Ressourceneffizienz im Software Lifecycle

Wie Ressourcenschonung, Langlebigkeit und Nachhaltigkeit in der Softwareentwicklung berücksichtigt werden können

Band 3 – Leitfaden für Entwicklung & Betrieb

bitkom

Herausgeber
Bitkom e. V.
Albrechtstraße 10
10117 Berlin
T 030 27576-0
bitkom@bitkom.org
www.bitkom.org

Ansprechpartner
Dr. Frank Termer
Bereichsleiter Software
Bitkom e.V.
T 030 27576-232 | f.termer@bitkom.org

Verantwortliches Bitkom-Gremium
AK Software Engineering & Software Architektur

Autorinnen und Autoren
Stan Bühne (IREB GmbH) | Sabine Büsing (Diamant Software GmbH) | Dr. Leif Geiger (Yatta Solutions GmbH) | Christoph Hein (DB Systel GmbH) | Patricia Kelbert (Fraunhofer IESE) | Fabrizio Kuruc (Algonaut) | Dr. Kim Lauenroth (FH Dortmund) | Yelle Lieder (adesso SE) | Hendrik Lösch (ZEISS Digital Innovation) | Kay Makowsky (hitabis) | Franziska Petrovsky (ZEISS Digital Innovation) | Hagen Rahn (Stackmeister GmbH) | Dr. Andreas Scharf (OctaVIA AG) | Jan Tschada (Esri Deutschland GmbH) | Dr. Joachim Weber (Fraunhofer IESE) | Dr. Nicolas Wellmann (Deutsche Telekom IT GmbH)

Layout
Lea Joisten | Bitkom

Titelbild
© Jeremy Bishop – unsplash.com

Dieser Leitfaden ist Teil der Publikationsreihe **Ressourceneffizienz im Software Lifecycle – Wie Ressourcenschonung, Langlebigkeit und Nachhaltigkeit in der Softwareentwicklung berücksichtigt werden können.**

In dieser Reihe sind folgende Leitfäden erschienen:

- ↗ Band 1 – Eine Landkarte für einen ressourceneffizienten und nachhaltigen digitalen Wandel
- ↗ Band 2 – Auftragsklärung & Konzeptarbeit
- Band 3 – Entwicklung & Betrieb

Inhalt

Geleitwort

Die Entwicklung und der Betrieb digitaler Lösungen sind weit mehr als technologische Prozesse. Sie formen die Grundlage einer modernen, vernetzten Gesellschaft und haben eine immense Auswirkung auf den Energieverbrauch und die Ressourcennutzung. Der vorliegende Leitfaden »Ressourceneffizienz im Software Lifecycle – Band 3 – Entwicklung & Betrieb« widmet sich der entscheidenden Phase, in der nachhaltige digitale Lösungen nicht nur konzipiert, sondern praktisch umgesetzt werden.

Die nachhaltige Softwareentwicklung verlangt nach einer neuen Perspektive: Sie geht über die klassische Effizienzsteigerung hinaus und zielt darauf ab, sowohl ökonomische als auch ökologische Ziele in Einklang zu bringen. Dies erfordert ein tiefes Verständnis der Prozesse, Technologien und Architekturen, die die Grundlage digitaler Lösungen bilden.

Dieser Leitfaden bietet wertvolle Einblicke und praxisnahe Ansätze, wie Teams entlang des gesamten Entwicklungsprozesses nachhaltiger arbeiten können. Er soll als Inspiration und Werkzeug dienen, um den ökologischen Fußabdruck digitaler Technologien zu minimieren und langfristig zukunftsfähige Lösungen zu schaffen.

Dr. Kim Lauenroth, Dr. Leif Geiger, Holger Koch für den Vorstand des Lenkungsausschuss Software.

Danksagung

Ein besonderer Dank gilt allen Expertinnen und Experten, die an diesem Leitfaden mitgewirkt haben. Ihre fundierten Kenntnisse und innovativen Ideen sind der Kern dieses wertvollen Werkes. Sie haben nicht nur theoretische Grundlagen vermittelt, sondern auch praxisorientierte Ansätze entwickelt, die den Leserinnen und Lesern als Richtschnur dienen können.

Ein ebenso herzlicher Dank gebührt den vielen Unterstützerinnen und Unterstützern, die durch ihre Rückmeldungen und ihre kritische Begleitung entscheidend zur Qualität dieses Leitfadens beigetragen haben. Gemeinsam ist es gelungen, ein umfassendes Werk zu schaffen, das allen, die an der Entwicklung und dem Betrieb nachhaltiger Softwarelösungen beteiligt sind, ein verlässlicher Begleiter sein wird.

Zum Team der Autorinnen und Autoren gehören:

- Stan Bühne, IREB GmbH
- Sabine Büsing, Diamant Software GmbH
- Dr. Leif Geiger, Yatta Solutions GmbH
- Christoph Hein, DB Systel GmbH
- Patricia Kelbert, Fraunhofer IESE
- Fabrizio Kuruc, Algonaut GmbH
- Dr. Kim Lauenroth, FH Dortmund
- Yelle Lieder, adesso SE
- Hendrik Lösch, ZEISS Digital Innovation
- Kay Makowsky, hitabis
- Franziska Petrovsky, ZEISS Digital Innovation
- Hagen Rahn, Stackmeister GmbH
- Dr. Andreas Scharf, OctaVIA AG
- Jan Tschada, Esri Deutschland GmbH
- Dr. Joachim Weber, Fraunhofer IESE
- Dr. Nicolas Wellmann, Deutsche Telekom IT GmbH

Berlin, im Januar 2025

1 Einleitung

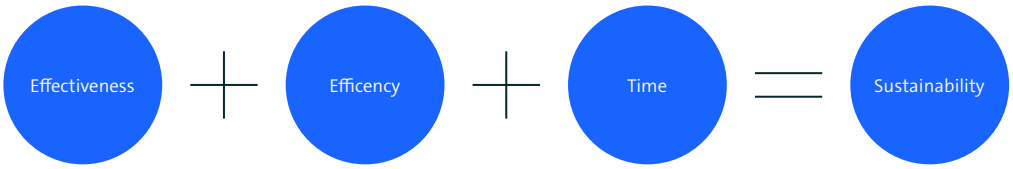
Software wird oft als Werkzeug betrachtet, das anderen Bereichen hilft, nachhaltiger zu arbeiten. Dieser Umstand wird auch als »Sustainability By Software«[1] bezeichnet und durch diverse Studien untermauert. Im Folgenden soll der Blick aber nicht nach außen, sondern nach innen, also in den Entwicklungsprozess und die Software-Systeme selbst gerichtet werden, um zu betrachten, wie Ressourcen effizienter und damit nachhaltiger entwickelt und betrieben werden können.

Sustainability By Software

Sustainability In Software

Dabei sei schon zu Beginn gesagt, dass Nachhaltigkeit kein neues Konzept in der IT ist. Um im Wettbewerb bestehen zu können, waren auch IT-Unternehmen schon immer gezwungen, effektiv zu arbeiten und unnötige Kosten zu vermeiden. Im Softwareentwicklungsprozess hat sich das in der Vergangenheit aber meist auf Themen wie Wartbarkeit oder Leistungseffizienz konzentriert und somit eher in finanzieller Nachhaltigkeit manifestiert. Darüber hinaus wurde Software eher als ein Werkzeug gesehen, um nachhaltigere Produkte zu schaffen oder Unternehmen zu helfen, nachhaltiger zu arbeiten.

Mit der Klimakrise und einem stetig steigenden Ressourcenverbrauch in der Digitalisierung hat sich der Begriff der Nachhaltigkeit gewandelt und an Bedeutung gewonnen. Einerseits hinterfragen Kunden zunehmend kritisch, wie Produkte hergestellt werden, und berücksichtigen das bei ihren Kauf- oder Investitionsentscheidungen. Dieses Bewusstsein hat beispielsweise zur Schaffung von Gütesiegeln wie dem Blauen Engel für ressourcen- und energieeffiziente Softwareprodukte[2] geführt. Andererseits wird das Thema häufig durch die Politik adressiert, um den Auswirkungen der Klimakrise entgegenzuwirken. Das führt dazu, dass auch IT-Unternehmen ihre CO2-Emissionen offenlegen und langfristig reduzieren müssen[3].



