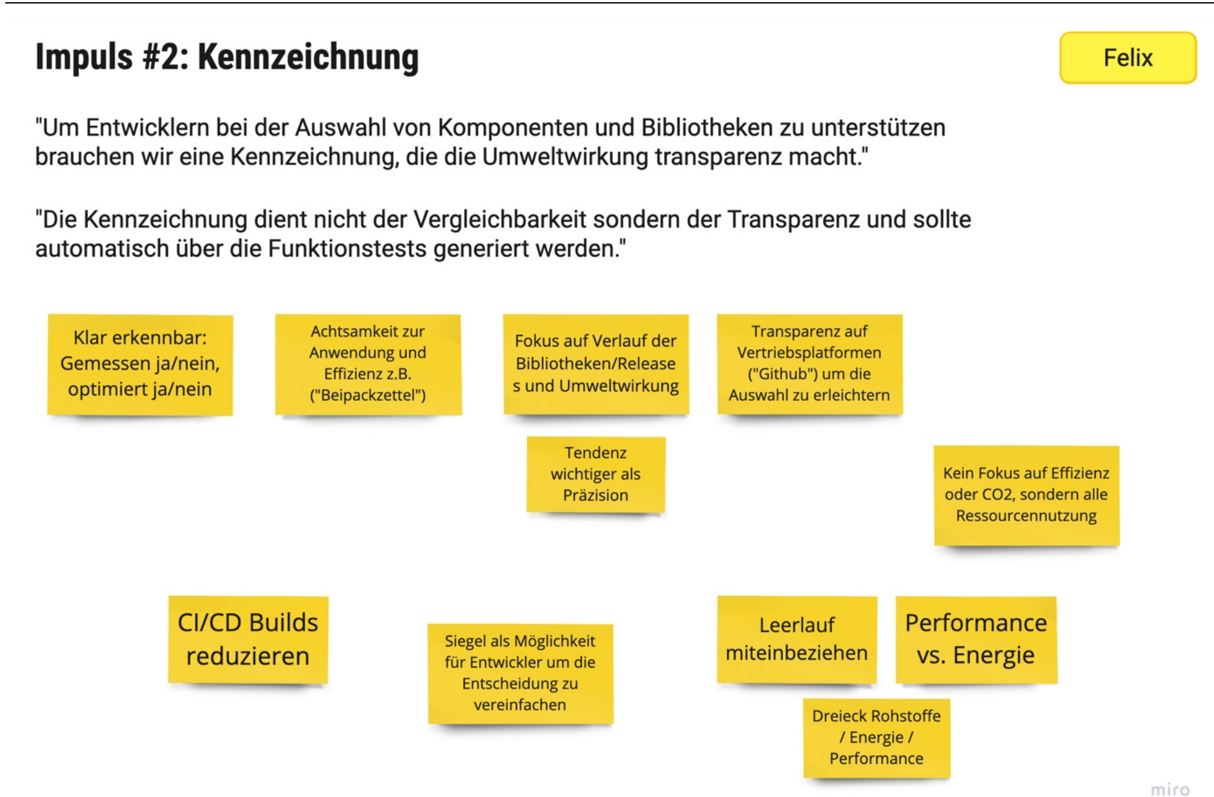
Quelle: eigene Darstellung, Sustainable Digital Infrastructure Alliance e.V.

Abbildung 54: Impuls #1 & Kommentare der Begleitkreismitglieder



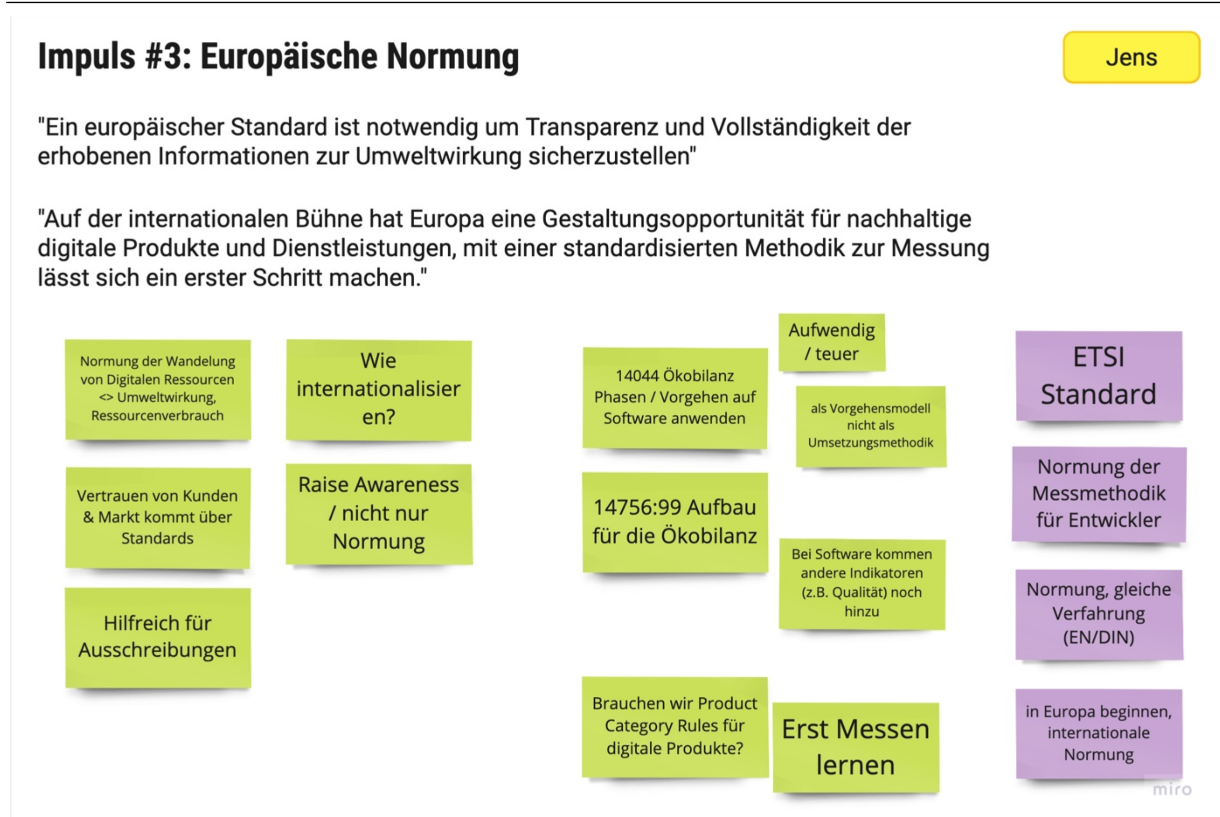
Quelle: eigene Darstellung, Sustainable Digital Infrastructure Alliance e.V.

Abbildung 55: Impuls #2 & Kommentare der Begleitkreismitglieder



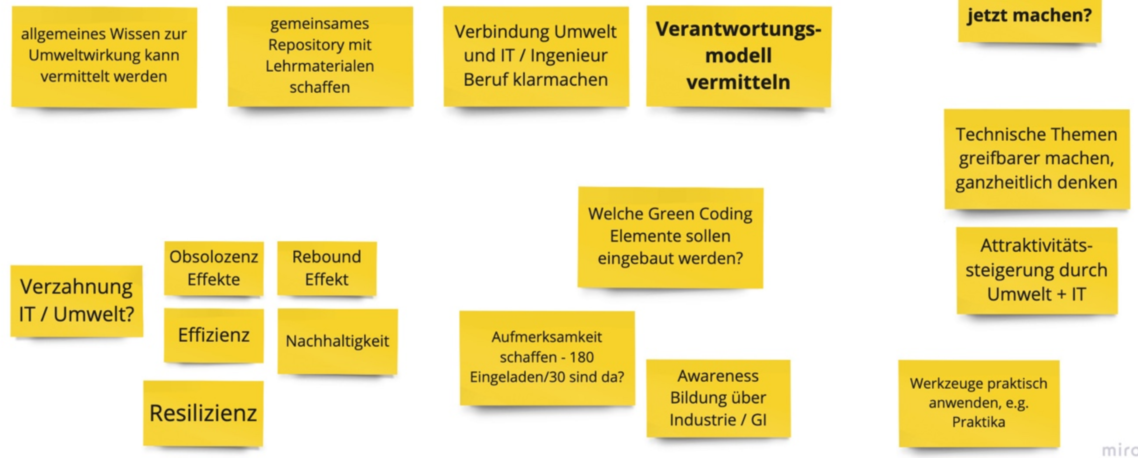
Quelle: eigene Darstellung, Sustainable Digital Infrastructure Alliance e.V.

Abbildung 56: Impuls #3 & Kommentare der Begleitkreismitglieder



Quelle: eigene Darstellung, Sustainable Digital Infrastructure Alliance e.V.

Abbildung 57: Impuls #4 & Kommentare der Begleitkreismitglieder



Quelle: eigene Darstellung, Sustainable Digital Infrastructure Alliance e.V.

B.3 Experten Workshop

In einem weiteren **Workshop**, der an die Expert*innen-Gemeinschaft und die Open-Source Gemeinschaft gerichtet war, konnte wertvolles Feedback gesammelt werden. Die Entscheidung, diesen virtuell und in englischer Sprache durchzuführen, ist der internationalen Zielgruppe des Vorhabens geschuldet. Es konnte dadurch eine breitere Gruppe zusammengebracht werden. Der Workshop hatte 23 Teilnehmende.

Der Workshop wurde sowohl interaktiv durch Break-Out-Räume gestaltet als auch durch eine Vorhabensvorstellung und eine Vorstellung des vorhergegangenen UFOPlan-Projekts „Sustainable Software Design“ (Kern et al. 2018) durch Jens Gröger vom Öko Institut inhaltlich ausgefüllt. Als Moderator wurde zusätzlich Chris Adams von der Green Web Foundation[122] als Unterstützung mit eingebunden.

Insbesondere die Diskussionen in den Break-Out-Räumen führten zu vielen neuen Einsichten, Ideen und Feedback. An jede diese Räume war eine konkrete Fragestellung gerichtet:

- Methodology - how can energy efficiency of software be measured? (moderiert von Jens Gröger).
- Application: What usage scenarios do we envision and where do we see the highest impact on potential software efficiency improvements? (moderiert von Chris Adams).

[122] https://greenwebfoundation.org

- Outlook: What is the current perception of energy and resource efficiency inside the software community? Where are boundaries towards sustainability? (moderiert von Max Schulze).

Abbildung 58: Ergebnisse Arbeitsgruppe „Methodology“

ROOM 1

Methodology

How can energy efficiency of software be measured?

Standardization of the interface of resource consumption <-> software

For mobile applications improving the battery lifetime is a regular habit

There is a lack of tools and methodologies to do this

On server side it is easy to measure the energy consumption...

...but it's getting complicated if it comes to the actual effects of the software

Installing Software in containers would be an approach to get an interface to these values

Libraries? Modular design of libraries would be fine and tests, which parts are really used (to leave out the unused parts).

