



Community
Nachhaltige
Digitalisierung

Green Coding

Booklet zur Workshopreihe



Bundesministerium
für Umwelt, Naturschutz, nukleare Sicherheit
und Verbraucherschutz

Inhaltsverzeichnis

1.	Einleitung	4
2.	Einführung Green Coding	5
2.1	Definitionen	5
2.2	Unternehmensarchitektur	5
2.3	IT-Architektur	6
2.3.1	Energiesparen durch Herunterfahren	7
2.3.2	Lose Kopplung	7
2.3.3	Reaktive Programmierung	7
2.4	Anforderungsanalyse und Entwicklung	8
2.4.1	Funktionale Anforderungen	8
2.4.2	Nicht-Funktionale Anforderungen	9
2.5	Ressourcenverbrauch	12
3.	Messtools	14
3.1	Performance Engineering	14
3.2	Stromverbrauch auf Prozessebene messen	17
3.3	Energieverbrauchsmessung auf Funktionsebene	18
3.4	jPowerMonitor	19
3.4.1	Stromverbrauchmessungstools	19
3.4.2	JUnit Extension	20
3.4.3	Java Agent	22
3.4.4	Fazit	24
4.	Optimierung von Testing und Betriebsplattform	25
4.1	Testing	25
4.1.1	Wie oft sollen die Tests laufen?	25
4.1.2	Wie viele Tests sollen laufen?	25

4.1.3	Wie wird getestet?	26
4.1.4	Wann wird getestet?	26
4.2	Betriebsplattform	27
4.2.1	Containerisierung	27
4.2.2	Containerorchestrierung	27
4.3	Monitoring	28
5.	Digitalisierung und Nachhaltigkeit	30
5.1	Gedankenanstöße zu möglichen Maßnahmen für nachhaltige Digitalisierung	30
5.1.1	Deployment von SW-Komponenten auf notwendiges Maß reduzieren	30
5.1.2	Aufbau und Etablieren selektiver, intelligenter CI/CD Pipeline	30
5.1.3	Nachhaltigkeitsaspekte als Erfolgsindikatoren einführen	30
5.1.4	CO2-Fußabdruck als Metrik für Projekte einführen und kontrollen	31
5.1.5	Einsatzdauer und Beschaffung für IT-Infrastruktur	31
5.1.6	Nachhaltige Beschaffung von Hardware honorieren	31
5.1.7	Aktive Reduzierung von Dokumentenablagen und Speicherbereichen	31
5.1.8	Definition und Aufbau einer Referenz-Energiemessumgebung	32
5.1.9	Zentrale Kompetenzstelle für Nachhaltigkeit im SE-Lifecycle aufbauen	32
5.1.10	Zentrale Expertise zu Nachhaltigkeit aufbauen und vermarkten	32
5.1.11	Aktives HW/SW-Sizing im Kontext Anwendungsbetrieb ermöglichen	32
5.1.12	Aktives Reporting/Monitoring der Auslastung von IT-Lösungen	33
5.1.13	Aufbau einer unternehmensweit nutzbaren Werkzeugkette	33
5.1.14	Aufbau einer Softwarekomponenten-Empfehlung bzw. Grey-List	33
5.1.15	Werkzeugbasierte Projektmanagement-Dokumentationsplattform	34
5.1.16	Klimafreundliche Architekturmuster und Empfehlungen etablieren	34
5.1.17	Bewusstsein zur Nachhaltigkeit im Projekt schärfen	34
5.1.18	Schulung und Training zu Nachhaltigkeitsaspekten in der IT	34
5.1.19	Entwicklung eines Use-Case Effizienz-Labels	35
6.	Deutsches Umweltzeichen Blauer Engel	36

6.1	Blauer Engel für Rechenzentren	36
6.2	Blauer Engel für Softwareprodukte	37
7.	Checkliste	38
8.	Fazit	39
9.	Abbildungsverzeichnis	40
10.	Literaturverzeichnis	41

1. Einleitung

Der Klimawandel ist die größte Herausforderung, der die Menschheit zu Beginn des 21. Jahrhunderts ausgesetzt ist. Niemand zweifelt mehr daran, dass der **Energieverbrauch global reduziert** werden muss, um die **CO₂-Emissionen zu senken** und die Klimaveränderung zumindest abzubremesen. Welche Möglichkeiten hat die IT-Industrie, um Energie und Ressourcen einzusparen und zu mehr Nachhaltigkeit beizutragen? Eine zentrale Technik, um diese Ziele zu erreichen ist **Green Coding**.

Um Nachhaltigkeitsbemühungen in der Softwareentwicklung zu stärken, wurde Rahmen des Projekts **Community Nachhaltige Digitalisierung** des Bundesministeriums für Umwelt, Naturschutz, nukleare Sicherheit und Verbraucherschutz die Workshopreihe „Green Coding“ aufgesetzt.

Die Workshopreihe „Green Coding“ richtete sich an Softwareentwickler*innen und Softwarearchitekt*innen in Unternehmen und vermittelte Skills zur nachhaltigen Softwareentwicklung / Green Coding. Sie wurde im Oktober und November 2023 in sechs Terminen online durchgeführt. Im Folgenden werden die **wichtigsten Erkenntnisse aus den Workshops zusammengefasst**. Die einzelnen Kapitel gliedern sich nach den Inhalten der jeweiligen Workshops.

2. Einführung Green Coding

2.1 Definitionen

Unter **grüner IT** verstehen wir im Folgenden den **Hardwarebetrieb mit grünem Strom** oder allgemein geringem Energieverbrauch. Unter **grüner IT** kann aber auch insgesamt die Betrachtung von nachhaltigen digitalen Infrastrukturen verstanden werden, die auch ressourcenschonende Rechenzentren, nachhaltige IT-Beschaffung im Sinne einer Kreislaufwirtschaft und auch **energie- und ressourceneffiziente Software** verstanden werden.

Green Coding bezieht sich dagegen auf den möglichst **geringen Energieverbrauch der Software**. Bezogen auf die Software geht es insbesondere um die Bereiche **Softwarearchitektur, die konkreten Softwarekomponenten und die Betriebsplattform**. Bezogen auf die Hardwareressourcen ist das Ziel von **Green Coding** die Ermöglichung einer **langen Nutzung** der technischen Infrastruktur durch effizientere Software.

2.2 Unternehmensarchitektur

Um die Nachhaltigkeitstransformation in Unternehmen voranzutreiben, besteht im IT-Bereich die Kernaufgabe der Unternehmensarchitektur darin, das Unternehmensziel eines geringeren ökologischen Fußabdrucks durch geeignete Maßnahmen zu unterstützen.

- **Forcieren von grüner IT**
Eine erste Maßnahme mit unmittelbarer Auswirkung ist das Forcieren von grüner IT. Dazu zählt das Etablieren grüner Rechenzentren sowie der Betrieb von **Software in einer grünen Cloud** - vorausgesetzt, dass die Software überhaupt in der Cloud betrieben werden kann. Die Berechnungen eines Hyperscalers ergaben, dass Rechenzentren in der Cloud um bis zu 93 Prozent energie- und bis zu 98 Prozent CO₂-effizienter seien als firmeneigene Rechenzentren (Becker, et al., 2021).
- **Green Cloud**
Wie grün eine Cloud tatsächlich ist, hängt vor allem vom **Energieverbrauch ihrer Rechenzentren** und dem **Einsatz erneuerbarer Energien** ab. **Effizienter Hardware-Einsatz und Ressourcennutzung** sind weitere Faktoren, die zur

Nachhaltigkeit beitragen, indem sie den Bedarf an lokaler Hardware reduzieren und dynamische Skalierung ermöglichen.

- **Nutzung von Abwärme und Ökostrom**

Falls dedizierte Rechenzentren unverzichtbar sind, sollte zumindest die entstehende **Abwärme sinnvoll genutzt** werden, zum Beispiel, um Firmengebäude zu beheizen. Der **Strom** sollte **möglichst aus erneuerbaren Energien** stammen, dies gilt gleichermaßen für die gesamte Unternehmens-IT. Durch intelligentes Scheduling des Rechenzentrums können komplexe Berechnungen bevorzugt zu Zeiten geplant werden, in denen viel Ökostrom zur Verfügung steht. Wenn beispielsweise die firmeneigene Photovoltaikanlage an sonnigen Tagen zur Mittagszeit größere Mengen Solarstrom produziert, kann dieser optimal für teure Batchläufe genutzt werden.

- **Unternehmensarchitektur**

Eine weitere wichtige Maßnahme ist, dass die Unternehmensarchitektur Green Coding in den Fokus der IT-Architektur und Entwicklung rückt. **Green Coding** muss **als unternehmensweite, nichtfunktionale Anforderung** mit hoher Priorität definiert werden. Daraus sollten konkrete nichtfunktionale Anforderungen sowie „Best Practices“ für IT-Architekten und Entwickler abgeleitet werden. Eine Auswahl an Richtlinien und Ideen wird in den Abschnitten 2.4 und 5.1 vorgestellt. Um ein gemeinsames Verständnis sicherzustellen, sollten darüber hinaus **Schulungsprogramme** für Architekten und Entwickler aufgesetzt werden. Non Functional Requirements (NFRs) und empfohlene Vorgehensweisen lassen sich am besten anhand verständlicher Praxisbeispiele verdeutlichen. Wichtig ist, dass sich die Unternehmensarchitektur auf die firmeneigenen Softwareprodukte mit dem potenziell höchsten Energieverbrauch konzentriert. Dasselbe gilt für sehr große Projekte. Für kleinere bis mittelgroße Projekte obliegt die Verantwortung für die Beachtung von Green-Coding Kriterien der jeweiligen IT-Architektur. So ist die Einhaltung der Green Coding-NFRs bei einer Webanwendung mit mehreren tausend parallelen Nutzern viel kritischer als bei einer Desktop-Anwendung für zehn bis fünfzehn Nutzern.

2.3 IT-Architektur

Unterhalb der Unternehmensarchitekturebene kann die IT-Architektur auf System- oder Komponentenebene einen positiven Beitrag zum Umweltziel leisten.

Um weniger CO₂-Emissionen zu erzeugen, müssen zunächst die **Komponenten mit dem höchsten Energiebedarf ermittelt** werden. Die Ermittlung des Energiebedarfs erfolgt dabei in der Regel ausschließlich über die **Messung des Stromverbrauchs** der

Softwarekomponente. Im Abschnitt 3.1 Performance Engineering gehen wir näher auf die verschiedenen Messverfahren ein. Anschließend können die ermittelten Komponenten idealerweise optimiert werden.

2.3.1 Energiesparen durch Herunterfahren

Die Optimierung kann zum Beispiel darin bestehen, dass Teilkomponenten **bei Leerlauf heruntergefahren** werden. Eine weitere Möglichkeit ist die Abschaltung von Komponenten außerhalb der Geschäftszeiten, z. B. nachts und an Wochenenden. Beim Betrieb eines Clusters mit Docker-Containern kann ein Knoten situationsabhängig hinzugefügt oder entfernt werden.

2.3.2 Lose Kopplung

Durch strikte Entkopplung von Softwarekomponenten verbessert sich nicht nur die Verfügbarkeit und Widerstandsfähigkeit, sondern auch die Ressourceneffizienz. **Kleine Services** mit geringen Hardwareanforderungen **können effizienter sein** als ein großer Monolith, was zu einer sparsameren und insgesamt leistungsfähigeren Lösung führt. Die Entkopplung von Komponenten und Verarbeitungsschritten ermöglicht eine drastische Reduzierung des Energieverbrauchs. Zum Beispiel kann die Auslagerung rechenintensiver Aufgaben in separate Umgebungen während ihrer Ausführung dazu beitragen, eine beträchtliche Menge an Energie zu sparen und das Hauptsystem effizienter zu gestalten (Dr. Geiger, et al., 2021).

2.3.3 Reaktive Programmierung

Die Überlegung, eine **reaktive Gestaltung** statt des traditionellen »One-Thread-per-Request-Modells« für eine HTTP-Schnittstelle einer Webapplikation zu nutzen, bringt trotz höherer Komplexität zahlreiche Vorteile wie gesteigerte Antwortbereitschaft, Widerstandsfähigkeit, Elastizität und eine **effizientere Hardwareauslastung** mit sich. Das reaktive Modell ermöglicht eine Event-basierte Kommunikation über wenige Threads im Gegensatz zu einer Vielzahl von Threads mit langen I/O-waits, was die Leistung verbessert (Dr. Geiger, et al., 2021).

2.4 Anforderungsanalyse und Entwicklung

Im Folgenden stellen wir Green Coding-Verfahren vor, die von Facharchitektur, IT-Architektur und Entwicklung gleichermaßen berücksichtigt werden sollten,

2.4.1 Funktionale Anforderungen

Energieverbrauch **beginnt bei den Anforderungen**. Der Requirements Engineer muss während der Anforderungsanalyse für ein Softwareprojekt **kritisch hinterfragen**, ob Anforderungen wie Echtzeitverarbeitung oder hoch dynamische Inhalte zwingend benötigt werden. Sind sie verzichtbar, können **Lastspitzen vermieden** und dadurch Energie gespart werden.

Natürlich muss ein vernünftiges Gleichgewicht zwischen der Forderung nach einer Minimierung des Energieverbrauchs und den funktionalen Anforderungen immer in Absprache mit dem Kunden gefunden werden.

- **Weglassen von Features**

Ein Mittel zur Steigerung der Effizienz von Produkten ist daher das Weglassen von unsinnigen Anforderungen und Features, die nur möglicherweise in Zukunft benötigt werden. **Das Feature, das erst gar nicht implementiert wird, ist am energieeffizientesten.**

- **Vermeidung von Feature Creep**

"Feature Creep" bezieht sich, insbesondere in der Entwicklung nach Scrum, auf die Tendenz, zusätzliche Anforderungen oder Funktionen zu einem Projekt hinzuzufügen, nachdem bereits mit der Entwicklung begonnen wurde.

Dies kann dazu führen, dass das Produktteam ständig neue Funktionen hinzufügt, was den Fokus von wichtigen Zielen ablenkt und die Sprintziele gefährdet, was letztendlich unnötige Komplexität, Verzögerungen oder Qualitätsprobleme hervorbringt.

Eine Zunahme an Komplexität und zusätzlichen Funktionen in der Software hat ebenfalls negative Auswirkungen auf die Energieeffizienz. Mit **wachsender Komplexität** und steigender Anzahl an Features **steigt** normalerweise **auch der Bedarf an Rechenleistung**. Gleichzeitig bleiben oft notwendige Optimierungen bestehender Softwareteile aufgrund des Hinzufügens neuer Funktionen unberücksichtigt.

- **Optimierte Algorithmen**

Eine der wohl offensichtlichsten Optimierungen von Software in Bezug auf den Energieverbrauch ist die Verwendung des für die Aufgabe am besten geeigneten und leistungsfähigsten Algorithmus. Mit diesem Problem beschäftigen sich bereits ganze Generationen von Programmierer*innen.