1 ↗ https://link.springer.com/book/10.1007/978-3-319-08581-4
2 ↗ https://www.blauer-engel.de/de/produktwelt/software
3 ↗ https://finance.ec.europa.eu/capital-markets-union-and-financial-markets/company-reporting-and-auditing/company-reporting/corporate-sustainability-reporting_en

In der nachhaltigen Softwareentwicklung liegt heute ein besonderer Fokus auf dem effizienten Umgang mit Ressourcen, welchen man früher eher indirekt betrachtet hat. Dazu gehören die CO2-Emissionen, die mit der Entwicklung und dem Betrieb von Software verbunden sind. Diese Emissionen stellen einen bedeutenden Kostenfaktor dar, der sowohl finanzielle als auch ökologische Auswirkungen hat[4].

Wenn Teams bei der Entwicklung und dem Betrieb energieeffiziente Methoden und Technologien nutzen, können die CO2e-Kosten reduziert werden. Das beinhaltet die optimale Ausnutzung von Hardware, die Senkung des Energiebedarfs, die Nutzung klimaneutraler Energiequellen und die Anwendung effizienter Entwicklungspraktiken. Dafür ist ein interdisziplinäres Denken über alle Phasen des Entwicklungsprozesses notwendig.

Ziel dieses Leitfadens ist es daher auch, sich dem Thema Nachhaltigkeit über die Betrachtung des Entwicklungsprozesses zu nähern, um über Zusammenhänge aufzuklären und mögliche Einsparpotenziale offenzulegen. Er richtet sich dabei insbesondere an Software-Architekten, -Entwickler und -Tester, kann aber auch anderen an der Entwicklung beteiligten Personengruppen helfen, neue Perspektiven zu gewinnen. Dieser Band 3 ist so aufgebaut, dass er sich an den Phasen des Software Lebenszyklus orientiert, wobei Kapitel 5 eine detaillierte Auseinandersetzung zu Software Carbon Intensity enthält.

Dieser Leitfaden folgt auf zwei weitere Leitfäden, die sich einerseits mit ↗ der Nachhaltigkeit von digitalen Lösungen, andererseits mit ↗ Nachhaltigkeit in Auftragsklärung und Konzeptarbeit beschäftigen.

Vorweg gesagt sei, dass nachhaltige Softwareentwicklung ein Bewusstsein und eine aktive Auseinandersetzung mit den Auswirkungen der eigenen Arbeit erfordert. Als Ergebnis belohnt sie nicht nur mit niedrigeren CO2-Emissionen, sondern auch mit sinkenden Betriebskosten.

4 ↗ https://dserver.bundestag.de/btd/20/036/2003650.pdf

2 CO2-Emissionen & Nachhaltigkeit als Teil des Entwicklungsprozesses

Grundbetrachtung

Nachhaltigkeit in der Softwareentwicklung ist ein vielschichtiges und komplexes Thema. Es gibt kaum einzelne Maßnahmen, durch die Softwaresysteme oder -unternehmen schlagartig nachhaltiger werden. Vielmehr besteht die Herausforderung darin, viele kleinere Ansätze zu verfolgen, die einzeln oft nur einen geringen Einfluss haben, sich aber in Summe positiv auswirken.

Dieses Dokument sammelt verschiedene Ansätze, um fundierte Entscheidungsprozesse zu ermöglichen. Um nicht sofort mit der Vielzahl an Optionen zu überwältigen, beginnen wir zunächst mit Kernpunkten, die bei der nachhaltigen Softwareentwicklung berücksichtigt werden sollten und sich wie eine Klammer um das gesamte Dokument legen.

CO2-Emissionen sind ein Kostenfaktor,

da Unternehmen zunehmend gesetzliche Vorgaben und Umweltauflagen erfüllen müssen, die mit Kosten für CO2-Kompensation und Energieverbrauch verbunden sind. Zudem können hohe CO2-Emissionen das Image eines Unternehmens beeinträchtigen und dadurch indirekte Kosten durch Kunden- und Marktverluste verursachen.

Ansätze zur Reduktion von CO2-Emissionen

variieren stark je nach Anwendungsgebiet, da unterschiedliche Softwaresysteme unterschiedlichen Rahmenanforderungen unterliegen. Daher können allgemeine Leitfäden nur Hilfestellungen bieten, während spezifische Lösungen individuell an die jeweilige Problemstellung und die spezifischen Qualitätskriterien angepasst werden müssen

CO2-Reduktion braucht Balance,

da CO2-Emissionen nicht vollständig vermeidbar sind und ihre Vermeidung selten primärer Kaufanreiz sind, müssen Entscheidungen in der Softwareentwicklung nicht nur auf Verbrauchszielen basieren, sondern auch andere Qualitätskriterien wie Leistung, Skalierbarkeit und Wartbarkeit berücksichtigen, um eine Balance zwischen Nachhaltigkeit und anderen Erfolgsfaktoren zu erreichen.

Nachhaltigkeit muss Teil des Entwicklungsprozesses sein,

da ökologische Überlegungen nur so konsequent und systematisch in jede Phase der Softwareentwicklung einfließen können, von der Planung bis zur Wartung. Das gewährleistet, dass nachhaltige Praktiken nicht nur sporadisch angewendet, sondern fest in der operativen Tätigkeit eines Unternehmens integriert werden.

Nachhaltigkeit erfordert Monitoring,

da einmalige Lösungen meist nur kurzfristig wirken und neue Ineffizienzen schnell auftreten können. Durch kontinuierliche Überwachung und Analyse können sich nachhaltige Lösungsstrategien dauerhaft im Entwicklungsprozess verankern. Zudem kann auf neue Problemstellen schnell reagiert werden, bevor sie sich unkontrolliert ausweiten.

Anpassungen nur auf Basis einer gesicherten Datenbasis

umsetzen, da nur so fundierte Entscheidungen über Optimierungsmaßnahmen getroffen werden können. Ohne eine solide Datengrundlage besteht das Risiko, dass Änderungen ineffektiv bleiben oder sogar kontraproduktive Effekte haben, wodurch nachhaltige Ziele verfehlt werden könnten.

Betrachten wir diese Punkte näher, stellt sich jedoch die Frage, wo genau sich Einsparungs- und Nachhaltigkeitseffekte finden, wie sie sich äußern und wie sie demzufolge überwacht werden sollten. Hierzu findet man Informationen im ISO-Standard 21031[5], der drei Hauptbereiche hervorhebt.

Energieeffizienz

Maßnahmen, die darauf abzielen, dass die Software weniger Strom verbraucht, aber dieselbe Funktion erfüllt.

Hardware-Effizienz

Maßnahmen, die ergriffen werden, damit Software weniger physische Ressourcen benötigt, um die gleiche Funktion zu erfüllen.

Kohlenstoff-Bewusstsein

Maßnahmen zur zeitlichen oder regionalen Verlagerung von Berechnungen, um saubere, erneuerbare oder kohlenstoffärmere Quellen für Elektrizität zu nutzen.

Im Entwicklungsprozess müssen wir die sechs Grundprinzipien und drei Hauptbereiche auf die einzelnen Phasen anwenden und die Zusammenhänge in jeder Phase gesondert betrachten, um insgesamt nachhaltiger zu arbeiten.

5 ↗ https://www.iso.org/standard/86612.html

Anknüpfungspunkte und Einflussfaktoren

Entwickelt man Software, kommt man schnell mit dem Begriff der Softwarearchitektur in Berührung. Sie beschreibt sowohl die strukturelle Beschaffenheit eines Softwaresystems unter Berücksichtigung der Anforderungen, als auch das Vorgehen, um jene Strukturen zu schaffen. Die Begrifflichkeit ist leider nicht sehr eindeutig und es gibt viele unterschiedliche Betrachtungsweisen[6].

