HAW HAMBURG

BACHELORTHESIS
Bassel Nasser

Green Software Engineering

FAKULTÄT TECHNIK UND INFORMATIK
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Bassel Nasser

Green Software Engineering

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Wirtschaftsinformatik*
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Stefan Sarstedt
Zweitgutachter: Prof. Dr. Ulrike Steffens

Eingereicht am: 30. August 2022

Bassel Nasser

Thema der Arbeit

Green Software Engineering

Stichworte

Green Software, Nachhaltigkeit, Energieverbrauch, Benchmark, Java

Kurzzusammenfassung

Im Rahmen der vorliegenden Bachelorarbeit soll die Frage beantwortet werden, ob eine Evaluation von Frameworks einen sinnvollen Aufwand im Entwicklungsprozess darstellt.

Auf Grundlage der theoretischen Ausarbeitung wurden bestehende Forschungsansätze für eine nachhaltige Softwareentwicklung untersucht, die im weiteren Verlauf in bestehende Verfahrensmodelle integriert wurden.

Basierend auf einem Benchmarking-Experiment soll verdeutlicht werden, wie der Prozess der Datenerhebung und -auswertung zur Evaluation von Software aufgebaut sein könnte. Zur Vorbereitung des Benchmarkings wurden vier REST-Clients in der Programmiersprache Java entwickelt und mit einer Datenbank verbunden. Durch automatisierte HTTP-Requests soll schließlich in mehreren Iterationen ein möglichst genaues Ergebnis zum durchschnittlichen Energieverbrauch des Systems entstehen.

Das durchgeführte Benchmarking-Experiment zeigte, dass der CPU- und DRAM-basierte Energieverbrauch der jeweiligen Anwendungen unterschiedlich ausfällt, das Verhältnis zwischen diesen Verbrauchswerten bei allen Anwendungen jedoch fast identisch war. Bezug nehmend auf die initiale Forschungsfrage lässt sich daher festhalten, dass die Forschung hinsichtlich eines nachhaltigen Software Engineering Prozesses noch am Anfang steht. Richtlinien und Verbraucherhinweise könnten hier zukünftig für mehr Transparenz sorgen und die Softwareunternehmen stärker in die Pflicht nehmen.

Bassel Nasser

Title of Thesis

Green Software Engineering

Keywords

Green Software, Sustainability, Energy consumption, Benchmark, Java

Abstract

Based on the present Bachelor thesis the question should be answered whether an evaluation of Frameworks represents a meaningful expenditure in the development process.

Based on the theoretical elaboration, existing research approaches for sustainable software development were studied, which were integrated into existing process models in the further process.

Based on a Benchmarking experiment it is to be clarified how the process for the data collection and evaluation, for the evaluation of software, could be structured. In preparation for the benchmarking, four REST clients were developed in the Java programming language and connected to a database. Through automated HTTP requests, a result that is as accurate as possible regarding the average energy consumption of the system should be generated in several iterations.

The benchmarking experiment showed that the CPU- and DRAM-based energy consumption of the respective applications differed, but the ratio between these consumption values was almost identical for all applications. With reference to the initial research question, it can therefore be said that research into a sustainable software engineering process is still in its early stages. Guidelines and consumer information could provide more transparency in the future and make software companies more responsible.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Abkürzungsverzeichnis

AOT	Ahead of time
API	Application Programming Interface
CLBG	Computer Language Benchmarks Game
CPU	Central Processing Unit
DBMS	Datenbankmanagementsystem
DRAM	Dynamic Random Access Memory
et al.	et alii
GSF	Green Software Foundation
HVAC	Heizung, Lüftung, und Klimatechnik
IaaS	Infrastructure as a Service
IKT	Informations- und Kommunikationstechnik
IT	Informationstechnik

JPA	Java Persistence API
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
LCT	Life Cycle Thinking
REST	Representational State Transfer
UI	User Interface

Glossar

Content-Type	Gibt den ursprünglichen Mediantyp der Ressource innerhalb einer HTTP-Kommunikation an.
CPU	Zentrale Prozessoreinheit in einem elektrischen Gerät, die im Zuge der Verarbeitung der daran übergebenen Befehle seine Funktionsweise steuert.
Dependency Injection	Programmierkonzept, bei dem Objekte andere benötigte Objekte von außerhalb der eigentlichen Anwendung erhalten.
DRAM	Technologie für einen elektronischen Speicher, welcher für die temporäre Speicherung genutzt wird.
GraalVM	Virtuelle Maschine, die auf hohe Performance abzielt, für Microservices geeignet ist, sowie die Leistung und Effizienz von Anwendungen erheblich verbessern soll.
Green Computing	Ansatz für eine umweltverträgliche Nutzung von Computern.
Green IT	Bestrebung, die Nutzung von IKT umwelt- und ressourcenschonend zu gestalten.

IaaS	Cloud-Computing-Servicemodell, bei dem Compute-Ressourcen in einer Cloud gehostet werden.
Joule	Internationale Maßeinheit für Energie mit dem Einheitszeichen J. 3600 J entsprechen 1 Wattstunde und 1000 Wattstunden entsprechen 1 Kilowattstunde.
Microservice	Architekturmuster, bei dem komplexe Anwendungssoftware aus unabhängigen Prozessen generiert wird.
Native-Image	Eigenständig ausführbare Dateien, die neben den übersetzten Klassen der eigentlichen Anwendung die Klassen der Laufzeit Library und der Dependencies sowie nativen Code des JDK enthalten.
Reaktive Programmierung	Programmierparadigma, das mit asynchronen Datenströmen verbunden ist, die auf jegliche Änderungen oder Ereignisse reagieren.
REST-API	Paradigma für die Softwarearchitektur von verteilten Systemen, insbesondere für Webservices.
Service Discovery	Bezeichnet die automatische Erkennung von Diensten in einem Rechnernetz.

1 Einführung

1.1 Motivation

Am 12. Dezember 2015 wurde auf der Weltklimakonferenz in Paris das „Pariser Klimaabkommen“ beschlossen. In diesem Übereinkommen haben sich 195 Staaten verpflichtet, den Klimawandel einzudämmen und die weltweite Wirtschaft klimafreundlich und nachhaltiger zu gestalten (Vgl. United Nations, 2015).

Das Abkommen beinhaltet eine Verpflichtung zur Einhaltung des weltweiten Temperaturanstiegs auf maximal 2 Grad Celsius, im Idealfall jedoch auf 1,5 Grad Celsius. Diese Temperaturangaben sind im Vergleich zum vorindustriellen Zeitalter zu betrachten. Eine Umsetzung dieser Maßnahmen ist erforderlich, um die Folgen des Klimawandels einzudämmen und eine widerstandsfähige Entwicklung der Weltwirtschaft zu gewährleisten (Vgl. United Nations, 2015).

Zur Erreichung der im Pariser Klimaabkommen formulierten Ziele sind alle Wirtschaftszeige in der Pflicht, bisherige Strukturen und Anforderungen zu überdenken. Die Erreichung des deutschen Klimaziels (BMU, 2019) erfordert technische Lösungen zur Verringerung des CO-Verbrauchs digitaler Technologien.

Die Green Software Foundation ist eine gemeinnützige Organisation, die es sich zur Aufgabe gemacht hat, die Kultur der Softwareentwicklung in der Technologiebranche zu verändern. Das Ziel der GSF ist es, den Aspekt der Nachhaltigkeit als eine zentrale Priorität für Entwicklungsteams zu verankern und dadurch auf eine Betrachtungsstufe mit der Leistung, Sicherheit, den Kosten und der Zugänglichkeit zu heben (Vgl. Green Software Foundation, 2022). Die Ziele der GSF zeigen, dass bei der Frage nach Optimierungspotenzialen der IKT-Branche neben der Hardware auch die Software stärker in den Fokus rückt.

Das Umweltbundesamt hat eine Studie in Auftrag gegeben, welche den Einfluss von Software auf die indirekte Inanspruchnahme natürlicher Ressourcen durch Hardware untersucht hat. Durch eine Standardisierung von Nutzungsmustern und Benchmarks sowie die Definition und Umsetzung von Nachhaltigkeitsanforderungen im Entwicklungsprozess von Software sollen potenzielle Lösungswege aufgezeigt werden (Vgl. Prof. Dr. Lorenz Hilty, 2015, S. 7).

Formulierte Leitlinien der Autorenschaft zielen darauf ab, die durch Software indirekt ausgelöste Inanspruchnahme natürlicher Ressourcen zu minimieren. Durch die Entwicklung von Methoden und Standards sowie einer regelmäßigen Erhebung von Daten zur Messung des Energieverbrauchs soll dieser Prozess unterstützt werden. Maßnahmen wie die Vergabe von Siegeln und die Aus- und Weiterbildung der Verbrauchenden und Entwickelnden werden zudem empfohlen (Vgl. Prof. Dr. Lorenz Hilty, 2015, S. 56 ff.).

Die objektorientierte Programmiersprache *Java* könnte dabei als eine der meistgenutzten Programmiersprachen (GitHub, 2022) einen übergreifenden Einfluss bei der Umsetzung nachhaltiger Software ausüben. In einer Studie von R. Pereira et al. (Rui Pereira, 2017) wurden Methoden zur Messung des Energie- und Speicherverbrauchs verschiedener Programmiersprachen aufgezeigt. Zur Messung des Verbrauchs wurden bekannte Softwareprobleme innerhalb eines automatisierten Testverfahrens in der jeweiligen Programmiersprache durchgeführt. Ziel der Studie war es, Softwareentwickelnde bei der Wahl einer geeigneten Programmiersprache zu unterstützen.

Die angeführten Studien (Rui Pereira, 2017) (Prof. Dr. Lorenz Hilty, 2015) haben zur Inspiration dieser Bachelorarbeit beigetragen und werden im weiteren Verlauf näher beschrieben.

1.2 Ziel der Arbeit

Das Ziel dieser Bachelorarbeit ist es, den gegenwärtigen Forschungsstand zur Entwicklung nachhaltiger Software zu untersuchen und verschiedene Forschungsansätze aufzuzeigen. Anschließend soll in einem Benchmarking-Experiment demonstriert werden, wie sich der Energie- und Speicherverbrauch verschiedener Software miteinander vergleichen lässt und welche Erkenntnisse sich daraus gewinnen lassen.

Zur Durchführung des Benchmarkings ist die Entwicklung von Test-Anwendungen erforderlich. Hierzu wurden die *Java*-Frameworks *Spring Boot*, *Micronaut*, *Quarkus* und *Vert.x* verwendet. Mithilfe dieser Frameworks sollen REST-Clients zur Kommunikation mit einer Datenbank entwickelt werden. Ein Tool zur Erhebung der Energie- und Speicherdaten des Prozessors sowie die Automatisierung von API-Tests sollen zur Realisierung des Benchmarkings beitragen.

1.3 Aufbau der Arbeit

Die Bachelorarbeit ist wie folgt aufgebaut:

Kapitel 2 – Nachhaltigkeit in der IT – definiert den Begriff der Nachhaltigkeit im Allgemeinen sowie dessen Bedeutung in der IT.

Kapitel 3 – Relevante Forschung – fasst die für diese Arbeit relevanten Forschungsergebnisse hinsichtlich Green Software und dessen Messverfahren zusammen.

Kapitel 4 – (Green) Software Engineering – fasst die relevanten Grundlagenkenntnisse für die vorliegende Arbeit zusammen. In Unterkapitel 4.1 wird der Begriff des Software Engineering definiert. In 4.2 werden mittels vorhandener Literatur alternative Methoden und Einflussfaktoren für ein nachhaltiges Software Engineering beschrieben. Das Unterkapitel 4.3 gibt einen Überblick der relevanten Forschungen auf dem Gebiet der nachhaltigen Softwareentwicklung.

Kapitel 5 – Frameworks für das Benchmarking – beschreibt die Charakteristika der für das Benchmarking genutzten Frameworks.

Kapitel 6 – Benchmarking – beschreibt die Planung, Vorbereitung, Spezifikation und Durchführung des Benchmarking-Prozesses.

Kapitel 7 – Auswertung der Testergebnisse – stellt die Ergebnisse des Benchmarkings vor und beschreibt die gewonnenen Erkenntnisse.

Kapitel 8 – Fazit – fasst die Forschungsergebnisse zusammen und gibt einen Ausblick über die Zukunft von nachhaltiger Softwareentwicklung.

2 Nachhaltigkeit in der IT

In diesem Kapitel wird der Begriff der Nachhaltigkeit im Allgemeinen definiert. Anschließend wird die Verantwortung der IT für die Nachhaltigkeit verdeutlicht und der Begriff Green Software abgegrenzt.

2.1 Begriffsdefinition von Nachhaltigkeit

Es gibt wenige Begriffe, die sich in den letzten Jahren so universell verwenden ließen wie der Begriff der „Nachhaltigkeit“. Geprägt wurde der Begriff im deutschsprachigen Raum durch Hans Carl von Carlowitz (1645 – 1714), der diesen als eine kontrollierte Abholzung der Waldwirtschaft verstand (Vgl. Reineke, 2020, S. 618).

Als im Jahr 1980 erstmals das Strategiepapier „World Conservation Strategy“ der *International Union for the Conservation of Nature* (IUCN; Weltnaturschutzunion) erschien und ins Deutsche übersetzt wurde, tauchte der Begriff als „Sustainable Development“ im Zusammenhang mit einer zukunftsfähigen, dauerhaften, umweltgerechten oder nachhaltigen Entwicklung wieder auf. Der Begriff „nachhaltige Entwicklung“ hat sich im deutschsprachigen Raum durchgesetzt und soll zwei Aspekte vereinen. Der Begriff nachhaltig wird als das Konservative, Bewahrende verstanden und Entwicklung als das Fortschreitende sich verändernde (Vgl. Zimmermann, 2016, S. 3 ff.).

In der Literatur hat sich das Nachhaltigkeitsdreieck etabliert, welches die drei Dimensionen der Nachhaltigkeit (Umwelt, Gesellschaft und Wirtschaft) in Balance bringen soll. Im Fokus der ökologischen Nachhaltigkeit stehen eine maßvolle Nutzung der natürlichen Lebensgrundlage sowie die Erhaltung der Artenvielfalt. Die soziale Nachhaltigkeit beschreibt die Sicherstellung weltweiten Wohlstands und Friedens. Ökonomische Nachhaltigkeit ist der Einklang eines

Wirtschaftssystems innerhalb dessen die ökologischen Grenzen langfristig bestehen können (Vgl. Pufé, 2014).

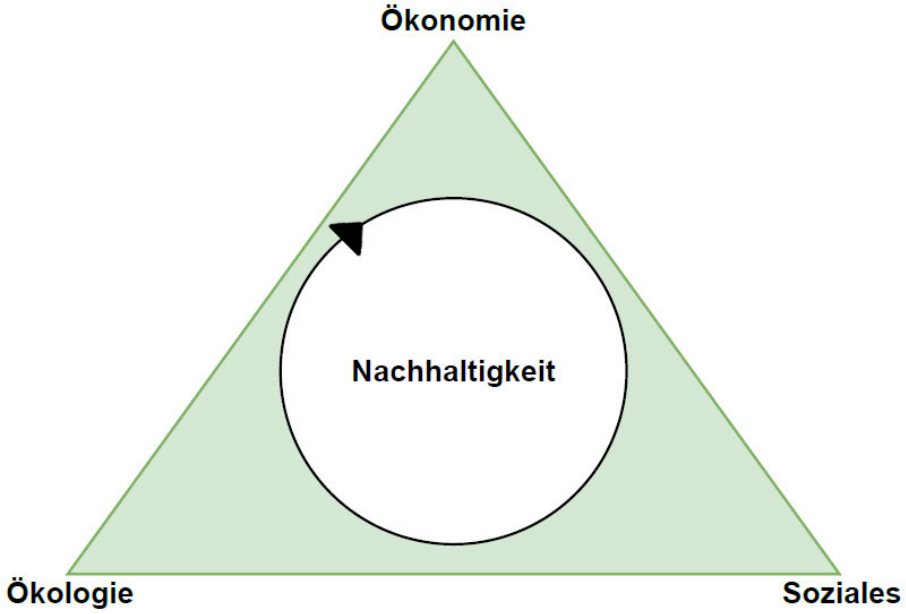

Abbildung 1: Nachhaltigkeitsdreieck
Quelle: In Anlehnung an (Pufé, 2014)

2.2 Kohlendioxidemissionen der IKT-Branche

Das Thema der globalen Erderwärmung ist spätestens seit der Entstehung der Klimastreik-Bewegung „Fridays for Future“ omnipräsent. Fabel et al. haben zudem eine Zunahme des Umweltbewusstseins bei Wählenden durch die Klimastreikbewegung festgestellt (Dr. Marc Fabel, 2022). Primär durch die Verbrennung fossiler Rohstoffe führen anwachsende Kohlendioxidemissionen langfristig zu Problemen ungeahnten Ausmaßes. Extremwetterlagen, Dürren und schmelzende Pole sind nur ein Teil der Auswirkungen gleichbleibender oder steigender Kohlenstoffemissionen (Vgl. Dr. Christian Anton, 2021).

Eine von dem Unternehmen *Ericsson* durchgeführte Studie zur Analyse des CO2-Fußabdrucks der IKT-Branche hat ergeben, dass allein erneuerbare Energien den Fußabdruck der Branche um bis zu 80 Prozent reduzieren könnten. Laut der Studie liegt der Anteil an globalen CO2-

Emissionen der IKT-Branche bei circa 1,4 %, der Anteil am weltweiten Stromverbrauch bei circa 3,6 % (Vgl. Ericsson, 2020).

Ericsson gibt in seiner Studie zudem Handlungsempfehlungen, wie Einzelpersonen ihren CO2-Fußabdruck reduzieren können. Die Lebensdauer von IKT-Geräten zu verlängern und die Akkus der Geräte mit Strom aus erneuerbaren Energien aufzuladen, sind nur ein Teil Vorschläge. Des Weiteren empfiehlt das Unternehmen, IKT-Dienste zu verwenden, die zur Reduzierung der CO2-Emissionen beitragen (Vgl. Ericsson, 2020, S. 15).

Um schließlich eine höchstmögliche Einsparung an CO2-Emissionen zu erzielen, ist es erforderlich, neben den Energiequellen auch die Energietreiber an die sich verändernden Gegebenheiten anzupassen. Die Optimierung der Software zur Reduzierung des Energiebedarfs ist somit ein zentraler Baustein für den Beitrag der IKT-Branche an der Vermeidung der globalen Erderwärmung.