Programmierer*innen sollten daher für jede Aufgabe **prüfen, ob** sich ein **bekannter** und **bereits optimierter Algorithmus** anwenden lässt, beziehungsweise ob sich die Aufgabe auf einen bekannten Algorithmus zurückführen lässt. **Eigene Algorithmen zu entwickeln und zu optimieren ist sowohl in wirtschaftlicher als auch ökologischer Hinsicht der teuerste Weg** und sollte daher vermieden werden. Diese Vorgehensweise empfiehlt sich im Übrigen unabhängig von den Green Coding-Anforderungen.

Beispiele:

Der Einsatz einer Berechnungsformel statt einer Schleife ist in der Regel performanter und damit energieeffizienter.

Wenn die Präzision der Berechnung nicht von entscheidender Bedeutung ist, kann anstelle der exakten mathematischen Berechnung eine Näherungsberechnung implementiert werden. Ein anschauliches Beispiel hierfür ist der Fast Inverse Square Root Algorithmus (Plummer, 2021).

2.4.2 Nicht-Funktionale Anforderungen

- **Verminderung der Anzahl von Serveranfragen**

Die Anzahl der Serveranfragen trägt wesentlich zum Energieverbrauch bei. Es muss daher für jede Anfrage abgewogen werden, ob sie wirklich notwendig ist.

So können zum Beispiel Anfragen zusammengefasst, Daten auf dem Client zwischengespeichert oder **Caches** für häufige gleichartige Requests eingerichtet werden. Durch das Nutzen von Caches werden unnötige Roundtrips über das Netzwerk und teure Festplatten- oder Datenbankzugriffe vermieden. Caches sind daher nicht nur ein Weg die Performance zu erhöhen, sondern auch den Energieverbrauch der Software zu senken.

- **Verminderung des Datenvolumens**

Das Datenvolumen kann in der Regel durch Verwendung geeigneter schlanker Dateiformate und Komprimierung verkleinert werden. Ein Beispiel hierfür ist die Verwendung von JSON statt XML für Web-Services, da dieses Format wesentlich weniger Overhead produziert.

Wenn ein Zugriff über das Netzwerk erfolgen muss, so sollten die Daten **komprimiert** übertragen werden. Das spart Bandbreite und vermindert die Kosten der Übertragung. Jeder Router, der an einer Netzwerkübertragung beteiligt ist, benötigt Strom. **Je mehr Daten also übertragen werden, desto länger wird der Strom auch verbraucht.** Das Komprimieren der Daten ist daher eine einfache Möglichkeit, um Energie zu sparen. Für Medien verringern verlustbehaftete Formate wie JPEG, AAC und AV1 die benötigte Bandbreite um ein Vielfaches.

- **Datenbankindizes**

Indizes auf Tabellen gehören zum Standardrepertoire eines Entwicklers. Trotzdem findet man oft Software, die ohne Index auf die Daten zugreift. Mit wachsender Datenmenge verschärft sich das Problem – die Zugriffe werden langsam. Dadurch steigt die CPU- und IO-Last auf der Datenbank, was zu einem hohen Energieverbrauch führt.

- **Explain Plans**

Relationale Datenbankmanagementsysteme werden mit der Anfragesprache **SQL** angesteuert. Dabei wird vom Entwickler im SQL-Statement spezifiziert, was als Ergebnismenge geliefert werden soll – jedoch nicht, wie der Zugriff auf die Daten zur Laufzeit erfolgt. Für diesen Zweck enthält das Datenbanksystem eine Anfrageoptimierungskomponente, die einen möglichst effizienten Ausführungsplan für das eingegebene SQL zu erzeugen versucht. Einschränkend muss man hinzufügen, dass dies leider nicht immer funktioniert, respektive das SQL selbst oftmals noch **Optimierungspotenzial** hat.

Die meisten Datenbanksysteme bieten ein Explain Plan-Tool, mit dem sich die Datenbankentwickler den Ausführungsplan anzeigen lassen und ihre Statements hinsichtlich geringer Kosten optimieren können.

Datenbankentwicklungs- und Abfragetools wie zum Beispiel das Postgres Admin Tool bieten dazu die Möglichkeit, den Ausführungsplan aufzurufen. Ohne die Detailinterpretation verstehen zu müssen, bekommen die Entwickler bei jeder Anpassung der Abfragen Feedback, ob sich die Gesamtkosten, CPU-Kosten sowie IO-Kosten und Speicherverbrauch dadurch positiv oder negativ entwickeln.

QUERY PLAN	
	text 🔒
1	Gather (cost=1000.00..12795.12 rows=1007 width=18) (actual time=62.230..65.675 rows=0 loops=1)
2	Workers Planned: 2
3	Workers Launched: 2
4	-> Parallel Seq Scan on orders (cost=0.00..11694.42 rows=420 width=18) (actual time=12.002..12.003 rows=0 loop...
5	Filter: (customer_id = 5000)
6	Rows Removed by Filter: 336667
7	Planning Time: 0.053 ms
8	Execution Time: 65.691 ms

Abbildung 1: Postgres Ausführungsplan

- **Zero-Waste-Code**

Während in den Anfängen der Softwareentwicklung für jedes Problem eine eigene Lösung erstellt werden musste, sind heute unzählige Open-Source-Bibliotheken für fast jeden Zweck frei verfügbar. Grundsätzlich ist das eine sehr begrüßenswerte Entwicklung (vgl. „Proudly found elsewhere“ statt „Not invented here“). Das Einbeziehen externer Abhängigkeiten erhöht jedoch auch das Risiko, eine Menge überflüssigen Code auszuliefern. Insbesondere im Kontext von Webanwendungen, die bei jeder neuen Benutzersitzung den Download des Codes einschließlich der Abhängigkeiten erfordern, wird durch weitgehend ungenutzte Bibliotheken potenziell viel Bandbreite und CPU-Parsing-Zeit verschwendet.

Eine gute Möglichkeit, „toten“ Code sicher zu entfernen, ist **der Einsatz von Tree-Shaking-Engines**. Im JavaScript-Umfeld sind „module bundlers“ wie „webpack“ oder „Rollup“ weit verbreitet. Sie entfernen beim Zusammenfassen mehrerer JavaScript-Dateien in eine Datei automatisch nicht verwendeten Code. Für Java und Kotlin existiert mit „ProGuard“ ebenfalls ein bekanntes Tool.

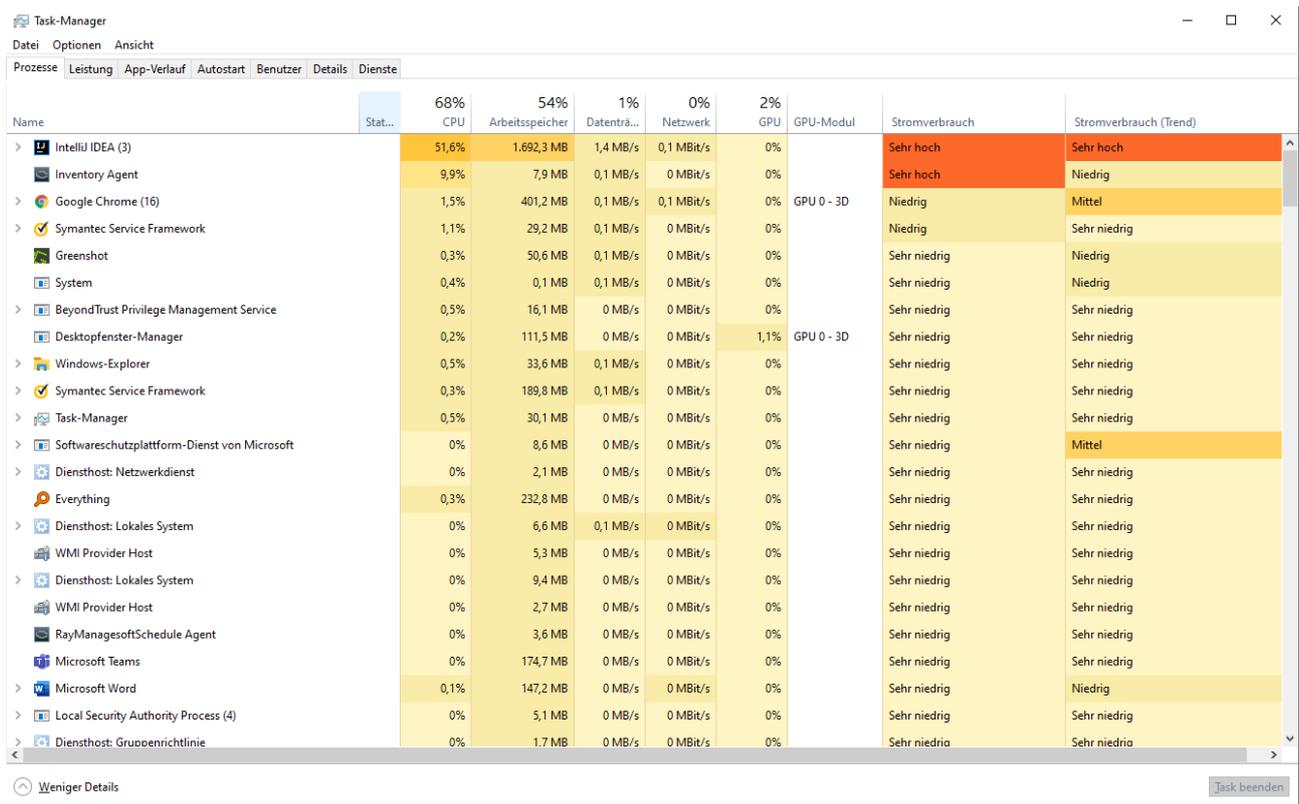
Ein innovativer Ansatz, der Funktionen wie **Bytecode-Optimierung und -Verkleinerung** bietet, ist „GraalVM“. Die Plattform ist eine vielseitige virtuelle Maschine von Oracle, die Java und mehrere andere Programmiersprachen (darunter Java, JavaScript, Python, Ruby, R, C, und C++) unterstützt. Sie verbessert die Leistung von Anwendungen, die in interpretierten Sprachen erstellt wurden, durch einen optimierten Compiler, ermöglicht die Erstellung von **nativen Binärdateien** für schnelleren Start ohne Aufwärmzeit und **geringeren Ressourcenverbrauch** und bietet die Möglichkeit, verschiedene Sprachen in einer einzigen Laufzeitumgebung zu integrieren (GraalVM, 2023).

2.5 Ressourcenverbrauch

Über den gesamten Softwareentwicklungszyklus – von den ersten Inkrementen bis zum Go-Live – ist es wichtig, den Ressourcenverbrauch im Blick zu behalten und darauf hinzuwirken, dass er im Sinne einer hohen Energieeffizienz möglichst niedrig gehalten wird.

Der Ressourcenverbrauch einer Applikation beinhaltet sowohl die **CPU-Zeit als auch die CPU-Auslastung, den Arbeitsspeicherbedarf, Datenträgerspeicherplatz und Datenträgerauslastung sowie die Netzwerklast**. Für die meisten Business-Anwendungen wird der GPU-Verbrauch eher zweitrangig sein. Aber zum Beispiel für Desktop-Anwendungen mit aufwändig gestalteter UI oder Web-Anwendungen mit vielen Medieninhalten sollte er mitberücksichtigt werden.

Einen ersten Anhaltspunkt zur Messung des Ressourcenverbrauchs auf Prozessebene liefert der in Windows enthaltene Task-Manager.



Name	Stat...	68% CPU	54% Arbeitsspeicher	1% Datenträ...	0% Netzwerk	2% GPU	GPU-Modul	Stromverbrauch	Stromverbrauch (Trend)
IntelliJ IDEA (3)		51,6%	1.692,3 MB	1,4 MB/s	0,1 MBit/s	0%		Sehr hoch	Sehr hoch
Inventory Agent		9,9%	7,9 MB	0,1 MB/s	0 MBit/s	0%		Sehr hoch	Niedrig
Google Chrome (16)		1,5%	401,2 MB	0,1 MB/s	0,1 MBit/s	0%	GPU 0 - 3D	Niedrig	Mittel
Symantec Service Framework		1,1%	29,2 MB	0,1 MB/s	0 MBit/s	0%		Niedrig	Sehr niedrig
Greenshot		0,3%	50,6 MB	0,1 MB/s	0 MBit/s	0%		Sehr niedrig	Niedrig
System		0,4%	0,1 MB	0,1 MB/s	0 MBit/s	0%		Sehr niedrig	Niedrig
BeyondTrust Privilege Management Service		0,5%	16,1 MB	0 MB/s	0 MBit/s	0%		Sehr niedrig	Sehr niedrig
Desktopfenster-Manager		0,2%	111,5 MB	0 MB/s	0 MBit/s	1,1%	GPU 0 - 3D	Sehr niedrig	Sehr niedrig
Windows-Explorer		0,5%	33,6 MB	0,1 MB/s	0 MBit/s	0%		Sehr niedrig	Sehr niedrig
Symantec Service Framework		0,3%	189,8 MB	0,1 MB/s	0 MBit/s	0%		Sehr niedrig	Sehr niedrig
Task-Manager		0,5%	30,1 MB	0 MB/s	0 MBit/s	0%		Sehr niedrig	Sehr niedrig
Softwareschutzplattform-Dienst von Microsoft		0%	8,6 MB	0 MB/s	0 MBit/s	0%		Sehr niedrig	Mittel
Diensthost: Netzwerkdienst		0%	2,1 MB	0 MB/s	0 MBit/s	0%		Sehr niedrig	Sehr niedrig
Everything		0,3%	232,8 MB	0 MB/s	0 MBit/s	0%		Sehr niedrig	Sehr niedrig
Diensthost: Lokales System		0%	6,6 MB	0,1 MB/s	0 MBit/s	0%		Sehr niedrig	Sehr niedrig
WMI Provider Host		0%	5,3 MB	0 MB/s	0 MBit/s	0%		Sehr niedrig	Sehr niedrig
Diensthost: Lokales System		0%	9,4 MB	0 MB/s	0 MBit/s	0%		Sehr niedrig	Sehr niedrig
WMI Provider Host		0%	2,7 MB	0 MB/s	0 MBit/s	0%		Sehr niedrig	Sehr niedrig
RayManagesoftSchedule Agent		0%	3,6 MB	0 MB/s	0 MBit/s	0%		Sehr niedrig	Sehr niedrig
Microsoft Teams		0%	174,7 MB	0 MB/s	0 MBit/s	0%		Sehr niedrig	Sehr niedrig
Microsoft Word		0,1%	147,2 MB	0 MB/s	0 MBit/s	0%		Sehr niedrig	Niedrig
Local Security Authority Process (4)		0%	5,1 MB	0 MB/s	0 MBit/s	0%		Sehr niedrig	Sehr niedrig
Diensthost: Gruppenrichtlinie		0%	1,7 MB	0 MB/s	0 MBit/s	0%		Sehr niedrig	Sehr niedrig

Abbildung 2: Windows Task-Manager

Seit Windows 10 Version 1803 zeigt der Task-Manager zusätzlich den Stromverbrauch eines Prozesses an, wenngleich nur als grobe Orientierung.