In Bezug auf Ressourceneffizienz und Nachhaltigkeit muss beachtet werden, dass einzelne Softwaresysteme und deren Architektur sich nur begrenzt auf den gesamten Ressourcenbedarf eines Unternehmens auswirken. Im Folgenden werden weitere Perspektiven innerhalb eines Unternehmens aufgezeigt, um andere wichtige Einflussfaktoren hervorzuheben. Das bedeutet nicht zwingend, dass im Softwareentwicklungsprozess auch automatisch auf Forderungen dieser Bereiche direkt reagiert werden muss. Durch zum Beispiel Änderungen von gesetzlichen Vorschriften oder der Unternehmensstrategie können jedoch zusätzliche Anforderungen an die Softwareentwicklung auch in diesen Bereichen entstehen.

Name	Business-Architektur
Ziel	Stellt die Grundlage für die Gestaltung von Geschäftsprozessen und die Entwicklung von IT-Systemen dar
Umfang	Gesamte Organisation, einschließlich Geschäftsstrategien, Geschäftsfunktionen, Geschäftsobjekte und Geschäftsprozesse
Perspektive	Langfristige Perspektive unter Berücksichtigung der strategischen Ziele des Unternehmens
Elemente	Geschäftsmodelle, Geschäftsprozessmodelle, Leistungsindikatoren und Organisationsstrukturen
Bezug zur Ressourceneffizienz	Gibt Nachhaltigkeitsziele über alle Geschäftsbereiche hinweg vor und ist wichtiger Bestandteil bei Sustainability Reporting und ESG-Kriterien

6 ↗ https://gi.de/informatiklexikon/software-architektur

Name	Enterprise-Architektur
Ziel	Zielt darauf ab, die IT-Infrastruktur so zu gestalten, dass sie die Geschäftsanforderungen optimal unterstützt und langfristig flexibel bleibt
Umfang	Gesamte IT-Landschaft, einschließlich Anwendungen, Daten, Infrastruktur, Prozesse und Standards
Perspektive	Langfristige Perspektive unter Berücksichtigung der aktuellen und zukünftigen Anforderungen an das Unternehmen
Elemente	Architekturmodelle, Roadmaps, Standards und Richtlinien
Bezug zur Ressourceneffizienz	Bei der Auswahl von Technologien und Lösungsalternativen können Nachhaltigkeitskriterien wie Energieverbrauch und Langlebigkeit eine Rolle spielen.

Name	Data-Architektur
Ziel	Strebt an, dass Daten konsistent, zugänglich und nutzbar sind, um fundierte Entscheidungen zu treffen
Umfang	Gesamte Datenlandschaft, einschließlich der Datenquellen, Datenmodelle, Datenqualität und Datenintegration
Perspektive	Langfristige Perspektive unter Berücksichtigung der sich ändernden Datenanforderungen des Unternehmens
Elemente	Datenmodelle, Metadaten-Management, Datenqualitätssicherung und Data Governance
Bezug zur Ressourceneffizienz	Klärt, welche Daten unbedingt erhoben und gespeichert werden müssen. Damit ergibt sich ein umfangreiches Einsparpotenzial, falls bisher keine klare Datenstrategie vorlag.

Name	Solution-Architektur
Ziel	Zielt darauf ab, eine technische Lösung zu entwerfen, die die Anforderungen einer bestimmten Anwendung oder eines bestimmten Geschäftsprozesses erfüllt
Umfang	Spezifische Lösung und ihre Komponenten
Perspektive	Hat eine eher kurz- bis mittelfristige Perspektive und konzentriert sich auf die technische Umsetzung
Elemente	Komponentenmodelle, konkrete Datenmodelle, Schnittstellendefinitionen und Deployment-Pläne
Bezug zur Ressourceneffizienz	Die Auswirkungen einer Lösung auf die Umwelt sollten über den gesamten Lebenszyklus betrachtet werden, von der Entwicklung bis zur Abschaltung.

Diese Liste der Perspektiven und Domänen ist sicher nicht vollständig, zeigt aber, wie eng die Softwareentwicklung mit anderen Sichten und strategischen Betrachtungen eines Unternehmens verknüpft ist und wie sich diese gegenseitig beeinflussen.

Kurzfassung: »Architekturen« und wie sie sich aufeinander auswirken

- Business-Architektur: definiert die Geschäftsziele und -prozesse, die durch die IT unterstützt werden sollen.
- Enterprise-Architektur: setzt die strategischen Ziele der Business-Architektur in eine IT-Strategie um.
- Data-Architektur: unterstützt die Business-Architektur und die Enterprise-Architektur durch Datenstrategien und kann zur Datensparsamkeit beitragen.
- Solution-Architektur: realisiert die technischen Lösungen, die die Anforderungen der Business-Architektur, der Enterprise-Architektur und der Data-Architektur erfüllen.

In Bezug auf die Nachhaltigkeit von Lösungen ist insbesondere in Großunternehmen eine Anpassung der eigenen Business-Architektur aufgrund gesetzlicher und gesellschaftlicher Rahmenbedingungen erforderlich. Im Anschluss müssen die Enterprise- und Data-Architektur optimiert werden, was letztlich Nachhaltigkeitsanforderungen gegenüber den Architekturen der verschiedenen Lösungssysteme im Unternehmen zur Folge hat.

Der Softwareentwicklungsprozess

Nachhaltige Softwareentwicklung beginnt mit einem strukturierten und durchdachten Entwicklungsprozess, der in mehrere Phasen unterteilt ist. Nachfolgend zeigen wir die üblichen Stationen eines solchen Prozesses und beschreiben, wie sie sich beim Thema Nachhaltigkeit gestalten. Eine genauere Betrachtung des Gesamtprozesses findet sich im ↗ ersten Leitfaden der Reihe. Natürlich können sich die Abläufe insbesondere in iterativ inkrementellen Vorgehen und durch agile Arbeitsweisen anders gestalten. Zur Vereinfachung der Erläuterungen werden jene jedoch nachfolgend nicht gesondert betrachtet.

Auftragsklärung

In der Planungsphase werden die Anforderungen an die Software definiert und die Projektziele festgelegt. Es werden Ressourcen und Zeitpläne bestimmt, um eine effiziente und zielgerichtete Entwicklung zu gewährleisten. In dieser Phase können wichtige Grundlagen für nachfolgende Entscheidungen gelegt werden, da hier nicht selten die grundlegenden Betriebsmittel ausgewählt werden. Diese Phase wurde im ↗ zweiten Leitfaden ausgiebig behandelt und wird daher hier nicht mehr ausgiebig betrachtet.

Konzeptarbeit

Die Konzeptphase umfasst eine detaillierte Untersuchung der Softwareanforderungen. Dabei wird zunächst ein erster Grobentwurf der Softwarearchitektur erstellt. Sie sollte nicht nur rein funktionale Anforderungen berücksichtigen, sondern auch Wert auf Qualitätsanforderungen legen. Diese wirken sich stark auf die Ressourcennutzung der Architektur und Implementierung aus.

Im Anschluss an das Grobkonzept wird die Softwarearchitektur in detaillierte Pläne umgesetzt. Es wird über Programmiersprachen, Plattformen und Frameworks entschieden. Diese Entscheidungen sollten unter dem Aspekt der Nachhaltigkeit getroffen werden, um den Energieverbrauch und die Umweltbelastung zu minimieren.

Entwicklung & Test

In dieser Phase wird der Code gemäß den Plänen aus der Konzeptphase geschrieben. Neben der Funktionalität und Effizienz sollte auch auf die Energieeffizienz des Codes beachtet werden. Es werden Tests durchgeführt, um sicherzustellen, dass die Software den Anforderungen entspricht und nachhaltig betrieben werden kann.