2.3 Green Software

Aus Unternehmenssicht ist es heutzutage essenziell die Entwicklung seiner Produkte nachhaltig zu gestalten, da es ansonsten öffentliche Kritik oder sogar einen Verlust an Marktanteilen bedeuten kann (Vgl. Wenyu Du, 2013). Nachhaltige Produktentwicklung allein hilft nicht, eine starke Position am Markt zu generieren. Vor allem bei Software sind es Merkmale wie zum Beispiel die Usability, Performance, Code-Qualität und Sicherheitsaspekte, die eine Kaufentscheidung beeinflussen können. Die meisten Kaufenden sind darüber hinaus bereit, mehr für ein Produkt zu zahlen, welches nachhaltig ist (Vgl. Cazier J, 2011).

Während sich auf der einen Seite klassische Softwareprodukte durch die genannten Merkmale auszeichnen, zeichnet sich Green Software vor allem durch die Hinzunahme von Nachhaltigkeitsaspekten wie beispielsweise Stromverbrauch, Speicherbedarf und Modularität aus. Die individuelle Anpassung einer Benutzeroberfläche hinsichtlich der Helligkeit oder die Möglichkeit zur Abschaltung nicht verwendeter Funktionen durch den Anwender können hierbei einen positiven Effekt auf den Strombedarf der Software haben.

Green Software hat eine Vielzahl von Anwendungsbereichen, wie z. B. Softwaresysteme, Softwareprodukte, Webanwendungen und Rechenzentren, in denen der Nachhaltigkeitsansatz

umgesetzt werden kann. Das Augenmerk wurde lange Zeit auf die Verbesserung der Energieeffizienz der Hardware gelegt. Doch gerade die Software bietet ein großes Potenzial für Effizienzverbesserungen, die in der Vergangenheit jedoch selten bis nie eine Rolle in den Anforderungen bei der Entwicklung spielten (Vgl. The Climate Group, 2008).

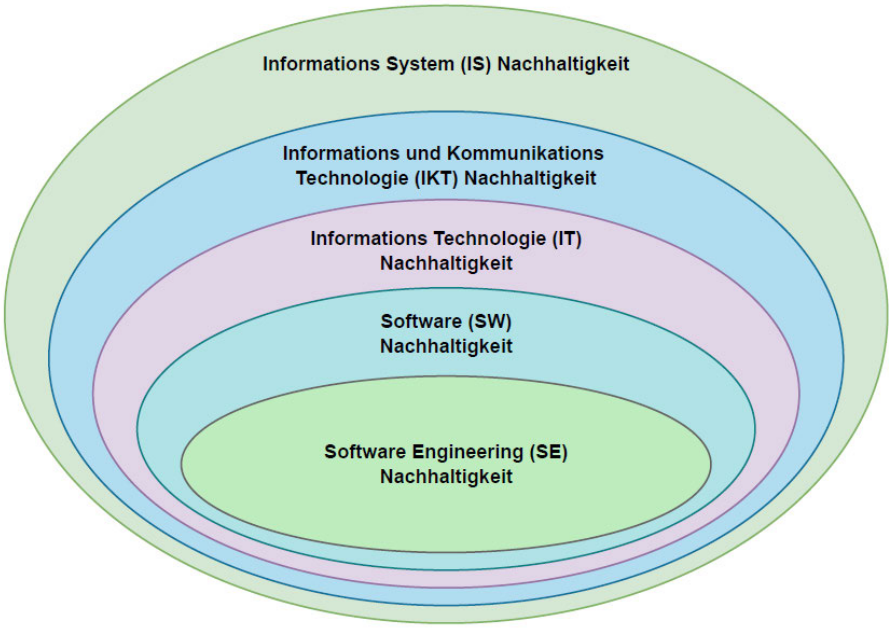

Abbildung 2: Nachhaltigkeitslevel von Software
Quelle: In Anlehnung an (Calero, 2021, S. 5)

In Abbildung 2 sind die verschiedenen Ebenen der Nachhaltigkeit dargestellt. In dieser Arbeit werden die zwei untersten Ebenen Software Nachhaltigkeit und Software Engineering betrachtet und herausgearbeitet, wie Software zu Green Software werden kann.

3 Relevante Forschung

Das folgende Kapitel soll einen Überblick über den für diese Arbeit relevanten Forschungsstand geben. Hierzu werden zwei ausgewählte Studien vorgestellt, die sich mit dem Thema „Green Software“ beschäftigen. Thematisiert wird zunächst das Umweltschutzpotenzial in der Softwareentwicklung. Anschließend wird eine Studie zum Thema Energieeffizienz von Programmiersprachen vorgestellt, welche Messverfahren und eine Auswertung dieser erhobenen Daten vorstellt.

3.1 Umweltschutzpotenziale in der Softwareentwicklung

Umweltschutzpotenziale sind in den meisten Softwarelösungen vorhanden. Viele Publikationen befassen sich folglich mit der Frage, wie Software nachhaltiger entwickelt werden kann und wo im Entwicklungsprozess diese Nachhaltigkeitsaspekte Beachtung finden sollten. Im folgenden Abschnitt wird eine Studie vorgestellt, die sich mit der Thematik der Erschließung von Umweltschutzpotenzialen in Softwareanwendungen befasst.

Das Umweltbundesamt hat eine Studie durchgeführt, in der die Ressourcennutzung von Software untersucht wurde. Dabei wurden verschiedene Funktionen von IKT-Systemen und deren Lebenszyklen betrachtet, um abschließend den Energieverbrauch zu ermitteln (Prof. Dr. Lorenz Hilty, 2015).

Im ersten Teil der Untersuchung wurde eine Trendanalyse durchgeführt, welche relevante Trends im Softwarebereich erfasst. Die Auswahl der Trends erfolgte nach Selektion von erkennbaren Entwicklungen, welche direkten Einfluss auf die Inanspruchnahme natürlicher Ressourcen haben (Vgl. Prof. Dr. Lorenz Hilty, 2015, S. 21 ff.).

Die Analyse ergab neben übergeordneten Trends auch solche im Bereich der Softwareentwicklung sowie Nutzungsformen. Bei den übergeordneten Trends wurde mobiler Internetzugang

genannt, welcher gegenüber dem Internetzugang via WLAN mit einem höheren Energiebedarf verbunden ist (CEET, 2013). Foto- und Videodaten mit immer höherer Auflösung und Spammails wurden als weitere Trends identifiziert (Vgl. Prof. Dr. Lorenz Hilty, 2015, S. 21 ff.). Im zweiten Analyseteil wurden die Trends App, webbasierte Anwendungssoftware, Virtualisierung und Cloud Computing sowie soziale Netzwerke erfasst (Vgl. Prof. Dr. Lorenz Hilty, 2015, S. 25 ff.).

Da die genannte Studie im Jahr 2015 erschienen ist, kann von einer Veränderung der Trends im Vergleich zum aktuellen Jahr ausgegangen werden. So hat eine Studie von Capgemini ergeben, dass sich das Thema „Open API“ auf Platz fünf der wichtigsten Technologie-Trends für das Jahr 2022 einreiht (Vgl. Dr. Sven L. Roth, 2022, S. 31).

Im zweiten Teil der Studie wurden Ansatzpunkte für die Ressourcenschonung genannt (Vgl. Prof. Dr. Lorenz Hilty, 2015, S. 32 ff.). Die Ansatzpunkte variieren je nach Anwendungsbereich, sollen aber ausnahmslos zu einer Ressourceneinsparung bei der Anwendung führen. Die wichtigsten Ansatzpunkte im Bereich Anwendungssoftware sind eine vom Nutzer wählbare Bildschirmauflösung, die Nutzung des mobilen Internets über WLAN, die verstärkte Nutzung von mobilen Apps, eine effizientere Entwicklung von webbasierten Anwendungen, skalierbare Software und die Nutzung von Open Source Software (Vgl. Prof. Dr. Lorenz Hilty, 2015, S. 32 ff.).

Abschließend wurden in der Studie Handlungsempfehlungen gegeben, welche als Inspiration der Fragestellung dieser Bachelorarbeit dienten. So wurde eine regelmäßige Datenerhebung von Webanwendungen empfohlen, um den Ressourcenbedarf zu erfassen (Vgl. Prof. Dr. Lorenz Hilty, 2015, S. 56 ff.).

3.2 Energy Efficiency across Programming Languages

Eine weitere, häufig gestellte Untersuchungsfrage in Bezug auf Green Software ist, wie man den Energieverbrauch von Software ermitteln kann. In der folgenden Studie wurden die Laufzeit, der Speicherverbrauch und der Energieverbrauch von 27 bekannten Programmiersprachen miteinander verglichen. Anschließend wurden die erhobenen Daten in Relation zueinander gebracht, um mögliche Abhängigkeiten zu identifizieren (Rui Pereira, 2017).

Zu Beginn der Studie haben sich die Forschenden die Frage gestellt, ob eine schnelle Programmiersprache auch gleichzeitig eine energieeffiziente Programmiersprache ist. Ein positives Ergebnis auf diese Frage würde bedeuten, dass die Optimierung der Geschwindigkeit auch eine Optimierung des Energieverbrauchs nach sich zieht (Rui Pereira, 2017). Die Gleichung *Energy = Zeit x Leistung* zeigt jedoch, dass der Energieverbrauch nicht nur von der Ausführungszeit abhängt, sondern auch von der Leistungsfähigkeit des Systems beeinflusst wird (Rui Pereira, 2016) (Luís Gabriel Lima, 2016).

Die Messung des Energieverbrauchs von verschiedenen Programmiersprachen war im zweiten Teil der Studie das zentrale Ziel. Hierzu wurde zur Testdurchführung auf das *The Computer Language Benchmarks Game* zurückgegriffen. Das CLBG beinhaltet ein Set an bekannten Programmierproblemen, welche die Stärken und Schwächen einer Programmiersprache aufdecken können (Gouy, 2008). Zur Datenerhebung wurde das *Intel Running Average Power Limit* Tool verwendet, welches in der Lage ist, genaue Energieabschätzungen des Systems auf feingranularer Ebene zu liefern (Vgl. Rui Pereira, 2017, S. 258 ff.).

Im Analyseteil der Studie wurden schließlich verschiedene Fragen hinsichtlich der Effizienz von schnellen Programmiersprachen, der Relation von Speicherbedarf zu Energieverbrauch und der Auswahl der besten Programmiersprache hinsichtlich der untersuchten Metriken beantwortet (Vgl. Rui Pereira, 2017, S. 260 ff.).

Die Ergebnisse zeigten, dass kompilierbare Programmiersprachen die schnellsten und energieeffizientesten sind. Durchschnittlich verbrauchen kompilierbare Programmiersprachen 120 J, virtuelle Maschinen 576 J und Interpretersprachen 2365 J. Auch bei der Ausführungszeit ist diese Tendenz zu erkennen. Zur Ausführung des Codes brauchte eine kompilierbare Programmiersprache im Durchschnitt 5s, eine virtuelle Maschine 20s und eine Interpretersprache 87s (Vgl. Rui Pereira, 2017, S. 260 ff.).

Mit durchschnittlich 88,94 % macht der CPU-basierte Energieverbrauch den größten Teil des Gesamtverbrauchs aus, wobei der restliche Anteil auf den DRAM entfallen ist. Während der durchschnittliche Gesamtverbrauch für diese drei Übersetzungstypen sehr unterschiedlich war, erschien das Verhältnis zwischen CPU- und DRAM-basiertem Energieverbrauch gleich zu sein. Wie in der Studie abschließend erwähnt, könnte dies darauf hindeuten, dass die Optimierung eines Programms zur Verringerung des CPU-basierten Energieverbrauchs auch den

DRAM-basierten Verbrauch verringert. Es wurde zudem beobachtet, dass die Messwerte bei den Interpretersprachen (min. 81,57 %, max. 92,90 %) stärker variieren als bei kompilierten Sprachen (min. 85,27 %, max. 91,75 %) oder virtuellen Sprachen (min. 86,10 %, max. 92,43 %) (Vgl. Rui Pereira, 2017, S. 260 ff.).

Abschließend wurde auf Grundlage der erhaltenen Daten die Auswirkung des genutzten Speichers auf den Energieverbrauch untersucht. Die Analyse zwischen dem Energieverbrauch des DRAMs und der Spitzenauslastung des Speichers zeigte jedoch, dass eine Beziehung zwischen diesen beiden Metriken fast nicht vorhanden war. Der Energieverbrauch des DRAMs hat laut der Studie wenig damit zu tun, wie viel Speicher zu einem bestimmten Zeitpunkt genutzt wird, sondern damit, wie er genutzt wird (Vgl. Rui Pereira, 2017, S. 260 ff.).

4 (Green) Software Engineering

4.1 Software Engineering

Unter Software Engineering versteht man eine technische Disziplin, welche sich mit allen Aspekten der Softwareerstellung befasst. Angefangen bei der Konzeption über den Betrieb bis hin zur Wartung werden verschiedene Aspekte berücksichtigt (Vgl. Sommerville, 2018, S. 27) (Vgl. IEEE, 1990, S. 69).

4.2 Green Software Engineering

„Green in Software Engineering“ lässt sich in den Teilbereich „Green in Software“ verorten, welcher wiederum ein Teilbereich von „Green Software“ ist. Dieser Bereich umfasst die Praktiken und Techniken, welche im klassischen Software Engineering angewendet werden und erweitert diese um die Einbeziehung von Nachhaltigkeitsaspekten. Die Entwicklung, der Betrieb und die Wartung von Software werden so durchgeführt, dass ein Green Software Produkt entsteht.

Green Software kann in drei verschiedene Kategorien unterteilt werden. Die Produktion von Green Software, die Produktion von Software mit positiven Einfluss auf die Umwelt und die Verringerung der Emissionen während des Entwicklungsprozesses dieser Software (K., 2013) (Bill Tomlinson, 2011).

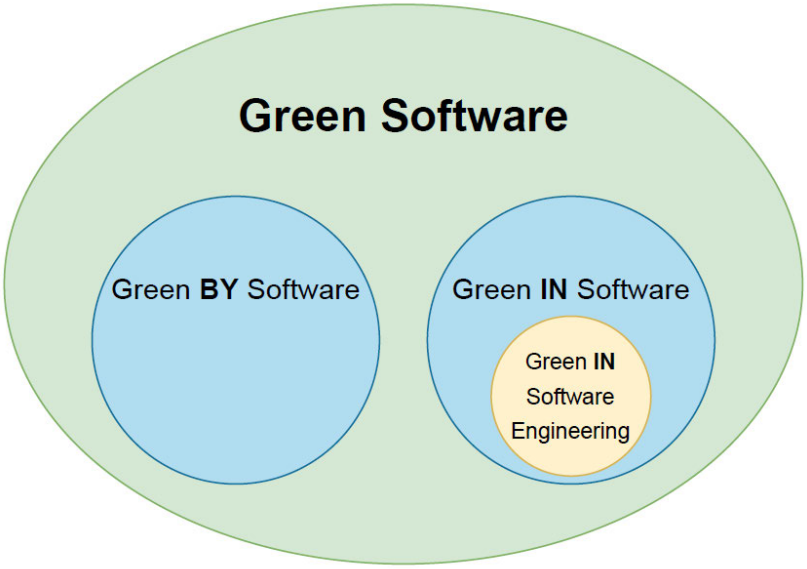

Abbildung 3: Verortung „Green in Software Engineering“
Quelle: In Anlehnung an (Vgl. Calero, 2015, S. 20)

Da es sich bei Green Software Engineering um ein junges Forschungsgebiet handelt (Calero, et al., 2019), gilt es im ersten Schritt das Bewusstsein für die Entwicklung nachhaltiger Software zu stärken. Dies kann einhergehen mit einer Verankerung dieses Themas in den Lehrplan von SchülerInnen und Studierenden sowie die Weiterbildung etablierter Software-EntwicklerInnen. Im Anschluss könnte man über weiterführende Maßnahmen, wie z. B. die Kennzeichnung des Energiebedarfs von Software und Apps nachdenken. Sobald sich das Bewusstsein für nachhaltige Software bei den Konsumierenden verankert hat, kann davon ausgegangen werden, dass dies Auswirkungen auf das gesamte Ökosystem haben wird und Softwareunternehmen sich verpflichtet fühlen, diese Aspekte in ihre Entwicklungsprozesse zu integrieren.

4.2.1 Definitionen

Die Grundidee jeder Software ist es, Tätigkeiten zu vereinfachen und Probleme für den Menschen zu lösen. Da auch Software seinen Teil zum Klimawandel beiträgt, ist es erforderlich sich auch bei der Entwicklung die Frage der Nachhaltigkeit zu stellen und Prozesse zu überdenken. Zum Verständnis der grundlegenden Anforderungen an grüne und nachhaltige Software sind die folgenden Definitionen erforderlich.

Definition: Grüne und nachhaltige Software

> [Green and Sustainable Software] is software, whose direct and indirect negative impacts on economy, society, human beings, and environment that result from development, deployment, and usage of the software are minimal and/or which has a positive effect on sustainable development (Stefan Naumann, 2011, S. 296).

Die Erreichung eines grünen und nachhaltigen Softwareprodukts kann nur dann erreicht werden, wenn sich die beteiligten Organisationen über die Auswirkungen eines nachhaltigen Produkts bewusst sind. Nachhaltigkeitsaspekte sollten in Maßnahmen und spezifischen Anforderungen integriert werden, um den beteiligten Akteuren die Umsetzung zu erleichtern. Ein nachhaltiger Entwicklungsprozess berücksichtigt Umweltauswirkungen während des gesamten Lebenszyklus der Software und verfolgt Nachhaltigkeitsziele (Vgl. Calero, 2015, S. 64).

Definition: Grünes und nachhaltiges Software Engineering

> Green and Sustainable Software Engineering is the art of developing green and sustainable software with a green and sustainable software engineering process. Therefore, it is the art of defining and developing software products in a way, so that the negative and positive impacts on sustainable development that result and/or are expected to result from the software product over its whole life cycle are continuously assessed, documented, and used for a further optimization of the software product (Stefan Naumann, 2011, S. 296).