Weitere Details liefert der Ressourcenmonitor, der über einen Link im Tab „Leistung“ des Task-Managers aufgerufen werden kann. Unter MacOS ist die Aktivitätsanzeige integriert, für

Unix-basierte Betriebssysteme erfüllen die Kommandozeilentools top und dessen Erweiterung htop ähnliche Zwecke.

Für die Analyse des Ressourcenverbrauchs innerhalb der Anwendung, sei es von einzelnen Komponenten, Objekten oder Methoden, eignen sich **Profiling-Tools**.

Viele bekannte Entwicklungsumgebungen wie IntelliJ, eclipse, Visual Studio oder VS Code bringen bereits Profiler für die geläufigsten Programmiersprachen Java, C/C++, C# und weitere mit. Für TypeScript und JavaScript enthalten die Entwicklertools der verbreitetsten Browser Chrome, Edge und Firefox entsprechende Werkzeuge. Falls für die Programmiersprache der Wahl kein Profiler verfügbar ist, kann zur Not auf trace-Logging mit Zeitstempel und den wichtigsten Parametern wie CPU-Last, RAM-Verbrauch etc. ausgewichen werden.

Für Software, die auf der Java Virtual Machine läuft, können **Lasttest-Frameworks** wie Java Microbench Harness und JMeter helfen, den Ressourcenverbrauch unter Stress einzuschätzen.

Durch Lasttests lassen sich in der Regel Speicherlecks oder Komponenten und Funktionen, die einen Flaschenhals mit übermäßigem CPU-Verbrauch darstellen, schnell ermitteln.

3. Messtools

3.1 Performance Engineering

„Das ‚Performance-Engineering‘ ist ein ganzheitlicher Ansatz, der Leistungsanforderungen von Beginn an als integralen Bestandteil der Entwicklung eines neuen Produktes sieht“ (Pustina, 2019). Performance Engineering etabliert dazu Techniken, um sicherzustellen, dass die nicht-funktionalen Anforderungen (wie Durchsatz, Latenzzeit oder Speichernutzung) erfüllt werden (Hager & Wellein, 2018).

Derartige Techniken gilt es auch für die nicht-funktionale Anforderung „geringer Energieverbrauch“ zu etablieren. Zu diesem Zweck muss der Energieverbrauch während der Entwicklung, aber auch später im Betrieb, gemessen und gegebenenfalls Maßnahmen ergriffen werden. Voraussetzung dafür ist wiederum, dass die Anforderung mit einer messbaren Bedingung formuliert wird.

Beispielsweise könnte eine nicht-funktionale Anforderung lauten:

Komponente X in System Y soll auf einem Intel Prozessor mit 2 Cores nicht mehr als 0.1 kWh Strom verbrauchen, wenn die Berechnung Z ausgeführt wird.

Energieverbrauch in der Softwareentwicklung wird im Folgenden vereinfacht mit Stromverbrauch gleichgesetzt. Grundsätzlich fallen auch bei der Herstellung und Entsorgung der benötigten Hardware CO₂-Kosten und weitere Ressourceninanspruchnahmen wie seltene Erden an. Auf diese sogenannten Scope 2 und 3 Kosten (vgl. zum Beispiel (Lehmkuhl, 2023)) gehen wir in diesem Abschnitt nicht näher ein, da sie im Unternehmenskontext beim Einkauf beziehungsweise der IT-Beschaffung berücksichtigt werden sollten. Das Umweltzeichen „**Blauer Engel**“ des Umweltbundesamtes versucht die **Lebensdauer** von Hardware durch energie- und ressourceneffiziente Software zu **verlängern**, in dem diese Aspekte adressiert werden (vgl. Abschnitt 6).

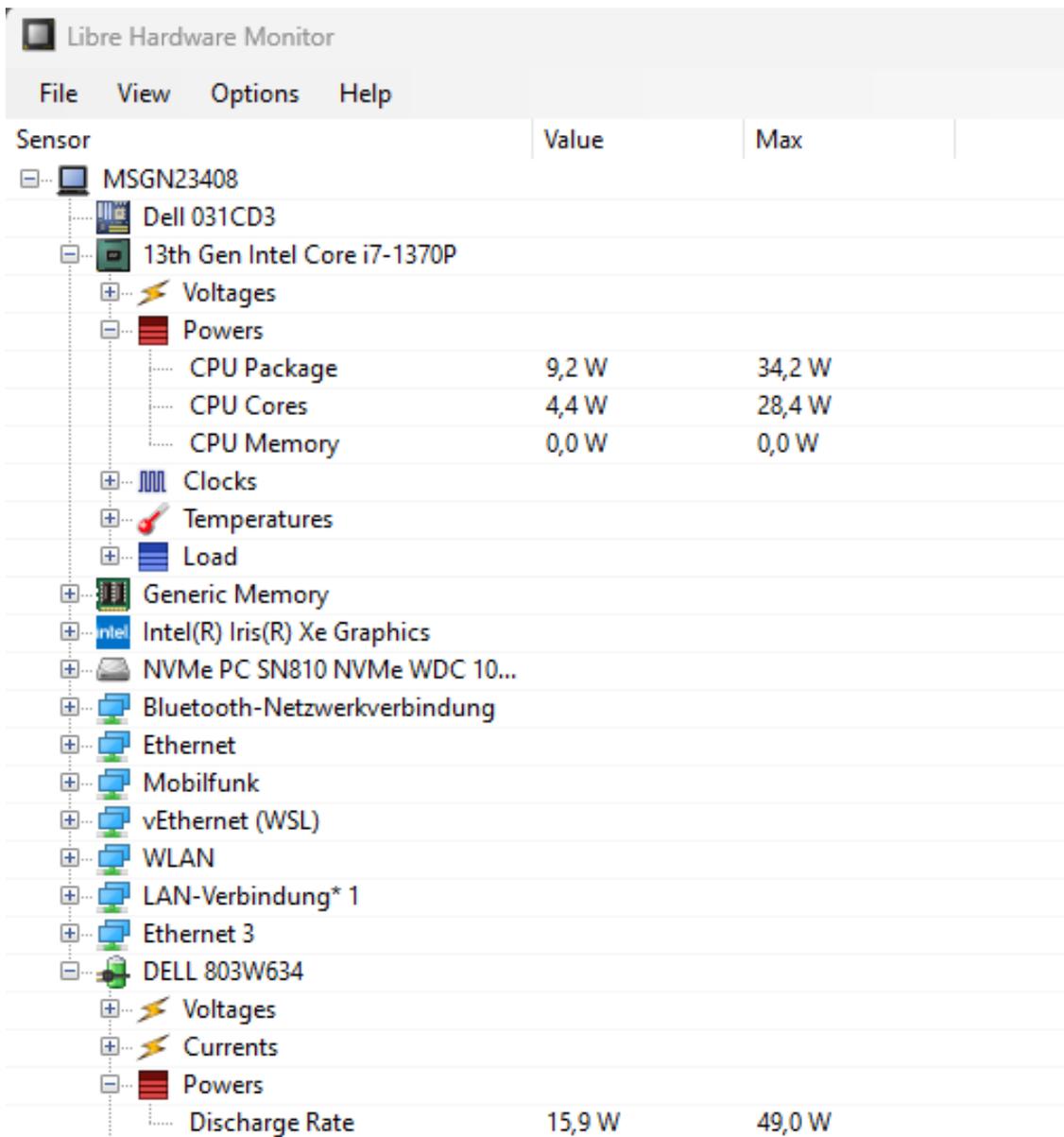
Wie kommt man nun zu der Kenngröße „Energieverbrauch“ für eine zu lösende Programmieraufgabe? Zunächst ist festzulegen, wie der Stromverbrauch gemessen wird. Grundsätzlich muss man unterscheiden, ob absolut oder relativ gemessen wird. **Relative Messungen** messen die Zunahme des Stroms in Prozent und sind in der Regel einfacher durchzuführen und allgemeingültiger. **Absolute Messungen** sind hingegen abhängig von der Zielplattform.

Entweder misst man den Stromverbrauch des gesamten Systems mittels eines Smart Plugs (Producto GmbH, 2023) **an der Steckdose**. Diese Messmethode ist jedoch sehr ungenau und man muss die Differenz des Stromverbrauchs ohne und mit laufendem Programm berechnen.

Alternativ kann **indirekt** über die **Anzahl** der benötigten **CPU-Zyklen** für eine Operation oder für ein ausgeführtes Programm der Verbrauch hochgerechnet werden.

Die dritte Möglichkeit ist, den Stromverbrauch der CPU über **hardwarenahe Software** auszulesen. Dies ist in der Regel die genaueste Messmethode, da bei modernen Hardwarebausteinen die Informationen zum gerade verbrauchten Strom über eine Schnittstelle ausgelesen werden können.

Beispielsweise kann das Tool „Libre Hardware Monitor“ (Möller, 2023) den Stromverbrauch auslesen.



Sensor	Value	Max
MSGN23408		
Dell 031CD3		
13th Gen Intel Core i7-1370P		
Voltages		
Powers		
CPU Package	9,2 W	34,2 W
CPU Cores	4,4 W	28,4 W
CPU Memory	0,0 W	0,0 W
Clocks		
Temperatures		
Load		
Generic Memory		
Intel(R) Iris(R) Xe Graphics		
NVMe PC SN810 NVMe WDC 10...		
Bluetooth-Netzwerkverbindung		
Ethernet		
Mobilfunk		
vEthernet (WSL)		
WLAN		
LAN-Verbindung* 1		
Ethernet 3		
DELL 803W634		
Voltages		
Currents		
Powers		
Discharge Rate	15,9 W	49,0 W

Abbildung 3: Beispielansicht Libre Hardware Monitor

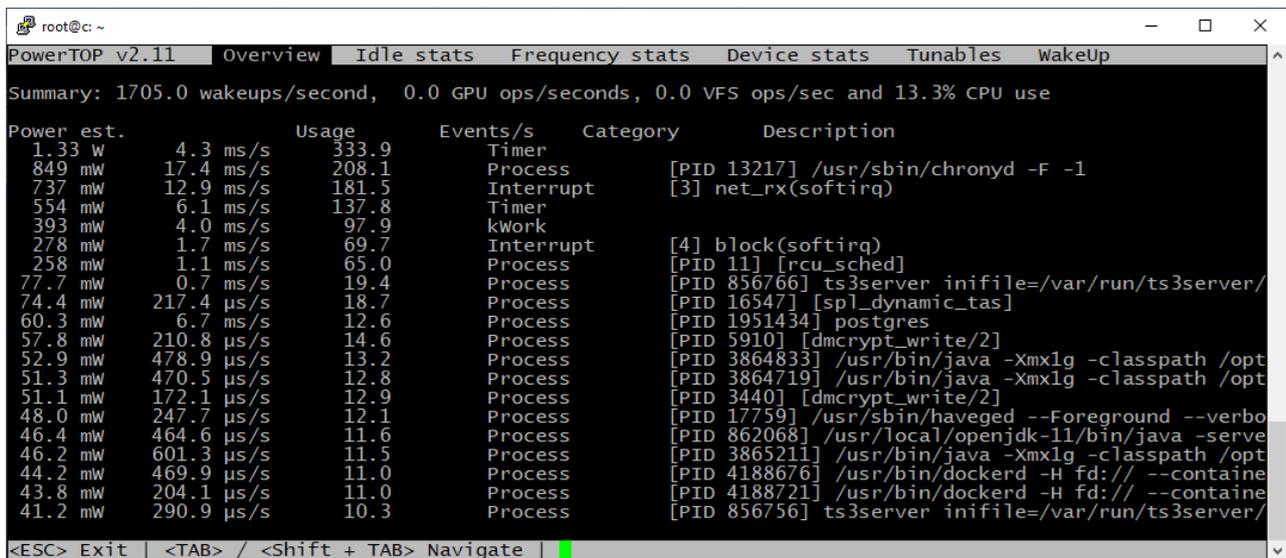
Neben der Messmethode – also dem wie – gibt es auch mehrere Ansätze, **wann im Softwareentwicklungszyklus der Stromverbrauch gemessen** werden soll.

So kann mit einem **Plugin für Unittests** neben den funktionalen Prüfungen auch der Stromverbrauch gemessen und mit einer Sollgröße abgeglichen werden (vgl. Abschnitt 3.4.2). Wie oben erwähnt, kann man damit die Messung über die verbrauchten CPU-Zyklen durchführen und dann auf die Plattform hochrechnen. In Integrationstests kann der Verbrauch einer Komponente gemessen werden.

Ein **Monitoring der Gesamtapplikation** kann auf der Zielplattform im laufenden Betrieb erfolgen (vgl. Abschnitte 4.3 und 5.1.12). Um die NFR „geringer Stromverbrauch“ einzuhalten, wird idealerweise **in allen Phasen der Entwicklung** gemessen. Je früher eine Komponente

auffällt, desto besser kann man auf negative Entwicklungen reagieren. Wie bei allen nicht-funktionalen Anforderungen muss auch der Energieverbrauch an die Gegebenheiten des Kunden, wie zum Beispiel Betriebssystem und Zielplattform, angepasst werden.