There is an incentive to reduce resource consumption on mini-computers (rasberry pi) and old computers (>10a)

Data transfer (e.g. from mobile applications) can be measured in emulators

Public authorities but also a number of companies are interested in getting external impacts (CO2) from their cloud activities

Providers of cloud services (e.g. usage time of HPC) should also report about energy consumption of their services

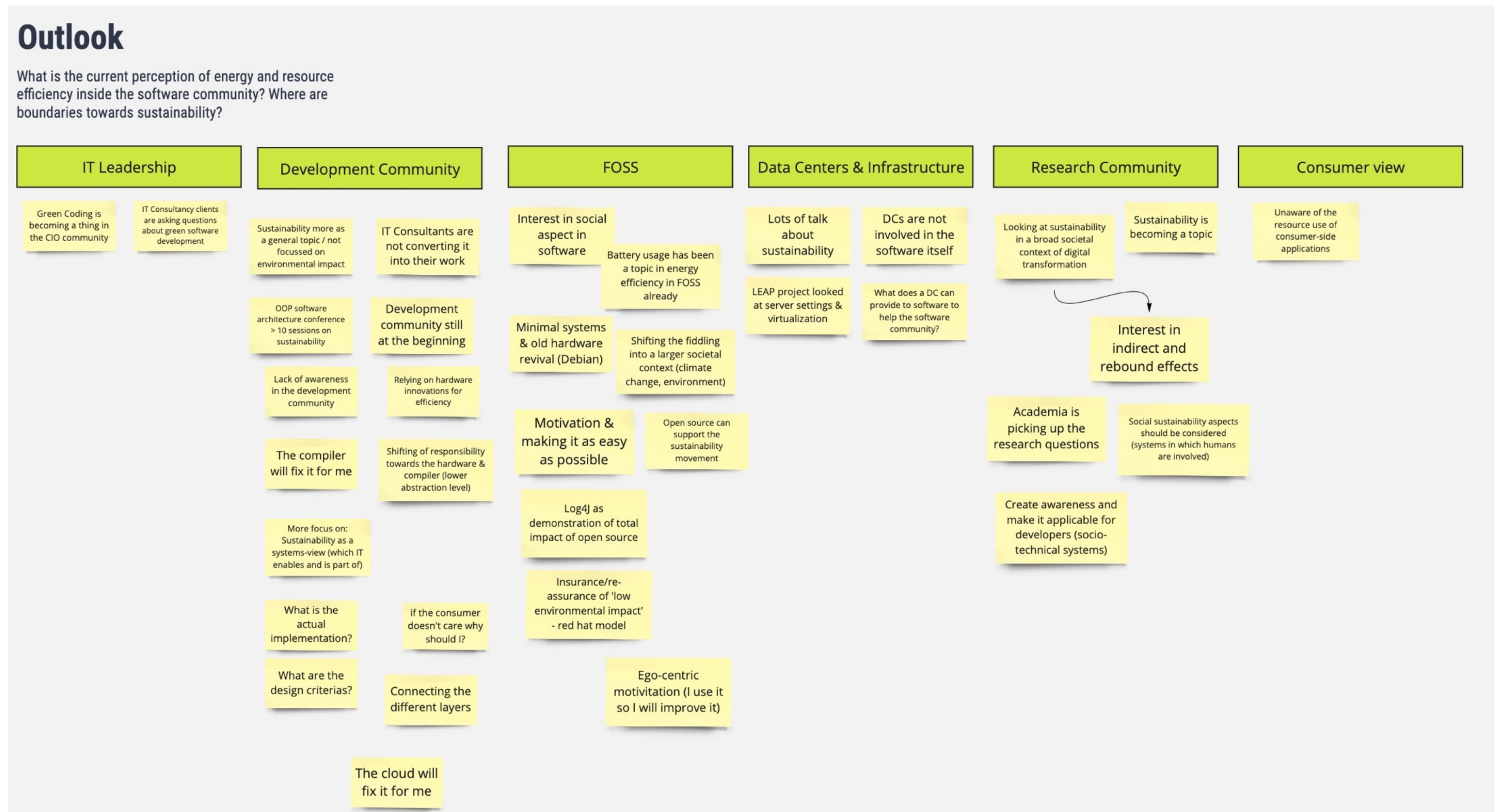
miro

Quelle: eigene Darstellung, Sustainable Digital Infrastructure Alliance e.V.

Abbildung 59: Ergebnisse Arbeitsgruppe „Application“

Quelle: eigene Darstellung, Sustainable Digital Infrastructure Alliance e.V.

Abbildung 60: Ergebnisse Arbeitsgruppe „Outlook“



Quelle: eigene Darstellung, Sustainable Digital Infrastructure Alliance e.V.

Die gesamten Ergebnisse und Präsentation des Workshops sind im Knowledge Hub der SDIA[123] zu finden.

B.4 Stakeholder Analyse

Für die hohe Verbreitung und für Feedback hinsichtlich der Anwendbarkeit der SoftAWERE Werkzeuge und Methodik ist es wichtig, die Open Source Gemeinschaft in das Projekt einzubinden. Um das erfolgreich durchzuführen und die richtigen Akteure zu identifizieren, hat der AN eine Stakeholder-Analyse durchgeführt. Die gesamte Analyse wurde im Knowledge Hub der SDIA in Englisch veröffentlicht[124].

Auf Basis dieser Analyse lassen sich zusammenfassend folgende Aussagen zur Open Source Gemeinschaft treffen:

- Stiftungen sind eines der zentralen Organisationsorgane der Open Source-Gemeinschaft und werden durch große Technologieunternehmen finanziert und unterstützt.
- Einzelne Kontributor*innen und Entwicklende haben weiterhin den größten Einfluss auf die Entwicklungen im Bereich Open Source-Software.
- Transparenz und offene Daten sind Kernprinzipien der Open Source-Gemeinschaft. Die Prinzipien passen zum SoftAWERE Projekt, und es ist wahrscheinlich, dass die Gemeinschaft Transparenz hinsichtlich der Energieverbräuche unterstützt.
- Trotz der starken Präsenz von kommerziellen Technologieunternehmen setzt die Gemeinschaft weiterhin auf intrinsische, nicht-monetäre Motivation, um weitreichende Vorteile für die Gesellschaft zu schaffen.
- Open Source-Komponenten und Bibliotheken implementieren bereits die Nachhaltigkeitsprinzipien der Wiederverwendbarkeit und Ressourcenschonung.

B.5 Hackathon 1

Der erste Hackathon wurde am 23. September 2022 in den Räumen des „Spielfelds“ in Berlin als hybride Veranstaltung ausgeführt. Übergreifendes Ziel des ersten Hackathons war die Wissensvermittlung und die Vermittlung der „Lessons learned“. Am Hackathon haben ca. 61 Personen teilgenommen, von denen 27 Vor-Ort waren und 34 sich per Video-Konferenz dazu geschaltet haben.

Der Hackathon wurde in drei Teilen aufgebaut:

- **Wissen vermitteln:** Vorträge verschiedener Personen sowohl aus dem Vorhabens-Team als auch aus den SDIA, Sustainable IT und Green Coding Communities
- **Anfassen:** In Arbeitsgruppen wurde mit dem Messlabor experimentiert und erste Versuche durchgeführt. Es wurde ausprobiert, viele Fragen gestellt und Grundlagen vermittelt

[123] SDIA Knowledge Hub – Workshop Dokumentation: https://knowledge.sdialliance.org/softawere/softawere-workshop-1

[124] SDIA Knowledge Hub, Stakeholder Analyse: https://knowledge.sdialliance.org/softawere/stakeholder-analysis

- **Feedback:** Um das Messsystem und Werkzeuge nutzerfreundlich zu gestalten und für die softwareentwickelnde Gemeinschaft brauchbar zu machen, wurde Feedback der Teilnehmer eingesammelt.

Alle Vorträge wurden aufgezeichnet und über den YouTube-Kanal der SDIA der Öffentlichkeit zur Verfügung gestellt[125]. Des Weiteren wurden die Aufzeichnungen über Social-Media-Kanäle verbreitet.

Das Feedback der Teilnehmer wurde in das Vorhaben integriert, und insbesondere die Dokumentation vereinfacht, und konkrete Beispiele über den Leitfaden verdeutlicht.

B.6 Hackathon 2

Der zweite Hackathon wurde in Berlin in den Räumen des „Space Shaks“ durchgeführt und hatte ca. 35 Teilnehmer. Ziel war es, den Teilnehmenden das weiterentwickelte Messsystem vorzustellen und in Arbeitsgruppen an verschiedenen Open-Source Projekten zu arbeiten, mit dem Ziel durch Experimentieren, Erproben und Diskutieren mögliche Verbesserungen zu identifizieren.

Um den Einstieg zu vereinfachen, hatte das Vorhaben-Team bereits Projekte vorbereitet, die bereits auf dem Messsystem aufgebaut worden sind und mit denen die anwesenden Entwickler*innen direkt einsteigen können.

Nach einer kurzen Einführung in das Messsystem haben sich Gruppen gebildet, die an verschiedenen Projekten gearbeitet haben:

- Eine Einsteigergruppe für Wissensvermittlung, durchgeführt von Jens Gröger mit einem Lasttreiber in Python implementiert
- Eine Arbeitsgruppe hat sich die Webseite der SDIA, welche auf Gatsby aufbaut und Open-Source ist, vorgenommen.
- Eine weitere Arbeitsgruppe hat sich bemüht, in einem Web-Server (Express) Verbesserungen vorzunehmen.

Es entstand eine positive Atmosphäre mit Austausch und einer gemeinsamen Lernkultur. Am Ende der Veranstaltung wurde bei einem gemeinsamen Abendessen noch weitere Vernetzung angeregt.

- Alle Teilnehmenden haben einstimmig am Ende des Hackathons festgestellt, dass die Optimierung hinsichtlich des Energieverbrauchs schwer ist und das entsprechende Wissen den meisten Software-Entwickler*innen fehlt.
- Open-Source Bibliotheken mit einer hohen Verbreitung und Nutzung (Beispiel Express[126]) sind bereits sehr stark optimiert, weitere Optimierungen erfordern ein sehr hohes Verständnis der Komplexität der Bibliothek oder Komponente bzw. setzen ein sehr spezialisiertes Wissen voraus, wie sich der Energieverbrauch noch weiter optimieren lässt.
- Bei einer Anwendung (Beispiel: sdia-website), ließen sich einfache Einsparungen schnell realisieren (z.B., in den unnötigen Bibliotheken entfernt, alle Abhängigkeiten auf den

[125] YouTube Playlist: https://www.youtube.com/playlist?list=PLOL05v40mafadtzWhGgXiN-jknvq20dWg

[126] ExpressJS ist eine Web-Server Implementierung für NodeJS: https://expressjs.com/

neusten Stand gebracht, und allgemeine Komplexität reduziert wurden). Einsparung am Beispiel der SDIA-Website mit einem Aufwand von 3-4 Stunden: 30%. Alle Messergebnisse finden sich in der NocoDB des Projekts[127].

Die Teilnehmenden beider Veranstaltungen haben sich dafür ausgesprochen, die Veranstaltungen regelmäßig zu wiederholen, um den Wissensstand zu erhöhen, Raum und Zeit zu schaffen, um über Energieverbrauch und Maßnahmen zur Reduktion nachzudenken und kontinuierlich über Verbesserungen in einer Gemeinschaft nachzudenken, auszuprobieren und zu dokumentieren.

Die Bilder der Veranstaltung hat die SDIA auf Ihrem Blog veröffentlicht[128].. Die Ergebnisse sind in der NocoDB des Projekts veröffentlicht worden.[129]

[127] https://sdia.io/sawe-hackathon2-results

[128] https://www.linkedin.com/posts/sustainable-digital-infrastructure-alliance_hackathon-gitlab-opensource-activity-7070019842472304642-hi9b/

[129] https://sdia.io/sawe-hackathon2-results

C Ergebnisse

C.1 Leitfaden für Energieeffiziente Software-Entwicklung

Der Leitfaden wurde in Englisch im Knowledge Hub der SDIA[130] veröffentlicht. Der Text ist im Folgenden aufgeführt.

Principles of Sustainable Software Design

The principles capture the essence of the sustainability criteria for software engineering defined[131] by the team at Umweltcampus Birkenfeld, the University of Zurich and the Oeko Institute as part of the Green Software Engineering research that was funded by the German Environmental Agency. Further we review & include the criteria defined by the German environmental label 'Blue Angel' (Resources and Energy-Efficient Software Products - DE-UZ 215)

Key concepts

- Unix philosophy[132]
- Minimalism in computing[133]
- Code bloating[134]
- Software bloating[135]
- Feature creep[136]

One overarching principle: Minimize resource usage

When looking at the core philosophies defined by Eric Raymond for Unix, there is one principle that should be revised given our global, society challenge of limiting resource usage, carbon emissions and overall moving to sustainable consumption & production across all sectors.