Im Zentrum der Definitionen für grüne und nachhaltige Software sowie für grünes und nachhaltiges Software Engineering stehen Produktlebenszyklen und deren Analyse. Es ist das Ziel, die negativen ökologischen, sozialen und wirtschaftlichen Auswirkungen des Softwareprodukts über den gesamten Lebenszyklus so gering wie möglich zu halten. Deutlich wird dies, wenn man die Effekte erster Ordnung betrachtet, die sogenannten Angebotseffekte. Dazu gehören Leistungsanforderungen, Netzbandbreite, Hardwareanforderungen oder die Verpackung des Produkts, welche die natürlichen Ressourcen und den Energieverbrauch beeinflussen (Vgl. Calero, 2015, S. 64).

Calero et al. führen zudem die Nutzungseffekte als Effekte zweiter Ordnung an, die sich aus dem Einsatz von IKT im Lebenszyklus anderer Dienstleistungen und Produkte ergeben. IT-Komponenten spielen heutzutage eine essenzielle Rolle für eine Vielzahl von Produkten bei der Umsetzung von Dienstleistungen. Effekte zweiter Ordnung wie Produktdesign,

Produktionsprozess und Abfallentsorgung, bei denen IT-Komponenten zum Einsatz kommen, sind schwieriger abzuschätzen als die Effekte erster Ordnung (Vgl. Calero, 2015, S. 65).

Die Effekte dritter Ordnung sind wiederum schwieriger zu beurteilen. Hier kommt der sogenannte Rebound-Effekt zum Tragen, welcher durch Optimierung Ressourcen freisetzt. Durch eine verstärkte Nutzung dieser neuen Ressourcen kann es schließlich zu einer Überkompensation kommen, wodurch sich die ursprünglichen Dimensionen verändern. Bei Aktivitäten von Green IT handelt es sich in erster Linie um Effekte erster Ordnung, die Angebotseffekte. Demgegenüber werden mit Green by IT die Effekte zweiter und dritter Ordnung, Nutzungseffekte und systematische Effekte betrachtet (Vgl. Calero, 2015, S. 65).

Dass systematisch eingesetzte Software zu einer Verbesserung der Energieeffizienz beitragen kann, zeigen Energiemanagementsysteme, mit deren Hilfe Energieeinsparpotenziale identifiziert und genutzt werden können (Vgl. Umweltbundesamt, 2021, S. 52). Diese Systeme können neben einer Steigerung der Energieeffizienz zudem dazu beitragen, dass sich Verhaltensweisen der Beschäftigten ändern und durch Delegation der Verantwortlichkeiten unternehmensweit Kompetenzen auf dem Gebiet des Energiemanagements entwickeln.

Ein Negativbeispiel für Green by IT ist die sich verändernde Arbeitskultur, welche durch die Corona-Pandemie beeinflusst wurde. Die Telearbeit birgt viele Vorteile wie z. B. Zeit- und Geldersparnis sowie eine höhere Flexibilität. Langfristig sollen sich die eingesparten Emissionen jedoch bedingt durch höhere Gebäudeemissionen und längere Anfahrtswege bei weniger zentralen Wohnorten ausgleichen bzw. sogar erhöhen. Aus diesem Grund sind Innovationen, welche sich auf alle Bereiche des Lebens auswirken, erforderlich. (Vgl. Marz, 2022, S. 18).

4.2.2 Das GREENSOFT-Modell

Das GREENSOFT-Modell ist ein Referenzmodell für die Entwicklung von grüner und nachhaltiger Software. Die Erstellung, Wartung und Nutzung von Software soll dadurch auf nachhaltige Weise erleichtert werden (Vgl. Stefan Naumann, 2011, S. 296). Das Modell (siehe Abbildung 4) umfasst ein ganzheitliches Lebenszyklusmodell für Softwareprodukte und dazugehörige Prozessmodelle. Des Weiteren umfasst es Empfehlungen und Werkzeuge zur Unterstützung der Entwicklung sowie Beschaffung und Nutzung von Softwareprodukten, welche mit Zielen einer nachhaltigen Entwicklung vereinbar sind.

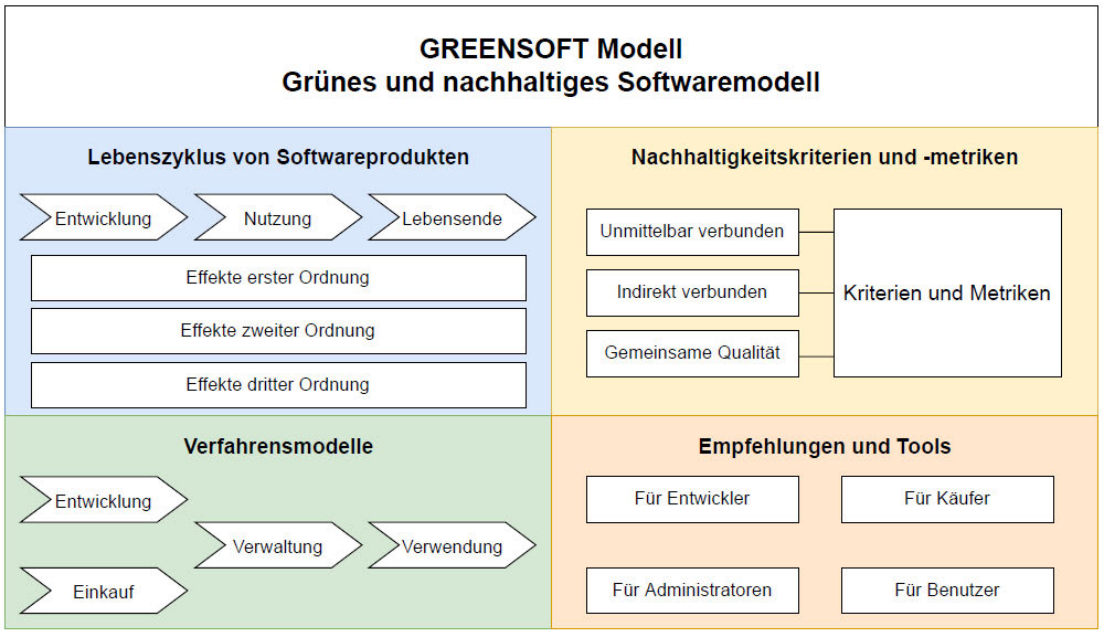

Abbildung 4: Das GREENSOFT Modell
Quelle: In Anlehnung an (Stefan Naumann, 2011, S. 297)

Das Referenzmodell ist in die vier Bereiche *Lebenszyklus von Softwareprodukten*, *Nachhaltigkeitskriterien und -metriken*, *Verfahrensmodelle* sowie *Empfehlungen und Tools* aufgeteilt.

Der erste Bereich des Modells – *Lebenszyklen von Softwareprodukten* – orientiert sich im Gegensatz zu traditionellen Lebenszyklen am Leitgedanken Life Cycle Thinking. Das Ziel von LCT ist es, die ökologische, soziale, menschliche und ökonomische Verträglichkeit eines Produktes während seines gesamten Lebenszyklus zu bewerten. Von der Entwicklungsphase bis hin zur Entsorgung können somit neue Erkenntnisse gewonnen werden, welche für eine Optimierung des eigenen Produkts oder den Vergleich mit einem Konkurrenzprodukt herangezogen werden können (Vgl. Stefan Naumann, 2011, S. 296).

Der zweite Bereich – *Nachhaltigkeitskriterien und -metriken* – umfasst allgemeine Metriken und Kriterien zur Messung der Softwarequalität. Dadurch lässt sich eine Klassifizierung zur Bewertung der Nachhaltigkeit eines Softwareprodukts durchführen. Hierbei wird zwischen direkten und indirekten Kriterien und Metriken unterschieden. Direkte Auswirkungen (Effekte erster Ordnung) lassen sich leichter messen als indirekte (Effekte zweiter und dritter Ordnung) (Vgl. Stefan Naumann, 2011, S. 298).

Im dritten Bereich geht es um die Klassifizierung von Verfahrensmodellen, die die Beschaffung, Entwicklung und Wartung von Software und IT-Systemen sowie deren Anwendersupport abdecken. Dabei soll während des Entwicklungsprozesses von Software systematisch die Berücksichtigung von Nachhaltigkeitsaspekten evaluiert werden (Vgl. Stefan Naumann, 2011, S. 299).

Im Bereich *Empfehlungen und Tools* werden unterstützende Werkzeuge und Techniken im Allgemeinen empfohlen, um die Entwicklung, den Kauf, die Verwaltung oder die Nutzung von Softwareprodukten nachhaltig zu gestalten (Vgl. Stefan Naumann, 2011, S. 301).

Relevanz des GREENSOFT-Modells

Das im Jahr 2011 entwickelte GREENSOFT-Referenzmodell stößt selbst nach mehr als zehn Jahren seit der Veröffentlichung auf große Zustimmung innerhalb des Forschungsgebiets und sogar darüber hinaus.

Im Forschungsbericht *Sustainability in Software Engineering - A Systematic Mapping* von 2016 wird das GREENSOFT-Modell als Startpunkt einer Entwicklung weiterer nachhaltiger Softwaremodelle und Richtlinien beschrieben (Vgl. Kristina Rakneberg Berntsen, 2016, S. 24).

Eine Vielzahl von Studien und Literaturwerke beziehen sich auf das Modell oder nutzen es für weiterführende Forschungen (Nelly Condori-Fernandez, 2019) (Siti Rohana Ahmad Ibrahim, 2019) (Weng, 2021) (Harika Jayanthi, 2021) (Deneckère, 2020) (Calero, et al., 2019).

Im *Green Model for Sustainable Software Engineering* von Mahmoud und Ahmad beziehen sich die Schreibenden auf das von Naumann et al. (Stefan Naumann, 2011) entwickelte Referenzmodell. Es wird explizit hervorgehoben, dass dieses Modell nicht nur auf die ökologische Nachhaltigkeit beschränkt ist, sondern auch Fragen der menschlichen und sozialen Nachhaltigkeit umfasst (Vgl. Sara S. Mahmoud, 2013, S. 57).

4.2.3 Life Cycle von Software

Der Life Cycle von Software beschreibt den Entwicklungsprozess mit dem Ziel der Bereitstellung für einen beliebigen Stakeholder. Wenn das Ziel jedoch um die Berücksichtigung von ökologischen, sozialen, menschlichen und wirtschaftlichen Auswirkungen erweitert wird,

spricht man vom LCT. Dabei werden neben dem Entwicklungsprozess auch die Prozesse von der Rohstoffgewinnung über die Nutzung bis hin zur Entsorgung betrachtet, um ein optimiertes Produkt zu erzeugen.

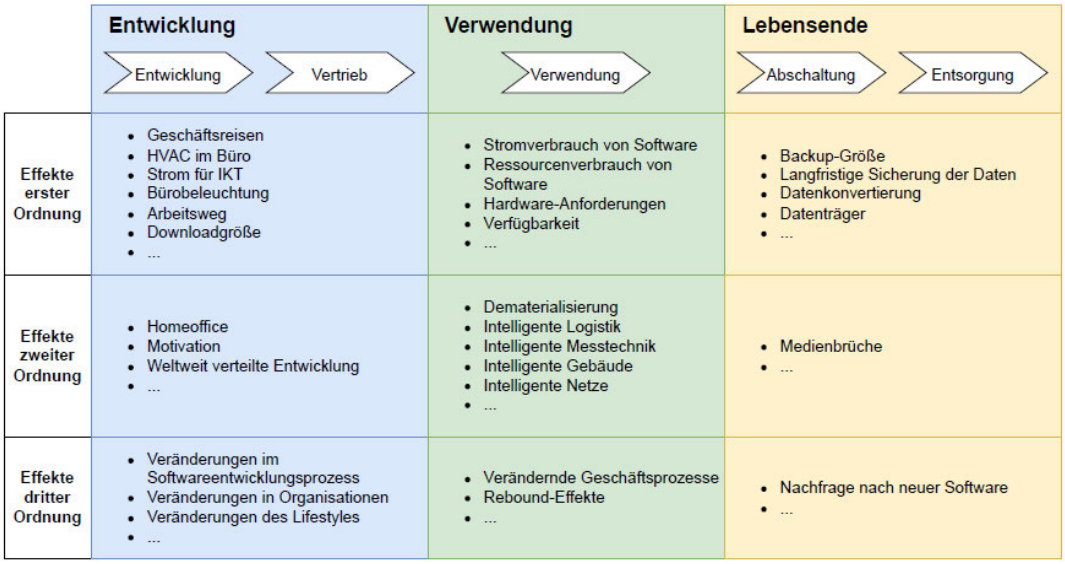

Abbildung 5: LCT von Softwareprodukten
Quelle: In Anlehnung an (Stefan Naumann, 2011, S. 297)

Entwicklungsphase

Während der Entwicklungsphase werden die direkten Auswirkungen des Softwareentwicklungsprozesses, sowie indirekte Aktivitäten berücksichtigt. Zu berücksichtigende Umweltauswirkungen sind z. B. die elektrische Energie und natürliche Ressourcen, die für die Arbeitsplätze und den Betrieb der IT-Infrastruktur benötigt werden. Des Weiteren wird der Energieverbrauch für den Arbeitsweg oder Geschäftsreisen berücksichtigt sowie die verbrauchte Energie beim Herunterladen von Daten. Durch flexible Arbeitsmodelle, wie z. B. Homeoffice lassen sich einige dieser Effekte zumindest kurzfristig reduzieren. Veränderungen auf Ebenen der Softwareentwicklung, Organisation oder des eigenen Lifestyles sind in dieser Phase ebenfalls zu berücksichtigen. Die Messung dieser Effekte ist jedoch mit erhöhtem Aufwand verbunden (Vgl. Stefan Naumann, 2011, S. 298).

Vertriebs- und Entsorgungsphase

Die Vertriebsphase berücksichtigt die Umweltauswirkungen während des Vertriebs des Softwareproduktes. In dieser Phase werden Umweltauswirkungen von z. B. gedruckten Handbüchern, gewählten Transportmitteln, Verpackungsdesigns (Kunststoff, biologisch abbaubares Material) oder Datenträger, z. B. USB-Sticks oder CDs/DVDs evaluiert. Üblicherweise wird Software heutzutage als Download zur Verfügung gestellt, wobei die Größe der Datei sowie die elektrischen Energie- und Materialressourcen für den Betrieb berücksichtigt werden. In der Entsorgungsphase werden Auswirkungen, die sich aus der Entsorgung und dem Recycling der genannten Materialien ergeben, berücksichtigt (Vgl. Stefan Naumann, 2011, S. 298).

Nutzungsphase

Die Nutzungsphase berücksichtigt Auswirkungen, die aus dem Einsatz, der Nutzung und Wartung des Produkts resultieren. Die Wartung umschließt in diesem Fall die Administration von Software in Organisationen und kann Tätigkeiten wie z. B. die Installation, das Patchen, die Konfiguration oder die Schulung von Mitarbeitenden beinhalten. Durch die Schulung der Mitarbeitenden können Effekte wie eine effizientere Nutzung der Software und somit eine Senkung des Energiebedarfs erzielt werden (Vgl. Stefan Naumann, 2011, S. 298). Eine geeignete Update-Strategie könnte zudem dazu beitragen, den Ressourcenverbrauch zu senken. So könnte die Installation von Updates abteilungsweise erfolgen, um Probleme frühzeitig erkennen und beheben zu können.

Bei der Einführung moderner Software ist in der Regel der Einsatz leistungsfähigerer Hardware erforderlich. Dies sollte deshalb bei der Wahl einer neuen Software berücksichtigt werden, um einen Ressourcen- und Energieverbrauch bei der Herstellung neuer Hardware und der Entsorgung der aktuellen entgegenzuwirken (Hilty, 2008) (Umweltbundesamt, 2021).

Die Auswirkungen zweiter und dritter Ordnung, die sich aus der Nutzungsphase ergeben, hängen vom Zweck des Produkts ab. Künstliche Intelligenz trägt bereits dazu bei, Produktionsprozesse effizienter zu gestalten und somit Ressourcen und Energie zu sparen (VDI Zentrums Ressourceneffizienz, 2021).

Deaktivierungsphase

Bei der Außerbetriebnahme eines Softwareproduktes ist es meist notwendig, vorhandene Daten zu konvertieren. Diese Konvertierung ist erforderlich, damit das nachfolgende Produkt die Daten aus dem Altsystem verarbeiten kann. Unter Umständen ist auch die digitale Archivierung von Daten sinnvoll. Zusätzlich muss der erforderliche Speicherbedarf für die Sicherung der Daten berücksichtigt werden (Vgl. Stefan Naumann, 2011, S. 298).

4.2.4 Nachhaltiges Verfahrensmodell

Um grüne und nachhaltige Software zu produzieren, muss während des Entwicklungsprozesses der gesamte Life Cycle eines Softwareprodukts sowie die Umstände, unter denen es produziert wird, berücksichtigt werden.

Das Referenzmodell von Neumann et al. umfasst zusätzlich ein Verfahrensmodell für den Entwicklungsprozess von Software (Markus Dick, 2010). Dieses Verfahrensmodell umfasst keinen vollständigen Softwareentwicklungsprozess, sondern gibt den Entwicklern Werkzeuge an die Hand, bestehende Prozesse zu erweitern. Vorgeschlagen werden Nachhaltigkeits-Reviews und -Previews, eine Prozessbewertung, das Dokumentieren der Nachhaltigkeitsaspekte des Produkts und eine Nachhaltigkeits-Retrospektive. Die Erweiterungen bilden einen kontinuierlichen Verbesserungszyklus, der sich mit Nachhaltigkeitsthemen befasst. Durch die Prozessbewertung kann der Entwicklungsprozess optimiert werden, während die Reviews und Previews helfen sollen, die Nachhaltigkeit des entstehenden Produkts zu erhöhen. Um eine Retrospektive erweitert, ergibt sich somit am Ende der Entwicklungsphase eine vollständige Abdeckung der Nachhaltigkeitserweiterungen.