Es gibt bereits verschiedene Tools für Unix basierte Systeme, wie zum Beispiel PowerTOP, das neben der CPU-Usage auch den Stromverbrauch in Watt ausgibt.



```

root@ci: ~
PowerTOP v2.11  Overview  Idle stats  Frequency stats  Device stats  Tunables  WakeUp
Summary: 1705.0 wakeups/second, 0.0 GPU ops/seconds, 0.0 VFS ops/sec and 13.3% CPU use

Power est.      Usage      Events/s    Category    Description
1.33 W          4.3 ms/s   333.9       Timer       [PID 13217] /usr/sbin/chronyd -F -1
849 mW         17.4 ms/s  208.1       Process
737 mW         12.9 ms/s  181.5       Interrupt   [3] net_rx(softirq)
554 mW         6.1 ms/s   137.8       Timer
393 mW         4.0 ms/s   97.9        kWork
278 mW         1.7 ms/s   69.7       Interrupt   [4] block(softirq)
258 mW         1.1 ms/s   65.0       Process     [PID 11] [rcu_sched]
77.7 mW        0.7 ms/s   19.4       Process     [PID 856766] ts3server inifile=/var/run/ts3server/
74.4 mW        217.4 µs/s  18.7       Process     [PID 16547] [spl_dynamic_tas]
60.3 mW        6.7 ms/s   12.6       Process     [PID 1951434] postgres
57.8 mW        210.8 µs/s  14.6       Process     [PID 5910] [dmccrypt_write/2]
52.9 mW        478.9 µs/s  13.2       Process     [PID 3864833] /usr/bin/java -Xmx1g -classpath /opt
51.3 mW        470.5 µs/s  12.8       Process     [PID 3864719] /usr/bin/java -Xmx1g -classpath /opt
51.1 mW        172.1 µs/s  12.9       Process     [PID 3440] [dmccrypt_write/2]
48.0 mW        247.7 µs/s  12.1       Process     [PID 17759] /usr/sbin/haveged --Foreground --verbo
46.4 mW        464.6 µs/s  11.6       Process     [PID 862068] /usr/local/openjdk-11/bin/java -serve
46.2 mW        601.3 µs/s  11.5       Process     [PID 3865211] /usr/bin/java -Xmx1g -classpath /opt
44.2 mW        469.9 µs/s  11.0       Process     [PID 4188676] /usr/bin/dockerd -H fd:// --containe
43.8 mW        204.1 µs/s  11.0       Process     [PID 4188721] /usr/bin/dockerd -H fd:// --containe
41.2 mW        290.9 µs/s  10.3       Process     [PID 856756] ts3server inifile=/var/run/ts3server/

<ESC> Exit | <TAB> / <Shift + TAB> Navigate
  
```

Abbildung 4: PowerTOP Tool

3.2 Stromverbrauch auf Prozessebene messen

Windows bringt mit der Windows Energy Estimation Engine (E3) (Chamberlin, 2020) ein Werkzeug mit, das den Energieverbrauch auf Prozess-Ebene misst.

Prozess bezeichnet in diesem Kontext die Ausführung eines einzelnen Programms.

E3 bedingt, dass es auf einem Gerät ausgeführt wird, das mit Akku betrieben werden kann.

Für erste „Gehversuche“ kann auf einem Notebook in einer Kommandozeile mit Admin-Rechten der Befehl **powercfg.exe /srumutil** aufgerufen werden.

Das Ergebnis ist eine srumutil.csv-Datei, die den Energieverbrauch aller Prozesse der letzten 7 Tage auf dem Computer enthält.

Um das Resultat möglichst auf die eigene Anwendung zu beschränken können folgende Schritte in cmd durchgeführt werden:

- 1.) Mit `sc stop dps` die Datenaufzeichnung stoppen.
- 2.) Zur Sicherheit mit `move C:\Windows\System32\sru\SRUDB.dat C:\Windows\System32\sru\SRUDB.dat.bak` die historischen Daten sichern.
- 3.) Mit `sc start dps` die Datenerhebung erneut starten.

4.) Jetzt ausschließlich die zu messende Anwendung ausführen.

5.) Nach Programmende powercfg.exe /srumutil aufrufen.

Aufbereitet und um drei Spalten ergänzt (Messwerte werden von E3 in Millijoule ausgegeben, Umrechnungs-Formeln siehe Abbildung 5) ist erkennbar, dass eine Woche Microsoft Teams auf einem Business-Notebook gut 0,05kWh und damit, basierend auf dem aktuellen Strommix Deutschland (Umweltbundesamt, 2023), ca. 0,02kg CO2 verbraucht.

AppId	TimeStamp	CPUEnergyConsumption	TotalEnergyConsumption	Joule	kWh	gCO ₂ , 485g/kWh
Teams.exe	2022-07-11:13:24:00.0000	8367366	8613698	8613,6980000000	0,0023926939	1,160456536
Teams.exe	2022-07-11:07:19:00.0000	7591854	7771766	7771,7660000000	0,0021588239	1,047029586
Teams.exe	2022-07-15:08:17:00.0000	6054067	6300375	6300,3750000000	0,0017501042	0,848800521
Teams.exe	2022-07-15:07:16:00.0000	5639669	5975514	5975,5140000000	0,0016598650	0,805034525
Teams.exe	2022-07-11:13:24:00.0000	93601	5726417	5726,4170000000	0,0015906714	0,771475624
Teams.exe	2022-07-14:13:45:00.0000	4662304	4771426	4771,4260000000	0,0013253961	0,642817114
Teams.exe	2022-07-12:11:50:00.0000	4117680	4203548	4203,5480000000	0,0011676522	0,566311328
Teams.exe	2022-07-14:12:44:00.0000	20931	4151090	4151,0900000000	0,0011530806	0,559244069
Teams.exe	2022-07-11:12:23:00.0000	3884994	3965081	3965,0810000000	0,0011014114	0,534184524
Teams.exe	2022-07-12:12:51:00.0000	3755729	3812311	3812,3110000000	0,0010589753	0,51360301
Teams.exe	2022-07-15:07:16:00.0000	56793	3622730	3622,7300000000	0,0010063139	0,488062236
Teams.exe	2022-07-14:13:45:00.0000	65229	3547574	3547,5740000000	0,0009854372	0,477937053
Teams.exe	2022-07-13:11:47:00.0000	17818	3525679	3525,6790000000	0,0009793553	0,47498731
Teams.exe	2022-07-15:10:19:00.0000	31506	3389300	3389,3000000000	0,0009414722	0,456614028
Teams.exe	2022-07-13:10:46:00.0000	3224	3177837	3177,8370000000	0,0008827325	0,428125263
Teams.exe	2022-07-11:09:20:00.0000	28444	2974994	2974,9940000000	0,0008263872	0,400797803
Teams.exe	2022-07-13:11:47:00.0000	2590548	2671823	2671,8230000000	0,0007421731	0,359953932
Teams.exe	2022-07-11:09:20:00	Formel Joule		Formel kWh	Formel gCO ₂	
Teams.exe	2022-07-13:07:44:00	=@[[TotalEnergyConsumption]]/1000		=[@Joule]/36000000	=[@kWh]*485	Energiemix DE 2021 485 gCO ₂ / kWh
Teams.exe	2022-07-15:10:19:00					
Teams.exe	2022-07-08:12:19:00	Σ Joule		Σ kWh	Σ gCO ₂	
Teams.exe	2022-07-14:07:41:00	164153,0150000000		0,0455980597	22,1150589653	
Teams.exe	2022-07-14:11:44:00					

Abbildung 5: Energieverbrauch MS Teams in ca. einer Arbeitswoche

Es gilt zu beachten, dass die ermittelten Messwerte nicht 1:1 den Verbrauch an der Steckdose wiedergeben, da Faktoren wie Verluste der AC/DC Wandler, Lüfter, GPU-Stromverbrauch etc. nicht berücksichtigt werden.

Dennoch gibt E3 einen Anhaltspunkt, wieviel Energie ein Prozess benötigt.

3.3 Energieverbrauchsmessung auf Funktionsebene

In einen Prozess "hineinsehen" kann man mit E3 nicht. D. h. es kann keinerlei Aussage darüber getroffen werden, **welche Komponente oder Funktion innerhalb des Prozesses am meisten Energie verbraucht**. Exakt diese Information ist für Entwickler*innen jedoch essenziell – wie soll gezielt hinsichtlich Energieeffizienz optimiert werden, wenn der genaue Ansatzpunkt unbekannt ist?

Man könnte mit einem Profiler die Funktionen der Applikation mit der höchsten CPU-Last ermitteln. In Kombination mit bekannter TDP (Wikipedia, 2023) der CPU oder Messwerten zum Stromverbrauch wäre es möglich, aus den gewonnenen Profildaten Schlüsse auf den Energieverbrauch zu extrapolieren. Dieses Vorgehen halten wir jedoch für unbequem und ungenau.

Um Stromverbrauchsmessungen als festen Bestandteil des Entwicklungsprozesses zu etablieren, muss dies für Entwickler*innen möglichst komfortabel sein. Die Messwerte sollten in einer passenden Skala einfach ablesbar sein. Ungenau ist das Vorgehen, da Energieverbrauchsmessung und Profiling nicht unmittelbar miteinander korrelieren - man misst jeden Faktor einzeln.

3.4 jPowerMonitor

Diese Lücke füllt die jPowerMonitor Java-Library (Deiner & Keilhofer, 2023). Die enthaltenen Komponenten **jUnit-Extension** und **Java Agent** ermöglichen direkte Messungen des Energieverbrauchs, auf Unit-Test respektive Methoden-Ebene eines beliebigen Java-Prozesses. Einzig ein ausführbares JAR-File wird benötigt - für Server-Anwendungen muss ein Java Agent konfiguriert werden.

Beide Erweiterungen schreiben während des Messlaufs den Momentan-Verbrauch in Watt sowie den kumulierten Energieverbrauch pro Testfall in Wattstunden bzw. pro Methode in Joule in separate CSV-Dateien.

3.4.1 Stromverbrauchsmessungstools

jPowerMonitor überwacht zur Laufzeit der Applikation, welche Funktionen gerade aktiv sind und wie viel CPU-Zeit sie in Anspruch nehmen (analog einem Profiler). Die zur Ermittlung des Energieverbrauchs zusätzlich notwendigen Messdaten wie z. B. „CPU Package Power“ (aktueller Gesamtverbrauch der CPU in Watt) kann jPowerMonitor nicht selbst abgreifen.

Zu diesem Zweck ist eine **konfigurierbare Schnittstelle** implementiert, welche die beiden Windows-Tools Libre Hardware Monitor (LHM) (Möller, 2023) per REST sowie HWiNFO (Malík, 2023) per CSV-Dateischnittstelle anbindet. Die CSV-Schnittstelle ist generisch, sodass sich beliebige Strommess-Tools verwenden lassen, die Messwerte periodisch nach CSV exportieren.

LHM und HWiNFO liefern eine Fülle an Messpunkten. Für Strommesszwecke empfehlen wir „CPU Package Power“ (HWiNFO) respektive „Powers“ -> „CPU Package“ (LHM).

Rechenintensive **Business-Applikationen** dürften den **Hauptteil ihres Stromverbrauchs durch CPU-Last** erzeugen.

Ist bekannt, dass die Anwendung eher RAM-, HDD/SSD- oder GPU-lastig ist, kann man stattdessen die korrespondierenden Sensoren oder „Total System Power“ (HWiNFO) konfigurieren.

Damit jPowerMonitor Verbrauchsdaten bekommt, ist bei LHM der „Remote Web Server“ zu aktivieren. Im Fall von HWiNFO ist das CSV-Monitoring einzuschalten. Eine Kurzübersicht der Konfigurationen ist in Abbildung 6 zusammengefasst.

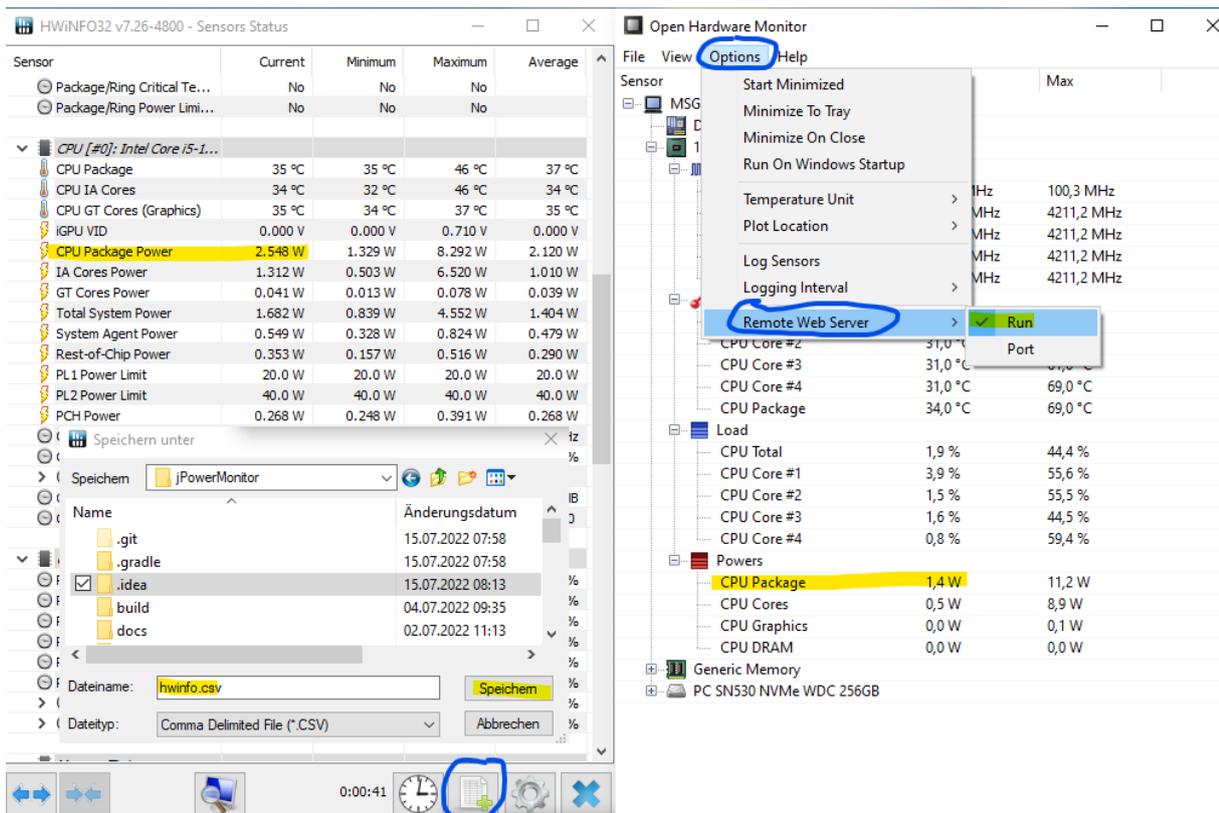


Abbildung 6: Messwert und Konfiguration HWiNFO und Libre Hardware Monitor

Wir empfehlen, die exportierten Sensoren zu minimieren. Standardmäßig schreibt HWiNFO mehr als 100 Spalten pro Zeile – jede Sekunde. Das Logging kann im Einstellungsdialog konfiguriert werden.

3.4.2 JUnit Extension

Unittests sind seit langem **fester Bestandteil des Softwareentwicklungsprozesses**. Tests bieten diverse Vorteile und werden regelmäßig über den Entwicklungs- und Wartungsprozess hinweg verwendet (Roden, 2021)

Die Extension könnte neuer Bestandteil des etablierten Java-Unittestings mit JUnit werden. Softwareentwickler sollen in ihrer täglichen Arbeit unterstützt werden. Die Verwendung soll möglichst einfach und unkompliziert sein.

Es kann konfiguriert werden, ob in einer Initialisierungsphase die **Grundlast** gemessen und später vom Verbrauch abgezogen wird. Oder alternativ, ob die Grundlast fix vorgegeben wird. Auf jedem PC laufen in der Regel verschiedenste Hintergrundprozesse, die den Stromverbrauch auch während des Tests beeinflussen. Um beispielsweise zwei Algorithmen in ihrer Energieeffizienz zu vergleichen, sollten daher möglichst alle anderen Programme geschlossen, die **Tests** mehrfach **wiederholt** und dann die **Mittelwerte** der Messungen **verglichen** werden.