Betrieb & Wartung

In der Bereitstellungsphase wird die Software in die Produktionsumgebung überführt und den Nutzern zur Verfügung gestellt. Darauf folgt die eigentliche Nutzung und somit der reguläre Betrieb. Die Nutzung energieeffizienter Server und optimierter Datenbanken kann den Betrieb umweltfreundlicher und nachhaltiger gestalten. Ein kontinuierliches Monitoring und eine adäquate Planung von Reserven sind wichtige Hebel, um Einsparpotenziale zu heben.

Die Wartung umfasst die laufende Pflege der Software, um sicherzustellen, dass sie weiterhin den Anforderungen entspricht und fehlerfrei funktioniert. Nachhaltige Wartungspraktiken beinhalten regelmäßige Überprüfungen und Optimierungen zur Minimierung des Energieverbrauchs und zur Verbesserung der Performance.

3 Konzeptarbeit mit Fokus auf die Entwicklung

Anforderungen und Architekturauswahl

Die Softwarearchitektur ist das Bindeglied zwischen Anforderungen und Umsetzung. Ihr kommt damit eine besondere Bedeutung zu, da sie die Leitplanken vorgibt, innerhalb derer detaillierte Entscheidungen getroffen werden. Das geschieht anhand von Qualitätsanforderungen, welche neben der reinen Benutzbarkeit und Stabilität auch den Ressourcenverbrauch betrachten sollten. Jene müssen in realen Umgebungen selbstverständlich mit anderen bestehenden Anforderungen abgewogen werden.

Energieeffizienz: Die Architektur sollte darauf ausgelegt sein, Energie effizient zu nutzen, um den Betrieb der Anwendung zu unterstützen und den Energieverbrauch zu minimieren.

Hardwarenutzung: Die Architektur sollte darauf ausgelegt sein, nur so viel Hardware wie CPU, Speicher oder Netzwerkbandbreite zu verwenden, wie zur Erfüllung der Aufgaben tatsächlich benötigt wird. Das gewährleistet eine optimale Nutzung der verfügbaren Ressourcen und vermeidet unnötigen Verbrauch, was sowohl die Leistung als auch die Skalierbarkeit der Systeme verbessert.

Skalierbarkeit: Die Architektur sollte es ermöglichen, Systembestandteile bedarfsgerecht zu skalieren, um auf sich ändernde Leistungsanforderungen reagieren zu können und somit Verschwendung vorzubeugen.

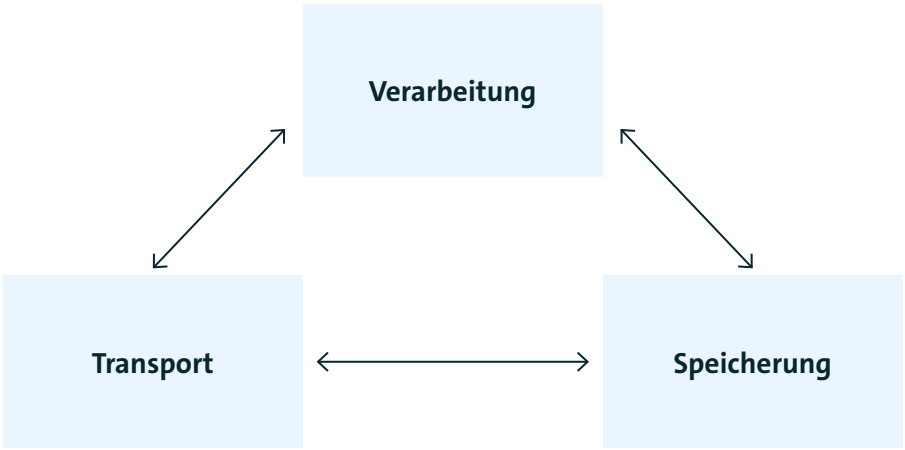
Redundanz und Ausfallsicherheit: Die Architektur sollte Mechanismen zur Fehlererkennung, Fehlerbehebung und Ausfallsicherheit enthalten, um die Verfügbarkeit der Anwendung zu verbessern und Ressourcenverluste durch Ausfälle zu minimieren. Die Redundanz selbst muss allerdings so bedarfsgerecht geschehen, dass sie nicht zur Ressourcenverschwendung beiträgt.

Monitoring und Optimierung: Die Architektur sollte Mechanismen zur Überwachung der Leistung und Ressourcennutzung der Anwendung enthalten, um Engpässe zu identifizieren und Optimierungsmöglichkeiten zu erkennen.

Wartbarkeit: Eine nachhaltige Architektur sollte darauf ausgelegt sein, langfristig gewartet, erweitert und angepasst werden zu können, ohne dass dies zur ineffizienten Nutzung von Ressourcen führt.

Auswirkungen der Grundstruktur

Die Softwarearchitektur muss den Ressourcenverbrauch berücksichtigen, da je nach gewählter Architektur mehr oder weniger Daten erfasst werden, die Berechnung entweder zentralisiert oder verteilt, und unterschiedliche Eingriffsmöglichkeiten auf die Speicherung ermöglicht werden können. So kann es beispielsweise sehr hilfreich sein, die Bestandteile eines Softwaresystems möglichst stark aufzuteilen, um eine bessere Skalierbarkeit zu erreichen. So können jedoch auch die Transportkosten steigen, da der Datenaustausch zwangsläufig über ein Netzwerk statt innerhalb eines Prozesses erfolgt, wodurch diverse Optimierungsverfahren nicht mehr wirksam sind.



Am offensichtlichsten entstehen Emissionen bei der Verarbeitung von Daten, da für die notwendigen Berechnungen sowohl Energie für die Berechnung als auch die Kühlung der entsprechenden Hardware benötigt wird. Hinzu kommt das in der Hardware gebundene CO2. Bei besonders hoher Leistungsfähigkeit der Hardware ist es hoch und muss bei jedem Hardwaretausch erneut in die Gesamtbilanz der CO2-Kosten des Systems aufgenommen werden.

Bei der Speicherung von Daten entstehen CO2-Emissionen durch den kontinuierlichen Energieverbrauch der Speichersysteme und Rechenzentren, die für den Betrieb und die Kühlung der Speichergeräte erforderlich sind. Zudem wird bei der Herstellung der Speichermedien CO2 gebunden, insbesondere bei Geräten mit hoher Speicherkapazität.

Beim Transport von Daten entstehen CO2-Emissionen durch den Energieverbrauch der Netzwerkgeräte wie Router, Switches und Mobilfunkmasten, die für die Datenübertragung notwendig sind. Auch bei der Herstellung dieser Netzwerkkomponenten wird CO2 gebunden.

Hier offenbaren sich auch Zielkonflikte. Verteilt man etwa die Verarbeitung und Speicherung von Daten auf unterschiedliche Systeme, die für die jeweiligen Aufgaben optimiert sind, erhöhen sich dadurch die Transportaufwände.

Auch wenn der CO2-Ausstoß eines Systems gerade nicht im Fokus steht, eröffnet die Auseinandersetzung mit der CO2-Bilanz eine neue Perspektive auf bestehende Kostenstrukturen. Tendenziell führen eine ineffiziente Nutzung der Hardware, übermäßiger Datenverkehr innerhalb des Systems oder unnötig beanspruchter Speicherplatz zu vermeidbaren Betriebskosten. Indem man sich auf die Reduzierung des CO2-Ausstoßes konzentriert, ergeben sich somit wertvolle Ansätze zur Optimierung und potenziellen Reduktion der Betriebskosten.

Vergleich von Architekturmustern

Basierend auf den zuvor dargestellten Zusammenhängen soll dieses Kapitel einen Überblick über verschiedene Architekturmuster geben und ihren sinnvollen Einsatz im Sinne der Ressourceneffizienz erläutern. Diese Übersicht soll damit als Denkanstoß und Entscheidungshilfe dienen. Sie stellt explizit kein Ranking der Muster dar, da ihr Einsatz von vielen verschiedenen Faktoren beeinflusst wird. Hierzu zählen etwa das Vorwissen im Team, die Qualitätsanforderungen an das Gesamtsystem und das bereits etablierte Ökosystem. Bei hinreichend komplexen Systemen werden die Muster ohnehin meist gemischt verwendet.