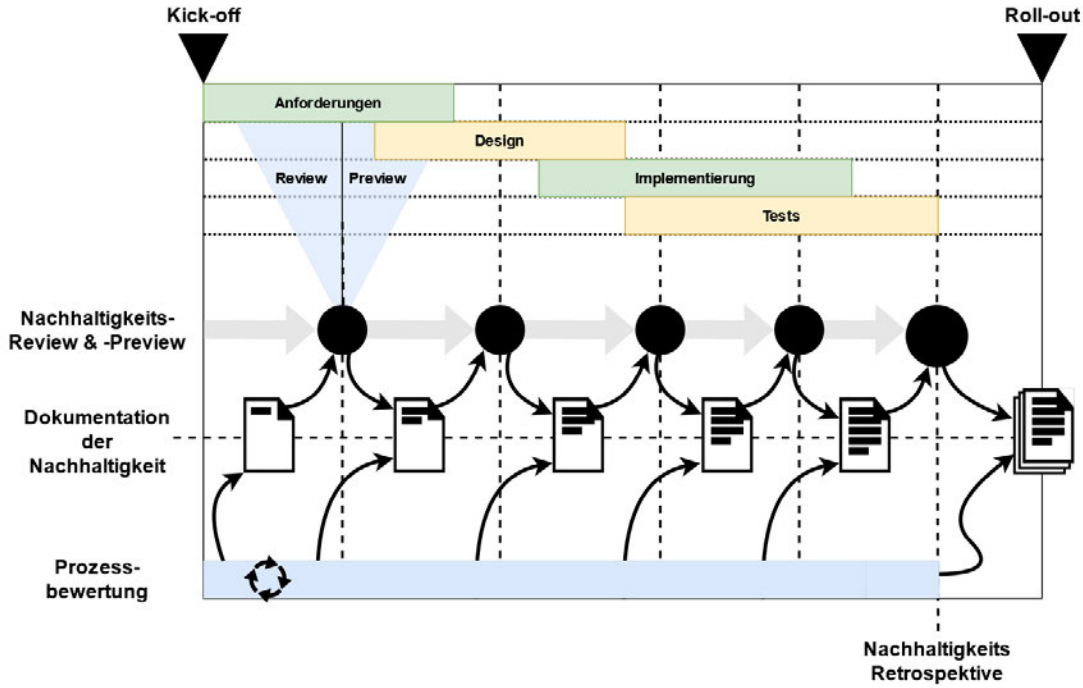

Abbildung 6: Beispiel eines erweiterten Verfahrensmodells
Quelle: In Anlehnung an (Stefan Naumann, 2011, S. 300)

Rollen

Zur Umsetzung des Verfahrensmodells werden drei Rollen empfohlen:

- Das Entwicklungsteam
- Eine Kundschaftsvertretung
- Eine für Nachhaltigkeit beauftragte Person

Das Entwicklungsteam treibt die Entwicklung des Produkts voran und ergänzt diesen Vorgang durch die Nachhaltigkeits-Reviews und Previews sowie die Dokumentation der Nachhaltigkeitsaspekte. Zum Abschluss des Prozesses führt das Team eine Nachhaltigkeits-Retrospektive durch (Vgl. Calero, 2015, S. 71).

Die Kundschaftsvertretung ist für die Anforderungen und Erwartungen der Kundschaft an die Nachhaltigkeit des Produkts verantwortlich. Außerdem plant diese Person die Zwischenpräsentation und die finale Abgabe des Abschlussberichts (Vgl. Calero, 2015, S. 71).

Eine für Nachhaltigkeit beauftragte Person ist für die Organisation rund um die Nachhaltigkeitsaufgaben und die Planung der Termine des Entwicklungsteams verantwortlich. Des Weiteren übernimmt diese Person die Erstellung des Nachhaltigkeitsberichts und die Pflege der Dokumentation (Vgl. Calero, 2015, S. 71).

Kick-off

Das Entwicklungsteam sollte sich vor Beginn des Projekts bewusst darüber sein, wie Umweltauswirkungen durch die entstehende Software minimiert werden können. Hierfür sollte eine einheitliche Basis geschaffen werden.

Die GSF hat zehn Empfehlungen verfasst, die bei der Entwicklung von Green Software unterstützen sollen (Murugesan, 2021):

1. Fokussierung auf Funktionen mit höherem Stromverbrauch und häufigen Anwendungsfällen
2. Reduzierung der Datennutzung und Einführung eines Cache-Speichers, sowie die Nutzung von kleineren Medien und Bildern
3. Ungenutzte Funktionen zur Verbesserung der Energieeffizienz entfernen
4. Entfernen von unnötigen Schleifen, die ihren Zweck nicht erfüllen
5. Anpassung der Software an den Energiesparmodus des Betriebsgeräts
6. Begrenzung der Berechnungsgenauigkeit auf des gewünschte Niveau, das den Anforderungen entspricht, z.B. bei Geodaten
7. Überwachung des Energieverbrauchs der Anwendung in Echtzeit, um Optimierungspotenziale zu identifizieren
8. Auswahl einer energieeffizienten Programmiersprache, wie in einer Studie von Pereira et al. bereits untersucht (Rui Pereira, 2017)
9. Codeoptimierung durch KI zu Nutze machen, um doppelten Aufwand zu reduzieren und somit die Effizienz zu steigern (Roy Schwartz, 2019)
10. Nutzung der dynamischen Codeanalyse zur Überwachung des Stromverbrauchs

Nachhaltigkeits-Review und -Preview

Die Nachhaltigkeits-Reviews und Previews werden hauptsächlich vom Entwicklungsteam durchgeführt. Beim Review wird die geleistete Arbeit betrachtet, während die Preview, auf Grundlage der im Review getroffenen Entscheidungen versucht einen Ausblick auf künftige Implementierungen zu geben. Neben einer statischen sollte in jedem Review eine dynamische Codeanalyse durchgeführt werden, um in Echtzeit weitere Energieeinsparungen identifizieren zu können (Brain, 2010). Somit bilden diese zwei Werkzeuge einen kontinuierlichen Verbesserungszyklus innerhalb einer Iteration.

Die Ergebnisse sollten in einer Zwischenpräsentation am Ende einer Iteration vorgestellt und in die Dokumentation übernommen werden, um später die erfolgreiche Umsetzung der Maßnahmen prüfen zu können (Vgl. Calero, 2015, S. 72).

Prozessbewertung

Die Prozessbewertung bewertet kontinuierlich die positiven und negativen Auswirkungen der nachhaltigen Entwicklung. Die Verantwortung dieser Aufgabe liegt bei der für Nachhaltigkeit zuständigen Person.

Ziel ist es, die Daten so detailliert zu erfassen, dass eine fortlaufende Berechnung zu einer Kalkulation des CO2-Fußabdrucks führt. Relevante Daten hierfür lassen sich aus dem Life Cycle der Softwareprodukte ableiten. Dazu gehören neben der Energie für das Büro vor allem die Energie für die IT-Infrastruktur. Die gesammelten Daten werden in der Dokumentation der Nachhaltigkeit erfasst (Vgl. Calero, 2015, S. 73).

Nachhaltigkeits-Retrospektive

Vor dem Projektabschluss findet die Nachhaltigkeits-Retrospektive statt. Im Rahmen einer Sitzung wird vom Entwicklungsteam und der für Nachhaltigkeit zuständigen Person der Entwicklungsprozess reflektiert, um Optimierungen für zukünftige Projekte zu identifizieren. Die Anregungen für eine Diskussion kommen hauptsächlich aus der Nachhaltigkeits-Dokumentation. Eine weitere Diskussionsgrundlage können mit dem Projekt verbundene Quellen liefern, die einen zusätzlichen Nutzen für das gesamte Team darstellen. Die Ergebnisse aus der

Retrospektive sollten in einer Wissensdatenbank festgehalten und zugänglich gemacht werden (Vgl. Calero, 2015, S. 74).

Dokumentation der Nachhaltigkeit

Die Dokumentation der Nachhaltigkeit ist ein strukturiertes Dokument, in dem alle Daten und Ergebnisse der Prozessbewertung sowie Probleme und Lösungen festgehalten werden.

E. Kern et al. empfiehlt, das Dokument in die folgenden drei Abschnitte zu gliedern (Calero, 2015, S. 74):

1. Erfasste und berechnete Umweltbelastungen und -auswirkungen aus Prozessbewertung
2. Maßnahmen, die in Reviews und Previews nach folgendem Muster getroffen wurden:
    a. Ausgangssituation (Daten aus Messungen, Softwaretests, etc.)
    b. Getroffene Entscheidung oder Maßnahme
    c. Überprüfung der Wirksamkeit der Entscheidungen/Maßnahmen
3. Anhang mit Daten und deren Herkunft, die zur Berechnung der Zahlen verwendet wurden, z.B. Energieverbrauch, CO2-Emissionen pro kWh, etc.

4.2.5 Integration von GREENSOFT-Aspekten in bestehende Vorgehensmodelle

Im Folgenden werden die zuvor beschriebenen Verfahrensoptimierungen des GREENSOFT-Referenzmodells in bestehende Prozessmodelle integriert werden. Hierzu wird der bestehende Prozess um den Nachhaltigkeitsaspekt erweitert.

GREENSOFT-Aspekte in Scrum integrieren

Bei Scrum handelt es sich um ein agiles, iteratives Vorgehen zur Softwareentwicklung, bei dem versucht wird, in jeder Iteration ein potenziell auslieferbares Softwareinkrement zu produzieren. Jeder Sprint beginnt mit der Sprint-Planung und endet mit einem Sprint-Review sowie einer Sprint-Retrospektive. Der Product Owner vertritt die Interessen aller Stakeholder und verhandelt mit dem Scrum-Team in Sprint-Planungssitzungen die Entwicklungsziele für den

folgenden Sprint. Diese Ziele werden in den Sprint-Reviews zur Beurteilung der Sprint-Ziele verwendet (Kent Beck, 2001).

Rollenverteilung

Die Rollen des GREENSOFT-Modells lassen sich ohne Veränderung der vorhandenen Teamstruktur in Scrum integrieren. Der Product Owner könne die Rolle der Kundschaftsvertretung übernehmen und das Scrum-Team als Entwicklungsteam agieren. Eine für Nachhaltigkeit beauftragte Person entspricht keiner vorhandenen Rolle in Scrum, weshalb sich hier die Auswahl einer Person des Scrum-Teams anbietet.

Kick-off

Zu Beginn jeder Iteration findet in Scrum ein Sprint-Planungstreffen statt. Dieses Treffen könnte dazu genutzt werden, die Prinzipien einer nachhaltigen Softwareentwicklung zu kommunizieren und im Team zu diskutieren. So kann eine gemeinsame Basis für das Verständnis von Nachhaltigkeit geschaffen werden.

Nachhaltigkeits-Review und -Preview

Eine Iteration entspricht einem Sprint, weshalb das Nachhaltigkeits-Review bzw. -Preview im letzten Drittel des Sprints erfolgen könnte. Die Präsentation der Zwischenergebnisse könnte in das Sprint-Review fallen.

Prozessbewertung

Da die Prozessbewertung einen kontinuierlichen Prozess darstellt, würde diese parallel zu den durchgeführten Sprints laufen.

Nachhaltigkeits-Retrospektive

Eine Nachhaltigkeits-Retrospektive verfolgt ein anderes Ziel als ein klassisches Sprint-Review. Für diese Retrospektive wäre somit ein eigenständiger Termin notwendig.

Dokumentation der Nachhaltigkeit

Die Dokumentation der Nachhaltigkeit ist wie die Prozessbewertung, ein kontinuierlicher Prozess, der parallel zum jeweiligen Sprint durchgeführt wird. Hierfür ist die für Nachhaltigkeit beauftragte Person im gesamten Zeitraum des gesamten Entwicklungsprozesses zuständig.

GREENSOFT-Aspekte in DevOps integrieren

DevOps umfasst diverse Praktiken, Tools und eine Kulturphilosophie zur Integration und Automatisierung der Prozesse von Softwareentwicklungs- und IT-Teams. Der Fokus liegt auf die Einbeziehung von Teams, teamübergreifende Kommunikation und Zusammenarbeit.

Ein DevOps-Team besteht aus Entwickelnden und operativen Teams, die zur Erhöhung der Geschwindigkeit und Qualität der Softwareverteilung zusammenarbeiten. Dadurch, dass die Teams nicht voneinander isoliert sind, können multidisziplinäre Fähigkeiten entstehen (Vgl. Atlassian, o. D.).

Der DevOps-Lebenszyklus besteht aus sechs Phasen (Plan, Build, CI/CD, Monitoring, Operate, Continuous Feedback), welche für die Entwicklung und den Betrieb erforderlich sind. Die Teams sind zu jeder Phase dieses Prozesses im Austausch, um eine konsistente Geschwindigkeit und Qualität zu gewährleisten.

Rollenverteilung

Die Rollen des GREENSOFT-Modells lassen sich wie bei Scrum ohne Veränderung der vorhandenen Teamstruktur integrieren. Eine Person aus dem Operate-Team könne die Rolle der Kundschaftsvertretung übernehmen und das Build-Team als Entwicklungsteam agieren. Eine Person des Monitorings könnte zusätzlich die Nachhaltigkeitsrolle übernehmen.

Kick-off

Zum Start eines neuen Projektes sollte das Kick-off stattfinden, um die Teammitglieder auf eine gemeinsame Basis hinsichtlich der Nachhaltigkeit von Software zu bringen. Da DevOps die Charakteristik verfolgt, ein Produkt über den gesamten Lebenszyklus zu begleiten, ist es

hier gegebenenfalls erforderlich, den Kick-off-Prozess zu wiederholen, falls sich das Team verändert.

Nachhaltigkeits-Review und -Preview

Die bereits integrierte Phase des Feedbacks könnte um die Nachhaltigkeits-Reviews und -Previes erweitert werden. Diese würden somit in regelmäßigen Abständen analog zum Scrum-Sprint durchgeführt werden.

Prozessbewertung

Das DevOps-Monitoring könnte die Prozessbewertung in seine bestehenden Überwachungsaufgaben integrieren. So kann eine Prozessbewertung über den gesamten Lebenszyklus sichergestellt werden.

Nachhaltigkeits-Retrospektive

Die Nachhaltigkeits-Retrospektive könnte bei Fertigstellung eines neuen Releases durchgeführt werden, um auch bei diesem Prozess einen kontinuierlichen Charakter zu schaffen.

Dokumentation der Nachhaltigkeit

Da die Dokumentation der Nachhaltigkeit ebenso einen kontinuierlichen Prozess darstellt, würde diese Tätigkeit auch vom Monitoring übernommen werden.

4.3 Weiterführende Forschung

Das folgende Kapitel gibt einen Überblick über weiterführende Forschungsansätze. Hierzu werden ausgewählte Studien vorgestellt, die im Zusammenhang mit nachhaltiger Softwareentwicklung stehen.

4.3.1 Quality Model Green and Sustainable Software (2013)

Im GREENSOFT-Referenzmodell von 2011 wurde die Erfassung von direkten (Effekte erster Ordnung) und indirekte Auswirkungen (Effekte zweiter und dritter Ordnung) nicht

berücksichtigt (Stefan Naumann, 2011). Im Jahr 2013 wurde das Referenzmodell deshalb um eine weitere Perspektive erweitert. Die Forschung *Quality Model Green and Sustainable Software* legt den Fokus auf die Definition von Kriterien und Metriken zur Klassifikation der vorliegenden Software in Green Software. Die Dimension *Nachhaltigkeitskriterien und -metriken* des GREENSOFT-Modells wurde dabei um die Analyse der Effekte der ersten, zweiten und dritten Ordnung erweitert. Zudem sollen in dem erweiterten Modell soziale und ökologische Aspekte berücksichtigt werden. Das vorgeschlagene Qualitätsmodell berücksichtigt neben Aspekten wie Effizienz z. B. auch Wiederverwendbarkeit, Modifizierbarkeit und Benutzbarkeit (Kern E, 2013).

Darüber hinaus haben die Forschenden folgende Kriterien der ISO/IEC 25010 (ISO, 2011) in die Kategorie *Common Criteria* und *Directly related Criteria* aufgenommen:

- Resource utilization
- Usability
- Portability

4.3.2 Software Product Quality and (Software Product) Greenability (2015)

Calero et al. haben in ihrer Forschung untersucht, ob die ISO/IEC 25010 (ISO, 2011) um den Faktor Nachhaltigkeit erweitert werden muss. Konkret schlagen die Forschenden vor, die Norm um die Dimension Greenability zu erweitern. Durch eine Erweiterung des *Software-Product-Quality-Model* der Norm erhoffen sie sich, den Nachhaltigkeitsfaktor im Software Engineering zu etablieren. Im Wesentlichen wird genannt, die Kriterien *Energieeffizienz, Ressourcenoptimierung, Optimierung der Kapazität* und *Beständigkeit von Software* zu integrieren. Durch das Kriterium *Beständigkeit von Software* soll die Langlebigkeit einer Software gewährleistet werden (Vgl. Calero, 2015, S. 240).

Im weiteren Verlauf der Forschung wird zudem eine Erweiterung des *Quality-in-use* Modells der ISO/IEC 25010 (ISO, 2011) um Nachhaltigkeitskriterien empfohlen. Das Modell könnte um eine sechste Hauptkategorie mit dem Namen *Greenability* erweitert werden und folgende Kriterien beinhalten:

- Optimierung der Effizienz

- Umweltwahrnehmung der Nutzer
- Minimierung der Umweltauswirkungen

4.3.3 Green software requirements and measurement (2015)

Beghoura et al. entwickelten in ihrer Arbeit ein *Profiling-Tool*, womit sich die Qualität und Effizienz einer Software beurteilen lassen. Anhand eines multiplen Regressionsmodells, welches einen überwachten Lernalgorithmus verwendet, untersucht das Tool den Energieverbrauch verschiedener Geräte. Das Tool simuliert verschiedene Bedingungen, die bei der Arbeit auftreten können und versucht basierend darauf, das Energieverbrauchsverhalten zu untersuchen (Vgl. Mohamed Amine Beghoura, 2015, S. 30).