- **Verwendung der Extension**

Um die Extension zu registrieren und damit für die eigenen Unittests zu verwenden, kann die Annotation **@ExtendWith** verwendet werden.

Die Annotation kann mehrfach zu einem Test hinzugefügt werden oder eine Liste von Extensions als Parameter enthalten (Baeldung, 2021).

Beispiel für Registrierung der jPowerMonitor Unittest Extension:

```

@EnabledOnOs(OS.WINDOWS)
@ExtendWith({JPowerMonitorExtension.class})
@Slf4j
public class EndlessLoopTest {

    @RepeatedTest(10)
    void endlessLoopCPUStressTest() {
        long ranSecs = StressCpuExample.runMeasurment();
        Assertions.assertTrue(condition: StressCpuExample.isStressCpuStable(ranSecs));
    }
}
    
```

Abbildung 7: Verwendung der JUnit Extension und @RepeatedTests Annotation

Am besten werden die Testfälle als **@RepeatedTests** (JUnit, 2023) ausgeführt, um ein genaueres Messergebnis zu erzielen. Bei der Annotation kann mitgegeben werden, wie oft ein Test wiederholt werden soll. Zum Beispiel immer 10-mal wiederholen. Im Ergebnis-CSV kann dann der Durchschnitt der Ergebnisse berechnet werden.

- **Messergebnisse**

Die Extension ermöglicht es den **Energieverbrauch einzelner Unittests** zu ermitteln und schreibt diese in eine **CSV-Datei** für spätere Überwachung und Analysen. In der CSV-Datei sind alle berücksichtigten Messpunkte enthalten.

Hier ein Beispiel, wie die Ergebnis-CSV-Datei aussehen könnte:

Uhrzeit	Name	Sensor	Wert	Einheit	Grundlast	Einheit	Wert-Grundl	Einheit	Energie(Wert	Einheit	Energie(Wert	Einheit
2022/07/20T11:57:50	EndlessLoop	MSGN19884	25,25	W	0	W	25,25	W	0,10536	Wh	0,10536	Wh
2022/07/20T11:58:05	EndlessLoop	MSGN19884	14,23	W	0	W	14,23	W	0,0593	Wh	0,0593	Wh
2022/07/20T11:58:10	MyTest->myF	MSGN19884	13,42	W	0	W	13,42	W	0,01818	Wh	0,01818	Wh
2022/07/20T11:58:15	MyTest->myF	MSGN19884	12,14	W	0	W	12,14	W	0,01517	Wh	0,01517	Wh
2022/07/20T11:58:20	MyTest->myF	MSGN19884	11,48	W	0	W	11,48	W	0,014	Wh	0,014	Wh
2022/07/20T12:16:14	EndlessLoop	MSGN19884	25,36	W	0	W	25,36	W	0,10582	Wh	0,10582	Wh
2022/07/20T12:16:29	EndlessLoop	MSGN19884	14,35	W	0	W	14,35	W	0,0598	Wh	0,0598	Wh

Abbildung 8: Beispiel Ergebnis CSV Extension

3.4.3 Java Agent

Java Agents sind Teil der Java Instrumentation API. Die Instrumentation API (Oracle, 2023) bietet die Möglichkeit zur Änderung des Java Bytecodes und **kann dynamisch Code zu einem Programm hinzufügen**. Beispielsweise nutzen Profiling Tools zur Messung von Performance und Speicherverbrauch Java Agents.

jPowerMonitor implementiert ebenfalls einen Java Agent und **klinkt sich somit in bestehenden Quellcode ein, ohne dass dieser angepasst werden muss**.

jPowerMonitor protokolliert den Energieverbrauch der gerade laufenden Methode in dem Java-Programm, an das der Agent angehängt wird.

- **Startup und Konfiguration**

Der Java Agent wird mit der Option `-javaagent:./jpowermonitor-1.1.2-all.jar` an den Java Prozess übergeben.

jPowerMonitor sucht per Default nach einer Konfigurationsdatei `jpowermonitor.yaml` im aktuellen Verzeichnis und im Classpath. Die Konfiguration kann alternativ auch direkt übergeben werden, indem sie mit einem Gleichheitszeichen an den Jar Namen angehängt wird:

```
-javaagent:./jpowermonitor-1.1.2-all.jar=my-config.yaml
```

Die Konfiguration legt fest, mit welchem Strommesstool gemessen wird. Aktuell kann hier „lhm“ für Libre Hardware Monitor oder „csv“ für ein beliebiges Tool, das CSV-Dateien erzeugt, z.B. HWINFO (Malík, 2023) und die Messwerte darin ablegt, eingetragen werden. Die Messwerte aus dem Libre Hardware Monitor werden über die JSON Schnittstelle des Tools abgefragt. In der Konfiguration muss demnach der Pfad zum gewünschten Messwert hinterlegt werden. Beispielsweise würde der Pfad ['<Computername>', 'Intel Core i7-9850H', 'Powers', 'CPU Package'] den CPU Package Stromverbrauch auf einem Intel Prozessor auslesen. Die genauen Bezeichnungen müssen dem Tool Libre Hardware Monitor entnommen werden.

In der Konfigurationsdatei muss die Spaltennummer (0-basiert) des gewünschten Messwerts für die CSV-Datei angegeben werden, zusammen mit der Angabe, ob der jüngste Datensatz in der ersten oder letzten Zeile der CSV-Datei zu finden ist.

In der Konfiguration des Java Agents kann darüber hinaus angegeben werden, wie oft die Messwerte ausgelesen werden. **Es ist möglich, die Messung auf Java Packages einzuschränken**. So können zum Beispiel nur Klassen aus dem Package `group.msg.*` gemessen werden.

Weitere **Konfigurationsdetails entnehmen Sie bitte der Dokumentation** des jPowerMonitor Tools. Beispielsweise ist es auch möglich, das Abfrageintervall oder den CO₂-Emissionsfaktor für den Strommix von kWh in Gramm CO₂ zu konfigurieren.

Das Aufrufbeispiel in Abbildung 9 zeigt die Verwendung des Java Agent mit einer Spring Boot Batch Applikation:

```

$ ./startup.sh --profile=batch --opts=javaagent:jPowerMonitor-0.1.0-SNAPSHOT-all.jar
PWD=/c:/ws/eva_real.pa/build/distributions/eva_real.pa-2022.7.0-SNAPSHOT
JAVA_HOME=C:\dev\jdk-11.0.6.10-hotspot
JAVA_OPTS=-Xms128m -Xmx1024m -Dfile.encoding=UTF-8 -Xshare:auto -Djava.awt.headless=true -XX:+UseG1GC -javaagent:jPowerMonitor-0.1.0-SNAPSHOT-all.jar
SERVER_PORT=8080
NON_INTERACTIVES
Executing bin/eva_real.pa --server.port=8080 --spring.profiles.active=batch --spring.config.location=classpath:/,file:./etc/ --logging.config=./etc/logback.xml
measuring power with jPowerMonitor, Version 0.1.0-SNAPSHOT
-----
Reading jPowerMonitor configuration from filesystem: 'jPowerMonitor.yaml'
main: Start monitoring application with PID 17128

Spring
:: Spring Boot :: (v2.7.1)

2022-07-30 16:04:38.293 INFO 17128 --- [main] com.msg.evareal.EvaRealPaApplication : Starting EvaRealPaApplication v2022.7.0-SNAPSHOT using Java 11.0.6
2022-07-30 16:04:38.295 INFO 17128 --- [main] com.msg.evareal.EvaRealPaApplication : The following 1 profile is active: 'batch'
2022-07-30 16:04:39.124 INFO 17128 --- [main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFAULT mode.
2022-07-30 16:04:39.210 INFO 17128 --- [main] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 75 ms. Found 3 JPA repositories
2022-07-30 16:04:39.848 INFO 17128 --- [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Starting...
2022-07-30 16:04:40.152 INFO 17128 --- [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Start completed.
2022-07-30 16:04:40.332 INFO 17128 --- [main] o.hibernate.jpa.internal.util.LogHelper : HHH000204: Processing PersistenceUnitInfo [name: default]
2022-07-30 16:04:40.405 INFO 17128 --- [main] org.hibernate.Version : HHH000412: Hibernate ORM core version 5.6.9.Final
2022-07-30 16:04:40.637 INFO 17128 --- [main] o.hibernate.annotations.common.Version : HCANN000001: Hibernate Commons Annotations {5.1.2.Final}
2022-07-30 16:04:41.129 INFO 17128 --- [main] org.hibernate.dialect.Dialect : HHH000400: Using dialect: org.hibernate.dialect.H2Dialect
2022-07-30 16:04:41.422 WARN 17128 --- [main] org.hibernate.id.UUIDHexGenerator : HHH000409: Using org.hibernate.id.UUIDHexGenerator which does not
  UUIDGenerator instead
2022-07-30 16:04:42.314 INFO 17128 --- [main] o.h.e.t.j.p.i.JtaPlatformInitiator : HHH000490: Using JtaPlatform implementation: [org.hibernate.engine
2022-07-30 16:04:42.435 INFO 17128 --- [main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
    
```

Abbildung 9: Aufrufbeispiel Java Agent mit Spring Anwendung

Beim Beenden des Java Prozesses wird eine Zusammenfassung der verbrauchten Energie ausgegeben (Abbildung 10):

```

2022-07-30 16:07:06.659 INFO 17128 --- [main] e.p.l.JobCompletionNotificationListener : Batch Job 2/2 finished
2022-07-30 16:07:06.664 INFO 17128 --- [main] c.m.e.p.e.CalculationResultsExporter : exportToExcel - creating workbook: c:/ws/eva_real.pa/build/distrib
utions/eva_real.pa-2022.7.0-SNAPSHOT/eva-output/Gesamtergebnis-Job-Id-272.xlsx
2022-07-30 16:07:13.320 INFO 17128 --- [main] c.m.e.p.e.CalculationResultsExporter : Written 9184 rows to sheet Ergebnisse
2022-07-30 16:07:13.981 INFO 17128 --- [main] o.s.b.c.l.support.SimpleJobLauncher : Job: [SimpleJob: [name=assessEnvironmentalRisk]] completed with th
e following parameters: {run_id=247} and the following status: {COMPLETED} in 2019s88ms
Power measurement ended gracefully
2022-07-30 16:07:13.990 INFO 17128 --- [ionShutdownHook] j.LocalContainerEntityManagerFactoryBean : Closing JPA EntityManagerFactory for persistence unit 'default'
2022-07-30 16:07:13.993 INFO 17128 --- [ionShutdownHook] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Shutdown initiated...
-----
jPowerMonitorAgent successfully finished monitoring application with PID 17128
Application consumed 1160,68 joule - 0,322 wh - 0,000322 kwh - 0,156 gCO2 total
Energy consumption per method and filtered methods written to 'jPowerMonitor_17128_energy_per_method.csv' / 'jPowerMonitor_17128_energy_per_method_filtered.csv'
-----
2022-07-30 16:07:14.290 INFO 17128 --- [ionShutdownHook] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Shutdown completed.
    
```

Abbildung 10: Ergebnis der Messung Java Agent

Im vorliegenden Fall wurden ca. 0,32 Wh Strom bzw. 0,16 g CO2 für den Programmlauf verbraucht.

3.4.4 Fazit

jPowerMonitor hilft Entwicklern, herauszufinden, **an welchen Stellen des eigenen Programms der meiste Strom verbraucht** wird. Dies ermöglicht es, den Code im Sinne von Green Coding zu optimieren und zum Ziel der Reduzierung von CO2-Emissionen beizutragen.

Das Tool erhebt nicht den Anspruch, absolute Stromverbrauchswerte zu liefern, bietet aber die Möglichkeit, **verschiedene Implementierungen bezüglich des Stromverbrauchs zu vergleichen**.

Auf Windows kann die Open-Source-Software "Libre Hardware Monitor" zum Auslesen der Stromverbrauchswerte genutzt werden, mittels CSV-Schnittstelle lassen sich durch Konfigurationsanpassungen Tools für Linux oder MacOS anbinden. Sowohl der Java Agent als auch die JUnit Extension des jPowerMonitor schließen eine Lücke in den bereits etablierten Messtools und stehen Open-Source zur Verfügung (Deiner & Keilhofer, 2023).

jPowerMonitor ist exklusiv in Java implementiert, **das Konzept lässt sich auf weitere Sprachen übertragen**, die Introspektion bzw. Reflection unterstützen, wenngleich die Library in der jeweiligen Programmiersprache neu zu implementieren ist. Mit jPowerMonitor wird der Stromverbrauch einer Java Software transparent.

4. Optimierung von Testing und Betriebsplattform

4.1 Testing

Moderne CI/CD-Pipelines, realisiert durch Build-Server wie Jenkins oder Bamboo, sind einerseits ein Segen bezüglich Qualität. In vielen Konfigurationen laufen zum Beispiel automatisiert alle Tests, bevor ein neues Feature oder ein Bugfix in den Hauptentwicklungszweig des Quellcodeverwaltungssystems integriert werden darf.

Andererseits erhöht dieses Pattern unzweifelhaft den Energieverbrauch während der Entwicklung beträchtlich – man stelle sich vor, bei jeder noch so kleinen Änderung laufen 1000 oder mehr Tests. Ein Prozess, der mehrere Minuten bis Stunden dauern kann.

4.1.1 Wie oft sollen die Tests laufen?

Aus Green Coding-Perspektive sollte man sich daher durchaus die Frage stellen, **wie oft Unittests und Integrationstest wirklich laufen müssen?** Tatsächlich nach jedem einzelnen Check-in? Es gilt hier abzuwägen zwischen der Sicherheit, die Tests bieten und dem Ressourcenverbrauch.

Wie kann man nun mehrere Commits zu einem Testlauf zusammenfassen?

- Eine mögliche Lösung ist, dass Pipeline Job periodisch prüft, ob Änderungen im Repository vorliegen und nur dann losläuft. So kann die Pipeline zum Beispiel alle acht Stunden auf Änderungen prüfen.
- Alternativ kann der Pipeline Job beim Commit prüfen, ob seit dem letzten Lauf eine bestimmte Anzahl von Commits überschritten wurden. Dies hat Nachteile, wenn nur wenige Änderungen vorgenommen werden.
- Der flexibelste Weg ist wohl eine Verhaltensänderung der Entwickler. So kann man im Team eine Regel aufstellen, dass zuerst lokal entwickelt wird und beispielsweise immer am Morgen die Arbeit des Vortags zum Server gepushed wird.

4.1.2 Wie viele Tests sollen laufen?

Die Ermittlung der Teilmenge von Tests, die von Codeänderungen betroffen sind, wird **Test Impact Analysis (TIA)** genannt. Test Impact Analysis kann auf verschiedene Arten die Lösungsmenge bestimmen.