Name	Monolith / Modulith
Beschreibung	Die Anwendung ist zur Laufzeit eine geschlossene Einheit. In der Entwicklungssicht unterscheiden sich Monolith und Modulith darin, dass der Monolith aus einer eng gekoppelten Codebasis besteht, während der Modulith stärker in getrennt entwickelten Einheiten strukturiert ist, die über APIs miteinander interagieren.
Bezug auf Ressourceneffizienz	Der Ressourcenverbrauch von Monolith oder Modulith zur Laufzeit ist als einzelner Prozess und durch In-Memory-Kommunikation seiner Codebestandteile sparsamer gegenüber einem verteilten System. Modulithen sind gegenüber Monolithen anfänglich etwas ressourcenintensiver, die bessere Wartbarkeit äußert sich langfristig jedoch zumeist in der Schonung von Ressourcen in Entwicklung, Test und Laufzeit. Die enge Code-Kopplung in Mono- oder Modulithen verursacht oft überproportional steigende Wartungs- und Testaufwände, wenn die Anwendung im Funktionsumfang und in Komplexität wächst. Weiterhin verhindert die enge Kopplung die getrennte Skalierung einzelner, häufiger benötigter Funktionen und kann in der vertikalen oder horizontalen Skalierung des gesamten Mono- oder Modulithen zu hohen Laufzeitkosten führen.
Einschätzung	Das Muster eignet sich insbesondere für kleinere Applikationen mit geringem Änderungsumfang, die keine besonderen Anforderungen an Skalierbarkeit und Ausfallsicherheit haben (z. B. eine Desktopanwendung).

Name	Microservices / Service Oriented Architecture (SOA)
Beschreibung	Mit SOA bezeichnet man Anwendungen, die als lose gekoppelte Dienste organisiert sind. Sie kommunizieren über das Netzwerk und können unabhängig voneinander bereitgestellt werden. Microservices sind eine spezialisierte Form der serviceorientierten Architektur, bei der Anwendungen in sich geschlossene, eigenständige, meist feingranulare Dienste bereitstellen, die jeweils eine spezifische Geschäftsfunktion erfüllen. Die Kommunikation bei Microservices gegenüber SOA erfolgt mittels schlanker Netzwerkprotokolle. Dies erleichtert die lose Kopplung von Komponenten.
Bezug auf Ressourceneffizienz	Gegenüber Mono- oder Modulithen verursacht eine verteilte Architektur durch die Organisation in getrennte Prozesse und die Datenübertragung zwischen den (Micro-)Services einen generell höheren Ressourcenverbrauch, solange keine Skalierung erforderlich ist. In der Skalierung einzelner Funktionen ergeben sich zur Laufzeit jedoch in der Regel signifikante Vorteile bezüglich des Ressourcenverbrauchs im Vergleich zu Mono- oder Modulith. Auch die Reduktion der Komplexität in den einzelnen Diensten auf ihre jeweils begrenzte Aufgabenstellung verbessern Wartbarkeit und Ausfallsicherheit und schonen damit langfristig Ressourcen.
Einschätzung	Serviceorientierte und Microservice-Architekturen eignen sich besonders für größere Softwaresysteme mit dedizierten Einzelbestandteilen, die getrennt skaliert und bereitgestellt werden können. Sie müssen aber sorgfältig geplant werden, um nicht unnötig Ressourcen zu verbrauchen. Microservices gelten als das bevorzugte Vorgehen zur Umsetzung von serviceorientierten Architekturen. Es muss jedoch darauf geachtet werden, dass die Dienste tatsächlich in sich geschlossen sind und unabhängig voneinander in loser Kopplung bereitgestellt werden können. Andernfalls droht die Gefahr eines verteilten Monolithen, der statt der Vorteile die Nachteile aller Architekturmuster miteinander verbindet.

Name	Serverless
Beschreibung	Weniger ein Architekturmuster als ein Ausführungsmodell. Es ermöglicht die Entwicklung und Bereitstellung von Anwendungen, ohne sich um die zugrunde liegende Infrastruktur kümmern zu müssen. Die Ressourcen werden dynamisch und automatisch bereitgestellt, basierend auf Anforderungen und Nutzung.
Bezug auf Ressourceneffizienz	Serverless erlaubt die bei weitem beste Skalierung aller vorgeschlagenen Muster und Modelle. Es wird insbesondere im Kontext von Microservice-Architekturen eingesetzt, um den Verwaltungsaufwand an Cloudbetreiber zu externalisieren. Dass viele Serverless-Dienste proprietär sind, kann die Portabilität erschweren und zu einer Abhängigkeit führen, die im Laufe der Zeit höhere Betriebskosten nach sich ziehen kann.
Einschätzung	Grundsätzlich ist Serverless hervorragend geeignet, um die Ressourceneffizienz zu verbessern, da es eine sehr gute Skalierung erlaubt. Es muss jedoch darauf geachtet werden, dass das Softwaresystem nicht zu stark entkoppelt wird, da sonst sehr viele Daten übertragen werden müssen, was die Kosten- und Ressourceneinsparungen negiert. Das Muster eignet sich daher für Systeme, die ein zeitlich sehr stark variables Datenverarbeitungsvolumen benötigen.

Folgt man diesen Erläuterungen, ergibt sich für neu entwickelte Software ein Idealweg in Bezug auf die Ressourceneffizienz, der bereits 2005 von Martin Fowler in seinem Blog erläutert wurde[7]. Zu Beginn werden die wichtigsten Bestandteile in Form eines modularen Monolithen (Modulith) erstellt, um dann abhängig vom Skalierungsbedarf einzelner Module eigenständige Dienste herauszulösen.

7 ↗ https://martinfowler.com/bliki/MonolithFirst.html

Mögliche Maßnahmen und Ansätze

Mit einer steigenden Menge an Berechnungen und zu verarbeitenden Daten, sowie einer wachsenden Entfernung der Bestandteile, steigen auch die CO2-Emissionen des Systems. Der Entwurf eines Softwaresystems hat massiven Einfluss darauf, wie sich diese Faktoren auswirken, da er bestimmt, welche Bestandteile das System hat, welche Verantwortlichkeiten sie haben, in welchen Beziehungen sie stehen und wie sie zukünftig zum Einsatz kommen.

Um Nachhaltigkeit in den Softwareentwurf und damit die Software-Architektur zu integrieren, lohnt es sich, während des Entwurfs Ansätze und Strategien wie die folgenden zu bedenken. Die genannten Maßnahmen führen zunächst allerdings nur theoretisch zu Einsparungen. Ob sie auch realisiert werden, hängt maßgeblich vom realen Kontext ab. Entsprechend ist die folgende Liste als eine Sammlung von Denkanstößen und nicht als strikter Maßnahmenkatalog zu sehen.

Datenminimierung und -effizienz

Datenvolumen reduzieren: Je weniger Daten verarbeitet und übertragen werden, desto geringer ist der Energiebedarf für Berechnung und Speicherung. Daher sollte die Software-Architektur sinnvolle Datenvolumen nennen und angemessene Lösungen skizzieren.

Datenredundanz vermeiden: Mehrfachspeicherung oder unnötige Berechnungen erhöhen den Ressourcenverbrauch. Folglich sollte die Software-Architektur auch Speicherung und Datenredundanz berücksichtigen.

Datenkomprimierung: Effiziente Komprimierungsalgorithmen reduzieren den Umfang der übertragenen Daten und damit den Ressourcenverbrauch. Deshalb sollte die Software-Architektur gezielt Komprimierungsverfahren einsetzen.

Verteilte vs. zentrale Systeme

Abwägung von Skalierbarkeit und Energieeffizienz: Eine dezentrale Architektur ermöglicht eine bessere Skalierung, kann aber zu höheren Kommunikationskosten führen.

Lokale Berechnungen bevorzugen: Innerhalb einer einzigen Laufzeitumgebung – sei es ein Prozess, ein Betriebssystem oder ein Rechenzentrum – lassen sich Datenströme und Berechnungen effizienter optimieren, als wenn sie über mehrere ähnliche Umgebungen verteilt sind.