Des Weiteren haben die Autoren Kriterien entwickelt, die Entwicklungsteams dabei unterstützen sollen, die betrachtete Hard- und Software zu differenzieren (Vgl. Mohamed Amine Beghoura, 2015, S. 29):

- *Green computation efficiency* beschreibt die Fähigkeit der Software die geforderte Arbeitslast effizient zu verarbeiten und dabei eine optimale Menge an Energie zu verbrauchen
- *Green data management* beschreibt die Effektivität der implementierten Datenverwaltungsstrategie zur Durchführung von I/O-Operationen bei niedrigem Energieverbrauch
- *Green data communication* beschreibt die Effizienz der Energieverwaltungsstrategie, wenn die Software Daten empfängt und versendet
- *Energy consumption awareness* beschreibt den gesamten Energieverbrauch der Software, um die verschiedenen Verbrauchsniveaus, bei hoher, mittlerer und niedriger Auslastung, zu definieren

4.3.4 Entwicklung und Anwendung von Bewertungsgrundlagen für ressourceneffiziente Software unter Berücksichtigung bestehender Methodik (2018)

Gröger et al. haben im Auftrag des Umweltbundesamtes methodische Grundlagen zur Ermittlung vom Ressourcenverbrauch durch Software erarbeitet. Durch eine Systematisierung der

Software-Eigenschaften innerhalb eines definierten Kriterienkatalogs konnte die wesentliche Wirkung von Software praktikabel strukturiert werden. Mit den Ergebnissen der Forschung war es möglich, Anforderungen an ressourceneffiziente Softwareprodukte zu stellen. Folgende Indikatoren wurden als Vergabekriterien eines Umweltzeichens definiert (Vgl. Jens Gröger, 2018, S. 70):

- Vergabekriterien adressieren die wesentlichen Umweltauswirkungen eines Produktes entlang dessen Produktlebenszyklus
- Kriterien müssen richtungssicher sein, die Erfüllung der Kriterien muss Vorteile für Mensch und Umwelt bieten
- Es werden nur Produkteigenschaften abgefragt, die zur Unterscheidung von Produkten beitragen
- Die Anforderungen müssen mit überprüfbaren Indikatoren hinterlegt sein, die das Kriterium bestätigen
- Zur Quantifizierung der Indikatoren muss eine unabhängige und reproduzierbare Überprüfung ermöglicht werden
- Externe Nachweise müssen gegenüber einer Vergabestelle bestätigt werden

4.3.5 Software Sustainability Model (2018)

Candori-Fernandez et al. haben in ihrer Arbeit *Characterizing the contribution of quality requirements to software sustainability* von 2018 untersucht, welche Qualitätsanforderungen der ISO/IEC 25010 (ISO, 2011) zu ökonomischer, technischer, ökologischer und sozialer Nachhaltigkeit beitragen könnten (Nelly Condori Fernandez, 2018-1). Die gewonnenen Erkenntnisse haben zur Entwicklung des *Software sustainability Quality Model* beigetragen, welches aus einer tabellarischen Zuordnung der ISO-25010-Kriterien besteht. Diese wurden in die Dimensionen soziale, technische, ökologische und ökonomische Nachhaltigkeit gruppiert (Nelly Condori-Fernandezab, 2018-2) (Nelly Condori-Fernandez, 2019). Folgende Kriterien der ISO 25010 wurden für die Entwicklung von Green Software als relevant betrachtet (Nelly Condori-Fernandez, 2020):

Eine grüne Zelle symbolisiert die Erfüllung des Nachhaltigkeitskriteriums, eine weiße Zelle, die Nicht-Erfüllung.

Kriterium	Technische Nachhaltigkeit	Soziale Nachhaltigkeit	Ökologische Nachhaltigkeit	Ökonomische Nachhaltigkeit
Compatibility				
Effectiveness				
Efficiency				
Maintainability				
Trust				
Usability				
Usefulness				

Tabelle 1: Kriterien des Sustainability-quality Modells

4.3.6 5Ws of Green and Sustainable Software (2019)

Calero et al. haben sich in ihren Forschungen über mehrere Jahre mit dem Thema grüne und nachhaltige Software beschäftigt. Darauf aufbauend haben die Forschenden im Rahmen der Studie *5Ws of Green and Sustainable Software* eine Momentaufnahme dieses Themas erfasst. Hierfür wurden die 5 Ws (warum, wann, wer, wo und was) von grüner und nachhaltiger Software definiert und beantwortet. Insgesamt wurden 542 Artikel, im Zeitraum von 2000 bis 2018 zu dem Thema untersucht. Das Ergebnis ist wie folgt (Calero, et al., 2019):

- *Warum*: Grüne und nachhaltige Software ist ein sehr aktiver Bereich der Forschung
- *Wann*: Es handelt sich um einen interaktiven Bereich mit einer großen Anzahl von multinationalen Veröffentlichungen
- *Wer*: Einige Teile der Welt arbeiten noch nicht an diesem Thema. Die USA, Europa und Kanada sind sehr aktiv
- *Wo*: Obwohl die Erforschung von Green Hardware/Green IT ein Trend war, hat grüne und nachhaltige Software einen guten Reifegrad erreicht und ist jetzt ein stabiler Forschungszweig

- *Was*: Die am häufigsten verwendeten Schlüsselwörter im Zusammenhang mit Aspekten grüner und nachhaltiger Software sind *Sustainable Development*, *Green Software*, *Green IT*, *Software Sustainability*, *Energy Consumption*, and *Energy Efficiency*

4.3.7 A Green Model for Sustainable Software Engineering (2013)

Mahmoud und Ahmad haben in Anlehnung an das GREENSOFT-Referenzmodell (Stefan Naumann, 2011) ihre Arbeit *A Green Model for Sustainable Software Engineering* im Jahr 2013 veröffentlicht. Dieses Green Software Modell basiert auf zwei Ebenen.

Auf der ersten Ebene wurde ein nachhaltiger Software Engineering Prozess vorgeschlagen, der auf Grundlage der sequenziellen, iterativen und agilen Methoden entwickelt wurde. Innerhalb dieser Ebene soll jede Entwicklungsphase durch grüne Prozesse sowie bei Bedarf grüne Richtlinien ökologisch und nachhaltig gestaltet werden. Hierzu wurden jeder Entwicklungsphase relevante Metriken zur Steigerung der Umweltfreundlichkeit hinzugefügt. Folgende Ziele werden bei nicht-funktionalen Anforderungen empfohlen (Vgl. Sara S. Mahmoud, 2013, S. 60):

- Verwendung von umweltverträglichen Produkten für die Entwicklung
- Reduzierung der Transportmittel und stattdessen Nutzung des Internets für die Kommunikation
- Nutzung von Cloud Computing im Sinne von IaaS
- Einsatz von serviceorientierter Software
- Verzicht auf Leistung oberhalb einer bestimmten Grenze für die Energieeffizienz
- Betrieb des Systems auf Computern mit Leistungsprofilen
- Verringerung der Anzahl unnötiger Aktivitäten im System
- Verwendung eines wiederverwendbaren Systems

Die zweite Ebene besteht aus den Ansätzen, die von der Software selbst verfolgt werden, um zum Green Computing beizutragen. Diese Ansätze wurden in fünf Hauptkategorien unterteilt. Abschließend wurden beide Ebenen miteinander in Beziehung gesetzt. Dies sollte verdeutlichen, wie Green-Computing-Konzepte in den einzelnen Phasen des Software Engineering Prozesses verwendet werden können (Vgl. Sara S. Mahmoud, 2013, S. 67).

5 Frameworks für das Benchmarking

5.1 Spring Boot

Spring Boot ist ein „Konvention vor Konfiguration“-Tool des Open-Source-Frameworks *Spring*. Es wurde zur Reduzierung der Komplexität neuer *Spring*-Applikationen auf Grundlage von Microservices entwickelt. Jeder Dienst hat seinen eigenen Prozess, wodurch ein leichtgewichtiges Modell erreicht werden soll. Die Vereinfachung wird dadurch erreicht, dass Grundkonfigurationen für die Nutzung des Frameworks sowie alle relevanten Bibliotheken von Drittanbietern festgelegt sind (Vgl. Augsten, 2021).

Zu den Vorteilen von *Spring Boot* gehört die flexible Möglichkeit zur Konfiguration von *Java*-Beans, XML-Konfigurationen und Datenbanktransaktionen. Zudem bietet es eine leistungsstarke Batch-Verarbeitung und die Möglichkeit zur Verwaltung von REST-Endpunkten sowie eine Erleichterung bei der Verwaltung von Abhängigkeiten. Spring-Boot-Anwendungen sind annotationsbasiert und enthalten darüber hinaus einen eingebetteten Servlet-Container. Die Möglichkeit zur Verknüpfung von SQL- oder NoSQL-Datenbanken sind ein weiterer Vorteil von *Spring Boot*.

Spring Web

Spring Web ist eine Dependency, die allgemeine webspezifische Werkzeuge für Servlet- und Portlet-Umgebungen enthält.

Spring Data JPA

Spring Data JPA erleichtert die Erstellung von JPA-basierten Repositories, indem der Implementierungsaufwand durch eine Reihe von Hilfestellungen wie z. B. eine dynamische Abfrageausführung reduziert werden.

5.2 Micronaut

Micronaut ist ein JVM-basiertes Framework zur Entwicklung schlanker, modularer Anwendungen. Entwickelt wurde es von Oracle Cloud Infrastructure (OCI) und eignet sich dazu, Microservices zu erstellen. *Micronaut* enthält zwar einige Funktionen, die bestehenden Frameworks wie *Spring* ähneln, bietet aber auch neue Funktionen, die es davon abheben. Mit der Unterstützung für *Java*, *Groovy* und *Kotlin* bietet es zudem eine Vielzahl von Möglichkeiten, Anwendungen zu erstellen (Vgl. Micronaut, 2022).

Ein Feature von *Micronaut* ist der Mechanismus zur Injektion von Dependencies zur Kompilierzeit. *Micronaut* baut seine Dependency-Injection-Daten zur Kompilierzeit auf. Daraus resultiert ein schnellerer Start der Anwendung und ein geringerer Speicherbedarf (Micronaut, 2022).

5.3 Quarkus

Quarkus ist ein von der Firma RedHat entwickeltes Full-Stack Java Framework für JVMs und die native Kompilierung, das speziell für Container-Anwendungen geeignet ist. Das Augenmerk der Quarkus-Entwicklung liegt auf die Nutzung von etablierten Java-Bibliotheken und Standards (Vgl. Red Hat, 2022).

Damit eine Optimierung für Container-Anwendungen erreicht wird, verarbeitet *Quarkus* bereits während des Build-Vorgangs den Code. Daraus resultiert, dass nur die Klassen zur Laufzeit geladen werden, die auch benötigt werden. Das sorgt dafür, dass der Speicherverbrauch einer Quarkus-Anwendung verringert wird und darüber hinaus die Start- und Antwortzeit verringert werden (Red Hat, 2022).

Quarkus bietet zudem die Möglichkeit, ein *Native Image*, eine direkt ausführbare Datei, die keine JVM benötigt, erzeugen zu lassen. Dadurch sollen Faktoren wie Speicherverbrauch, sowie Start- und Antwortzeit zusätzlich positiv beeinflusst werden. Für den Betrieb von Container-Anwendungen sind diese Faktoren besonders wichtig, denn je geringer der Speicherverbrauch ist, desto mehr Container lassen sich gleichzeitig betreiben (Red Hat, 2022).

5.4 Vert.x

Vert.x ist ein Open-Source Toolkit, welches von der Eclipse Foundation entwickelt wurde. Es soll die Entwicklung reaktiver, skalierbarer Anwendungen für die JVM unterstützen, welche widerstandsfähig hinsichtlich von Ausfällen sind (Vgl. Eclipse Vert.x, 2022).

5.4.1 Verticles

Verticles sind Codeteile, die von der *Vert.x*-Engine ausgeführt werden. Das Toolkit bietet eine Vielzahl von abstrakten Verticle-Klassen, die beliebig erweitert und implementiert werden können (Eclipse Vert.x, 2022).

Da Verticles polyglott sind, können diese in jeder unterstützen Sprache geschrieben werden. Eine Anwendung besteht typischerweise aus mehreren Verticles, die in der gleichen *Vert.x*-Instanz laufen und über den Event-Bus miteinander kommunizieren (Eclipse Vert.x, 2022).

5.4.2 Event Bus

Da Verticles reaktiv sind, bleiben sie inaktiv, bis sie eine Nachricht oder ein Event erhalten. Eine Nachricht kann von einer Zeichenkette bis hin zu einem komplexen Objekt reichen. Die Nachrichtenverarbeitung erfolgt idealerweise asynchron. Die Nachrichten werden in eine Warteschlange des Event Bus gestellt und die Kontrolle an den Absender zurückgegeben. Anschließend wird die Warteschlange an das abhörende Verticle übergeben. Antworten werden mit Future- und Callback-Methoden gesendet (Eclipse Vert.x, 2022).

6 Benchmarking

In diesem Kapitel werden die Testphasen bis zur Durchführung des Benchmarkings beschrieben. Die Testplanung und das Ziel werden im Unterkapitel 6.1 erläutert. Anschließend werden im Unterkapitel 6.2 das Design und der Aufbau der Anwendungen beschrieben. Das Unterkapitel 6.3 beschreibt die Spezifikationen und Vorbereitungen zur Durchführung des Benchmarkings. Im letzten Unterkapitel wird schließlich die Testdurchführung dargestellt.

6.1 Testplanung

6.1.1 Testziele

Das Ziel des Benchmarkings ist es, die erhaltenen Daten zum Energie- und Speicherbedarf zu vergleichen, um Messergebnisse zur Effizienz des jeweiligen Frameworks zu erhalten. Anhand der Messergebnisse und einer Evaluation des Verfahrens soll abschließend eine Aussage darüber getroffen werden, ob dieser Prozess zur Integration in ein Verfahrensmodell geeignet ist.

6.1.2 Teststrategie

Für die Durchführung des Benchmarkings ist neben der in Java entwickelten REST-Clients eine Datenbank zur Kommunikation zwischen den Clients erforderlich. Hierzu wird eine Datenbank aufgesetzt, welche von jedem REST-Client gleichermaßen verwendet wird.

Mittels automatisierter HTTP-Requests soll anschließend in mehreren Iterationen ein möglichst genaues Ergebnis zum durchschnittlichen Verbrauch entstehen. Zur Erhöhung der Konsistenz der Ergebnisse werden je Iteration die vier folgenden HTTP-Requests durchgeführt:

- **POST**: Request zum Hinzufügen eines Datensatzes in die Datenbank
- **GET**: Request zum Auslesen aller Datensätze der Datenbank

- **PUT**: Request zum Ändern eines Datensatzes auf der Datenbank
- **DELETE**: Request zum Löschen eines Datensatzes von der Datenbank

Request	Pfad	Header
POST	/customers	application/json
GET	/customers	application/json
PUT	/customers{id}	application/json
DELETE	/customers{id}	application/json

Tabelle 2: Übersicht der REST-Schnittstellen

6.1.3 Testwerkzeuge

Datenbank

Um die entwickelten REST-Clients testen und das Benchmarking durchführen zu können, ist die Anbindung einer Datenbank erforderlich. Hierzu wird das relationale Datenbankmanagementsystem *MySQL Workbench* in der Version 8.0.28 verwendet. Bei *MySQL Workbench* handelt es sich um ein Open-Source-Werkzeug zur Verwaltung von Daten (Vgl. Oracle, 2022).

Postman

Postman ist ein Entwicklungswerkzeug, das bei der Erstellung, Prüfung und Änderung von APIs hilft. Hauptsächlich wird es zum Testen von REST-APIs auf HTTP-Basis eingesetzt. Der grundlegende Aufbau ist auf das Verarbeiten und Validieren von Requests und deren Responses fokussiert. Über einen visuellen Request-Builder lassen sich HTTP-Requests mit den zugehörigen Parametern spezifizieren und absenden. Die GUI bietet zudem die Möglichkeit, die Responses über eine integrierte Programmierschnittstelle mit Javascript auszuwerten. Ein Feature von *Postman* ist die Erstellung von Collections, welche ein Set von Requests bündeln. Die Collection kann automatisiert ausgeführt und deren Responses im gleichen Durchlauf auswerten werden (Vgl. SimplyTest, 2022).

Intel Power Gadget

Intel Power Gadget ist ein softwarebasiertes Tool zur Überwachung des Stromverbrauchs, das mit Intel Core Prozessoren der 2. Generation oder höher kompatibel ist. Es wird von Windows und OS X unterstützt und umfasst eine Anwendung, einen Treiber und Bibliotheken. Die Bibliotheken werden zur Überwachung und Schätzung der Echtzeit-Leistungsdaten des Prozessors in Watt unter Verwendung der Energiezähler im Prozessor genutzt. Als Output generiert das Tool eine CSV-Datei mit den Messwerten. Für das Benchmarking kommt das Tool in der Version 3.6 zum Einsatz (Vgl. Intel Corporation, 2019).

6.2 Testvorbereitung

6.2.1 Design der REST-Clients

Architektur

Die Architektur der Anwendungen orientiert sich an einer Schichtenarchitektur, bei der jede Schicht mit der Nachbarschicht kommuniziert.

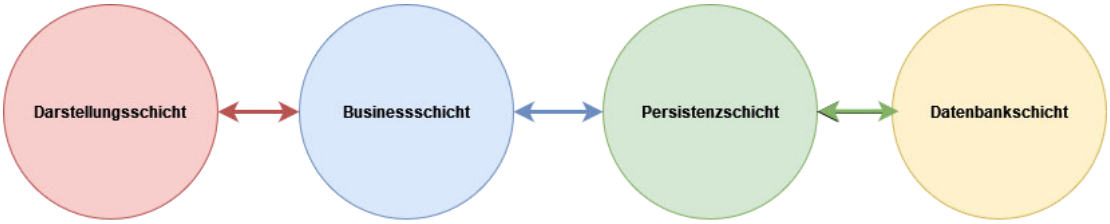

Abbildung 7: Schichten von REST-Clients

Darstellungsschicht

Die Darstellungsschicht ist die oberste Schicht der Architektur. Sie besteht aus Views, dem Front-End-Teil der Anwendung. In dieser Schicht werden die HTTP-Anfragen bearbeitet und die Authentifizierung durchgeführt. Die Konvertierung der HTTP-Parameter aus einer JSON-Datei in Java-Objekte sowie der umgekehrte Weg gehören ebenfalls zu den Aufgaben dieser Schicht. Nach erfolgreicher Authentifizierung der Anfrage erfolgt die Weiterleitung an die nächste Schicht.

Businessschicht

Die Businessschicht enthält die gesamte Business-Logik und besteht aus Service-Klassen. Sie ist für die Validierung und Autorisierung zuständig.

Persistenzschicht

Die Persistenzschicht enthält die gesamte Datenbank-Speicherlogik. Sie ist für die Umwandlung von Geschäftsobjekten in Datenbankzeilen und umgekehrt zuständig.