Value developer time over machine time → Eric Raymond's rules[137]

Machine time, or as we consider them - digital resource primitives (compute, memory, storage, network) - are not infinite, as the term cloud computing might suggest. They are finite, and ties to physical consumption of natural resources - since the end of Moore's Law - in an almost linear relationship. If an application consumes more digital resource primitives, it consumes more hardware resources and though that energy and natural resources.

In simpler terms: Every CPU cycle, every GB of memory, GB of storage and network has an impact on the environment.

[130] "Principles for Sustainable Software Design im SDIA Knowledge Hub": https://knowledge.sdialliance.org/softawere/principles-for-sustainable-software-design

[131] siehe https://www.umwelt-campus.de/forschung/projekte/green-software-engineering/kriterienkatalog/einleitung, abgerufen am 14. Februar 2024

[132] siehe https://en.wikipedia.org/wiki/Unix_philosophy, abgerufen am 14. Februar 2024

[133] siehe https://en.wikipedia.org/wiki/Minimalism_(computing), abgerufen am 14. Februar 2024

[134] siehe https://en.wikipedia.org/wiki/Code_bloat, abgerufen am 14. Februar 2024

[135] siehe https://en.wikipedia.org/wiki/Software_bloat, abgerufen am 14. Februar 2024

[136] siehe https://en.wikipedia.org/wiki/Feature_creep, abgerufen am 14. Februar 2024

[137] Wikipedia Artikel: https://en.wikipedia.org/wiki/Unix_philosophy#Eric_Raymond's_17_Unix_Rules, abgerufen am 18. September 2023

Hence the overarching principle for every software developer, product manager, designer or architect should be to minimize resource use of an application.

Recommendations

Our recommendations are building on the existing studies referenced below. However, we divide them into three groups:

- Functional recommendations, such as the ability to turn off functionalities, for roles/stakeholders such as user interface designers, user experience teams, product management, business analysts, etc.
- Architecture recommendations, such as designing a modular technical architecture, for software architects, technical management, etc.
- Development recommendations, such as avoiding blocks on future resources, for software developers, site reliability engineers, and operational engineering teams.

With this separation we want to ensure that the recommendations are relevant for the different roles who are involved in the making of a Digital Product. If a person occupies multiple roles (e.g. in a small startup) where product management and architecture might be combined, you can freely combine them.

Further, we need to segment the recommendations for different types of software applications. The categories have been defined in previous studies[138]. They are **numbered and referenced in the headline** of each recommendation:

3. Local or client-side application (DE: lokale Anwendung)
4. Client-side application with remote data storage (DE: Anwendung mit entfernter Datenhaltung)
5. Client-side application with remote processing (DE: Anwendung mit entfernter Verarbeitung)
6. Server-side application (DE: Serverdienst)

Lastly, we have ordered the recommendations **by importance.**

Functional Recommendations

Define requirements to minimize the hardware requirements (1, 2, 3, 4)

Defining a clear requirement to minimize the amount of hardware resources used by the applications has many benefits: it enables hardware to be used for a longer time, it is often also increasing performance & the responsiveness of the application, and it allows the application to run on many more devices.

Many times, the user is not using a state-of-the-art machine like the ones the development and product teams are using, but rather older, much less powerful machines. By designing for minimum hardware utilization, most of these users benefits while having also an overall reduced environmental impact.

From a product management & design perspective, the question to the development teams should be: How will this increase our hardware resource usage? Or: How many hardware resources will this feature use?

[138] Siehe https://www.umwelt-campus.de/forschung/projekte/green-software-engineering/kriterienkatalog/einleitung

Instead of doing performance optimization later, define minimal resource usage as a requirement for each functionality from the start.

Design the application so that the user has control and visibility over the resource-consumption from their usage (1, 2, 3, 4)

It should be visible to the user which function (e.g., performing an export of an image) triggers how much resource consumption, both locally and on any involved server-side (remote) systems. The user should hence be enabled to choose not to perform or use a function due to its potentially excessive resource usage.

Ability for the user to turn off not-needed or unused functions of the software (1, 2, 3, 4)

It should be possible for a user to turn off functionalities of the software that are not needed. These 'feature toggles' should be connected to the underlying code & functionality, as this is an existing technique in software engineering (see Feature Flags by Pete Hodgson). This means that when a user disables a feature, it should result in less code being executed in total, and thus should translate in a reduction in resource usage.

Support delay tolerance and slow connectivity (2)

When remote data sources are used, a slow or delayed connectivity should be supported by design. This not only leads to a better user experience, but it enables the use of the application in low- or limited-bandwidth settings as well as reduces the network capacity use. This must be considered both in the user interface (to not block the entire functionality of the application from a delayed data connection) as well as in the technical architecture (e.g., pre-loading critical data when a connection is available, local caching, etc.)

Support an offline version of the application (2, 3)

As very common in mobile applications, any software application that works on a client (mobile or desktop) should support basic functionality when there is no connectivity. The application's enhanced featured (e.g., that are resource intensive and therefore require remote processing or that depend on accessing data that cannot be stored offline, e.g., from real-time data sources) can be disabled, but functionality that is technically possible to be provided offline should be present. This increases usability, leads to reduced network usage through local caching and less obsolescence, e.g., if the software vendor does not support the remote services anymore, the application continues to provide basic functionality.

Ability to completely remove the application (1, 2, 3, 4)

It should be possible to remove the application in its entirety from a client (e.g. a desktop computer) with residual data removed (e.g. caches, log files, user data). In the case of remote storage or processing - any associated data with the user (or users in a multi-tenant environment) should be completely wiped to avoid stranded data which uses storage resources.

Turn off features that are not supported on an older hardware (1, 2, 3, 4)

Instead of making the entire application unavailable on older operating systems and client environments, rather opt for disabling features, e.g. using feature toggles. The application then may have less up-to-date functionality, but continues to perform the functionality the user has originally installed the application for on the client.

The other way to look at this is to not stop supporting a device (e.g. laptop or phone model), but rather unlock features only for capable hardware; this also expand the market potential of the application and helps expand the lifetime of hardware in general.

A well-known design principle is responsive design, e.g. responding to the screen size of the user when delivering a user interface. The same principle can be applied to the available hardware resources and functionalities. The application can simply offer features that respond to what is available on the client and hide the ones that cannot be run in the client environment.

Delivery of the software only via internet (1, 2, 3, 4)

Modern software applications should not be shipped on physical media (CD, DVD, or any other form of physical storage) but rather be transferred through networks/internet.

Consider removing outdated user data or suggest to the user to remove data (1, 2, 3, 4)

Many applications deal with large amounts of data which are never being accessed, because the user has stopped using the product (but not deleted the account), or because the data is long obsolete. No company benefits from storing the unused information of its users. Therefore, implement functionalities that help users outdated data, give indicators and suggestion where data can be removed, and make it simple to execute a recommendation.

Where possible also delete, archive, or delete data that is obsolete and does not require user intervention or permission. Further, archive information that is infrequently accessed and integrate a 'recovery from archive' delay-interface into the user interactions. Data storage is not unlimited and free, it has a large environmental impact and should not be wasted.

Consider monitoring feature creep (1, 2, 3, 4)

Many software applications suffer growing more and more features, of which each bloats the software more, which leads to more hardware resources being used over time, even though there might be no new benefit to the user. Especially feature creep without feature toggles, e.g., letting user's opt-in to new features if they need them, rather than continuously expanding functionalities, creates enormous pain for users who constantly have to adapt and leads to ever increasing resource-hunger of applications.

Consider monitoring your feature development and prioritization process either monitor it (e.g., performing some kind of cost-benefit analysis on each feature before developing it) or having an equal, feature-removal process to counterbalance the constant adding of new features.

Architecture Recommendations

Micro-services that represent application functionalities & core features (4)

The movement towards micro-services has been well underway in the software architecture ecosystem. The benefits are wide, to name a few: Independent scaling of components, separation of concerns and better collaboration between large development teams.

A micro-service architecture can have a big impact In terms of resource-usage especially in conjunction with a functional requirement/design that enables the user to turn off/on functionalities. If the micro-service architecture breaks up core services (such as storage, simple utility functions and other support services) and functionalities, when users all disable a functionality, the respective services can be turned off without impacting the overall application.

Further, using container-based orchestration, this scaling up and down can be directly reflected in the underlying infrastructure and lead to a measurable reduction in hardware resource usage.

Consider web-based user-interfaces that are operating-system independent (1, 2, 3)

Browser-based or web-based user interfaces have become more common, especially as they are operating-system independent. However, they often do not have access to the same the resource optimization opportunities of native applications that can directly interact with the operating

system. The resource optimization of a web-based application is mostly managed by the browser environment.

However, for most digital products consider a lightweight, web-based interface first, reducing the hardware resource required on the client-computer to a minimum (e.g. must be enough to run a browser). This does not apply for CLI or command-line based tools (hence we refer to digital products and not applications in this case).

Before building native client-applications perform a proper analysis on performance & resource usage (1, 2, 3)

When building a native client application, consider the actual improvement in performance against a web-based client, especially considering maintainability and interoperability. Also consider how the potential additional hardware resources on the client may limit the types of devices that can be used to run the application.

Consider decoupling frontend and backend

Most modern applications have already embraced breaking up applications between the user interface (frontend) and the data storage & processing layer (backend). This design pattern in conjunction with separate micro-services for each API endpoint, can lead to a lot of resource savings, as APIs can be scaled independently based on the amount of requests that a single endpoint receives from the frontend application.

Further, it enables single features to be turned off in the frontend, and equally be shut off on the backend, without affecting the rest of the application.

Lastly the frontend is only running when someone is using the application and can be delivered statically (see static delivery).

Consider delivering a static frontend to and to shift most of processing and storage to server-side micro-services (1, 2, 3)

When web-based frontends are statically rendered at build time, the result is a fixed bundle of HTML, CSS and JavaScript, all static files that can be rendered within the browser (client-side) without a roundtrip to a server which needs to re-render the same page for each user.

Static frontends, further have the advantage that they can be served via content delivery networks - efficient caching systems that can deliver millions of static frontend applications with mostly storage & network resources. This delivery mechanism is much more compute efficient - for each client the compilation of the page only happens once, and is then delivered to each client where it’s rendered on the client-side.

When building static web-based frontends, the paradigm of shifting heavy computing and data storage to the server-side is more or less enforced, hence removing potential high hardware resource requirements.

Consider operating micro-services as function-like services that are switched off when not used (4)

Functions are the the next computing paradigm that is gaining traction - breaking down each functional unit of code into a micro-application that is executed only when someone is ‘calling’ it. In principle this is very resource efficient, e.g. an API which is never called, is never loaded into memory or allocated computing time.

You should consider if you can turn very basic, shared services, such as sending an email, processing/compressing an image, preparing a data export, into functions that are only running when they are triggered.

Consider moving as many operations as possible into asynchronous background processes that can be delayed or moved (4)

This is a common practice in the software development world, but it's worth mentioning that all tasks that can be postponed in time or that can be shifted to a different location should be designed as a background task.

Background task should use a queue system that has a scheduler that can be optimized or extended to include considerations for the availability of renewable energy, shifting to a different data center or simply delaying the task.

You should also consider if you can move the queue and background processing to low-grade machines, or in Cloud environments to use spot instances for these tasks.

Support incremental, over the air updates that do not replace the entire application on each update-cycle (1, 2, 3)

When delivering updates to client-side applications, deliver them incrementally, e.g. do not require to replace all of the program code every time an update is delivered.

Use a full program code update only for 'reset' or self-repair functionalities.

Support public APIs on all main functionalities for the programmatic use & as well as data export (2, 4)

Making the data within the application technically accessible, is a key feature to enable composability between applications as well as the exporting of data in a technical format (e.g. to migrate into another application).

These public APIs should be well documented, and accessible to the user of the application.

Support data interoperability or export into common formats (2, 4)

Either in conjunction with a Public API or as a standalone functionality, support the ability to export all of the user-data in a standardized and machine-readable format, e.g. a ZIP file of JSON documents, YAML or XML. Enable the exporting of all the information the user has inputted to the system.