Entfernungen minimieren: Kurze Übertragungswege reduzieren den Ressourcenverbrauch.

Auswahl von Softwarealgorithmen und -sprachen

Cloud-Technologien: Cloud-Technologien ermöglichen eine flexible und effiziente Nutzung von Hardware-Ressourcen, da Rechenzentren Workloads dynamisch verteilen und optimieren können. Durch zentrale Verwaltung und Automatisierung lassen sich Ressourcen besser skalieren und an den tatsächlichen Bedarf anpassen, wodurch eine höhere Effizienz im Betrieb erreicht wird.

Effiziente Algorithmen: Bei hohem Berechnungsaufwand können optimierte Algorithmen den Rechenaufwand erheblich reduzieren. Die Architektur sollte berücksichtigen, welche Algorithmen besonders optimiert werden sollten.

Energieeffiziente Programmiersprachen: Nicht alle Programmiersprachen sind in allen Situationen gleich ressourceneffizient. Das sollte bereits bei ihrer Auswahl während des Entwurfs berücksichtigt werden.

Parallelisierung: Die volle Ausnutzung der Parallelisierungsfeatures von Laufzeitumgebungen kann die Rechenzeit verkürzen und Energie sparen. Inwiefern Abläufe parallelisiert werden können, sollte bereits in der Entwurfsphase berücksichtigt werden.

Serverless Computing: Serverless-Funktionen sind für sporadisch ausgeführte Aufgaben zu bevorzugen, da sie automatisch skalieren und Ressourcen nur bei Bedarf nutzen.

4 Entwicklung

Um Ressourcen besser zu nutzen und CO2-Emissionen wirksam zu reduzieren, ist ein gemeinschaftliches Vorgehen während der Umsetzung notwendig. Durch die Kombination verschiedener Optimierungen können sich die Effekte summieren und insgesamt zu einer erheblichen Verbesserung führen:

- Softwarearchitekten berücksichtigen die Ressourceneffizienz beim Entwurf und helfen bei der Etablierung der Konzepte während der Entwicklung.
- Softwareentwickler schreiben energieeffizienten Code, optimieren Datenbankabfragen und nutzen effiziente Datenstrukturen.
- Entwickler optimieren Datenbankabfragen und bauen effiziente Datenstrukturen auf.
- DevOps Engineers beschränken den Ressourcenaufwand einer Software in der Laufzeit durch sorgfältiges Design von Pipelines, Deployment und Skalierung.

Quelle: ↗ Software Carbon Intensity (SCI) Specification (greensoftware.foundation)

Programmierung

Nachhaltigkeit und Ressourceneffizienz spielen in der Programmierung eine wichtige Rolle. Es reicht von der Auswahl notwendiger Frameworks bis hin zur Betrachtung ganzer Ökosysteme[8] und analysiert auch die optimale Nutzung von Algorithmen. Die damit verbundenen Handlungsempfehlungen müssen im Kontext gesehen und im Verhältnis zum Ressourcenverbrauch in der zukünftigen Laufzeitumgebung betrachtet werden. So unterscheidet sich etwa das Vorgehen zur Einsparung von Ressourcen bei der Front-End-Entwicklung deutlich von dem bei der Datenbankentwicklung.

Demzufolge ergibt es Sinn, Guidelines wie jenen des W3C zu folgen und die damit verbundenen Best Practices während der Umsetzung zu berücksichtigen. Die Menge jener Guidelines und deren Detailgrad ist in den vergangenen Jahren stark gestiegen. Einige Empfehlungen sind jedoch widersprüchlich oder so detailliert, dass man ihnen nur schwer folgen kann.

Daher empfiehlt es sich, zunächst den generellen Best Practices des jeweiligen Anwendungsgebietes zu folgen. Arbeitet man sehr viel mit Datenbanken, sollte man sich mit der optimalen Struktur und Verwendung von Features des jeweiligen Datenbankmanagementsystems befassen.

Darauf aufsetzend können allgemeinere Patterns zur nachhaltigen Softwareentwicklung angewendet werden, wie sie sich zum Beispiel in diesem Dokument oder bei der Green Software Foundation[9] finden. Dabei sollten insbesondere ständig ausgeführte Programmstellen oder

8 ↗ https://haslab.github.io/SAFER/
9 ↗ https://patterns.greensoftware.foundation/

andersartig kritische Programmbestandteile optimiert werden. Das hilft, die Balance zwischen Feature-Umsetzung und Optimierung zu wahren und die kognitive Lasten der Entwickler in Grenzen zu halten.

In diesem Zusammenhang sind die reine Programmierung und der Betrieb der Software nur schwer voneinander zu trennen, wenn es um Ressourceneffizenz geht. Deswegen findet sich in ↗ Kapitel 5 ein Deep Dive in die Software Carbon Intensity, welche in beiden Phasen eine erhebliche Rolle spielt.

Softwaretest

Die Ressourcenverbräuche im Softwaretest ergeben sich vorrangig durch die Testausführung, dafür verwendete Testdaten und die Laufzeitumgebungen. Auch hier können durch die Umsetzung von Best Practices Einsparungen erzielt werden. Dazu zählen beispielsweise die Minimierung von manuellen Tests des Gesamtsystems und eine möglichst umfangreiche Verwendung automatisierter Unit-Tests, welche durch Integrationstests ergänzt werden. Da nicht das gesamte Realsystem zur Verfügung stehen muss und Fehler schon frühzeitig auffallen, können Ressourcen gespart werden.

Weitere Einsparungen können durch eine effiziente Auswahl der notwendigen Testdaten, die gezielte Ausführung von Tests sowie den Einsatz effizienterer Testumgebungen erzielt werden. Werden aufwändige Tests beispielsweise nicht mehr täglich innerhalb einer virtuellen Maschine durchgeführt, sondern zunächst durch einfache Smoke Tests innerhalb einer containerisierten Umgebung ersetzt, liegen die Ergebnisse schneller vor und verbrauchen weniger Energie.

Tests bieten aber noch einen weiteren Ansatzpunkt für die Ressourceneffizienz: Sie können Engpässe und Verschwendung aufdecken, wenn gezielt darauf getestet wird. Es ergibt Sinn, neben rein funktionalen Tests auch regelmäßig Last- und Leistungstests durchzuführen, um aufzudecken, an welchen Stellen das System seine Ressourcen nicht optimal einsetzt.

DevOps

DevOps stellt den Übergang von der Entwicklung zum Betrieb dar. Daher gelten viele Erläuterungen zum Betrieb aus ↗ Kapitel 6. DevOps kommt damit aber auch in der Entwicklungsphase eine besondere Bedeutung zu, ermöglicht es doch den Rückfluss von Informationen aus dem Einsatz einer Software in die Entwicklungsteams. Hierfür ist ein konstantes Monitoring der Arbeitsergebnisse im Betrieb und während des Tests nötig, welches durch Werkzeuge realisiert wird.

Neben den Werkzeugen für die Analyse und das Profiling der Softwarestrukturen sind auch Werkzeuge für Release(-planung), Build und Testausführung zu berücksichtigen. Sie können zu einem erhöhten Ressourcenverbrauch während der Entwicklung führen, wenn sie nicht korrekt eingesetzt werden. Dazu gehören beispielsweise zu häufige und zu umfangreiche Builds, ungünstig gewählte Build-Umgebungen oder Deployment-Strategien.

Die Hebel zur Ressourcenoptimierung sind gerade im Bereich des DevOps sehr umfangreich und belohnen nicht zuletzt mit Zeitersparnis sowie höherer Entwicklerzufriedenheit.

Mögliche Maßnahmen und Ansätze

Aus den geschilderten Zusammenhängen lassen sich folgende Ansätze für die Entwicklung ableiten. Dabei zeigen sich die Vorteile gängiger Praktiken bei Continuous Integration und Continuous Deployment durch den Einsatz von Monitoring- und Analyselösungen. Mit ihnen lassen sich Schwächen im Ressourcenbedarf schnell aufdecken und beseitigen, bevor sie zu erhöhten Betriebskosten führen.