Datenbankschicht

Die Datenbankschicht enthält alle Datenbanken, in diesem Fall eine MySQL-Datenbank und kann aus mehreren Datenbanken bestehen. Sie ist für die CRUD-Operationen zuständig:

- C: Create (Erstellen)
- R: Read/Retrieve (Lesen/Abrufen)
- U: Update (Aktualisieren)
- D: Delete (Löschen)

Workflow

Spring Boot nutzt alle Funktionen von *Spring* wie *Spring MVC*, *Spring Data* und *JPA*. Eine generische Spring-Boot-Anwendung hat einen Controller, der die HTTP-Anfragen des Clients bedient. Dieser Controller interagiert mit der Serviceschicht, die die Anfragen zur Änderung des Modells und der Datenbank unter Verwendung des JPA-Repositories verarbeitet. Dieser Vorgang wird durch Dependency-Injection ermöglicht. Wenn keine Fehler auftreten, wird die Darstellungsschicht an den Client zurückgegeben. Der Workflow der implementierten Spring-Boot-Anwendung ist in Abbildung 8 dargestellt. Diesen Workflow nennt man Repository-Pattern, welcher in ähnlicher Form von allen Anwendungen dieses Benchmarkings implementiert wird.

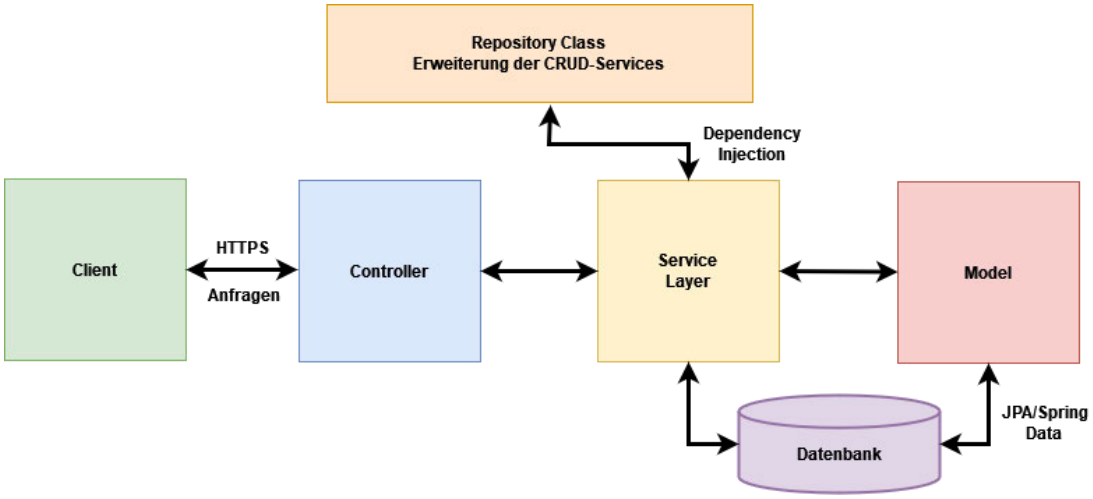

Abbildung 8: Repository-Pattern

6.2.2 Aufbau der Spring-Boot-Anwendung

Entität

Das Datenmodell ist durch eine Entitätsklasse abgebildet, damit dessen Persistenz durch *Spring Data* gewährleistet wird. Die Annotation *@Entity* sorgt dafür, dass die Klasse durch JPA als Tabelle abgebildet werden kann.

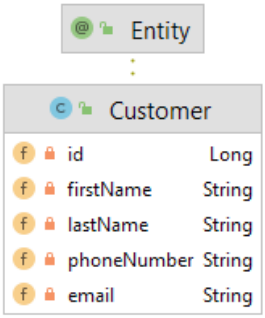

Abbildung 9: Klassendiagramm Spring-Boot-Entität

Um die Persistenz und den Zugriff auf die Entität zu ermöglichen, wird ein Repository benötigt. Im Fall der Spring-Boot-Anwendung ist dies als Interface implementiert und wird um das *JpaRepository* aus dem JPA-Package erweitert. *JpaRepository* stellt vordefinierte Methoden zur Durchführung von CRUD-Operationen bereit.

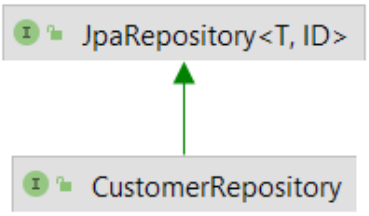

Abbildung 10: Klassendiagramm Spring-Boot-Repository

REST-Schnittstelle

Die REST-Schnittstelle wird durch die Controller-Klasse zur Verfügung gestellt, um den Zugriff von außen zu ermöglichen. Hier sind Endpunkte definiert, die über HTTP angesprochen werden können. In dieser Klasse kommen neben den Annotationen für die verschiedenen Schnittstellen die Annotationen *@Autowired*, *@RestController* und *@RequestMapping* zum Einsatz.

- *@Autowired* sorgt dafür, dass *Spring* die Abhängigkeiten anderer Objekte zur Laufzeit einbringt und somit die Dependency Injection ermöglicht wird. In diesem Fall ist das Objekt *CustomerService* mit der Annotation versehen
- *@RestController* markiert die Klasse als Schnittstelle für Web-Anfragen, wodurch *Spring* die eingehenden Anfragen mit den entsprechenden Mapping-Methoden verarbeitet
- *@RequestMapping* wird in der Anwendung auf Klassenebene als Anfragepfad für HTTP-Anfragen verwendet. Mit dieser Annotation können zusätzlich auf der Methodenebene spezifische Anfragepfade konfiguriert werden

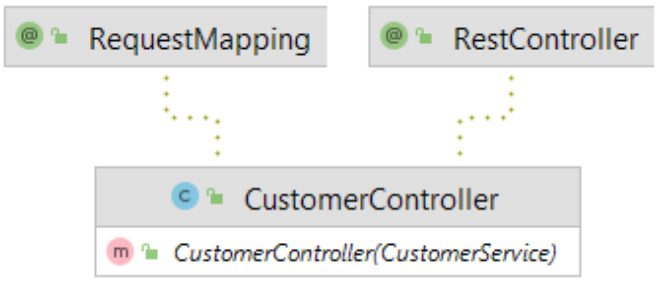

Abbildung 11: Klassendiagramm Spring-Boot-Controller

Konfiguration

Sprint Boot Anwendungen lassen sich durch Properties in der Konfigurationsdatei *application.properties* konfigurieren. Properties sind Key-Value-Paare, mit deren Hilfe in dieser Anwendung der Port und die Verbindung zur Datenbank festgelegt sind. Die Keys mit den Attributen *spring.datasource* definieren die Zugriffsdaten zur Datenbank, wohingegen die *spring.jpa.hibernate* Keys die Datenbank individuell konfigurieren.

```
spring.datasource.url=jdbc:mysql://<DATABASE_URL>
spring.datasource.username=<USERNAME>
spring.datasource.password=<PASSWORD>
spring.jpa.hibernate.ddl-auto=update
spring.jpa.hibernate.naming.physical-strategy=org.hiber-
nate.boot.model.naming.PhysicalNamingStrategyStandardImpl
```

Listing 1: Konfigurationsdatei der Spring-Boot-Anwendung

6.2.3 Aufbau der Micronaut-Anwendung

Entität

Das Datenmodell der Micronaut-Anwendung entspricht dem der Spring-Boot-Anwendung. Die Entitätsklasse ist durch die *@Entity*-Annotation definiert und auch das Repository-Interface wird um das *JpaRepository* erweitert. Entsprechend diesem Design sind die Klassendiagramme der Spring-Boot-Anwendung auch als solche der Micronaut-Anwendung zu verstehen.

REST-Schnittstelle

Der Zugriff von außen wird durch die Controller-Klasse ermöglicht. Die Annotationen der Micronaut-Anwendung unterscheiden sich jedoch von denen der Spring-Boot-Anwendung. Die Schnittstelle verwendet bei dieser Implementierung die Annotationen *@Controller* und *@Inject*.

- *@Controller* definiert die Klasse als einen Controller und implementiert einen Anfragepfad auf Klassenebene

- *@Inject* implementiert die Dependency Injection von Micronaut-Anwendungen. In diesem Fall ist das Objekt *CustomerRepository* mit der Annotation versehen

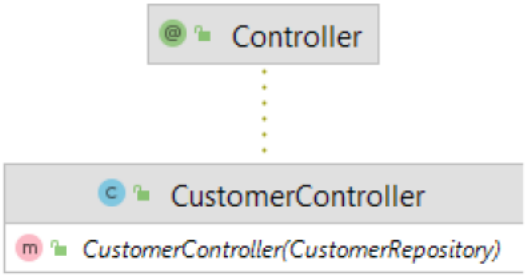

Abbildung 12: Klassendiagramm Micronaut-Controller

Konfiguration

Micronaut-Anwendungen lassen sich ebenfalls durch Properties in einer Konfigurationsdatei konfigurieren. Dazu wird die *application.yml* verwendet, welche ebenfalls Key-Value-Paare verarbeitet und für diese Anwendung die Verbindung zur Datenbank herstellt.

```
micronaut:
  application:
    name: <APPLICATION_NAME>
  server:
    port: <PORT>
datasources:
  default:
    url: jdbc:mysql://<DATABASE_URL>
    driverClassName: com.mysql.cj.jdbc.Driver
    username: <USERNAME>
    password: <PASSWORD>
    db-type: mysql
    schema-generate: UPDATE
    dialect: MYSQL
netty:
  default:
    allocator:
      max-order: 3
jpa.default.properties.hibernate.hbm2ddl.auto: update
```

```
jpa.default.properties.hibernate.physical_naming_strategy: "org.hiber-
nate.boot.model.naming.PhysicalNamingStrategyStandardImpl"
```

Listing 2: Konfigurationsdatei der Micronaut-Anwendung

6.2.4 Aufbau der Quarkus-Anwendung

Entität

Der grundlegende Aufbau der Quarkus-Anwendung ist vergleichbar mit der Spring-Boot-Anwendung. So wird bei dieser Anwendung die Entitätsklasse auch durch die *@Entity*-Annotation definiert. Anders als bei den Spring-Boot- und Micronaut-Anwendungen implementiert *Quarkus* nicht das *JpaRepository*, sondern die Quarkus-spezifische Bibliothek *Panache*. Dabei handelt es sich um eine Bibliothek, die die Entwicklung einer Hibernate-basierten Persistenzschicht vereinfacht. Ähnlich wie ein *JpaRepository* bietet *Panache* vordefinierte CRUD-Methoden und vereinfacht somit die Kommunikation mit der Datenbank. Eine weitere Unterscheidung liegt darin, dass das Repository nicht als Interface, sondern als Klasse implementiert ist. Die Verwendung einer Klasse sorgt dafür, dass der Primärschlüssel standardmäßig als Long-Datentyp erzeugt wird.

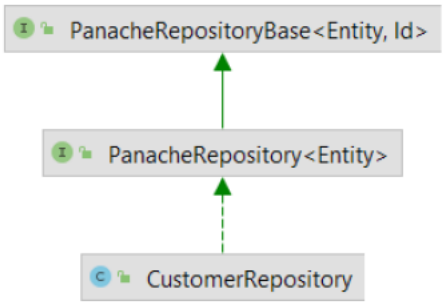

Abbildung 13: Klassendiagramm Quarkus-Repository

REST-Schnittstelle

Die Schnittstelle der Quarkus-Anwendung wird durch die Ressource-Klasse definiert. Diese Controller-Klasse entspricht vom Aufbau und den definierten Funktionen den Spring-Boot- und Micronaut-Anwendungen. Unterschiede finden sich bei der Namenskonvention von *Quarkus* und den spezifischen Klassen- und Methoden-Annotationen. Eine Quarkus-Schnittstelle

wird auf Klassenebene mit der Annotation *@Path* gekennzeichnet gefolgt von dem initialen Pfad des Zugriffs. Zudem werden auf Klassenebene noch die Annotationen *@Produces* und *@Consumes* verwendet.

- *@Path* kennzeichnet die Endpunkte und kann sowohl auf Klassen- als auch auf Methodenebene verwendet werden
- *@Produces* definiert den Content-Type, den die Methoden einer Ressourcenklasse erzeugen können
- *@Consumes* definiert den Content-Type, den die Methoden einer Ressourcenklasse verarbeiten können

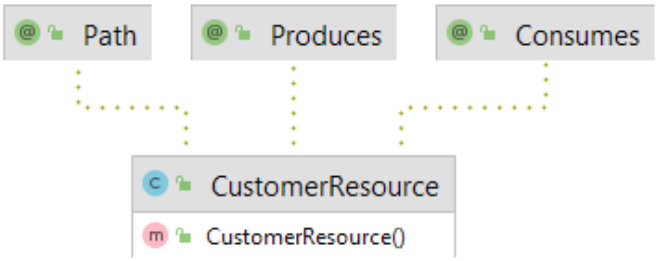

Abbildung 14: Klassendiagramm Quarkus-Resource

Konfiguration

Quarkus-Anwendungen lassen sich wie Spring-Boot-Anwendungen über Properties konfigurieren. Die hierbei verwendete Konfigurationsdatei entspricht der Datei der Spring-Boot-Anwendung.

6.2.5 Aufbau der Vert.x-Anwendung

Entität

Der Aufbau der Entitätsklasse der Vert.x-Anwendung unterscheidet sich darin, dass keine Annotation auf Klassenebene zur Erzeugung einer Entität verwendet wird. Stattdessen wird eine Java-übliche Bean-Klasse mit Gettern und Settern verwendet. Vert.x-Anwendungen binden zur Verarbeitung von JSON-Dateien die Bibliothek *Jackson* ein. *Jackson* automatisiert die Serialisierung und Deserialisierung von Bean-Klassen.

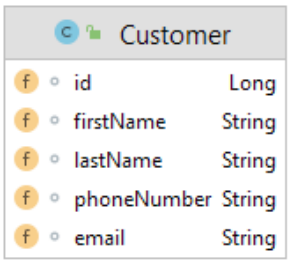

Abbildung 15: Klassendiagramm Vert.x-Entität

Anders als bei den vorher beschriebenen Anwendungen wird im Vert.x-Repository kein Interface eingebunden. Stattdessen werden zur Persistenz der Daten die CRUD-Methoden manuell als SQL-Statements in Methoden implementiert.

REST-Schnittstelle

Der Mittelpunkt einer Vert.x-Anwendung ist die MainVerticle-Klasse, über die der HTTP-Server gestartet wird und die Pfade für die Schnittstellen in Form eines Routers definiert werden. Der Router ist ein spezifischer Handler, der die Bearbeitung von HTTP-Anfragen mittels vordefinierter Methoden unterstützt und die Verkettung einer Reihe von Handlern ermöglicht. Zum Starten der Anwendung ist es erforderlich, den HTTP-Server mit dem Router in der *start()*-Methode an das Vertx-Objekt zu übergeben.

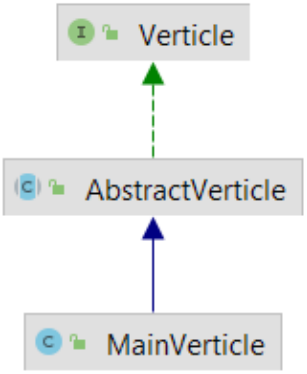

Abbildung 16: Klassendiagramm Vert.x-MainVerticle

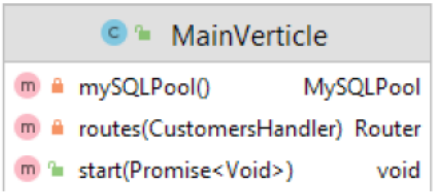

Abbildung 17: Klassendiagramm Vert.x-MainVerticle mit Methoden

Konfiguration

Die Konfiguration der Datenbank wird in *Vert.x* über ein datenbankspezifisches Interface geregelt. Da es sich bei der verwendeten Datenbank um eine MqSQL-Datenbank handelt, wurde zur Konfiguration das *MySQLPool*-Interface eingebunden. Das daraus resultierende *MySQLPool*-Objekt erhält die Konfigurationen und wird schließlich dem Repository übergeben, damit dieses die CRUD-Methoden darauf ausführen kann.

```
private MySQLPool mySQLPool() {
    MySQLConnectOptions connectOptions = new MySQLConnectOptions()
      .setPort(<PORT>)
      .setHost(<HOSTNAME>)
      .setDatabase(<DATABASE>)
      .setUser(<USERNAME>)
      .setPassword(<PASSWORD>);
    PoolOptions poolOptions = new PoolOptions().setMaxSize(5);
    MySQLPool pool = MySQLPool.pool(vertx, connectOptions, poolOptions);
    return pool;
  }
```

Listing 3: Konfiguration der Datenbank in Vert.x

6.3 Testspezifikation

6.3.1 Gerätespezifikation

Alle Tests wurden auf demselben System mit den in Tabelle 3 abgebildeten Spezifikationen, unter denselben Bedingungen durchgeführt.

HP Spectre x360 Convertible 13-ac0XX	
Prozessor	Intel(R) Core(TM) i7-7500U CPU @ 2.70GHz 2.90 GHz
Installierter RAM	8,00 GB (7,88 GB verwendbar)
Systemtyp	64-Bit-Betriebssystem, x64-basierter Prozessor
Betriebssystem	Windows 10 Home

Tabelle 3: Gerätespezifikation des ausführenden Systems

6.3.2 Systemvorbereitung

Zur Ausführung der Anwendungen ist die Installation mehrerer Tools erforderlich. Alle Anwendungen wurden mit *Java* und *Apache Maven* entwickelt. *Java* wurde in der Version 11.0.16 und *Maven* in der Version 3.8.6 verwendet.

Zusätzlich ist es für die Ausführung der Quarkus-Anwendung erforderlich, *GraalVM* und *Native-Image* zu installieren. *GraalVM* wurde in der Version CE 22.2.0 und *Native-Image* in der Version 22.2.0 verwendet. Nach dem Download von *GraalVM* über die offizielle Webseite ist es erforderlich, die Umgebungsvariablen des Systems zu konfigurieren.