Consider measuring the total resource use as part of your integration tests to provide insights for product management and development on areas of improvement (1, 2, 3, 4)

When running your integration tests, e.g., testing that the functional requirements are met, consider also recording the total hardware resource usage of each functionality and reporting these metrics as part of your continuous integration process, or ticketing system, so that the organization that defines the functional requirements is aware of the resource costs of the functionality.

It should also help to inform your development teams to consider refactoring or code optimization and should overall also help improve application performance (less resource usage most likely will also convert into a performance gain).

The SoftAWERE tooling can support you to do those measurements.

Development Recommendations

Remove unnecessary libraries and favor libraries that provide required only functionality needed (1, 2, 3, 4)

Integrate the removal of unused libraries into your practice and workflows as it can reduce the size of the overall application and lead to an overall decrease in resource-usage.

Further consider integrating libraries that deliver a limited functionality, that meets current requirements rather than 'super libraries' that deliver functionality that may be useful in the future but are not needed right now. All of these bloat the size of the codebase and can (depending on the compiler, interpreter and programming language) also lead to increased runtime resource usage.

Minimize the hardware requirements for any client-side application (1, 2, 3)

Consider making hardware requirements as low as possible to support legacy hardware equipment as well. Today's software applications (especially client-side) are often driving constant, unnecessary hardware requirements, simply because a new iteration of an application requires more resources. This must avoided and maximizing the lifetime of the hardware should be paramount as it significantly contributes to reducing the environmental burden created by IT equipment and applications.

Furthermore, low hardware requirements on the client-side also lead to a better user experience and overall performance and expands the potential user-base of any application.

Lower hardware requirements can also be accomplished by delivering updates of applications with new functionality default 'switched off' and only loading the application code of a new feature when the user has explicitly switched it 'on'. This way a user can receive updates, without those updating driving the obsolescence of the equipment they are using.

Report the resource-usage of a single operation and make it visible to the client-side (1, 2, 3, 4)

In order to raise the awareness of users towards the resource usage of their actions within an application, or of the functionalities they are using as well as their overall usage behavior, it is paramount to make the underlying resource-usage transparent.

Each action of a user should have a clearly understandable label that informs about the energy usage, carbon emissions from the energy & remote infrastructure that was used to perform processing or store data.

When building a server-side application, e.g. an API, consider embedding the resource-usage information into the HTTP header or any other transport header, so it can be displayed in the user interface.

The SDIA's digital environmental footprint framework and open source tools can help make this information visible in an application.

Turn off staging & test environments (1, 2, 3, 4)

Unless your development is spread across many unrelated time-zones (e.g., West Coast of the United States and Barcelona, Spain), consider turning off development environments outside of office hours or at least at night, e.g. 22:00 - 04:00. Even the maybe minor resource usage of these systems creates waste if they are not utilized.

If possible, run these environments in containerized or at least virtualized, shared resource environments so that if these test & staging environments receive limited load or stress, the underlying hypervisor & scheduler can re-allocate the unused resources.

Do not over-provision these systems either, but rather constrain them to ensure that resource-usage is always minimized in new feature developments.

Don't run builds on each minor code-change (1, 2, 3, 4)

Most development environments are configured to trigger a new build or test-run on every file change. Even if a simple README file is changed, a new build maybe triggered. Make sure to

configure your systems so that new builds and test-runs are only triggered when a real code change has happened.

Consider switching continuous integration & delivery to a semi-manual process. Keep in mind that each run is using energy and resources, which are neither infinite nor do they have no environmental impact. The opposite is the case and resource usage should be minimized, also during the development process.

Shift processing and storage to server-side if it helps reduce client-side obsoleteness (1, 2, 3, 4)

When introducing new functionalities that would lead to a drastic increase of client-side resource usage or a change in the minimum hardware requirements, consider migrating the processing or storage into a remote-environment.

Driving the obsoleteness of client-side hardware should be avoided whenever possible, as hardware lifetime extensions can lead to significant environmental benefits and also ensure the application can be run on a much wider range of hardware.

Ensure restart-ability, and up- and down-scaling (4)

To ensure maximum optimization potential, e.g. in virtualized or containerized infrastructure environments, ensure that the application can be started/stopped without any intervention and performs necessary health-checks automatically, repairing itself (e.g. clearing caches and triggering another restart itself).

Provide proper signaling when the application is corrupted, a restart would interrupt a task or anything else so that schedulers and optimizers can migrate & optimize the application.

Try to avoid linking the application to any requirements on the physical infrastructure and allow it to be fully virtualized.

Apply 'mobile-first' principle to 'oldest machine first' (1, 2, 3, 4)

When it comes to developing web-based frontend applications, many software developers & vendors have opted for a mobile-first approach, meaning that the user interface will first be developed for the smallest, least capable screen.

The same principle can be applied to developing client-side applications as well as server-side applications, working 'bottom-up', e.g., assuming the smallest possible available resource or a old device as the baseline for development. As an example, what would an application look like that has to work Intel Celeron with 2 GB memory and a 80 GB HDD?

This principle can be found a lot across the linux community and has enabled very high-performing and resource efficient operating systems & tools, which are powering many server-environments today, because of their 'low overhead'. It's time to bring these principles back into application development as well.

Do not block resources that are not being used (1, 2, 3, 4)

It has become common practice for IT departments and developers to request hardware resources 'that might be needed when it scales' or that are provisioned because the actual load/stress on the system is still unknown. In some cases, this is warranted, however, the overall architecture should then quickly be resized to match the actual load & stress of the application.

The same goes for the application itself, e.g., reserving all available memory even though it's not needed removes any possibility of another application or process to use the memory while its not in use. Most hypervisors & resource provisioning systems are more than capable to reallocate the required resources back to the application when they are needed. Blocking

resources that could be used (even just temporarily) by another application or system, should be avoided.

Zero work should equal near-zero resource-use (except background processes) (1, 2, 3, 4)

When an application does not perform any action, it should consume as little hardware resources as possible. In an idling state (except background processes that might need to run to clean up, garbage collection, etc.), the hardware resources should be freed up again (e.g., memory) and the usage of CPU resources should be close to none.

Avoid code & software bloat (1, 2, 3, 4)

Resource usage should always be considered with each iteration of the software application. Often, when new features are introduced or 'speed', developer productivity is valued over efficiency (see Wikipedia excerpt below), application resource usage can grow quickly.

Hence each cycle of development (new feature, refactoring) should re-evaluate the total resource usage of the application and each respective feature. The tools developed by the SoftAWERE project can support this transparency during the development process.

Actual (measurable) bloat can occur due to de-emphasising algorithmic efficiency in favour of other concerns like developer productivity, or possibly through the introduction of new layers of abstraction like a virtual machine or other scripting engine for the purposes of convenience when developer constraints are reduced. The perception of improved developer productivity, in the case of practising development within virtual machine environments, comes from the developers no longer taking resource constraints and usage into consideration during design and development; this allows the product to be completed faster but it results in increases to the end user's hardware requirements to compensate. (Wikipedia)

Consider minimalism as your development practice (1, 2, 3, 4)

Many developers have already spoken about embracing Minimalism as a practice in software engineering. Consider reviewing theMinifesto[139], the Wikipedia page for an overview on Minimalism in computing[140] or read some of the articles below:

- My Minimalist Approach to Software Engineering by Benjamin Tanone[141]
- A Minimalist Approach to Software by Ammar Hakim[142]
- What Digital Minimalism Means to me as a Software Developer by Jiana Javier[143]

Remaining questions

- Should we actually still build native applications or do everything in the browser and rather assume better connectivity (so that resources can be used centrally and hence more efficiently)
- Chrome vs. Native App - what's the difference with the same scenario? Word vs. Google Docs

[139] siehe http://minifesto.org, abgerufen am 14.02.2024

[140] siehe https://en.wikipedia.org/wiki/Minimalism_(computing), abgerufen am 14.02.2024

[141] siehe https://medium.com/@benjamin.tanone/my-minimalist-approach-to-software-engineering-5bc743ecb295, abgerufen am 14.02.2024

[142] siehe https://ammar-hakim.org/minimalist-software.html, abgerufen am 14.02.2024

[143] siehe https://jianajavier.github.io/digital_minimalism/, abgerufen am 14.02.2024

- The studies before are mostly applicable for desktop software applications what about software-as-a-service?
- Sustainability vs. resource efficiency (principles, e.g. open source vs. techniques)

C.2 Auswahl an Untersuchungsgegenständen

Für den Test-Aufbau des Vorhabens hat das Forschungsteam sich dafür entschieden, die Unit-Tests der einzelnen Bibliotheken als Nutzungsszenario zu verwenden. So kann der Stromverbrauch für jeden Testlauf aufgezeichnet werden. Eine Test-Suite deckt in vielen Fällen mehr als 80% des Codes der Bibliotheken ab, sodass davon ausgegangen werden kann, dass eine Ausführung der Test-Suite mit der Ausführung des Codes der Bibliothek vergleichbar ist.

Um einen eher zufälligen Ansatz bei der Auswahl von Softwarekomponenten zu gewährleisten, wurde die Liste der beliebtesten Bibliotheken für jede Programmiersprache von Github verwendet. Github repräsentiert die Mehrheit der Open-Source-Code-Repositories weltweit, Das Beliebtheitsranking der Projekte auf Github kann als repräsentativ angesehen werden. Dies hat den zusätzlichen Nebeneffekt, dass die Messungen des Energieverbrauchs für eine sehr große Gruppe von Entwickelnden relevant sind, die diese Komponenten bereits verwenden.

Außerdem haben diese beliebten Komponenten oft eine große Gemeinschaft von Mitwirkenden, die aktiv am Softwarecode der Komponente arbeiten. Das bedeutet, dass rasche Verbesserungen möglich sind und es den Weg für viele potenzielle Nutzende von SoftAWERE ebnet. Diese Mitwirkenden können mit Hilfe von SoftAWERE den Gesamtenergieverbrauch der Komponenten senken, was sich wiederum auf Tausende von Softwareanwendungen auswirken könnte, die diese Komponente einsetzen.

Aus technischer Sicht müssen noch einige weitere Bedingungen erfüllt werden, damit die Tests im Labor ausgeführt werden können und unsere Experimente leicht wiederholbar sind:

- **Eine automatisierte Testabdeckung ist vorhanden**: Die Komponente muss mindestens über funktionale Tests verfügen, die ausgeführt werden können. Ein Großteil der beliebten Komponenten verfügt über Tests.
- **Open-Source-Lizenz:** Diese Lizenz ist erforderlich, da es sonst für das Forschungsteam schwierig ist, Ergebnisse zu den untersuchten Softwarekomponenten zu veröffentlichen.
- **Keine Benutzeroberfläche:** Die untersuchte Komponente sollte keine Nutzeroberfläche enthalten, sondern eine „einfache" Komponente oder Bibliotheken sein, nicht eine Anwendung selbst.
- **Die Tests sind erfolgreich:** Es wurde bei der Auswahl darauf geachtet, dass die meisten der ausgewählten Komponenten eine Test-Suite haben, die nicht nur existiert, sondern auch ohne Fehler ausgeführt wird.
- **Container-basierte Testsuite**: Optional -um die Wiederholbarkeit der Tests zu erhöhen, ist es einfacher, wenn die Komponente die Anweisungen enthält, wie die Entwicklungsumgebung einzurichten ist, damit die Tests ausgeführt werden können (z. B. alle Abhängigkeiten oder Umgebungskonfigurationen). Dies ist eine gute Praxis, die jedoch nicht so häufig vorzufinden ist und daher kein Ausschlusskriterium darstellt.
- **Aktualität:** Die Komponente sollte eine aktive Gemeinschaft von Mitwirkenden haben. Bei der Auswahl wurde darauf geachtet, dass es in den letzten 3 Monaten neue Beiträge zu der

Komponente gegeben hat (was sich bei den populären Komponenten als unproblematisch herausgestellt hat).

Tabelle 9: Übersicht Untersuchungsgestände – Python

Quelloffene Bibliotheken die in Python programmiert worden sind.