Im Falle von Unit Tests besteht meist eine Eins zu Eins Beziehung zwischen geändert Klasse und Testklasse.

Paul Hammant zeigt in seinem Artikel einen Weg über Shell Scripts und dem Einsatz des Code Coverage Tools JaCoCo (JaCoCo, 2023) auf. So kann man den Output von JaCoCo parsen und darüber die Beziehung der Testklassen zu den getesteten Klassen herstellen. Bei Änderungen werden nur die Tests aufgerufen, die von den Änderungen betroffen sind. Die Erkennung der geänderten Dateien kann über `git status` bzw. `git diff` zur Erkennung der geänderten Methoden erfolgen. (Hammant, 2017)

Alternativ kann man Build Tools einsetzen, die die Funktionalität bereits mitbringen: hier sind die Tools Bazel von Google, Buck von Facebook oder Pants von X zu nennen. Diese Tools erfordern jedoch alle eine beträchtliche Einarbeitungszeit, umfassende anfängliche Konfiguration und kontinuierliche intensive Wartung. Der Einsatz sollte daher nur in größeren Projekten erfolgen, wo dann aber auch der Nutzen entsprechend hoch sein wird.

4.1.3 Wie wird getestet?

Beim Test selbst kann man darauf achten, dass die Bibliotheken, die der Testlauf benötigt, möglichst **gecached** vorliegen. So sollte mindestens ein **Unternehmensrepository (Proxy)** eingerichtet werden, so dass nicht jede Bibliothek immer wieder über das Internet bezogen werden muss.

Werden Testcontainer benutzt, **kann man die Bibliotheken auch direkt in das Image des Containers packen**. So fällt zur Laufzeit kein Netzwerkverkehr an. Bietet die CI/CD Pipeline eigene Caching Mechanismen an, dann sollten diese genutzt werden.

4.1.4 Wann wird getestet?

Tests sollten bevorzugt zu einem **Zeitpunkt mit viel verfügbarer grüner Energie** laufen, d.h. zum Beispiel zur Mittagszeit und nicht nachts oder morgens.

Weitere Ideen zum energiebewussten Testen:

- **Redundanzfreiheit.** Tests sollten möglichst redundanzfrei erstellt werden, da das mehrfache Testen des Codes keinen Nutzen bringt. Obwohl das Kopieren von Tests schnelle Ergebnisse liefert, sollten Programmierer*innen darauf achten, den Code nur mit einem Test zu testen.
- **Energieverbrauchsdocumentation.** Um Energieeffizienz bewusst in den Entwicklungsprozess zu integrieren, sollten Protokolle über den Energieverbrauch während des Testprozesses zu erstellt werden. Warnungen sollten ausgelöst werden, wenn der Energieverbrauch akzeptable Grenzen überschreitet. Die Messung kann beispielsweise mit dem Tool `jPowerMonitor` erfolgen.

- **Vergleich mit Baseline:** Wenn ein Referenzwert mit einer spezifischen Hardwarekonfiguration und einem bestimmten Szenario festgelegt wird, dann können die nachfolgenden Tests mit dieser Baseline verglichen werden. Abweichungen zur Baseline sollten dann zu Warnungen führen, so dass ggf. darauf reagiert werden kann.
- **Stress- und Lasttests:** Mittels Simulationen sollte sichergestellt werden, dass die Software auch unter hoher Last effizient mit Energie umgeht.
- **Energieeffiziente Bibliotheken und Frameworks:** Es ist ratsam, Bibliotheken und Frameworks zu nutzen, die speziell auf Energieeffizienz optimiert sind. Für die Auswahl der Libraries können entweder eigene Messungen durchgeführt oder Erfahrungsberichte aus entsprechenden Communities genutzt werden. Dies gilt natürlich nicht nur für die Testframeworks, sondern noch viel mehr für die zur Produktionszeit eingesetzten Bibliotheken.

4.2 Betriebsplattform

4.2.1 Containerisierung

Als Mitte der 2000er Jahre virtuelle Maschinen durch Open-Source-Lösungen wie VirtualBox einen Aufschwung erlebten, wurden diese von der Entwicklergemeinde dankend adaptiert. Ein Nachteil von VMs ist die „Schwergewichtigkeit“ und der damit einhergehende hohe Ressourcen- und Energieverbrauch. Eine VM muss ein gesamtes Betriebssystem samt Kernel, Neben- und Hintergrundprozessen auf dem Host emulieren. Im Jahr 2013 kam mit Docker für viele Anwendungsfälle eine elegantere und leichtgewichtige Lösung. Container nutzen im Gegensatz zu VMs den Kernel und viele OS-Funktionen des Host-Systems mit. Dadurch **erzeugen Container-Instanzen viel weniger Overhead als VMs** und haben somit per se einen geringeren Energieverbrauch.

4.2.2 Containerorchestrierung

„Mit Hilfe der Container-Orchestrierung werden Deployment, Ressourcenzuweisung, Skalierung und Vernetzung von Containern automatisiert“ (Redhat, 2019). Container-Orchestrierungsplattformen wie das quelloffene Kubernetes oder das darauf basierende kommerzielle Openshift bieten diverse Mechanismen für optimierte Ressourcennutzung durch optimierte Konfiguration der Deployment-Rezepte.

Optimale Nutzung der zur Verfügung stehenden Hardware kann unter anderem dadurch erreicht werden, dass jedem einzelnen Pod fix Ressourcen zugewiesen werden (CPU / RAM / Disk space). Pod bezeichnet dabei eine Containerinstanz.

Werden die festgelegten Grenzen über- oder unterschritten, **skalieren** Orchestrierungsplattformen die Anzahl der Pods **automatisch** in beide Richtungen (horizontale Skalierung) (Kubernetes, 2020). Durch die Definition sowohl minimal zugesicherter als auch maximal erlaubter Ressourcen ist die automatische Verteilung und gegebenenfalls Verschiebung von Pods auf ausreichend leistungsfähige Cluster-Nodes umsetzbar (vertikale Skalierung) (Kubernetes, 2022). Für die Ersteller der Deployment-Rezepte gilt es dabei zu beachten, dass die Hardware-Limits nicht überschritten werden, um unerwünschten Nebeneffekten vorzubeugen (Credativ GmbH, 2020).

„Der Schlüssel zur effizienten Ressourcenauslastung ist die **Minimierung von Leerlauf**. Ein Server, der ungenutzte Kapazitäten hat, ist totes Kapital“ (Schneller & Pustina, 2015) beziehungsweise vermeidbarer Energieverbrauch.

4.3 Monitoring

Ein sehr wichtiger Faktor zum **effizienten Anwendungsbetrieb** ist die Implementierung einer **geeigneten Monitoring-Lösung**. Insbesondere im Container-Umfeld hat sich die Kombination aus **Prometheus** (Prometheus Authors, 2023) und **Grafana** (Grafana Labs, 2023) zunehmend als Industrie-Standard etabliert.

Vorbedingung ist, dass die zu überwachende Software einen HTTP-Endpoint mit Metriken im Prometheus-Format bereitstellt. Bekannte Frameworks wie Spring Boot für Java unterstützen in der Regel dabei bzw. lassen sich entsprechend konfigurieren (“Actuator”).

Durch **“Scraping”**, bei dem der Endpoint in zu konfigurierenden Intervallen abgerufen wird, bündelt Prometheus sämtliche verfügbaren **Anwendungsmetriken** inklusive Zeitstempel in seiner zentralen Datenbank.

Im Fall von Kubernetes / Openshift muss eine “ServiceMonitor” Custom Resource Definition (CRD, (The Kubernetes Authors, 2023)) spezifiziert werden, die Prometheus den aufzurufenden Endpoint sowie das Aufrufintervall vorgibt.

Die gesammelten Prometheus-Daten lassen sich als Datenquelle in Grafana (siehe Abbildung 11) einbinden. Mittels **“PromQL”** (Prometheus Authors, 2023) können die Metriken aufbereitet und durch Grafana grafisch ansprechend in verschiedensten Diagrammtypen auf einem Dashboard dargestellt werden.

In Bezug auf **geringen Stromverbrauch** ist es ratsam, insbesondere genau **die APIs** zu untersuchen und vorzugsweise **zu optimieren**, die entweder **am häufigsten aufgerufen** werden oder die **längste Ausführungszeit** beanspruchen.



Abbildung 11: Grafana Dashboard mit Spring Boot Actuator Prometheus Metriken

5. Digitalisierung und Nachhaltigkeit

5.1 Gedankenanstöße zu möglichen Maßnahmen für nachhaltige Digitalisierung

Der Wunsch nach mehr Nachhaltigkeit ist in allen Ecken und Enden auch in der IT-Branche angekommen- **Unklarheit** besteht jedoch oft **in der operativen Umsetzung** und der Entwicklung einer pragmatisch umsetzbaren Strategie. Wir skizzieren im Folgenden **mögliche Maßnahmen** für Unternehmen, die **Digitalisierung nachhaltiger gestalten** wollen.

5.1.1 Deployment von SW-Komponenten auf notwendiges Maß reduzieren

Jede Codeänderung führt zu einem geänderten Produktinkrement, Komponente oder Anwendungscontainer aufgrund automatisierter Generierungs- und Installationsprozessen - eine geschätzte technische Errungenschaft der CI/CD Pipelines und Verteilungssystemen. Man erhält diese Funktionalität quasi kostenlos und als Standard vorkonfektioniert. Dass diese Artefakte dann oft nicht relevant sind oder genutzt werden, wird dabei ignoriert. Lieber zehn Deployments umsonst erstellt, als auf ein erwartetes verzichtet. Diese Prozessdenke führt zu regelmäßiger Energie- und Ressourcenverschwendung. Deshalb sollen **Kriterien** festgelegt und eingeführt werden, um Continuous Deployment nicht als Standard, sondern **nur bei konkretem Bedarf** und nur für **produktiv**-einsetzbare Artefakte einzusetzen.

5.1.2 Aufbau und Etablieren selektiver, intelligenter CI/CD Pipeline

Moderne CI/CD Pipelines nehmen dem Entwickler viel Arbeit ab, indem mit jedem 'commit' des geänderten Source-Codes alle Produktinkremente automatisiert kompiliert, gebaut und (unit-) getestet werden. Die dabei erzeugte Menge an Artefakten und deren Übertragung in den Netzen ist enorm, und oftmals unnötig, da nicht jeder Zwischenstand an Artefakten auch benötigt wird. Deshalb soll eine CI/CD Pipeline-Strategie entwickelt, und Implementierung erfolgen, die **selektiv und intelligent** nur die tatsächlich von der Code-Änderung betroffenen Produktinkremente erstellt, und auch nur dann Artefakte generiert und testet, wenn **konkreter Bedarf** dafür existiert. Hierzu gehört u. a. auch der Aufbau von **Komponenten-Caches**.

5.1.3 Nachhaltigkeitsaspekte als Erfolgsindikatoren einführen

Projekterfolge werden meist anhand wirtschaftlicher Parameter gemessen und ausgewiesen - oft auch in Verbindung mit einer funktionalen und terminlichen Zielerreichung. Welche Qualitätsziele oder gar Nachhaltigkeitsziele damit erreicht (oder auch verfehlt) wurden, wird

nur bei Projektverfehlungen untersucht. Die Entwicklung einer Erfolgsmessung für **qualitative und nachhaltige Ziele** soll entwickelt und zukünftig angewendet werden.

5.1.4 CO₂-Fußabdruck als Metrik für Projekte einführen und kontrollen

Solange Nachhaltigkeit bzw. Klimaaspekte keiner messbaren Metrik unterliegen, ist deren Berücksichtigung von der individuellen Einschätzung der Beteiligten abhängig. Deshalb wird eine Bemessungsgrundlage für einen CO₂ Fußabdruck für ein Projekt benötigt (Project Carbon Intensity - PCI) vergleichbar zum SCI (Software Carbon Intensity) der Green Software Foundation (Dissanayake, 2022), und dessen Fortschritt im Projekt-Statusbericht rapportiert. Bereits zu Projektbeginn wird ein **CO₂ Budget** vereinbart, gegen das der aktuelle Wert geprüft und im Forecast berichtet wird. **Weitere Nachhaltigkeitsziele** werden als **Checkliste** hinterlegt.

5.1.5 Einsatzdauer und Beschaffung für IT-Infrastruktur

Technologische Innovationen im Notebookbereich oder bei Monitoren haben sich in den letzten Jahren verlangsamt, so dass ein Power-Notebook aus 2019 auch über 2022 hinaus seinen Dienst leisten könnte. Viele Beschaffungsprozesse lösen die Geräte, unabhängig vom konkreten Bedarf, jedoch z. B. nach 3 Jahren automatisch ab. Zudem ist der CO₂-Fußabdruck der Geräte bislang häufig kein Beschaffungskriterium.

Zur Minimierung von Ressourcen und Energieverschwendung soll deshalb eine Variante gefunden werden, um Geräte **bedarfsbezogen** nutzen zu können und deren **CO₂-Fußabdruck bekannt** zu machen, so dass er in die Entscheidungen für das Geräteportfolio einfließen kann.

5.1.6 Nachhaltige Beschaffung von Hardware honorieren

In der Beschaffungs-Datenbank den CO₂-Fußabdruck (oder eine andere passende Kennzahl) ausweisen (neben der Leasingrate) ggfs. visuelle Aufbereitung zum leichteren Verständnis; CO₂-Budget je MA, das bei der Beschaffung von Hardware herangezogen wird. **Sparsames Verhalten honorieren**, z. B. durch Auszahlung von Restbudget oder Spenden an ein Klimaschutzprojekt.

5.1.7 Aktive Reduzierung von Dokumentenablagen und Speicherbereichen

Zur Sicherheit nochmal eine Kopie speichern, E-Mail Accounts mit 2.000 ungelesenen Mails und Anhängen von 10 GB sind bequeme technische Errungenschaften, da die Infrastruktur dazu quasi umsonst erscheint – dennoch werden hierdurch Ressourcen verbraucht. Deshalb

sollten proaktiv und durch technische Möglichkeiten, **Redundanzen aufgespürt** und **Löschszenarien automatisiert** eine Bereinigung der Projektartefakte durchführen.

5.1.8 Definition und Aufbau einer Referenz-Energiemessumgebung

Nur die Kenntnis über Verbräuche und Metriken lassen Rückschlüsse auf potenzielle Defizite in Bezug auf Energieverbräuche von IT-Lösungen zu. Die eingesetzten Standard-Infrastrukturen sind jedoch sehr unterschiedlich und erlauben keine Messung von Energieverbräuchen auf Lösungs-, Komponenten- oder gar Funktionsebene. Deshalb soll eine **Referenzumgebung** definiert und ggfs. bereitgestellt werden, die eine entsprechende **Energiemessung** erlaubt. Damit sollen Verbrauchsänderungen in einer Anwendung qualifiziert, und qualitative Vergleiche mit anderen Lösungen und Architekturmustern unterstützen werden, um langfristig **ein Gespür und Kompetenz für Good-Climate-Practices** zu entwickeln.