Die Umgebungsvariable PATH muss um den Pfad des *GraalVM* bin-Verzeichnisses erweitert werden. Dies kann über eine Kommandozeile mit dem folgenden Befehl erreicht werden:

```
setx /M PATH "C:\Progra~1\Java\<graalvm install dir>\bin;%PATH%"
```

Anschließend muss die Umgebungsvariable JAVA_HOME so gesetzt werden, dass diese auf das *GraalVM*-Installationsverzeichnis zeigt. Der folgende Befehl kann ebenso über die Kommandozeile ausgeführt werden:

```
setx /M JAVA_HOME "C:\Progra~1\Java\<graalvm install dir>"
```

Sobald *GraalVM* installiert ist und die Umgebungsvariablen konfiguriert sind, muss *Native-Image*, über die Kommandozeile installiert werden.

```
gu install native-image
```

6.3.3 Konfiguration der Kommandodatei

Postman bietet als Standardfunktion die Möglichkeit, eine definierte Collection mit einer beliebigen Anzahl an Iterationen sowie weiteren Einstellungsmöglichkeiten auszuführen. Zur Ausführung der *Postman*-Collection ist das Kommandozeilen-Tool *Newman* erforderlich. *Newman* ermöglicht die Ausführung von Collections und zeigt die Ergebnisse an. Collections lassen sich im JSON-Format aus *Postman* exportieren.

Newman lässt sich via *NPM* installieren. Hierfür ist die Installation der Open-Source-Javascript-Laufzeitumgebung *Node.js* erforderlich. Über die offizielle Webseite lässt sich *Node.js* herunterladen und installieren. Durch die Installation von *Node.js* kann *NPM* verwendet und *Newman* installiert werden.

```
npm install -g newman
```

Anschließend kann die aus *Postman* exportierte Collections über die Kommandozeile ausgeführt werden.

```
newman run Benchmark.postman_collection.json
```

Intel Power Gadget kann ebenfalls über die Kommandozeile gestartet und gestoppt werden.

```
IntelPowerGadget.exe -start
```

```
IntelPowerGadget.exe -stop
```

Die resultierende Kommandodatei ist nun in der Lage, vor Beginn der Ausführung der *Postman* Collection das *Intel Power Gadget* Tool auszuführen und die Messung zu starten. Anschließend wird die Collections via *Newman* ausgeführt und das *Intel Power Gadget* Tool über die Kommandodatei gestoppt und beendet. Es resultiert eine CSV-Datei mit den detaillierten Leistungswerten des Prozessors.

Die auszuführende Kommandodatei sieht wie folgt aus:

```
SET postman_collection=Benchmark.postman_collection.json

SET IPG=C:\Program Files\Intel\Power Gadget 3.6\IntelPowerGadget.exe

START "" "%IPG%"

timeout /t 5

START "" "%IPG%" -start

CALL newman run %postman_collection% -n <ITERATIONS>

START "" "%IPG%" -stop

TASKKILL /F /IM IntelPowerGadget.exe /T

pause
```

Listing 4: Kommandodatei zur Testdurchführung

6.4 Testdurchführung

6.4.1 Spring-Boot-Anwendung

Nach der Navigation in das Verzeichnis der Spring-Boot-Anwendung muss die Anwendung gestartet werden.

```
mvn spring-boot:run
```

Der Server ist nun auf dem gewählten Port gestartet, sodass die Konfigurationsdatei ausgeführt werden kann. Nachdem die Collection ausgeführt und die CSV-Datei mit den Daten zur Prozessorauslastung generiert wurde, wird das Ergebnis des Tests im Kommandozeilen-Programm angezeigt.

	executed	failed
iterations	100	0
requests	400	0
test-scripts	800	0
prerequest-scripts	400	0
assertions	900	0
total run duration: 40.3s		
total data received: 33.4kB (approx)		
average response time: 14ms [min: 7ms, max: 100ms, s.d.: 7ms]		

Abbildung 18: Testergebnis Spring Boot

In der dargestellten Tabelle in Abbildung 18 ist zu sehen, dass 100 Iterationen mit insgesamt 400 Requests und 900 Assertions erfolgreich durchgeführt wurden. Der Test lief insgesamt 40,3 Sekunden und die durchschnittliche Antwortzeit des REST-Clients lag bei 14 ms.

6.4.2 Micronaut-Anwendung

Zum Starten der Micronaut-Anwendung muss in den Projektordner navigiert und folgender Befehl ausgeführt werden:

```
mvnw mn:run
```

Der Server ist nun gestartet und die Kommandozeilendatei kann ausgeführt werden.

	executed	failed
iterations	100	0
requests	400	0
test-scripts	800	0
prerequest-scripts	400	0
assertions	900	0
total run duration: 42.8s		
total data received: 11.3kB (approx)		
average response time: 17ms [min: 7ms, max: 713ms, s.d.: 37ms]		

Abbildung 19: Testergebnis Micronaut

Die Tabelle in Abbildung 19 zeigt bei 100 Iterationen, 400 Requests und 900 Assertions eine Laufzeit von 42,8 Sekunden und eine durchschnittliche Antwortzeit von 17 ms.

6.4.3 Quarkus-Anwendung

Der Test der Quarkus-Anwendung umfasst zwei verschiedene Arten. Um einen Vergleich zu haben, inwieweit *Native-Image* den Energieverbrauch beeinflusst, wird sowohl eine JAR-Datei, als auch ein *Native-Image* zum Testen ausgeführt.

JAR ausführen

Im ersten Schritt muss die JAR-Datei aus dem Projektordner heraus generiert werden.

```
quarkus build
```

Zum Starten der Anwendung muss die JAR-Datei ausgeführt werden.

```
java -jar target/quarkus_benchmark/quarkus-run.jar
```

	executed	failed
iterations	100	0
requests	400	0
test-scripts	800	0
prerequest-scripts	400	0
assertions	900	0
total run duration: 42.9s		
total data received: 22.3kB (approx)		
average response time: 17ms [min: 6ms, max: 751ms, s.d.: 37ms]		

Abbildung 20: Testergebnis Quarkus

Die Tabelle in Abbildung 20 zeigt bei 100 Iterationen, 400 Requests und 900 Assertions eine Laufzeit von 42,9 Sekunden und eine durchschnittliche Antwortzeit von 17 ms.

Native-Image ausführen

Im ersten Schritt muss ein *Native-Image* aus dem Projektordner heraus generiert werden.

```
mvnw package -Pnative
```

Anschließend muss die entstandene Runner.exe, welche sich im Target-Ordner befindet, ausgeführt werden.

```
quarkus_benchmark-1.0.0-SNAPSHOT-runner.exe
```

	executed	failed
iterations	100	0
requests	400	0
test-scripts	800	0
prerequest-scripts	400	0
assertions	900	0
total run duration: 39.8s		
total data received: 22.3kB (approx)		
average response time: 11ms [min: 5ms, max: 118ms, s.d.: 6ms]		

Abbildung 21: Testergebnis Quarkus-Native-Image

Die Tabelle in Abbildung 21 zeigt bei 100 Iterationen, 400 Requests und 900 Assertions eine Laufzeit von 39,8 Sekunden und eine durchschnittliche Antwortzeit von 11 ms.

6.4.4 Vert.x-Anwendung

Um die Vert.x-Anwendung ausführen zu können, muss vorerst eine ausführbare JAR-Datei erzeugt.

```
mvn clean package
```

Zum Starten der Anwendung muss der folgende Befehl ausgeführt werden:

```
java -jar target/vertx_benchmark-1.0-SNAPSHOT-fat.jar
```

Nachdem die Anwendung gestartet wurde, kann die Kommandozeilendatei ausgeführt werden.

	executed	failed
iterations	100	0
requests	400	0
test-scripts	800	0
prerequest-scripts	400	0
assertions	900	0
total run duration: 41.2s		
total data received: 11.09kB (approx)		
average response time: 13ms [min: 5ms, max: 264ms, s.d.: 13ms]		

Abbildung 22: Testergebnis Vert.x

Die Tabelle in Abbildung 22 zeigt bei 100 Iterationen, 400 Requests und 900 Assertions eine Laufzeit von 41,2 Sekunden und eine durchschnittliche Antwortzeit von 13 ms.

7 Auswertung der Testergebnisse

In diesem Kapitel werden die Ergebnisse des in Kapitel 6 durchgeführten Benchmarkings beschrieben. Das Unterkapitel 7.1 konzentriert sich auf die erhobenen Daten zum CPU- und DRAM-Energieverbrauch. Im Unterkapitel 7.2 werden die analysierten Daten miteinander verglichen und die daraus resultierenden Erkenntnisse beschrieben.

Zur Identifikation einer geeigneten Anzahl an Iterationen wurden mehrere Testdurchlaufe ausgeführt. Diese Testdurchläufe haben gezeigt, dass das Tool zur Datenerhebung die Erhebung der Leistungsdaten ab einem unbekannten Zeitpunkt gestoppt hat. Dies hatte zur Folge, dass eine Durchführung von mehr als 100 Iterationen der HTTP-Requests nicht realisierbar war.

Die Dauer der erfolgreichen Datenerhebungen lag, wie in Kapitel 6.4 beschrieben, zwischen 39,8 und 42,9 Sekunden.

7.1 Energieverbrauch

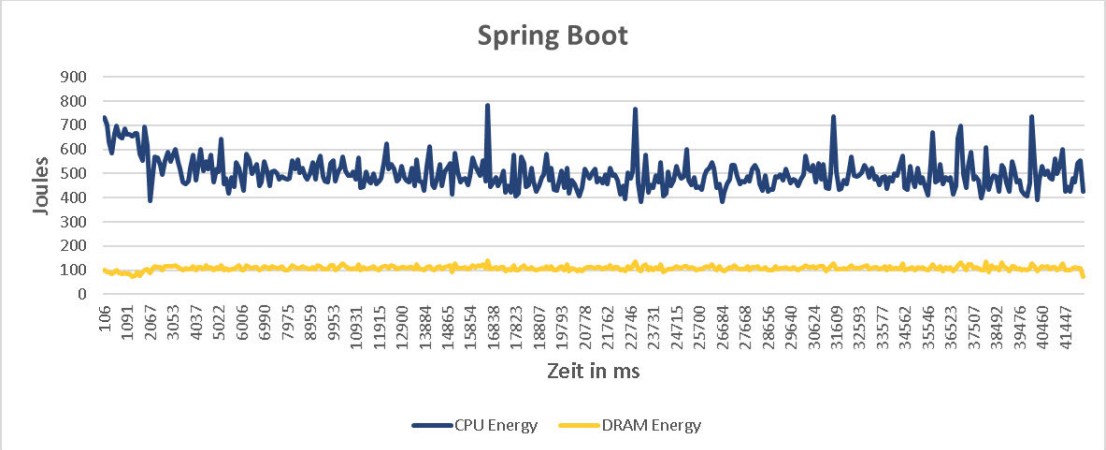

Abbildung 23: Spring-Boot-Energieverbrauch

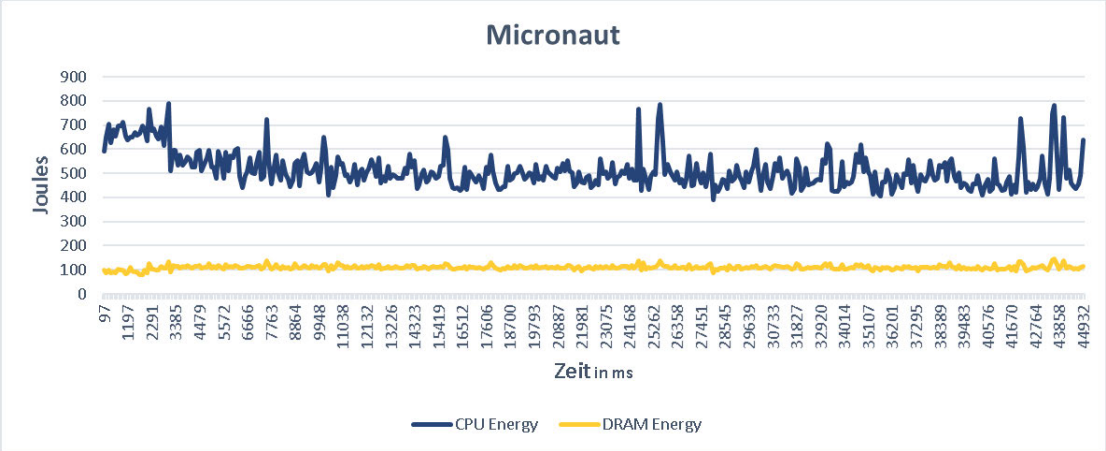

Abbildung 24: Micronaut-Energieverbrauch

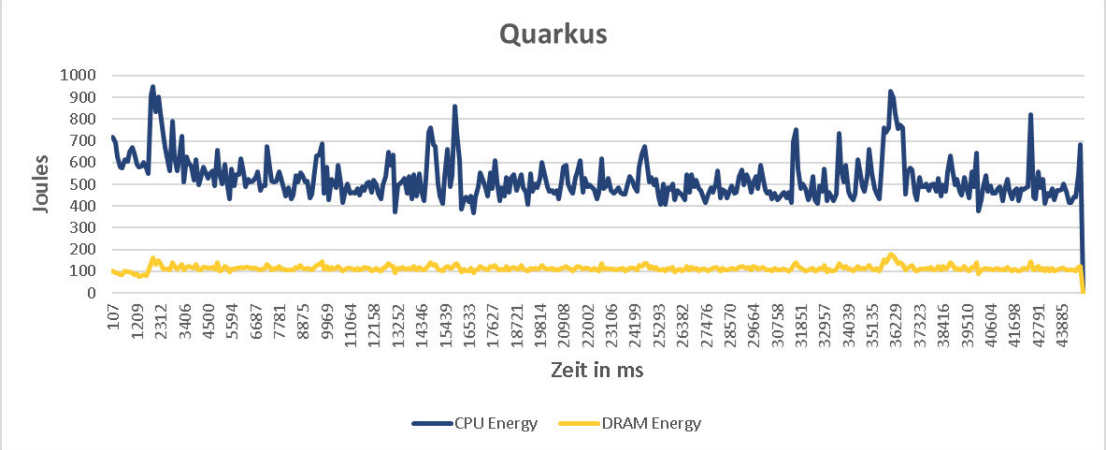

Abbildung 25: Quarkus-Energieverbrauch

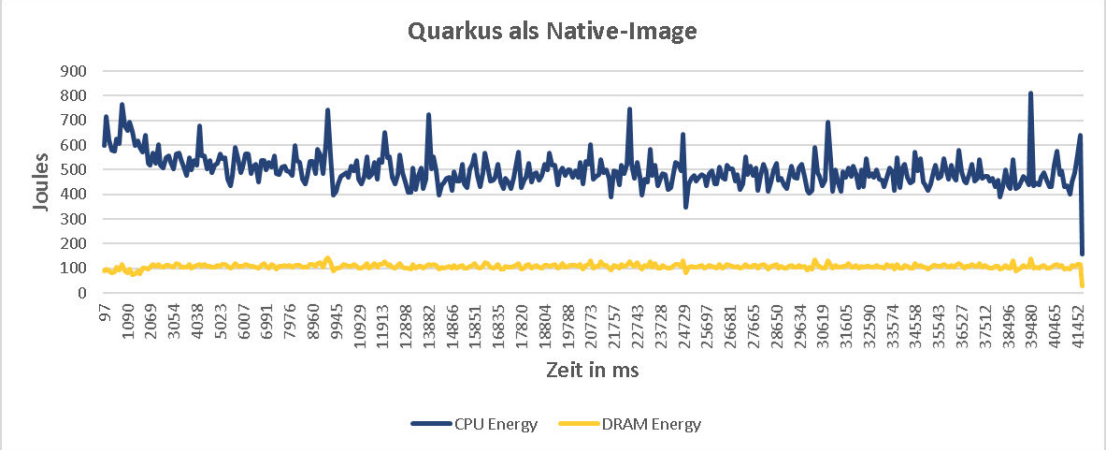

Abbildung 26: Quarkus-Native-Image-Energieverbrauch

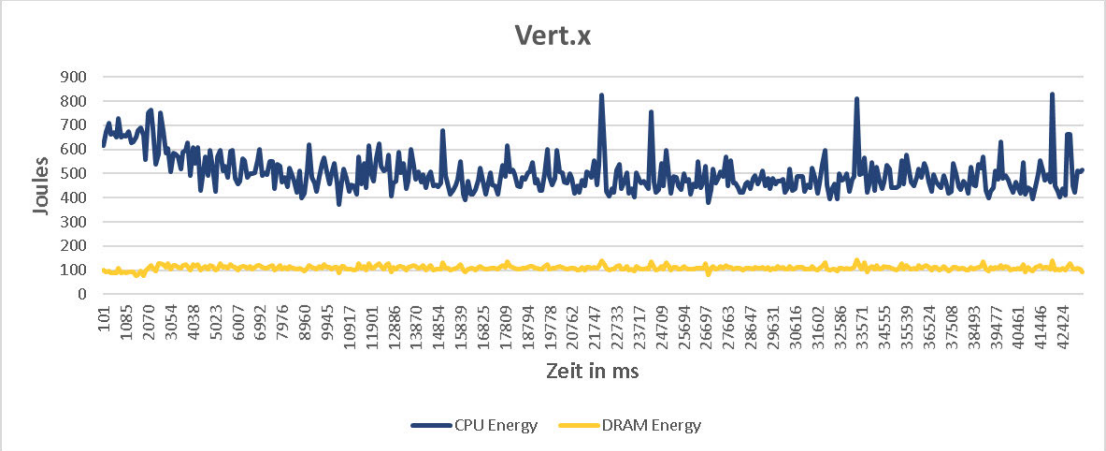

Abbildung 27: Vert.x-Energieverbrauch

Die Abbildungen 23, 24, 25, 26 und 27 zeigen den Energieverbrauch der Anwendungen während des Benchmarkings. Der CPU-basierte Verbrauch ist jeweils in Blau und der DRAM-basierte Verbrauch in Gelb dargestellt.

Der CPU-basierte Energieverbrauch stellt bei jeder Anwendung den größten Teil der verbrauchten Energie dar. Durchschnittlich 78,37 % der verbrauchten Energie geht vom CPU aus, der restliche Anteil fällt auf den DRAM.

Anwendungsübergreifend beträgt der höchst gemessene Wert für den CPU-Verbrauch 952 J. Dieser Wert wurde bei der Quarkus-Anwendung gemessen. Der niedrigste Wert lag bei 348 J und wurde bei der Quarkus-Native-Image-Anwendung gemessen.

Die Messung des DRAM-Energieverbrauchs zeigt anwendungsübergreifend einen Maximalwert von 180 J. Dieser Wert wurde bei der Quarkus-Anwendung gemessen. Der Minimalwert von 140 J wurde bei der Spring-Boot-Anwendung gemessen.