Bibliothek	Sterne	Nutzer	Tests verfügbar?	Alle Tests sind erfolgreich?	Letzte Aktualisierung	Keine Oberfläche	Open Source?	Container-basierte Tests
Django	61,900	825,000	JA	JA	~ 24 Stunden	JA	JA	NEIN
Flask	57,700	929,000	JA	NEIN	~ 7 Tage	JA	JA	NEIN
Transformers	57,200	20,400	JA	NEIN	~ 11 Stunden	JA	JA	JA
Ansible	51,400	19,300	JA	NEIN	~ 12 Stunden	JA	JA	JA
Pytorch	53,400	111,200	JA	JA	~ 5 Stunden	JA	JA	JA
Keras	53,800		JA	JA	~ 2 Stunden	JA	JA	JA
Httpie	53,400	9,800	JA	JA	~ 7 Tage	JA	JA	NEIN
Requests	46,700	1,200,000	JA	JA	~ 4 Tage	JA	JA	NEIN
Apache Superset	44,000		JA	NEIN	~ 12 Stunden	NEIN	JA	JA
Face_Recognition	42,900		JA	JA	~ 6 Monate	JA	JA	JA
Scrapy	42,500	27,400	JA	NEIN	~ 11 Tage	JA	JA	NEIN

Tabelle 10: Übersicht Untersuchungsgestände – Java

Quelloffene Bibliotheken die in Java programmiert worden sind.

Bibliothek	Sterne	Nutzer	Tests verfügbar?	Alle Tests sind erfolgreich?	Letzte Aktualisierung	Keine Oberfläche	Open Source?	Container-basierte Tests
spring-boot	59,200		JA	JA	~ 14 Stunden	JA	JA	NEIN
elasticsearch	58,200		JA	JA	~ 6 Stunden	JA	JA	NEIN
RxJava	45,700		JA	JA	~ 8 Tage	JA	JA	NEIN
guava	43,600	266,000	JA	JA	~ 2 Tage	JA	JA	NEIN
retrofitt	39,400	7,800	JA	JA	~ 24 Tage	JA	JA	NEIN
jadx	28,900		JA	JA	~ 2 Tage	NEIN	JA	NEIN
dbeaver	24,400		JA	JA	~ 4 Stunden	NEIN	JA	NEIN
jenkins	18,400		JA	JA	~ 4 Stunden	NEIN	JA	JA
redisson	18,200		JA	JA	~ 6 Stunden	JA	JA	NEIN
flink	18,100		JA	NEIN	~ 3 Stunden	JA	JA	NEIN
mockito	12,600		JA	JA	~ 2 Tage	JA	JA	NEIN

Tabelle 11: Übersicht Untersuchungsgestände – C

Quelloffene Bibliotheken die in C programmiert worden sind.

Bibliothek	Sterne	Nutzer	Tests verfügbar?	Alle Tests sind erfolgreich?	Letzte Aktualisierung	Keine Oberfläche	Open Source?	Container-basierte Tests
curl	23,600		JA	NEIN	~ 2 Tage	JA	JA	NEIN
radare2	15,800		JA	NEIN	~ 16 Stunden	JA	JA	NEIN
libsodium	9,900		JA	JA	~ 2 Tage	JA	JA	NEIN
glfw	8,600		JA	JA	~ 31 Dec, 2021	JA	JA	NEIN
vlc	8,500		JA	NEIN	~ 22 Stunden	JA	JA	NEIN
libevent	8,300		JA	NEIN	~ 23 Tage	JA	JA	NEIN
libgit2	8,300		JA	JA	~ 16 Stunden	JA	JA	JA
libvips	6,300		JA	JA	~ 4 Tage	JA	JA	NEIN
curl	23,600		JA	NEIN	~ 2 Tage	JA	JA	NEIN

Tabelle 12: Übersicht Untersuchungsgestände – C++

Quelloffene Bibliotheken die in C++ programmiert worden sind.

Bibliothek	Sterne	Nutzer	Tests verfügbar?	Alle Tests sind erfolgreich?	Letzte Aktualisierung	Keine Oberfläche	Open Source?	Container-basierte Tests
nghttp2	3,900		JA	JA	~ 5 Stunden	JA	JA	NEIN
arrayfire	3,700		JA	NEIN	~ 19 Stunden	JA	JA	NEIN
harfbuzz	2,300		JA	JA	~ 2 Tage	JA	JA	NEIN
c-ares	1,400		JA	JA	~ 21 Dec, 2021	JA	JA	NEIN
fann	1,300		JA	NEIN	~ 14 Mar, 2021	JA	JA	NEIN

Tabelle 13: Übersicht Untersuchungsgestände – JavaScript

Quelloffene Bibliotheken die in JavaScript programmiert worden sind.

Bibliothek	Sterne	Nutzer	Tests verfügbar?	Alle Tests sind erfolgreich?	Letzte Aktualisierung	Keine Oberfläche	Open Source?	Container-basierte Tests
vue	193,000		JA	JA	~ 21 Tage	JA	JA	NEIN
react	182,000	8,800,000	JA	JA	~ 12 Minuten	JA	JA	JA
axios	90,900	5,600,000	JA	JA	~ 2 Tage	JA	JA	NEIN
angular	79,400	2,100,000	JA	JA	~ 9 Stunden	JA	JA	JA
three,js	78,700		JA	JA	~ 14 Stunden	JA	JA	NEIN
material-ui	75,200	792,000	JA	JA	~ 12 Stunden	JA	JA	NEIN
Chart,js	56,100	472,000	JA	JA	~ 3 Tage	JA	JA	NEIN
express	55,800	12,400,000	JA	JA	~ 12 Stunden	JA	JA	NEIN
vue	193,000		JA	JA	~ 21 Tage	JA	JA	NEIN

Tabelle 14: Übersicht Untersuchungsgestände – Go Lang

Quelloffene Bibliotheken die in Go Lang programmiert worden sind.

Bibliothek	Sterne	Nutzer	Tests verfügbar?	Alle Tests sind erfolgreich?	Letzte Aktualisierung	Keine Oberfläche	Open Source?	Container-basierte Tests
kubernetes	85,200		JA	NEIN	~ 15 Minuten	JA	JA	JA
moby	62,200		JA	NEIN	~ 15 Minuten	JA	JA	JA
hugo	56,900		JA	JA	~ 3 Tage	JA	JA	JA
gin	55,300	53,300	JA	JA	~ 19 Stunden	JA	JA	NEIN
tidb	30,300		JA	NEIN	~ 3 Stunden	JA	JA	NEIN

C.3 Umrechnungsformel Digital Ressource nach Energie

Die aktuelle Version der Formeln wird im Knowledge Hub der SDIA[144] bereitgestellt.

C.4 Ausgabe des Messskriptes get_metrics.py (Beispiel)

```
total_operational_emissions :

    value = 7.6e-05

    description = GHG emissions related to usage, from start_time to
end_time.

    type = gauge

    unit = kg CO2eq

    long_unit = kilograms CO2 equivalent

total_operational_abiotic_resources_depletion :

    value = 5.6e-12

    description = Abiotic Resources Depletion (minerals & metals, ADPe)
due to the usage phase.

    type = gauge

    unit = kgSbeq

    long_unit = kilograms Antimony equivalent

total_operational_primary_energy_consumed :

    value = 0.0014

    description = Primary Energy consumed due to the usage phase.

    type = gauge

    unit = MJ

    long_unit = Mega Joules

calculated_emissions :

    value = 0.00010282648401826485

    description = Total Green House Gaz emissions calculated for
manufacturing and usage phases, between start_time and end_time
```

[144] https://knowledge.sdialliance.org/digital-environmental-footprint

```
    type = gauge

    unit = kg CO2eq

    long_unit = kilograms CO2 equivalent

start_time :

    value = 1682065095.0

    description = Start time for the evaluation, in timestamp format
(seconds since 1970)

    type = counter

    unit = s

    long_unit = seconds

end_time :

    value = 1682065104.0

    description = End time for the evaluation, in timestamp format
(seconds since 1970)

    type = counter

    unit = s

    long_unit = seconds

embedded_emissions :

    value = 2.682648401826484e-05

    description = Embedded carbon emissions (manufacturing phase)

    type = gauge

    unit = kg CO2eq

    long_unit = kilograms CO2 equivalent

embedded_abiotic_resources_depletion :

    value = 5.422374429223744e-09

    description = Embedded abiotic ressources consumed (manufacturing
phase)

    type = gauge

    unit = kg Sbeq

    long_unit = kilograms ADP equivalent
```

```
embedded_primary_energy :

    value = 0.000365296803652968O5

    description = Embedded primary energy consumed (manufacturing
phase)

    type = gauge

    unit = MJ

    long_unit = Mega Joules
```

C.5 EEA-Tabelle zu den THG-Emissionen

Jahr	EU-Mitgliedsstaat	g CO2-eq/kWh
2021	Sweden	9
2021	Luxembourg	45
2021	France	58
2021	Finland	70
2021	Austria	82
2021	Latvia	106
2021	Slovakia	113
2021	Denmark	123
2021	Lithuania	127
2021	Croatia	138
2021	Belgium	139
2021	Spain	165
2021	Portugal	167
2021	Hungary	188
2021	Slovenia	211
2021	Romania	212
2021	Italy	234
2021	Ireland	332
2021	Netherlands	339
2021	Germany	348
2021	Malta	349
2021	Greece	397
2021	Czechia	397

Jahr	EU-Mitgliedsstaat	g CO2-eq/kWh
2021	Bulgaria	398
2021	Cyprus	605
2021	Estonia	656
2021	Poland	721
2021	EU-27	238

Quelle: European Energy Agency (‘Greenhouse Gas Emission Intensity of Electricity Generation in Europe’, n.d.), veröffentlicht am: 2. Juni 2023: https://www.eea.europa.eu/ims/greenhouse-gas-emission-intensity-of-1

D Messaufbau

D.1 Erfahrungen mit der Durchführung der Messung

In einem ersten Anlauf soll der Energieverbrauch eines Test-Skriptes bestimmt werden, um Hürden in der Durchführung der Messung zu identifizieren. Bei dem Test-Skript handelt es sich um wenige Zeilen Code in der Programmiersprache *Python*, der in Abbildung 61 dokumentiert ist. In der ersten Zeile wird die Fakultätsfunktion aus der *Python Math*-Bibliothek importiert, in der zweiten Zeile wird die Fakultät der Zahl 1 Million berechnet. Nach dieser Berechnung wird das Skript beendet.