5.1.9 Zentrale Kompetenzstelle für Nachhaltigkeit im SE-Lifecycle aufbauen

Die fachliche, technische und methodische Ausrichtung als IT-Unternehmen erhält mit der Nachhaltigkeit eine zusätzliche Querschnittsdimension, vergleichbar mit "IT-Security". Der Kompetenzaufbau hierzu ist zum Teil branchenspezifisch, wie beispielsweise im Falle von ESG Richtlinien im Banking. Der Großteil der **Nachhaltigkeitskompetenz** ist jedoch branchenneutral, so dass sich eine Zentralisierung von Beratungs-Know-how empfiehlt. Abgeleitet vom übergreifenden unternehmensweiten Nachhaltigkeitsziel sind Nachhaltigkeitsziele auch in den einzelnen Abteilungen zu berücksichtigen, welche durch **zentralen und dezentralen Kompetenzaufbau** unterstützt werden müssen.

5.1.10 Zentrale Expertise zu Nachhaltigkeit aufbauen und vermarkten

Nachhaltigkeit kennt keine Branchengrenzen, sondern hat stark übergreifende Handlungsfelder. Branchenspezifisch können hierbei ggfs. Regularien, Gesetze oder konkrete Lösungen für Anwendung der Nachhaltigkeitsaspekte sein. Deshalb kann der **Aufbau zentraler Kompetenz** für diesen Bereich hilfreich sein, um Grundlagen zu erarbeiten, **Standards** vorzubereiten und die einzelnen Sparten zu unterstützen. Zudem ist eine Verankerung in Communities oder Kompetenzzentren sinnvoll.

5.1.11 Aktives HW/SW-Sizing im Kontext Anwendungsbetrieb ermöglichen

Lösungsspezifische Hard- und Softwarebeschaffung und deren operative Betreuung sind übliche Vorgehen, um eigene oder Kundensysteme stabil und performant zur Verfügung zu stellen. Da bei der Bedarfsermittlung oft mit unbekanntem Größen spekuliert werden muss, führt dies manchmal zur Über- oder Unterdimensionierung der Infrastruktur.

Deshalb muss die zugrundeliegende Lösungsarchitektur eine **aktive Anpassung an den aktuellen Leistungsbedarf** ermöglichen, Anwendungs-Container dynamisch erzeugt oder verworfen werden, **Deployments nur bei konkretem Bedarf** durchgeführt und die dabei notwendigen Testkonstellationen darauf angepasst und auswählbar sein. Eine **Bestandsaufnahme aller laufenden Lösungen** soll Auskunft über den aktuellen Zustand in Bezug auf Nachhaltigkeitskriterien geben.

5.1.12 Aktives Reporting/Monitoring der Auslastung von IT-Lösungen

Überlastung von Systemen wirkt sich negativ für den Anwender aus, **Unterlast bleibt hingegen unerkannt** und verschwendet auch im Leerlauf im ungünstigsten Fall Energie und erzeugt Emissionen.

Deshalb müssen alle Systeme eine **transparente Indikation** (z.B. Dashboard, Report) bzgl. der aktuellen bzw. durchschnittlichen Auslastung der Rechenkapazitäten liefern, um ein aktives oder gegebenenfalls **automatisiertes Sizing** zu vereinfachen. Regelmäßiges Reporting von Lastverteilung, Monitoring und Alarmierung bei dauerhaft geringer Auslastung, sowie Handlungsempfehlungen zur **nachhaltigkeitsbezogenen richtigen Dimensionierung** sind hier zu erarbeiten und anzuwenden.

5.1.13 Aufbau einer unternehmensweit nutzbaren Werkzeugkette

Ein hoher Aufwand bei der Einarbeitung verursacht nicht nur wirtschaftliche Kosten, sondern führt auch zu einem Verbrauch von Ressourcen.

Die Einstiegshürde, zum Einsatz konzeptioneller Werkzeuge für Kollaboration, Dokumentation, Tickets und Modellierung, ist gerade für kleinere Projekte oder unerfahrene Teams hoch und kostspielig. Ein **de facto-Standard einer Werkzeugkette** mit zentralem Betrieb und Beratung sowie attraktivem Lizenz Modell schafft zukünftig **Synergien** und reduziert ineffiziente Insellösungen in den Unternehmens-Einheiten.

5.1.14 Aufbau einer Softwarekomponenten-Empfehlung bzw. Grey-List

Die Verfügbarkeit und der Einsatz von Eigen-, Kauf- und Open-Source Komponenten ist unüberschaubar. Einen Überblick und zugleich einen Vergleich bzgl. ihrer Funktionalität zu erhalten ist für den Einzelnen kaum möglich. Fraglich ist allerdings, ob jeder aufkommende Komponentenstern den Weg in die Projekte finden muss, oder ob Synergien und Austausch zu Erfahrungen mit Komponenten zwischen den Einheiten für viele Standardfälle nicht zielführender wäre.

Der Aufbau eines **Komponenten-Standardverzeichnisses**, mit funktionaler Empfehlung und/oder Abgrenzung und ergänzender Aussage über eine **qualitative Energiebewertung**,

kann Effizienz in der Umsetzung fördern und zugleich **energiebewusste Komponentenlösungen** transparent machen.

5.1.15 Werkzeugbasierte Projektmanagement-Dokumentationsplattform

Viele Projektdokumente basieren auf einzelnen Office-Vorlagen und werden als Varianten, Kopien und Ableitungen in den Projektverzeichnissen gespeichert und zudem per E-Mail verteilt. Dies fördert unnötige Redundanzen, hemmt die Auffindbarkeit aktueller Informationen und schränkt die Möglichkeiten zur automatisierten Archivierung stark ein. Die Auswahl und Einführung einer **einheitlichen Dokumentationsplattform** für Projektartefakte mit vorgefertigten Vorlagen, automatischen Archivierungsprozessen und eindeutiger Informationsidentifikation, soll **nicht-nachhaltigen Umgang** mit technischen Ressourcen zukünftig **minimieren**.

5.1.16 Klimafreundliche Architekturmuster und Empfehlungen etablieren

Softwarearchitekturen haben vornehmlich den Zweck, das angestrebte Lösungsszenario bzgl. fachlicher und technischer Anforderungen optimal zu unterstützen und dabei den Anspruch an die Qualitätsanforderungen sicherzustellen.

Nachhaltigkeit gehört bislang nicht zu den ISO 25010 Qualitätskriterien, dennoch müssen die Weichen dafür in der Architektur gestellt werden.

Architekturbezogene Empfehlungen sollen deshalb gesammelt, entwickelt und bekannt gemacht werden, damit IT-Lösungen den zukünftigen Anforderungen an nachhaltige Entwicklung, Test, Distribution und vor allem Betrieb gerecht werden können.

5.1.17 Bewusstsein zur Nachhaltigkeit im Projekt schärfen

Dass Nachhaltigkeit in der IT an Bedeutung zunimmt, ist durchaus spürbar. Allerdings ist diesbezüglich bei vielen Unternehmen keine klare Strategie in Bezug auf deren Wertschöpfung erkennbar. Zudem ist nicht bei allen Mitarbeitenden ein **einheitliches Verständnis von Nachhaltigkeit** vorhanden oder ein Mindestkenntnisstand bzgl. Green IT erreicht.

Aus diesem Grund sind alle verfügbaren Formate wie Intranet, Communities, Poster, Kongresse etc. mit entsprechenden Inhalten zu nutzen und Verständnis und Unterstützung für weitere Nachhaltigkeitsmaßnahmen aufzubauen.

5.1.18 Schulung und Training zu Nachhaltigkeitsaspekten in der IT

Das **Nachhaltigkeitsuniversum im Zusammenhang mit IT** ist ungewohnt und Neuland für die meisten Betroffenen. Eine praxisnahe Sensibilisierung durch Schulungen,

Workshopreihen oder Kurzvideos soll Vorbehalte abbauen und eine bedarfsgerechte und alltägliche Umsetzung unterstützen.

5.1.19 Entwicklung eines Use-Case Effizienz-Labels

Ein Use-Case oder fachliches Szenario definiert die Rahmenbedingungen für die Umsetzung und ist somit mitverantwortlich für die daraus resultierende **Effizienz und Energiebedarf der Lösung**. Es soll eine Metrik entwickelt werden, die anhand eines fachlichen Szenarios, eine **Indikation für Energieprognose** ermittelt. Die Indikation soll mindestens einen Vergleich innerhalb der Lösung erlauben und perspektivisch eine grundsätzliche Indikation ermöglichen.

6. Deutsches Umweltzeichen Blauer Engel

Trotz gesteigerter Energieeffizienz von Geräten wie Laptops zeigt sich ein **paradoxe Anstieg im IT-Stromverbrauch**, da neuere Technologien **oft durch Softwareanwendungen mit höherem Energiebedarf** genutzt werden. Des Weiteren verbrauchen Rechenzentren oft Energie ineffizient, mit CPU-Auslastungen häufig unter 20%, was die Notwendigkeit einer Verbesserung der Effizienz unterstreicht (Umweltbundesamt, 2023).



Abbildung 12: Logo "Blauer Engel"

Diese Problematik verdeutlicht, dass neben hardwareseitigen Optimierungen **auch die Software** und Arbeitsprozesse **zur Reduzierung des Stromverbrauchs** in der IT **beitragen** müssen.

6.1 Blauer Engel für Rechenzentren

Durch die Digitalisierung hat der Bedarf an zentraler Rechen- und Speicherleistung deutlich zugenommen und somit auch der Bedarf an wertvollen Energie- und Rohstoffressourcen. Das Umweltzeichen Blauer Engel zeichnet Rechenzentren aus, die **besonders energieeffizient und ressourcenschonend** betrieben werden.

Für die Vergabe des Blauen Engels an Betreiber von Rechenzentren und Informationstechnik werden Mindestanforderungen vorgegeben, deren Einhaltung von einem unabhängigen Auditor überprüft werden.

Es können solche Rechenzentren ausgezeichnet werden (ibid.),

- deren technische Gebäudeausrüstung (TGA) besonders energieeffizient, klima- und ressourcenschonend betrieben wird,
- deren Betreiber eine langfristige Strategie zur Erhöhung der Energie- und Ressourceneffizienz für das Rechenzentrum erarbeiten und erfolgreich umsetzen,
- deren Informationstechnik effizient betreiben wird,
- die ihre Kunden in die Lage versetzen, Maßnahmen zur Steigerung der Energieeffizienz umzusetzen,
- und die durch garantierte Mindeststandards und transparente Berichterstattung die Voraussetzung für IT-Betreiber schaffen, Informationstechnik energieeffizient zu betreiben.

6.2 Blauer Engel für Softwareprodukte

Das Umweltzeichen für energieeffiziente Softwareprodukte strebt an, den **Gesamtenergieverbrauch von Informations- und Kommunikationstechnik zu senken** und die Ressourceneffizienz zu erhöhen. Durch den Einsatz einer mit dem Blauen Engel ausgezeichneten Software können Verbraucher und staatliche Beschaffungsstellen sicher sein, dass diese sparsam mit Hardwareressourcen umgeht und energiesparend ist.

Für Entwickler bietet das Umweltzeichen eine Leitlinie, um Umweltbelastungen durch Software zu identifizieren und geeignete Maßnahmen zu ergreifen, um diese zu reduzieren (Gröger, et al., 2018).

Drei Kriterien des Blauen Engel für ressourcen- und energieeffiziente Softwareprodukte beziehen im Wesentlichen folgende Punkte mit ein:

- **Ressourceneffizienz:** Ziel des Umweltzeichens ist es, den Energieverbrauch der Informations- und Kommunikationstechnik insgesamt zu reduzieren und die Ressourceneffizienz zu steigern.
- **Potenzielle Hardware-Nutzungsdauer:** Wachsende Software-Anforderungen verkürzen die Nutzungsdauer der Hardware. Die Herausforderung liegt darin, Hardware- und Software-Erneuerungszyklen zu entkoppeln, um Ressourcen zu sparen. Abwärtskompatibilität sichert, dass Software auf älteren Systemen läuft. Das Ziel ist eine längere Hardware-Nutzungsdauer und weniger Ressourcenverbrauch für neue Geräte.
- **Nutzungsautonomie:** Um einen ressourcenschonenden Softwareeinsatz zu ermöglichen, sollten Nutzende Einblick ins Produkt und Optionen zur Reduzierung von Kapazitäten haben. Dies beinhaltet unter anderem Aspekte wie Datenformate, Transparenz des Produkts, Deinstallierbarkeit und Offlinefähigkeit.

Weitere detaillierte Informationen zu den Kriterien und ihrer Operationalisierung finden sich im Abschlussbericht "Entwicklung und Anwendung von Bewertungsgrundlagen für ressourceneffiziente Software unter Berücksichtigung bestehender Methodik".

Der blaue Engel kann Stand Ende 2023 nur für Desktop-Anwendungssoftware, die über eine Benutzerschnittstelle verfügt, vergeben werden (Naumann, et al., 2021). Momentan wird aber der Geltungsbereich um Client-Server-Anwendungen sowie mobile Apps erweitert.

7. Checkliste

- ✓ **Nicht funktionale Anforderungen zu grüner IT und Green Coding sind definiert**
- ✓ **IT-Architekten und Entwickler sind geschult**
- ✓ **System skaliert dynamisch und fährt sich bei Leerlauf und ggf. zu Randzeiten herunter**
- ✓ **Lastspitzen werden vermieden**
 - Echtzeitverarbeitung wird vermieden, wo möglich
 - Dynamische Inhalte halten sich in vertretbarem Rahmen
- ✓ **Komponenten mit dem höchsten Energieverbrauch sind identifiziert**
 - Schätzung des Stromverbrauchs ist erfolgt
 - Messungen wurden durchgeführt
- ✓ **Green Coding Prinzipien und Entwicklungsmuster in folgenden Punkten berücksichtigt:**
 - Lose Kopplung der Komponenten
 - Reaktives Programmiermodell
 - Effizienter Code mit optimalen Algorithmen
 - Vermeidung unnötiger Round-Trips, Caching
 - Optimiertes Datenvolumen
 - Komprimierte Netzwerkkommunikation
 - Datenbankindizes und Explain Plans nutzen
 - Optimierte Ressourcennutzung durch optimierte Konfiguration
 - Zero-Waste-Code: Treeshaking und Bytecodeoptimierung z.B. mit GraalVM
 - Richtlinien zum green Testing werden beachtet
 - Prinzip: „Jede Codezeile hat das Potential den Energieverbrauch zu senken“
- ✓ **Performance Engineering**
 - Es ist klar, was, wann und wie gemessen wird
 - Was: Gesamtsystem/Prozess/Komponente/Methode
 - Wann: Unittest, Integrationstest, Betrieb (ideal: in jeder Phase)
 - Wie: Task-Manager, Libre Hardware Monitor oder Smart Plugs, Frameworks, z. B. jPowerMonitor und weitere
- ✓ **Ressourcenverbrauch**
 - CPU/RAM/Speicher/Netzwerk: Optimierung durch Profiler und SQL Explain Plans
 - Monitoring ist implementiert
- ✓ **DevOps und Betrieb**
 - DevOps Pipeline ist hinsichtlich Energieverbrauch optimiert
 - Containerbasierte Plattform mit intelligenter, dyn. Skalierung wird genutzt
 - Ressourcen werden optimal genutzt (CPU, RAM, Speicher, Netzwerk, GPU, ...)
- ✓ **Konkrete Green Coding NFRs sind berücksichtigt**

8. Fazit

Alle, die am Entwicklungsprozess und am Betrieb einer Software beteiligt sind, haben die Chance dazu beizutragen, **Strom und damit Energie zu sparen**. In diesem Beitrag haben wir gezeigt, wie die **nicht-funktionale Anforderung** „niedriger Energieverbrauch“ in Fachbereichen, IT-Architektur, Programmierung, Test und Betrieb berücksichtigt werden kann.