Programmierung

Effiziente Algorithmenauswahl: Bei sehr rechenintensiven Aufgaben oder solchen, die ständig wiederholt werden, sollten Algorithmen eingesetzt werden, die optimal für die Aufgaben geeignet sind und dabei Rechenkapazität sowie Speicherbedarf auf ein Mindestmaß begrenzen.

Deploymentoptimierung: Gemeinsam verwendete Codebestandteile sollten gemeinsam paketiert werden, um einen unnötigen Kommunikationsaufwand durch zu stark getrennte Programmbestandteile oder mehrfaches Deployment zu vermeiden.

Datenstrukturoptimierung: Es sollten einheitliche Datenstrukturen, -formate und -standards verwendet werden, um eine unnötige Übertragung und Konvertierung zu vermeiden.

Speicherauslastung: Es sollte darauf geachtet werden, dass die gewählten Datenstrukturen Memory und Storage effizient nutzen und keine unnötigen Daten vorhalten.

Auswahl der Programmiersprachen und Frameworks: Programmiersprachen und Frameworks sollten für ihr Einsatzgebiet optimiert sein und keine unnötigen Abhängigkeiten bedingen.

Profiling und Engpassanalyse: Profiling-Tools sollten bereits während der Entwicklung eingesetzt werden, um Daten- oder Verarbeitungsengpässe im Code zu identifizieren und zu optimieren.

Code-Optimierung: Redundante Berechnungen und überflüssige Zuweisungen sind genauso zu vermeiden wie Code-Clones, also Codestellen, die nahezu identisch zu anderen Codestellen sind.

Kontinuierliches Refactoring: Redundante und ineffiziente Codeabschnitte müssen regelmäßig identifiziert und optimiert werden. Das führt zu einer verbesserten Leistung, einem geringeren Speicherbedarf und einer einfacheren Wartbarkeit der Software.

Künstliche Intelligenz

Vortrainierte Modelle: Vortrainierte Modelle sind bereits auf umfangreiche Datensätze trainiert und verringern einen Teil des notwendigen Ressourcenbedarfs, um ein KI-Modell zur Betriebsreife zu führen.

Gezielter Trainingsstandort: KI benötigt erhebliche Mengen Energie. Indem ein Modell in einer Region trainiert wird, welche vorrangig CO2-neutrale Energieträger einsetzt, kann der CO2-Fußabdruck des entstehenden Modells stark gesenkt werden.

Optimierte Hardware: Speziell auf KI ausgelegte Hardware benötigt weit weniger Energie und führt wesentlich schneller zu Ergebnissen, als wenn übliche Standardhardware eingesetzt wird.

Nutzung kleinerer Modelle: Auch vergleichsweise kleine Modelle können gute Ergebnisse liefern. Es lohnt sich, ihren Einsatz zu prüfen, bevor man auf umfangreichere Modelle zurückgreift. Dadurch verringert sich nicht nur der Ressourcenverbrauch bei der Bearbeitung von Anfragen, sondern auch der Speicherbedarf.

Spezialisierte Modelle: Auch wenn einige Anbieter den Trend zu generalisierten Modellen verfolgen, sind spezialisierte Modelle wesentlich effektiver im Ressourcenverbrauch und der Bearbeitung. Das führt gleichermaßen zu geringeren Bearbeitungszeiten und Betriebskosten.

Test

Testdaten: Testdaten sollten so dimensioniert sein, dass sie die zu testenden Einsatzbedingungen angemessen widerspiegeln, aber unnötige Datenmengen vermeiden. Das verringert Speicher- und Verarbeitungsaufwände, senkt aber auch die Laufzeit der Tests.

Testumgebung: Für Testumgebungen empfiehlt sich der Einsatz von Containern, da diese im Vergleich zu Tests auf virtuellen Maschinen oder direkt auf der Hardware eine effizientere Ausnutzung der Ressourcen ermöglichen.

Testautomatisierung: Automatisierte Tests sollten insbesondere auf Unit- und Integrationsebene zum Einsatz kommen, da sie die Testaufwände pro Testfall vergleichsweise klein halten und weniger komplexe Laufzeitumgebungen benötigen.

Konzentrierte Systemtests: Systemtests sollten vorrangig als Smoke Tests oder konzentrierte manuelle Tests, die besonders kritische Bestandteile prüfen, ausgeführt werden. Detaillierte Prüfungen sollten eher als Unit- oder Integrationstests und ohne das Gesamtsystem umgesetzt werden.

Last- und Performance-Tests: Punktuelle Prüfung von rechenintensiven Aufgaben decken Ressourcenverschwendung und Optimierungspotenzial auf.

A/B-Tests: Vergleich unterschiedlicher Lösungen für die gleiche Aufgabe erlaubt die Auswahl optimaler Verfahren vor deren tatsächlichem Release.

Ressourcenverbrauch im Test messen: Monitoring des Ressourcenverbrauchs ergibt bereits während des Tests Sinn und muss nicht erst im Betrieb geschehen. So lassen sich die größten Schwachstellen finden, bevor sie in den Betrieb übertragen werden und nur noch schwer zu ändern sind.

Zusammenarbeit

Team-Schulung: Optimal geschulte Mitarbeiter vermeiden die Einführung neuer technischer Schulden und können die eigenen Entwicklungswerkzeuge effektiver verwenden.

Kultur: Eine Kultur der kontinuierlichen Verbesserung und des verantwortungsvollen Umgangs mit Ressourcen schafft die Grundlage, um ressourcenschonende Praktiken in der Entwicklung zu verankern.

5 Deep Dive zu Software Carbon Intensity

Die Software Carbon Intensity ist ein Spezialthema innerhalb der nachhaltigen Softwareentwicklung und betrifft insbesondere die Phasen der Entwicklung und des Betriebs. Sie ist sehr bedeutend innerhalb der Green-Software-Bewegung und soll vertieft behandelt werden.

Grundlagen der SCI-Reduzierung

Die Software Carbon Intensity (SCI) einer Software-Funktionseinheit R (z. B. einzelne User-Session, einzelner API-Aufruf oder gesamter ML-Trainingslauf) ermittelt sich laut ISO 21031:2024 zu

SCI = ((E·I) + M) pro R

mit den Einflussvariablen
E: Energie in kWh, die zum Ablauf der Software benötigt wird
I: Standortbezogene Kohlenstoffintensität, d. h. die reale Kohlenstoffmenge, die pro kWh ausgestoßen wird, gemessen in gCO2eq/kWh
M: Anteilige Embodied Energy in gCO2eq, d. h. der für die betrachtete Software anteilige Kohlenstoffausstoß der erforderlichen Hardware über deren gesamten Lebenszyklus

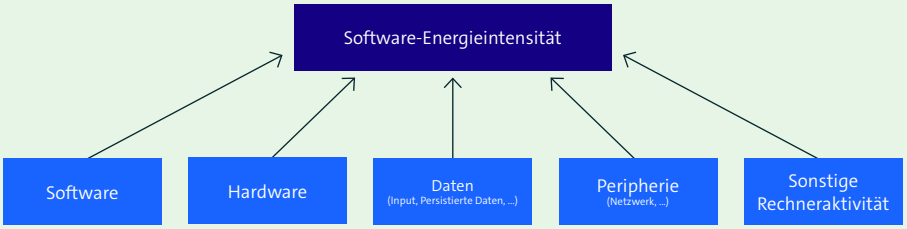
Entwickler und Betreiber von Software haben unterschiedliche Einflussmöglichkeiten:

- Die Entwickler beeinflussen in erster Linie die Variable E pro R, d. h. die Energieintensität der Software. (Wie viel Energie wird tatsächlich für die Herstellung einer bestimmten Funktionalität benötigt?) Außerdem beeinflussen sie stark die Variable M, indem sie Ansprüche an die Aktualität und Beschaffenheit der zum Betrieb benötigten Hardware definieren.
- Die Betreiber der Software bestimmen mit der Wahl der Energieversorgung die Variable I, also die standortbezogene Kohlenstoffintensität der zum Betrieb eingesetzten elektrischen Energie. (Was sind die realen Kohlenstoffemissionen pro kWh in der Energieversorgung ohne Beachtung von Einflüssen wie Emissions-Trading oder Ausgleichsmaßnahmen?) Ferner beeinflussen die Betreiber die Variable M durch die Häufigkeit und spezifische Natur von Hardware-Aktualisierungen. (Wird bei der Hardware-Aktualisierung auf eine möglichst nachhaltige Beschaffung geachtet?)