Abbildung 61: Test-Skript faculty.py

```
from math import factorial as fac
fac(10**6)
```

Quelle: https://gitlab.com/softawere-hackathon/hackathon/fork-felix-faculty

D.2 Messaufbau

Der Messaufbau besteht aus einem physischen Server und mehreren darauf installierten Softwaremodulen. Das Zusammenspiel der unterliegenden Softwaremodule des Messaufbaus ist in Abbildung 62 dargestellt. Die Softwaremodule auf dem Server sind:

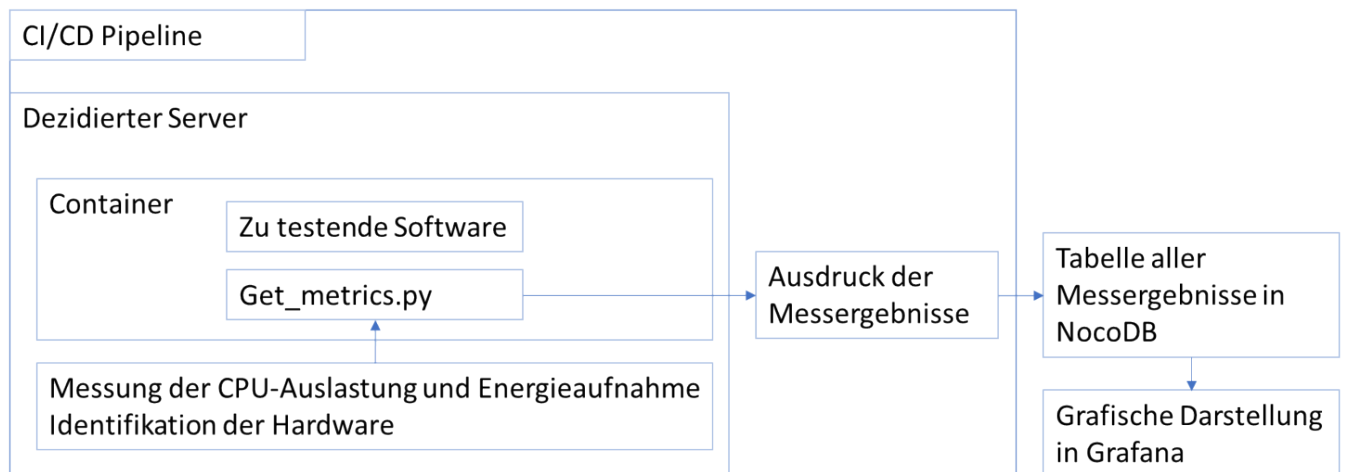
- Gitlab
- Continuous Integration/ Continuous Development (CI/CD) Pipeline innerhalb des Gitlab-Runners
- Module, welche die Hardwareauslastung und Energieaufnahme mitschreiben (Scaphandre, Boagent, boavistapi)
- Skript, welches Hardwareauslastung und Energieaufnahme der zu testenden Software zuordnet (Get_Metrics.py)
- Instruktionen, welche die zu testende Software in einem Container startet (.gitlab-ci.yml)
- Zu testende Software
- Externe Tools zum Speichern (Prometheus) und zur Visualisierung (nocoDB, Grafana) der Energieaufnahme und der Hardwareauslastung

Gitlab ist eine online-Plattform, auf der Software abgespeichert, geteilt und entwickelt werden kann. Der Gitlab-Runner stellt die Verknüpfung zur physikalischen Infrastruktur des Projektes dar: der Code wird auf diese Server-Infrastruktur übertragen und dort ausgeführt.

In der Mitte steht die zu testende Software, welche die Last erzeugt. Diese ist in einem Software-Container isoliert. Der Container wird auf einem dedizierten Server aufgesetzt, auf dem sonst ausschließlich die Messinfrastruktur läuft. Jeder Test wird in einem neu aufgesetzten Container gestartet. Tests können ausschließlich nacheinander starten, sodass jeweils genau die von einem einzelnen Container erzeugte Last gemessen wird. Die Messung der CPU-Laufzeit und die Messung der Energieaufnahme im *RAPL*-Chip (*RAPL – Running Average Power Limit*) verbirgt sich im Modul namens *Scaphandre*. Die zwei Module *Boagent* und *boaviztapi* identifizieren die

Spezifikationen der unterliegenden Hardware, z.B. die Anzahl der Prozessor-Kerne. Aus den Spezifikationen der unterliegenden Hardware wird der Herstellungsaufwand der Hardware berechnet und entsprechend der CPU-Laufzeit der zu testenden Software zugeordnet. „`Get_metrics.py`“ ist ein Python-Skript, das die Ergebnisse der Energieaufnahme und der Hardwareauslastung abfragt, die von außerhalb des Containers für den Inhalt des Containers gemessen wurden. Die Ergebnisse werden in der Laufzeitumgebung der CI/CD Pipeline angezeigt. Zusätzlich werden die Ergebnisse über *Prometheus* an *grafana* weitergeleitet und grafisch dargestellt und in der Datenbank *NocoDB* gespeichert und tabellarisch dargestellt. In Kapitel 3.1wird die Möglichkeit geprüft, die Ergebnisse zusätzlich in öffentlichen Repositorien wie Gitlab zu dokumentieren.

Abbildung 62: Schematischer Aufbau des Messlabors



Quelle: eigene Darstellung, Öko-Institut e.V.

D.3 Vorbereitende Schritte zur Messung

Um das Messsystem SoftAWERE Nutzenden zur Verfügung zu stellen, müssen Nutzer zunächst einen *Gitlab*-Account anlegen und der Administrator des Gitlab-Projektes *sdialliance*[145] muss eine Zugangsberechtigung erteilen, damit externe Nutzer eigenen Code auf dem Messystem ausführen können. Die Person, die eine Messung durchführen will, sollte außerdem mit der Funktionsweise von *Git* vertraut sein. *Git* ist ein Open-Source Versionsverwaltungstool für Software, das unter anderem mit der Plattform *Gitlab* zusammenarbeitet und ein kollaboratives Arbeiten ermöglicht. Statt über die Entwicklergruppe *sdialliance* die physische Infrastruktur der SDIA zu nutzen, ist es ebenso möglich, die Software der Messinfrastruktur auf einem eigenen Server einzurichten. Darauf wird an dieser Stelle nicht eingegangen.

Das Gitlab-Projekt „`Fork Me`“ im Unterordner *hackathon*[146] der Entwicklergruppe *sdialliance* dokumentiert die wesentlichen Schritte, die notwendige Ordnerstruktur und Abhängigkeiten, die man benötigt, um den Energieverbrauch der eigenen Software in einem *Gitlab-Runner* zu messen. Die wesentlichen Anpassungen an das jeweilige zu untersuchende Projekt werden in der Datei „`.gitlab-ci.yml`“ (*CI-Pipeline*) vorgenommen. Diese Datei steuert die Reihenfolge der folgenden auszuführenden *Bash*-Befehle:

[145] https://gitlab.com/softawere-hackathon

[146] https://gitlab.com/softawere-hackathon/hackathon/fork-me

- Auswahl des Container-Images,
- Installation aller Abhängigkeiten,
- Start der Energiemessung,
- Start der zu untersuchenden Software,
- Detektion des Endes der zu untersuchenden Software
- Ende der Energiemessung
- und Ausgabe der Messergebnisse mit „`get_metrics.py`".

D.4 Ausführung der zu messenden Software

Das Python-Skript „`faculty.py`" ist in diesem Absatz beispielhaft die zu untersuchende Software. Die zu untersuchende Software muss aus der *CI-Pipeline*-Datei „`.gitlab-ci.yml`" heraus gestartet werden. Die Datei zur Steuerung der kontinuierlichen Integration (CI) „`.gitlab-ci.yml`" beinhaltet die Befehle, welche die Messung ausführen. Der Inhalt ist in Abbildung 63 wörtlich nachzulesen. Er bedeutet folgendes:

- Zuerst wird ein Standard Python Container-Image der neuesten Version installiert. In diesem Container werden vorbereitend eine virtuelle Maschine, ein Paketmanagement *apt*, *command-line JSON processor* namens *jq* eingerichtet und externe Abhängigkeiten (siehe Abbildung 64) festgelegt.
- Der Zeitstempel zu Beginn der Messung wird festgehalten:
  ```
  export CI_JOB_AFTER_REQUIREMENTS_STARTED_AT=$(date +%s)
  ## Your test starts here !
  ```
- Die zu messende Software `faculty.py` (hier mit einer *for*-Schleife in *bash*-Schreibweise umfasst) wird gestartet:

```
for ((k=1; k<=1; k++)); do
      python faculty.py
done
```

- Die Messergebnisse werden ausgewertet: `## Collect results` usw.

Der eigentliche Test kann in diesem Beispiel mehrfach ausgeführt werden, indem die obere Grenze der Laufvariable k innerhalb der *for*-Schleife erhöht wird (hier nur einmalige Ausführung mit: `k<=1`).

Abbildung 63: Instruktionen der Messung „.gitlab-ci.yml“

```
image: python:latest

variables:
  PIP_CACHE_DIR: "$CI_PROJECT_DIR/.cache/pip"

before_script:
  - pip install virtualenv
  - virtualenv venv
  - source venv/bin/activate
# Dependencies for get_metrics script
  - apt update
  - apt install jq procps wget git-core -y
  - curl -O https://gitlab.com/softawere-hackathon/softawere/-/raw/main/
gitlab-metric-script/requirements-metrics.txt
  - pip install -r requirements-metrics.txt


test:
  script:
    - export CI_JOB_AFTER_REQUIREMENTS_STARTED_AT=$(date +%s)
## Your test starts here!
    - for ((k=1; k<=1; k++)); do
    - python faculty.py
    - done
## Collect results
    - curl -s https://gitlab.com/softawere-hackathon/softawere/-/raw/main
/gitlab-metric-script/get_metrics.py | python3
```

Quelle: Angelegtes Messprojekt https://gitlab.com/softawere-hackathon/hackathon/fork-felix-faculty

Abbildung 64: Inhalt von „requirements-metrics.txt“

```
prometheus-client==0.14.1
sty==1.0.4
docker==6.0.0
```

Quelle: Angelegtes Messprojekt https://gitlab.com/softawere-hackathon/hackathon/fork-felix-faculty

Wurden die Dateien vom lokalen Entwicklungsordner, dem sogenannten *Git-Repository*, erfolgreich in das *Gitlab*-Projekt hochgeladen (`git push origin main`), wird nach dem Hochladen auf den eingestellten Server automatisch die `.gitlab-ci.yml` Datei ausgeführt. Unter dem Menüpunkt *CI/CD>Jobs* innerhalb der *Gitlab*-Entwicklergruppe *sdialliance* werden der Fortschritt und die Ergebnisse von `get_metrics.py` ausgegeben. Die Ergebnisse für die Datenauswertung werden außerdem tabellarisch im Browser dargestellt, der Link wird zusammen mit dem *CI/CD*-Fortschritt am Ende ausgegeben. In dieser Tabelle ist jede Messung mit der Darstellung ihrer Ergebnisse in *Grafana* verlinkt. Die Tabelle kann als *csv*-Datei zur weiteren Auswertung heruntergeladen werden.

D.5 Interpretation der Messergebnisse

Das Messlabor misst die dynamischen Werte der CPU-Auslastung, Nutzung von RAM und Festplattenspeicher und der übertragenen Datenmengen, Leistungsaufnahme am IPMI-Chip (*IPMI – Intelligent Platform Management Interface*), Leistungsaufnahme am RAPL-Chip, Chip (*RAPL – Running Average Power Limit*) sowie die statischen Werte zur Anzahl der CPU-Kerne und anderer Systemgrößen.

Ausgehend von diesen Messwerten werden sowohl der anteilige Aufwand zur Herstellung der Hardware (*embedded X*) als auch der anteilige Umweltaufwand für die Betriebsphase (*operational X*) berechnet. Auch der PUE-Wert des Rechenzentrums wird als Wert in die Berechnung mit einbezogen.

Als Umweltwirkungs-Kennzahlen werden das **Treibhausgaspotenzial** (*GWP – Global Warming Potential*), der **Rohstoffverbrauch** (*ADP – Abiotic Resources Depletion*) und der **Primärenergieverbrauch** (*PEC – Primary Energy Consumption*) berechnet.

Der **Herstellungsaufwand** ergibt sich aus der anteiligen Inanspruchnahme der Hardware über die jeweilige Nutzungszeit. Der Anteil der Nutzung des Servers ist gleich dem Integral der jeweiligen Auslastung der Komponente über die Nutzungszeit, geteilt durch die erwartete Lebensdauer des Servers von vier Jahren. Die Berechnung erfolgt in dem Modul *boaviztapi*. In diesem Modul ist u.a. auch eine Tabelle der Umweltaufwände der Herstellung der Komponenten, z.B. Emissionsfaktoren für die Herstellung von CPU-Kernen und den Strommix, hinterlegt.

Der **Aufwand der Betriebsphase** wird über den gemessenen Stromverbrauch bestimmt. Die mittlere Leistungsaufnahme [W] multipliziert mit der Ausführungsdauer [s] ergibt den Stromverbrauch in der Nutzungsphase [Ws]. Daraus werden wiederum mit einem Emissionsfaktor für Elektrizität, einem Rohstofffaktor und einem Primärenergiefaktor die jeweiligen Umweltwirkungskategorien berechnet.

Als Umrechnungsfaktoren gibt die Bildschirmausgabe des Messlaufs die in Abbildung 65 genannten Umrechnungsfaktoren an.