7.2 Vergleich der Messungen

In Tabelle 4 sind die CPU-Messwerte der einzelnen Anwendungen dargestellt. Hier ist zu erkennen, dass der geringste Mittelwert aller Anwendungen bei der Quarkus-Native-Image-Anwendung gemessen wurde. Dieser lag 5,02 % unter dem höchsten Mittelwert. Der niedrigste Gesamtverbrauch wurde ebenso bei der Quarkus-Native-Image-Anwendung gemessen. Dieser lag bei 189.451 J und somit 13,25 % unter dem höchsten Verbrauch.

	Gesamt	Mittelwert	Minimalwert	Maximalwert
Spring Boot	193.753	501	385	783
Micronaut	210.692	512	391	789
Quarkus	214.555	523	370	952
Quarkus Native	189.451	498	348	810
Vert.x	198.047	500	370	828

Tabelle 4: CPU-Energieverbrauch in Joule

In Tabelle 5 sind die DRAM-Messwerte der einzelnen Anwendungen dargestellt. Der niedrigste Mittelwert wurde bei den Spring-Boot- und Quarkus-Native-Image-Anwendungen gemessen. Dieser lag bei 108 J und somit 3,70 % unter dem höchsten Mittelwert. Der niedrigste Gesamtverbrauch lag bei 41.192 J und somit 12,18 % unter dem höchsten Verbrauch.

	Gesamt	Mittelwert	Minimalwert	Maximalwert
Spring Boot	41.790	108	74	140
Micronaut	45.203	109	81	145
Quarkus	46.210	112	77	180
Quarkus Native	41.192	108	74	143
Vert.x	43.273	109	76	142

Tabelle 5: DRAM-Energieverbrauch in Joule

Der durchschnittliche Gesamtverbrauch fällt bei allen Anwendungen unterschiedlich aus. In Tabelle 6 wird jedoch deutlich, dass sich das Verhältnis zwischen CPU- und DRAM-basiertem Energieverbrauch bei allen Anwendungen in einem fast identischen Bereich bewegt. Die Optimierung einer Anwendung hinsichtlich des CPU-basierten Energieverbrauchs könnte zusätzlich zu einer Verringerung des DRAM-basierten Energieverbrauchs führen.

	CPU gesamt	CPU in %	DRAM gesamt	DRAM in %
Spring Boot	193.753	78,43	41.790	21,57
Micronaut	210.692	78,55	45.203	21,45
Quarkus	214.555	78,46	46.210	21,54
Quarkus Native	189.451	78,26	41.192	21,74
Vert.x	198.047	78,15	43.273	21,85

Tabelle 6: CPU- und DRAM-Energieverbrauch in Joule

8 Fazit

Das Ziel dieser Bachelorarbeit war es, den Prozess des Software Engineerings hinsichtlich der Nachhaltigkeit zu untersuchen und mittels Literaturrecherche den aktuellen Entwicklungsstand dieses Prozesses aufzuzeigen. Anschließend an die Recherche sollte anhand eines Benchmarking-Experiments verdeutlicht werden, dass die Evaluation der für ein Softwareprojekt erforderlichen Frameworks zu einer Reduzierung des Energieverbrauchs führen kann. Schließlich sollte eine Aussage darüber getroffen werden, ob der Evaluationsprozess eine geeignete Maßnahme in der Entwicklung nachhaltiger Software darstellt.

8.1 Bewertung

Im Benchmarking-Experiment wurde der Energie- und Speicherverbrauch von vier bekannten *Java*-Frameworks zur Entwicklung von REST-Clients erhoben und analysiert. Dazu wurden mittels *Postman* API-Tests entworfen und automatisiert. Zum Erheben der Daten wurde das *Intel Power Gadget* verwendet, welches die Leistungsdaten des Prozessors auslesen kann. Durch die Messung der Ausführungszeit und der maximalen Speicherauslastung konnten beide Messwerte in Beziehung zur verwendeten Energie gesetzt werden.

Den niedrigsten Verbrauch und somit das beste Ergebnis konnte die Anwendung *Quarkus-Native-Image* erzielen. Dieses Testobjekt verwendete die *GraalVM* mit der *Native-Image* Technologie zur Erzeugung einer eigenständig ausführbaren Datei. Beim CPU-basierten Energieverbrauch war der Verbrauch bis zu 13,25 % und beim DRAM-basierten Energieverbrauch bis zu 12,18 % geringer als beim Testobjekt mit dem höchsten Verbrauch. Die Nutzung eines *Native-Image* bei diesem Testobjekt könnte auf eine Abhängigkeit zwischen dieser Technologie und einem geringeren Energie- und Speicherbedarf schließen.

Mit der Entstehung und Entwicklung des GREENSOFT-Referenzmodells im Jahr 2011 befindet sich das Forschungsgebiet Green Software Engineering noch in einer frühen Entwicklung. Durch weiterführende Forschung zu grüner und nachhaltiger Software und dem steigenden Bewusstsein für nachhaltige Software wird die Optimierung von Software in Zukunft noch relevanter.

Das durchgeführte Benchmarking-Experiment hat verdeutlicht, dass die Messung des Energie- und Speicherbedarfs von Software zwar mit einem erhöhten Konfigurationsaufwand verbunden ist, das Ergebnis jedoch ein Anreiz dafür sein kann, dies durchzuführen. Durch standardisierte Bewertungskriterien und die Einführung von Siegeln für Software und Frameworks kann der Mess- und Analyse-Vorgang zukünftig im Idealfall aus dem Entwicklungsprozess entkoppelt werden. Dies würde dazu führen, dass die Konsumierenden sowohl den Aufwand der Evaluation als auch den Energieverbrauch reduzieren könnten.

8.2 Ausblick

Die Reduzierung des Energieverbrauchs von Software wird in den kommenden Jahren ein wichtiger Faktor der IKT-Branche werden. Ein wesentlicher Schritt wird es sein, das nötige Wissen darüber, wie sich Software nachhaltiger und weniger ressourcenverschwenderisch gestalten lässt, zu vermitteln. Angefangen bei der Klimatisierung eines Rechenzentrums über die Hardware bis hin zur Software gibt es in allen Bereichen Optimierungspotenziale.

Nachhaltige Software lässt sich nur entwickeln, wenn die Komplexität der Softwaresysteme nicht die Kompetenz der Entwickler übersteigt. Eine zu komplexe Software lässt unvorhersehbare Fehler entstehen und führt schließlich zu einem Kontrollverlust. Um diese Situation zu verändern, sind einerseits technische Innovationen notwendig. Andererseits sind neue Denkweisen im Management erforderlich.

Ein entscheidender Faktor wird auch die Optimierung von alter Software sein. Moderne Systeme werden häufig an die Anforderungen alter Systeme angepasst, obwohl es mit einer entsprechenden Strategie auch andersherum möglich wäre. Eine breite Testabdeckung der Software zur Aufdeckung von Problemen in der Infrastruktur würde dem Risiko vor ungewollten

Seiteneffekten entgegenwirken. Durch entsprechende Tests könnten entstandene Probleme beim Austausch von Abhängigkeiten frühzeitig aufgedeckt und behoben werden.

Seit mehr als 15 Jahren wird an den Charakteristika nachhaltiger Software geforscht, doch der Fokus liegt je nach Perspektive auf unterschiedlichen Aspekten. Der Blaue Engel für Desktop-Software (DE-UZ 215) könnte hier zukünftig dazu beitragen, ein einheitliches Verständnis von nachhaltiger Software zu schaffen.

Literaturverzeichnis

Atlassian, o. D.. *atlassian.* [Online]
erhältlich unter https://www.atlassian.com/devops
[Zugriff am 24 07 2022].

Augsten, S., 2021. *Dev-Insider.* [Online]
erhältlich unter https://www.dev-insider.de/was-ist-spring-boot-a-1009135/
[Zugriff am 07 August 2022].

BMU, 2019. *Klimaschutzprogramm 2030* , s.l.: BMU.

Brain, R., 2010. *ComputerWeekly.* [Online]
erhältlich unter https://www.computerweekly.com/answer/Dynamic-code-analysis-vs-static-analysis-source-code-testing
[Zugriff am 2 Juli 2022].

Calero, C. &. P. M., 2015. *Green in Software Engineering.* s.l.:Springer Publishing.

Calero, C. et al., 2019 . *5Ws of green and sustainable software,* s.l.: TUP.

Calero, C. M. M. Á. &. P. M., 2021. *Software Sustainability.* s.l.:Springer Publishing.

Cazier J, H. B., 2011. *Doing the right thing for the environment just got easier with a little,* s.l.: s.n.

CEET, 2013. *THE POWER OF WIRELESS CLOUD An analysis of the impact on energy consumption of the growing popularity of accessing cloud services via wireless devices,* s.l.: Centre for Energy-Efficient Telecommunications (CEET).

Deneckère, R. R. G., 2020. *EcoSoft: Proposition of an Eco-Label for Software Sustainability,* s.l.: Springer.

Dr. Christian Anton, J. M. D. E. W., 2021. *Klimawandel: Ursachen, Folgen und Handlungsmöglichkeiten,* s.l.: Deutsche Akademie der Naturforscher Leopoldina e. V..

Dr. Marc Fabel, D. M. F. L. A., 2022. *Trägt die Fridays-for-Future-Bewegung,* s.l.: ifo Institut.

Dr. Sven L. Roth, T. H. B. B. M. P. F. N. G.-d. F. X. K. D. K. D. C. N. J. R. D. S.-C., 2022. *IT WIRD KERN DER WERTSCHÖPFUNG Studie IT-Trends 2022,* s.l.: Capgemini.

Eclipse Vert.x, 2022. *Vertx.* [Online]
erhältlich unter https://vertx.io/docs/vertx-web/java/
[Zugriff am 07 August 2022].

Ericsson, 2020. *A quick guide to your digital carbon footprint,* s.l.: Ericsson.

GitHub, 2022. *PYPL PopularitY of Programming Language index.* [Online]
erhältlich unter https://pypl.github.io/PYPL.html
[Zugriff am 11 Juni 2022].

Gouy, I., 2008. *The Computer Language Benchmarks Game.* [Online]
erhältlich unter https://benchmarksgame-team.pages.debian.net/benchmarksgame/why-measure-toy-benchmark-programs.html
[Zugriff am 13 Juni 2022].

Green Software Foundation, 2022. *Manifest.* [Online]
erhältlich unter https://greensoftware.foundation/manifest-german/
[Zugriff am 2022 Juni 11].

Harika Jayanthi, A. K. A. E. P. S. S. V., 2021. *An Organizational Structure for Sustainable Software Development,* s.l.: IEEE.

Hilty, L. M., 2008. *Information Technology and Sustainability.* s.l.:Books on Demand.

IEEE, 1990. *IEEE Standard Glossary of Software Engineering Terminology,* s.l.: Computer Society of the IEEE.

Intel Corporation, 2019. *Intel® Power Gadget.* [Online]
erhältlich unter https://www.intel.com/content/www/us/en/developer/articles/tool/power-

gadget.html
[Zugriff am 11 Juni 2022].

ISO, 2011. *ISO/IEC 25010:2011,* s.l.: ISO.

Jens Gröger, D. A. K. D. S. N. A. F. A. G. E. K. D. L. M. H. Y. M., 2018. *Entwicklung und Anwendung von Bewertungsgrundlagen für ressourceneffiziente Software unter Berücksichtigung bestehender Methodik,* s.l.: Umweltbundesamtes.

Kent Beck, M. B. A. v. B. A. C. W. C. M. F. J. G. J. H. A. H. R. J. J. K. B. M. R. C. M. S. M. K. S. J. S. D. T., 2001. *Agile Manifesto,* s.l.: Die Autoren.

Kern E, D. M. N. S. G. A. J. T., 2013. *Green software and green software engineering - Definitions, Measurements, and Quality Aspects,* Zürich: ICT4S 2013 Proceedings.

Kristina Rakneberg Berntsen, M. R. O. N. L. A. T. T. R. C.-P., 2016. *Sustainability in Software Engineering - A Systematic Mapping.* s.l.:Springer.

Luís Gabriel Lima, G. M. F. S.-N. P. L. J. P. F. F. C., 2016. *Haskell in Green Land: Analyzing the Energy Behavior of a Purely Functional Language,* s.l.: 23rd IEEE Int. Conf. on Software Analysis, Evolution, and Reengineering (SANER'2016).

Markus Dick, S. N., 2010. *Enhancing Software Engineering Processes towards - Sustainable Software Product Design,* s.l.: Trier University of Applied Sciences, Umwelt-Campus Birkenfeld.

Marz, W., 2022. *GRÜNE TRANSFORMATION,* s.l.: ifo Zentrum für Energie, Klima und Ressourcen.

Micronaut, 2022. *Micronaut.* [Online]
erhältlich unter https://docs.micronaut.io/latest/guide/
[Zugriff am 07 August 2022].

Mohamed Amine Beghoura, A. B. A. B., 2015. *Green software requirements and measurement: random decision forests-based software energy consumption profiling,* London: Springer-Verlag.

Murugesan, S., 2021. *Green Software Foundation.* [Online]
erhältlich unter https://greensoftware.foundation/articles/10-recommendations-for-green-

software-development
[Zugriff am 02 Juli 2022].

Nelly Condori Fernandez, P. L., 2018-1. *A Software Sustainability-Quality Model,* s.l.: VU Technical Report.

Nelly Condori-Fernandezab, P. L., 2018-2. *Characterizing the contribution of quality requirements to software sustainability,* s.l.: Journal of Systems and Software.

Nelly Condori-Fernandez, P. L., 2019. *Towards a Software Sustainability-Quality Model: Insights from a Multi-Case Study,* s.l.: IEEE.

Nelly Condori-Fernandez, P. L. M. R. L. Á. S. P., 2020. *An Action Research for Improving the Sustainability Assessment Framework Instruments,* s.l.: MDPI.

Oracle, 2022. *MySQL.* [Online]
erhältlich unter https://www.mysql.com/de/products/workbench/
[Zugriff am 09 August 2022].

Prof. Dr. Lorenz Hilty, D. W. L. D. S. B. M. E.-W. P. D. K. F. D. R. H., 2015. *Grüne Software: Ermittlung und Erschließung von Umweltschutzpotenzialen der Informations- und Kommunikationstechnik (Green IT),* s.l.: Umweltbundesamt.

Pufé, I., 2014. *Nachhaltigkeit.* s.l.:UVK Verlagsgesellschaft mbH, Konstanz und München.

Red Hat, 2022. *Quarkus.* [Online]
erhältlich unter https://quarkus.io/about/
[Zugriff am 07 August 2022].

Reineke, W. &. S. M., 2020. *Umweltmikrobiologie (3. Aufl.).* s.l.:Springer Spektrum.

Roy Schwartz, J. D. N. A. S. O. E., 2019. *Green AI,* s.l.: Allen Institute for AI, Seattle - Washington (USA), Carnegie Mellon University, Pittsburgh - Pennsylvania (USA), University of Washington, Seattle, Washington (USA).

Rui Pereira, M. C. F. R. R. R. J. C. J. P. F. J. S., 2017. *Energy Efficiency across Programming Languages: How Do Energy, Time, and Memory Relate?,* s.l.: ACM SIGPLAN Interna-tional Conference on Software Language Engineering (SLE'17).

Rui Pereira, M. C. J. S. J. C. J. P. F., 2016. *The Influence of the Java Collection Framework on Overall Energy Consumption,* s.l.: 5th Int.Workshop on Green and Sustainable Software (GREENS ’16).

Sara S. Mahmoud, I. A., 2013. *A Green Model for Sustainable Software Engineering,* s.l.: International Journal of Software Engineering and Its Applications.

SimplyTest, 2022. *Testautomatisierung.* [Online]
erhältlich unter https://www.testautomatisierung.org/lexikon/postman-api/
[Zugriff am 09 August 2022].

Siti Rohana Ahmad Ibrahim, J. Y. H. S. N. H. B., 2019. *Towards Green Software Process: A Review on Integration of Sustainability Dimensions and Waste Management,* s.l.: IEEE.

Sommerville, I., 2018. *Software Engineering.* 10., aktualisierte Auflage Hrsg. s.l.:Pearson.

Stefan Naumann, M. D. E. K. T. J., 2011. *The GREENSOFT Model: A reference model for green and sustainable software,* s.l.: Trier University of Applied Sciences, Umwelt-Campus Birkenfeld (Environmental Campus Birkenfeld), ISS - Institute for Software Systems.

The Climate Group, 2008. *SMART 2020: enabling the low carbon economy in the information age,* s.l.: The Global eSustainability Initiative.

Umweltbundesamt, 2021. *Umweltbundesamt.* [Online]
erhältlich unter https://www.umweltbundesamt.de/themen/wirtschaft-konsum/wirtschaft-umwelt/umwelt-energiemanagement/energiemanagementsysteme
[Zugriff am 25 06 2022].

Umweltbundesamt, 2021. *Umweltbundesamt.* [Online]
erhältlich unter https://www.umweltbundesamt.de/umwelttipps-fuer-den-alltag/elektrogeraete/computer-pc-laptop#gewusst-wie
[Zugriff am 26 Juni 2022].

United Nations, 2015. *Übereinkommen von Paris.* [Online]
erhältlich unter https://ec.europa.eu/clima/eu-action/international-action-climate-

change/climate-negotiations/paris-agreement_de
[Zugriff am 11 Juni 2022].

VDI Zentrums Ressourceneffizienz, 2021. *BMUV.* [Online]
erhältlich unter https://www.bmuv.de/pressemitteilung/unternehmen-sparen-rohstoffe-und-energie-mit-kuenstlicher-intelligenz
[Zugriff am 26 Juni 2022].

Weng, W. H., 2021. *Examining Impact Factors for the Sustainability of Information Systems: A Relational Perspective,* s.l.: IEEE.

Wenyu Du, S. L. P. M. Z., 2013. *How to balance sustainability and profitability in technology,* s.l.: IEEE Trans Eng Manag 60(2):366–385.

Zimmermann, F. M., 2016. *Nachhaltigkeit wofür?: Von Chancen und Heraus-forderungen für eine nachhaltige Zukunft (1. Aufl. 2016 Aufl.).* s.l.:Springer Spektrum.

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

_____	_____	[redacted]
Ort	Datum	Unterschrift im Original