Die vorgestellten Aspekte können anhand der dargestellten **Checkliste** wieder ins Gedächtnis gerufen und in **jeder Phase eines Softwareprojekts** angewendet werden. In der Regel ergeben sich **positive Nebeneffekte** auf andere NFRs wie **Performance oder Benutzbarkeit** der Software.

Schon allein durch **Messungen**, die zum Beispiel mit unserer JUnit-Extension durchgeführt werden können, entsteht ein **Gefühl für die Größe des Energieverbrauchs** in Softwarekomponenten. So können sinnvolle Anforderungen an neue oder zu ändernde Komponenten gestellt werden.

9. Abbildungsverzeichnis

Abbildung 1: Postgres Ausführungsplan.....	11
Abbildung 2: Windows Task-Manager	12
Abbildung 3: Beispielansicht Libre Hardware Monitor	16
Abbildung 4: PowerTOP Tool.....	17
Abbildung 5: Energieverbrauch MS Teams in ca. einer Arbeitswoche	18
Abbildung 6: Messwert und Konfiguration HWiNFO und Libre Hardware Monitor.....	20
Abbildung 7: Verwendung der JUnit Extension und @RepeatedTests Annotation.....	21
Abbildung 8: Beispiel Ergebnis CSV Extension	21
Abbildung 9: Aufrufbeispiel Java Agent mit Spring Anwendung	23
Abbildung 10: Ergebnis der Messung Java Agent	23
Abbildung 11: Grafana Dashboard mit Spring Boot Actuator Prometheus Metriken	29
Abbildung 12: Logo "Blauer Engel"	36

10. Literaturverzeichnis

Baeldung, 2021. <https://www.baeldung.com/junit-5-extensions>. [Online]

Available at: <https://www.baeldung.com/junit-5-extensions>

[Zugriff am 8 12 2023].

Becker, M., Thorenz, L.-K. & Zacher, M., 2021. <https://info.microsoft.com/>. [Online]

Available at: https://info.microsoft.com/rs/157-GQE-382/images/de-ebook-SRGCM5002.pdf?mkt_tok=MTU3LUdRRS0zODIAAAGAo6eTYVcKOI5sIrMo75SWo-l1xcOVA-UPecECIzi_d5Ar2CTcgVUgCn47gkaWI75IYu7GCAw9-XdywVMDcT9alUMQBrSRheL_G1xkUpogaleSVqRDmmbGzIQI

[SRGCM5002.pdf?mkt_tok=MTU3LUdRRS0zODIAAAGAo6eTYVcKOI5sIrMo75SWo-l1xcOVA-UPecECIzi_d5Ar2CTcgVUgCn47gkaWI75IYu7GCAw9-](https://info.microsoft.com/rs/157-GQE-382/images/de-ebook-SRGCM5002.pdf?mkt_tok=MTU3LUdRRS0zODIAAAGAo6eTYVcKOI5sIrMo75SWo-l1xcOVA-UPecECIzi_d5Ar2CTcgVUgCn47gkaWI75IYu7GCAw9-XdywVMDcT9alUMQBrSRheL_G1xkUpogaleSVqRDmmbGzIQI)

[XdywVMDcT9alUMQBrSRheL_G1xkUpogaleSVqRDmmbGzIQI](https://info.microsoft.com/rs/157-GQE-382/images/de-ebook-SRGCM5002.pdf?mkt_tok=MTU3LUdRRS0zODIAAAGAo6eTYVcKOI5sIrMo75SWo-l1xcOVA-UPecECIzi_d5Ar2CTcgVUgCn47gkaWI75IYu7GCAw9-XdywVMDcT9alUMQBrSRheL_G1xkUpogaleSVqRDmmbGzIQI)

[Zugriff am 24 11 2023].

Chamberlin, S., 2020. *Measuring Your Application Power and Carbon Impact (Part 1)*. [Online]

Available at: <https://devblogs.microsoft.com/sustainable-software/measuring-your-application-power-and-carbon-impact-part-1/>

[application-power-and-carbon-impact-part-1/](https://devblogs.microsoft.com/sustainable-software/measuring-your-application-power-and-carbon-impact-part-1/)

[Zugriff am 1 12 2023].

Cornell University und Microsoft, 2023. *People who work from home all the time 'cut emissions by 54%' against those in office*. [Online]

Available at: <https://www.theguardian.com/environment/2023/sep/18/people-who-work-from-home-all-the-time-cut-emissions-by-54-against-those-in-office>

[from-home-all-the-time-cut-emissions-by-54-against-those-in-office](https://www.theguardian.com/environment/2023/sep/18/people-who-work-from-home-all-the-time-cut-emissions-by-54-against-those-in-office)

[Zugriff am 8 12 2023].

Credativ GmbH, 2020. *Verwaltung von Hardware-Ressourcen in Kubernetes*. [Online]

Available at: <https://www.credativ.de/blog/howtos/verwaltung-von-hardware-ressourcen-in-kubernetes>

[kubernetes](https://www.credativ.de/blog/howtos/verwaltung-von-hardware-ressourcen-in-kubernetes)

[Zugriff am 8 12 2023].

Deiner, J. & Keilhofer, H.-P., 2023. <https://github.com/msg-systems/jpowermonitor>. [Online]

Available at: <https://github.com/msg-systems/jpowermonitor>

[Zugriff am 8 12 2023].

Deiner, J. & Keilhofer, H.-P., 2023. <https://mvnrepository.com>. [Online]

Available at: <https://mvnrepository.com/artifact/io.github.msg-systems/jpowermonitor>

[Zugriff am 8 12 2023].

Dissanayake, N., 2022. <https://greensoftware.foundation/articles/software-carbon-intensity-sci-specification-project>. [Online]

Available at: <https://greensoftware.foundation/articles/software-carbon-intensity-sci-specification-project>

[specification-project](https://greensoftware.foundation/articles/software-carbon-intensity-sci-specification-project)

[Zugriff am 8 12 2023].

Dr. Geiger, L. et al., 2021. *Ressourceneffiziente Programmierung*. [Online]
Available at: https://www.bitkom.org/sites/main/files/2021-03/210329_If_ressourceneffiziente-programmierung.pdf
[Zugriff am 6 12 2024].

GraalVM, 2023. <https://www.graalvm.org/>. [Online]
Available at: <https://www.graalvm.org/>
[Zugriff am 1 12 2023].

Grafana Labs, 2023. *Grafana*. [Online]
Available at: <https://grafana.com/>
[Zugriff am 22 12 2023].

Green Software Foundation, 2023. *Green Software Foundation*. [Online]
Available at: <https://greensoftware.foundation/>
[Zugriff am 15 12 2023].

Gröger, J. et al., 2018. *Entwicklung und Anwendung von Bewertungsgrundlagen für ressourceneffiziente Software unter Berücksichtigung bestehender Methodik*. [Online]
Available at:
https://www.umweltbundesamt.de/sites/default/files/medien/1410/publikationen/2018-12-12_texte_105-2018_ressourceneffiziente-software_0.pdf
[Zugriff am 14 12 2023].

Hager, G. & Wellein, G., 2018. *Gesellschaft für Informatik*. [Online]
Available at: <https://gi.de/informatiklexikon/performance-engineering>
[Zugriff am 03 04 2024].

Hammant, P., 2017. *The Rise of Test Impact Analysis*. [Online]
Available at: <https://martinfowler.com/articles/rise-test-impact-analysis.html>
[Zugriff am 19 12 2023].

JaCoCo, 2023. *JaCoCo Java Code Coverage Library*. [Online]
Available at: <https://github.com/jacoco/jacoco>
[Zugriff am 19 12 2023].

JUnit, 2023. <https://junit.org/>. [Online]
Available at:
<https://junit.org/junit5/docs/current/api/org.junit.jupiter.api/org.junit.jupiter.api/RepeatedTest.html>
[Zugriff am 8 12 2023].

Kubernetes, 2020. *Horizontal Pod Autoscaler*. [Online]
Available at: <https://kubernetes.io/de/docs/tasks/run-application/horizontal-pod-autoscale>
[Zugriff am 8 12 2023].

Kubernetes, 2022. *Resource Management for Pods and Containers*. [Online]
Available at: <https://kubernetes.io/docs/concepts/configuration/manage-resources-containers>
[Zugriff am 8 12 2023].

Lehmkuhl, M., 2023. <https://www.climatepartner.com>. [Online]
Available at: <https://www.climatepartner.com/de/scope-1-2-3-complete-guide#was-sind-scope-1-scope-2-und-scope-3-emissionen>
[Zugriff am 8 12 2023].

Malík, M., 2023. <https://www.hwinfo.com/>. [Online]
Available at: <https://www.hwinfo.com/>
[Zugriff am 8 12 2023].

Möller, M., 2023. <https://github.com/>. [Online]
Available at: <https://github.com/LibreHardwareMonitor/LibreHardwareMonitor>
[Zugriff am 1 12 2023].

msg systems ag, 2022. *Workshop Digitalisierung und Nachhaltigkeit*. Online: s.n.

Naumann, S., Kern, E., Guldner, A. & Gröger, J., 2020. www.umweltbundesamt.de. [Online]
Available at:
https://www.umweltbundesamt.de/sites/default/files/medien/479/publikationen/texte_119-2021_umweltzeichen_blauer_engel_fuer_ressourcenund_energieeffiziente_softwareprodukte.pdf
[Zugriff am 17 11 2023].

Naumann, S., Kern, E., Guldner, A. & Gröger, J., 2021. *Umweltzeichen Blauer Engel für ressourcenund energieeffiziente Softwareprodukte*. [Online]
Available at:
https://www.umweltbundesamt.de/sites/default/files/medien/479/publikationen/texte_119-2021_umweltzeichen_blauer_engel_fuer_ressourcenund_energieeffiziente_softwareprodukte.pdf
[Zugriff am 14 12 2023].

Oracle, 2023. <https://docs.oracle.com/>. [Online]
Available at:
<https://docs.oracle.com/en/java/javase/21/docs/api/java.instrument/java/lang/instrument/package-summary.html>
[Zugriff am 8 12 2023].

Plummer, D., 2021. *youtube*. [Online]

Available at: <https://www.youtube.com/watch?v=uCv5VRf8op0>

[Zugriff am 22 12 2023].

Producto GmbH, 2023. *https://www.testberichte.de*. [Online]

Available at: <https://www.testberichte.de/f/1/2972/470861.470864/1.html>

[Zugriff am 1 12 2023].

Prometheus Authors, 2023. *Prometheus*. [Online]

Available at: <https://prometheus.io/docs/introduction/overview/>

[Zugriff am 22 12 2023].

Prometheus Authors, 2023. *Querying Prometheus*. [Online]

Available at: <https://prometheus.io/docs/prometheus/latest/querying/basics/>

[Zugriff am 22 12 2023].

Pustina, L., 2019. *Universität Bonn*. [Online]

Available at: <https://net.cs.uni-bonn.de/de/wg/cs/projekte/performance-engineering/>

[Zugriff am 24 11 2023].

Redhat, 2019. *https://www.redhat.com*. [Online]

Available at: <https://www.redhat.com/de/topics/containers/what-is-container-orchestration>

[Zugriff am 7 12 2023].

Roden, G., 2021. *https://www.heise.de*. [Online]

Available at: <https://www.heise.de/blog/Tests-schreiben-die-Grundlagen-5077457.html>

[Zugriff am 8 12 2023].

Schneller, D. & Pustina, L., 2015. *heise online: Operations heute und morgen, Teil 3: Virtualisierung und Containerisierung*. [Online]

Available at: <https://www.heise.de/ratgeber/Operations-heute-und-morgen-Teil-3-Virtualisierung-und-Containerisierung-2762412.html?seite=all>

[Zugriff am 8 12 2023].

Syngenio AG, 2023. *Green Software Design Label*. [Online]

Available at: <https://www.greensoftwaredesign.com/green-software-design-label>

[Zugriff am 15 12 2023].

The Kubernetes Authors, 2023. *Kubernetes*. [Online]

Available at: <https://kubernetes.io/docs/concepts/extend-kubernetes/api-extension/custom-resources/>

[Zugriff am 22 12 2023].

Umweltbundesamt, 2023. *Rechenzentren (DE-UZ 228)*. [Online]
Available at: <https://www.blauer-engel.de/de/produktwelt/rechenzentren>
[Zugriff am 15 12 2023].

Umweltbundesamt, 2023. *Strom- und Wärmeversorgung in Zahlen*. [Online]
Available at: <https://www.umweltbundesamt.de/themen/klima-energie/energieversorgung/strom-waermeversorgung-in-zahlen#Kraftwerke>
[Zugriff am 1 12 2023].

Umweltgutachterausschuss (UGA), 2023. *Emas*. [Online]
Available at: <https://www.emas.de/was-ist-emas>
[Zugriff am 22 12 2023].

Wikipedia, 2022. *Performance Engineering*. [Online]
Available at: https://en.wikipedia.org/wiki/Performance_engineering
[Zugriff am 24 11 2023].

Wikipedia, 2023. *Thermal Design Power*. [Online]
Available at: https://de.wikipedia.org/wiki/Thermal_Design_Power
[Zugriff am 1 12 2023].

Bundesministerium für Umwelt, Naturschutz, nukleare Sicherheit und Verbraucherschutz (BMUV)
Referat T I 1

Nachhaltige Digitalpolitik

Stresemannstraße 128 - 130
10117 Berlin

Telefon: 030 18 305-0

Mail: nachhaltige-digitalisierung@bmu.bund.de

Stand: Juni 2024



Community
**Nachhaltige
Digitalisierung**