Abbildung 65: Umrechnungsfaktoren von Stromverbrauch in ökologische Kennzahlen

```
gwp_factor = 0.45 kg CO2eq/kWh

adp_factor = 3.29e-08 kg Sbeq/kWh

pe_factor = 8.511 MJ/kWh
```

Quelle: Ausschnitt aus dem Messprotokoll (angepasst) von https://gitlab.com/softawere-hackathon/hackathon/

In Abbildung 66 ist beispielhaft das Ergebnis der CI/CD Pipeline für das Test-Skript „`faculty.py`“ bei sechsmaligem Durchlauf (`k<=6`) dargestellt. Die drei genannten Größen werden jeweils als eine Summe von Nutzungsphase und Herstellungsphase ausgegeben.

Abbildung 66: Ergebnisausgabe in der CI/CD Konsole

```
🏭 Global Warming Potential = Operational Emissions + Embdedded Emissions

🏭 0.0005371131405377981 kg CO2eq = 0.0004 + 0.00013711314053779807

⛏  Abiotic Resources Depletion = Operational Abiotic Resources depletion + Embdedded Abiotic
Resources depletion

⛏  2.7743558193810252e-08 kg Sbeq = 2.92e-11 + 2.771435819381025e-08

🛢 Primary Energy Consumption = Operational Primary Energy consumption + Embdedded Primary Energy
Consumption

🛢 0.00943107255200406 MJ = 0.007564 + 0.001867072552004059
```

Quelle: Ausschnitt aus dem Messprotokoll von https://gitlab.com/softawere-hackathon/hackathon/

Bei der Berechnung der ökologischen Kennzahlen (Umweltwirkungskategorien) bei der Ausführung des oben beschriebenen Test-Skripts fällt auf, dass die aus dem Stromverbrauch resultierenden betriebsbedingten Treibhausgasemissionen (GWP) rund drei Viertel (74%) des gesamten Treibhauspotenzials ausmachen. Beim Primärenergieverbrauch (PEC) sind es vier Fünftel (80%), die vom Stromverbrauch hervorgerufen werden. Beim Rohstoffverbrauch (ADP) sind es dagegen gerade mal rund ein Tausendstel (0,1%) des Gesamtwertes, die auf die Betriebsphase zurückgehen. Dies ist plausibel, da die Stromerzeugung viel weniger abiotische Ressourcen benötigt als die Herstellung von Servern.

D.6 Detaillierte Spezifikation des Testsystems

Jeder der Server im Testsystem verfügt über:

- 48 CPU-Kerne über 2 CPUs (Intel Xeon CPU E5-2678 v3 @ 2.50GHz, 6 MB L2, 60 MB L3)
- 32 GB Arbeitsspeicher
- 2 x 120 GB KINGSTON SUV500M - Solid State Drives

Die detaillierten CPU-Konfigurationsoptionen findet sich in Abbildung 67.

Ein Bild des wassergekühlten Rechenzentrums von Blockheating findet sich in Abbildung 68.

Abbildung 67: Detaillierte CPU-Spezifikationen

Detaillierte Spezifikationen der eingesetzten CPUs im Test- Labor

```
ubuntu@huge-filly:~$ lscpu
Architecture:                    x86_64
CPU op-mode(s):                  32-bit, 64-bit
Byte Order:                      Little Endian
Address sizes:                   46 bits physical, 48 bits virtual
CPU(s):                          48
On-line CPU(s) list:             0-47
Thread(s) per core:              2
Core(s) per socket:              12
Socket(s):                       2
NUMA node(s):                    2
Vendor ID:                       GenuineIntel
CPU family:                      6
Model:                           63
Model name:                      Intel(R) Xeon(R) CPU E5-2678 v3 @ 2.50GHz
Stepping:                        2
CPU MHz:                         1197.760
CPU max MHz:                     3300.0000
CPU min MHz:                     1200.0000
BogoMIPS:                        4988.39
Virtualization:                  VT-x
L1d cache:                       768 KiB
L1i cache:                       768 KiB
L2 cache:                        6 MiB
L3 cache:                        60 MiB
NUMA node0 CPU(s):               0-11,24-35
NUMA node1 CPU(s):               12-23,36-47
Vulnerability Itlb multihit:     KVM: Mitigation: Split huge pages
Vulnerability L1tf:              Mitigation; PTE Inversion; VMX conditional cache flushes, SMT vulnerable
Vulnerability Mds:               Mitigation; Clear CPU buffers; SMT vulnerable
Vulnerability Meltdown:          Mitigation; PTI
Vulnerability Mmio stale data:   Mitigation; Clear CPU buffers; SMT vulnerable
Vulnerability Spec store bypass: Mitigation; Speculative Store Bypass disabled via prctl and seccomp
Vulnerability Spectre v1:        Mitigation; usercopy/swapgs barriers and __user pointer sanitization
Vulnerability Spectre v2:        Mitigation; Retpolines, IBPB conditional, IBRS_FW, STIBP conditional, RSB filling
Vulnerability Srbds:             Not affected
Vulnerability Tsx async abort:   Not affected
Flags:                           fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2
                                  ss ht tm pbe syscall nx pdpe1gb rdtscp lm constant_tsc arch_perfmon pebs bts rep_good nopl xtopology nons
                                 top_tsc cpuid aperfmperf pni pclmulqdq dtes64 monitor ds_cpl vmx smx est tm2 ssse3 sdbg fma cx16 xtpr pdcm
                                  pcid dca sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm abm cpuid
                                 _fault epb invpcid_single pti intel_ppin ssbd ibrs ibpb stibp tpr_shadow vnmi flexpriority ept vpid ept_ad
                                  fsgsbase tsc_adjust bmi1 avx2 smep bmi2 erms invpcid cqm xsaveopt cqm_llc cqm_occup_llc dtherm ida arat p
                                 ln pts md_clear flush_l1d
```

Quelle: eigene Darstellung, Sustainable Digital Infrastructure Alliance e.V.

Abbildung 68: Rechenzentrum des Testlabors

Wassergekühlte Infrastruktur, in der sich das Testlabor auf wiederaufbereiteter Server-Hardware befindet

Quelle: Blockheating BV

D.7 Mess-Skript aus dem Test-Labor

Abbildung 69: Python Skript für die Sammlung der Messergebnisse im Test-Labor

```
#!/usr/bin/python3


import requests

import urllib.request

import urllib.parse

from os import environ

from datetime import datetime

import time

import json

from sty import fg, bg, ef

from prometheus_client import CollectorRegistry, Gauge,
push_to_gateway


DOCKER_NETWORK_GATEWAY="172.17.0.1"

BOAVIZTAPI_ENDPOINT =
"http://{}:8000".format(DOCKER_NETWORK_GATEWAY)

RUNNER_ENDPOINT =
"http://{}:8085".format(DOCKER_NETWORK_GATEWAY)

PUSHGATEWAY_ENDPOINT =
"{}:9091".format(DOCKER_NETWORK_GATEWAY)

NOCODB_ENDPOINT="https://db.softaware.hubblo.org"


def main():

    ci_job_started_at =
environ.get("CI_JOB_AFTER_REQUIREMENTS_STARTED_AT")

    ci_job_coverage = environ.get("CI_JOB_COVERAGE")
```

```
    ci_job_size = environ.get("CI_JOB_SIZE")

    ci_job_test_count = environ.get("CI_JOB_TEST_COUNT")

    ci_project_id = environ.get("CI_PROJECT_ID")

    ci_pipeline_id = environ.get("CI_PIPELINE_ID")

    ci_job_id = environ.get("CI_JOB_ID")


    grafana_link = "https://grafana.eco-
qube.eu/d/hzpu8ydVz/ci-cd-impacts?orgId=1&refresh=1m&var-
PROJECT_ID={}&var-PIPELINE_ID={}&var-
JOB_ID={}".format(ci_project_id, ci_pipeline_id,
ci_job_id)


    #ci_job_stopped_at = datetime.now().isoformat()

    ci_job_stopped_at = time.time()

    url =
"{}/query?verbose=true&location=NLD&start_time={}&end_time
={}".format(

        BOAVIZTAPI_ENDPOINT, ci_job_started_at,
ci_job_stopped_at

    )

    f = urllib.request.urlopen(url)

    res = f.read().decode('utf-8')

    d = json.loads(res)

    #push_to_nocodb(d, ci_job_started_at,
ci_job_stopped_at, ci_job_coverage, ci_job_size,
ci_job_test_count, grafana_link)

    push_to_prom(d, ci_job_started_at, ci_job_stopped_at)

    pretty_print_results(d)

    print_summary(d, grafana_link)
```

```
def add_gauge_to_registry(name: str, description: str,
value: str, registry: CollectorRegistry):

    g = Gauge(name, description, ['ci_job_id',
'project_name', 'ci_job_name', 'project_id',
'pipeline_id'], registry=registry)

    g.labels(environ.get("CI_JOB_ID"),
environ.get("CI_PROJECT_NAME"),
environ.get("CI_JOB_NAME"), environ.get("CI_PROJECT_ID"),
environ.get("CI_PIPELINE_ID")).set(value)


def push_to_prom(res, started_at, stopped_at):

    registry = CollectorRegistry()

    add_gauge_to_registry(

        name='softawere_job_operational_GWP',

        description="Global Warming Potential: Operational
(run) emissions attributed to this CI job. In
{}".format(res["total_operational_emissions"]["unit"]),

        value=res["total_operational_emissions"]["value"],

        registry=registry

    )

    add_gauge_to_registry(

        name='softawere_job_embodied_GWP',

        description="Global Warming Potential: Embodied
(manufacture, transport, eol) emissions attributed to this
CI job. In {}".format(res["embedded_emissions"]["unit"]),

        value=res["embedded_emissions"]["value"],

        registry=registry

    )

    add_gauge_to_registry(
```

```
        name='softawere_job_total_GWP',

        description="Global Warming Potential: Total (full
lifecycle) emissions attributed to this CI job. In
{}".format(res["embedded_emissions"]["unit"]),


value=res["embedded_emissions"]["value"]+res["total_operat
ional_emissions"]["value"],

        registry=registry

    )

    add_gauge_to_registry(

        name='softawere_job_operational_ADPe',

        description="Abiotic Resources Depletion:
Operational (run) minerals consumption attributed to this
CI job. In
{}".format(res["total_operational_abiotic_resources_deplet
ion"]["unit"]),


value=res["total_operational_abiotic_resources_depletion"]
["value"],

        registry=registry

    )

    add_gauge_to_registry(

        name='softawere_job_embodied_ADPe',

        description="Abiotic Resources Depletion: Embodied
(manufacture, transport, eol) minerals consumption
attributed to this CI job. In
{}".format(res["embedded_abiotic_resources_depletion"]["un
it"]),


value=res["embedded_abiotic_resources_depletion"]["value"]
,

        registry=registry
```

```
    )

    add_gauge_to_registry(

        name='softawere_job_total_ADPe',

        description="Abiotic Resources Depletion: Total
(full lifecycle) minerals consumption attributed to this
CI job. In
{}".format(res["embedded_abiotic_resources_depletion"]["un
it"]),


value=res["embedded_abiotic_resources_depletion"]["value"]
+res["total_operational_abiotic_resources_depletion"]["val
ue"],

        registry=registry

    )

    add_gauge_to_registry(

        name='softawere_job_operational_PE',

        description="Primary Energy: Operational (run)
primary energy consumption attributed to this CI job. In
{}".format(res["total_operational_primary_energy_consumed"
]["unit"]),


value=res["total_operational_primary_energy_consumed"]["va
lue"],

        registry=registry

    )

    add_gauge_to_registry(

        name='softawere_job_embodied_PE',

        description="Primary Energy: Embodied
(manufacture, transport, eol) primary energy consumption
attributed to this CI job. In
{}".format(res["embedded_primary_energy"]["unit"]),
```

```
        value=res["embedded_primary_energy"]["value"],
        registry=registry
    )
    add_gauge_to_registry(
        name='softawere_job_total_PE',
        description="Primary Energy: Total (full
lifecycle) primary energy consumption attributed to this
CI job. In
{}".format(res["total_operational_primary_energy_consumed"
]["unit"]),

value=res["embedded_primary_energy"]["value"]+res["total_o
perational_primary_energy_consumed"]["value"],
        registry=registry
    )
    add_gauge_to_registry(
        name='softawere_job_total_time',
        description="Time needed by the job. In seconds.",
        value=str(float(stopped_at) - float(started_at)),
        registry=registry
    )


    push_to_gateway(PUSHGATEWAY_ENDPOINT,
job="from_ci_jobs", registry=registry)


def pretty_print_results(res, level=0):
    indent = ""
    i = 0
    while i < level:
```

```
        indent += "\t"
        i+=1
    for k, v in res.items():
        if type(v) is dict:
            print("{}{}{} :".format(indent, fg.red, k))
            pretty_print_results(v, level+1)
        else:
            print("{}{}{} = {}{}".format(indent, fg.white,
k, fg.green, v))


def get_nocodb_auth_token() -> str:
    signin_url = NOCODB_ENDPOINT +
'/api/v1/auth/user/signin'
    print("Authentication at NocoDB:
{}".format(signin_url))
    response = requests.post(signin_url, json={
        'email': environ['NOCODB_USERNAME'],
        'password': environ['NOCODB_PASSWORD']
    }, verify=False)
    response.raise_for_status()
    return response.json().get('token')



def push_to_nocodb(res, started_at, stopped_at,
job_coverage, job_size, job_test_count, link):
    token = get_nocodb_auth_token()
    print("NocoDB Token received {}".format(token))
```

```
    headers = {'xc-auth': token}

    orgs = 'nc'

    project_name = 'Leaderboard'

    table_name = 'Leaderboard submissions'

    insert_url = NOCODB_ENDPOINT +
f'/api/v1/db/data/{orgs}/{project_name}/{table_name}'

    print("NocoDB reporting data to URL:
{}".format(insert_url))

    response = requests.post(insert_url, headers=headers,
json={

        'Team': environ.get("CI_PROJECT_NAME"),

        'Branch': environ.get("CI_COMMIT_BRANCH"),

        'Submission date': str(datetime.now()),

        'Job ID': environ.get("CI_JOB_ID"),

        'Job duration': float(stopped_at) -
float(started_at),

        'Job embedded impact CO2':
res["embedded_emissions"]["value"],

        'Job embedded impact ADP':
res["embedded_abiotic_resources_depletion"]["value"],

        'Job embedded impact PE':
res["embedded_primary_energy"]["value"],

        'Job operational impact CO2':
res["total_operational_emissions"]["value"],

        'Emission Factor':
float(res["emissions_calculation_data"]["electricity_carbo
n_intensity"]["value"]),

        'Average Power':
float(res["emissions_calculation_data"]["average_power_mea
sured"]["value"]),
```

```
        'Grafana Link': link,
        'Test Coverage': job_coverage,
        'Job Size': job_size,
        'Number of Tests': job_test_count,
    }, verify=False)
    if not response.ok:
        print('Error while pushing to nocodb.')


def print_summary(res, grafana_link):
    print(ef.bold)
    print(
        "🏭 {}Global Warming Potential = {}Operational 
Emissions {}+ {}Embdedded Emissions".format(
            fg.white, fg.grey, fg.white, fg.grey
        )
    )
    print(
        "🏭 {}{}{}{} {} = {}{} {}+ {}{}".format(
            bg.yellow,
            fg.white,
            res["calculated_emissions"]["value"],
            bg.rs,
            res["calculated_emissions"]["unit"],
            fg.grey,
            res["total_operational_emissions"]["value"],
            fg.white,
```

```
            fg.grey,
            res["embedded_emissions"]["value"]
        )
    )
    print(
        "⛏� {}Abiotic Resources Depletion = 
{}Operational Abiotic Resources depletion {}+ {}Embdedded 
Abiotic Resources depletion".format(
            fg.white, fg.grey, fg.white, fg.grey
        )
    )
    print(
        "⛏� {}{}{}{} {} = {}{} {}+ {}{}".format(
            bg.yellow,
            fg.white,

res["total_operational_abiotic_resources_depletion"]["valu
e"]+res["embedded_abiotic_resources_depletion"]["value"],
            bg.rs,

res["embedded_abiotic_resources_depletion"]["unit"],
            fg.grey,

res["total_operational_abiotic_resources_depletion"]["valu
e"],
            fg.white,
            fg.grey,

res["embedded_abiotic_resources_depletion"]["value"]
```

```
        )
    )
    print(
        "🛢️ {}Primary Energy Consumption = {}Operational
Primary Energy consumption {}+ {}Embdedded Primary Energy
Consumption".format(
            fg.white, fg.grey, fg.white, fg.grey
        )
    )
    print(
        "🛢️ {}{}{}{} {} = {}{} {}+ {}{}".format(
            bg.yellow,
            fg.white,

res["total_operational_primary_energy_consumed"]["value"]+
res["embedded_primary_energy"]["value"],
            bg.rs,
            res["embedded_primary_energy"]["unit"],
            fg.grey,

res["total_operational_primary_energy_consumed"]["value"],
            fg.white,
            fg.grey,
            res["embedded_primary_energy"]["value"]
        )
    )


    print(
```

```
        "🔬 🧪 {}{}Get complete data and analysis at {}{}
🧪 🔬".format(fg.white, bg.yellow, grafana_link, bg.rs)
    )


    print(
            "Aggregated data from all projects tested on
this instance are accessible here :
{}".format("https://db.softaware.hubblo.org/dashboard/#/nc
/view/66a34add-e5f3-4a6a-a1f6-4a42739af7a7")
    )


if __name__ == '__main__':
    main()
```

Quelle: eigene Darstellung, Sustainable Digital Infrastructure Alliance e.V.

D.8 Docker Compose Instruktionen für den Aufbau des Testlabors

Das Testlabor kann auf beliebigen Server-Systemen gestartet werden. Notwendig für die Ausführung ist lediglich, dass entweder die RAPL-Schnittstelle oder die IPMI-Schnittstelle verfügbar ist und die Docker Compose-Umgebung über ein Administratorenprivileg verfügt.

Die Docker-Compose Konfiguration wurde mit Docker Compose Version 3.8 getestet und wurde nicht auf Kompatibilität mit älteren Versionen überprüft. Die aktuelle Version der Konfiguration findet sich im quelloffenen Code-Repository des Vorhabens[147].

Die Konfigurationsdatei „docker-compose.yml“ ist in Abbildung 70 dargestellt.

Abbildung 70: docker-compose.yml Datei

Detaillierte Spezifikationen der eingesetzten CPUs im Test- Labor

```
version: '3.2'


services:
```

[147] Das Repository ist auf GitLab.com verfügbar: https://gitlab.opencode.de/OC00004041236/softawere, abgerufen am 13.10.2023

```
  gitlab-runner:

    image: gitlab/gitlab-runner:alpine

    privileged: true

    volumes:

      - "./gitlab-runner/config/config.toml:/etc/gitlab-runner/config.toml"

      - "/var/run/docker.sock:/var/run/docker.sock"

      - "./gitlab-runner/entrypoint:/home/gitlab-runner/entrypoint"

    entrypoint: ["/home/gitlab-runner/entrypoint/entrypoint.sh"]

    ports:

      -
"127.0.0.1:${GITL_RUNN_EXPORTER_LISTEN_PORT}:${GITL_RUNN_EXPORTER_LISTEN_PO
RT}"

      -
"${DOCKER_LOCAL_IP_GATEWAY}:${GITL_RUNN_EXPORTER_LISTEN_PORT}:${GITL_RUNN_E
XPORTER_LISTEN_PORT}"

    networks:

      - boagent-network


  container-stats:

    image: ghcr.io/bpetit/container-stats:0.0.4

    volumes:

      - "/var/run/docker.sock:/var/run/docker.sock"

    ports:

      - "127.0.0.1:${STATS_TCP_PORT}:${STATS_TCP_PORT}"

    networks:

      - boagent-network


  grafana:

    image: grafana/grafana-oss:latest

    ports:

      - "${GRAF_LISTEN_PORT}:${GRAF_LISTEN_PORT}"

    environment:
```

```
      GF_DASHBOARDS_DEFAULT_HOME_DASHBOARD_PATH:
"/var/lib/grafana/dashboards/sample/sample-dashboard.json"

    volumes:

      -
"./grafana/config/sample/dashboards:/var/lib/grafana/dashboards/sample"

      -
"./grafana/config/sample/datasources:/etc/grafana/provisioning/datasources:
ro"

      -
"./grafana/config/sample/dashboards:/etc/grafana/provisioning/dashboards:ro
"

    networks:

      - boagent-network


  prometheus:

    image: bitnami/prometheus

    ports:

      - "127.0.0.1:${PROM_LISTEN_PORT}:${PROM_LISTEN_PORT}"

    volumes:

      - "./prometheus/config:/opt/bitnami/prometheus/conf:ro"

    networks:

      - boagent-network


  pushgateway:

    image: bitnami/pushgateway:latest

    ports:

      - "127.0.0.1:${PROM_PUSH_LISTEN_PORT}:${PROM_PUSH_LISTEN_PORT}"

      -
"${DOCKER_LOCAL_IP_GATEWAY}:${PROM_PUSH_LISTEN_PORT}:${PROM_PUSH_LISTEN_POR
T}"

    networks:

      - boagent-network


  boagent:
```

```
    image: ghcr.io/boavizta/boagent:0.0.5
    environment:
      BOAVIZTAPI_ENDPOINT: "http://boaviztapi:${BOAV_LISTEN_PORT}"
      DEFAULT_LIFETIME: 5.0
      HARDWARE_FILE_PATH: "/home/boagent/hardware_data.json"
      POWER_FILE_PATH: "/app/data/power_data.json"
    depends_on:
      - boaviztapi
      - scaphandre
    ports:
      - "127.0.0.1:${BOAG_LISTEN_PORT}:${BOAG_LISTEN_PORT}"
      -
"${DOCKER_LOCAL_IP_GATEWAY}:${BOAG_LISTEN_PORT}:${BOAG_LISTEN_PORT}"
    networks:
      - boagent-network
    volumes:
      - "/proc:/proc"
      - "/sys:/sys:ro"
      - "powerdata:/app/data:ro"
      - "./db:/home/boagent/db:rw"

  scaphandre-json:
    image: hubblo/scaphandre:dev
    volumes:
      - type: bind
        source: /proc
        target: /proc
      - type: bind
        source: /sys/class/powercap
        target: /sys/class/powercap
      - "powerdata:/app/data:rw"
```

```
    command: [ "json", "-s", "1", "-f", "/app/data/power_data.json" ]
    networks:
      - boagent-network

  scaphandre:
    image: hubblo/scaphandre:dev
    ports:
      - "127.0.0.1:${SCAPH_LISTEN_PORT}:${SCAPH_LISTEN_PORT}"
    volumes:
      - type: bind
        source: /proc
        target: /proc
      - type: bind
        source: /sys/class/powercap
        target: /sys/class/powercap
    command: [ "prometheus", "--containers", "-p", "8080" ]
    networks:
      - boagent-network

  boaviztapi:
    image: ghcr.io/boavizta/boaviztapi:0.2.0a0
    ports:
      - "127.0.0.1:${BOAV_LISTEN_PORT}:${BOAV_LISTEN_PORT}"
    networks:
      - boagent-network

  node-exporter:
    image: bitnami/node-exporter:latest
    ports:
      -
"127.0.0.1:${NODE_EXPORTER_LISTEN_PORT}:${NODE_EXPORTER_LISTEN_PORT}"
```

```
    networks:
      - boagent-network

volumes:
  powerdata: {}

networks:
  boagent-network:
    driver: bridge
    ipam:
      config:
        - subnet: 192.168.33.0/24
```

Quelle: eigene Darstellung, Sustainable Digital Infrastructure Alliance e.V.