Da sich dieses Dokument in erster Linie an die Entwickler von Software richtet, stehen die zur Reduktion von E und M ausgerichteten Fragen im Vordergrund. Die relevanten Einflüsse werden im folgenden Abschnitt betrachtet.

Einflüsse auf die Software-Energieintensität

Die Software-Energieintensität wird nicht allein durch die Software bestimmt, sondern unterliegt einer Reihe von Einflüssen:



- Die Auswahl der technischen Infrastruktur der **Software**, d. h. Programmiersprachen, Laufzeitumgebung, Libraries, Algorithmen etc., hat entscheidenden Einfluss auf die Energieintensität.
- Ebenfalls entscheidend ist die Effizienz der **Rechnerhardware**. Durch die typischerweise höhere Energieeffizienz moderner Rechnerausstattung ergibt sich ein Break-Even-Point, an dem es günstiger wird, einen neuen Rechner einzusetzen, als die Lebensdauer eines Altgeräts zu verlängern.
- Der Energieverbrauch einer Software wird in der Regel auch durch die Menge und Art der **verarbeiteten Daten** bestimmt. Die Auswahl der Verarbeitungsalgorithmen bestimmt, wie gut der Rechenaufwand – und der damit korrespondierende Energieverbrauch – mit der Datenmenge skaliert (Rechenkomplexität, Big-O-Notation).
- Die unterschiedliche Aktivität der **Peripherie** (Datei-IO, Grafik, Netzwerk, Drucker etc.) kann einen signifikanten Einfluss auf den Energieverbrauch haben.
- Moderne Serveranwendungen, ggf. in virtualisierter Umgebung, beanspruchen den Rechnerknoten in der Regel nicht exklusiv. Die Energieintensität einer Software zu einem bestimmten Zeitpunkt wird also ebenfalls von **der parallelen Rechneraktivität** beeinflusst. Das bedeutet, dass der Energieverbrauch einer Software – sogar mit identischen Daten – auf demselben Rechnerknoten nicht immer gleich ist. In der Regel nimmt die Energieeffizienz zu, je stärker der Rechner ausgelastet ist. Eine hohe Auslastung der vorhandenen Hardware ist also sowohl unter ökonomischen als auch ökologischen Aspekten sinnvoll.

Metriken zur Reduzierung der Software-Energieintensität

Um eine Reduzierung der Software-Energieintensität – und damit des SCI – zu erreichen, können Entwickler zwei unterschiedliche Arten von Metriken zur Zielerreichung einsetzen: direkte (physikalische) und indirekte (Code-) Metriken.

Direkte Metriken treffen eine Aussage über den Energieverbrauch oder den resultierenden Kohlenstoffausstoß einer Software (in Bezug auf eine angenommene oder reale standortbezogene Kohlenstoffintensität). Sie können damit genutzt werden, um energieintensive Codeteile zu identifizieren. Weiterhin ermöglichen sie ein Monitoring, wie sich die Energieintensität einer Software über den Lebenszyklus verändert. Obwohl sie damit die Grundlage zur Messung des SCI legen und objektive Vergleiche ermöglichen, geben sie keinen Aufschluss darüber, was getan werden muss, um die Energieintensität einer Software zu senken. Auch ob eine Verbesserung erzielt werden kann, beantworten direkte Metriken nicht unmittelbar. Die Generierung detaillierter, kleinteiliger energetischer Metriken ist von hoher technischer Komplexität und wird im nächsten Abschnitt betrachtet.

Indirekte Metriken sind hingegen Codemetriken, die eine Indikation geben, welche Codestellen bezüglich ihrer Energieintensität optimiert werden können. Solche Metriken können manuell erstellt werden, z. B. für analytische Kriterien bezüglich Big-O-Notation oder zyklomatischer Komplexität des betrachteten Codes. Es existieren jedoch eine Reihe von freien oder kostenpflichtigen Analysewerkzeugen (z. B. Sonarqube ecoCode), die Code Smells hinsichtlich unnötigen Energieverbrauchs anzeigen. Der große Vorteil solcher indirekter Metriken ist, dass sie aufzeigen, wo es konkretes Verbesserungspotenzial gibt. In der Regel können sie jedoch im Voraus keine Aussage treffen, wie hoch der Einspareffekt einer Optimierung ist. Auch können sie energetisch ungünstig gewählte Algorithmen oder Architekturmuster in der Regel nicht aufdecken oder Aussagen über die Effizienz von Codeteilen (z. B. 3rd-Party-Libraries) treffen, die nicht im Quellcode vorliegen.

Energiemessverfahren

Die präziseste Messung des Energieverbrauchs der IT-Ausrüstung als direkte Metrik bedient sich Hardware-Messgeräten. Mit ihrer Hilfe können Kurz- oder Langzeitmessungen im Gegensatz zu anderen Ansätzen den Energieverbrauch der Peripherie miterfassen. Unter Laborbedingungen und mit spezieller Hardware können solche Energiemessungen bis auf die Komponentenebene hinab erfolgen. Die Energiemessung mittels dedizierter Hardware ist jedoch in der täglichen Softwareentwicklung in der Breite unpraktikabel. Deswegen wird sie absehbar nur für bestimmte Zwecke eingesetzt, z. B. in der Entwicklung energiesparender Hardware selbst oder zur Proto-

kollierung des Software-Energieverbrauchs in festgelegten Szenarien bei der Zertifizierung für das Umweltzeichen »Blauer Engel«.

Generell kann nicht überbetont werden, wie wichtig die Einbindung kontinuierlicher Energiemessungen in die Softwareentwicklung ist, um das in Kapitel 2 angesprochene Monitoring umzusetzen. Konkretes, feingranulares Messen ist erforderlich, um

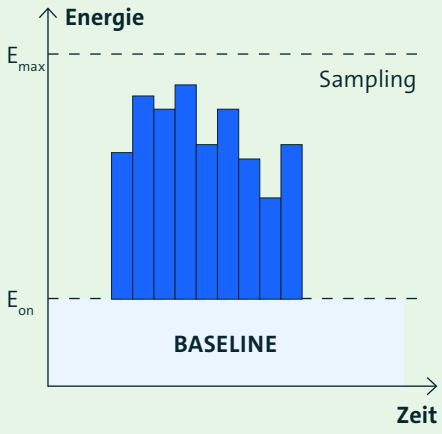
- Problemstellen aufdecken,
- Vergleiche durchführen und
- die Veränderung der Energieintensität über den Software-Lebenszyklus beurteilen zu können.

Folglich braucht es Softwarewerkzeuge, die kontinuierlich und ohne physischen Zusatzaufwand direkte Metriken zur Reduzierung des SCI generieren.

Dazu integrieren Hardware- und Betriebssystemhersteller seit mehreren Jahren zunehmend leistungsfähige Interfaces in ihre Produkte. Beispiele für solche Interfaces sind Intel RAPL oder Nvidia NVML. Auch für Hardwareplattformen, die noch keine solche Unterstützung aufweisen, existieren oft 3rd-Party-Anwendungen, die auf der Basis von Modellinformationen und aktueller Rechneraktivität eine Verbrauchsschätzung durchführen. Eine Reihe kommerzieller und Open-Source-Anwendungen ermöglichen auf dieser Grundlage auch im Entwicklungsalltag das Monitoring der Software-Energieintensität.

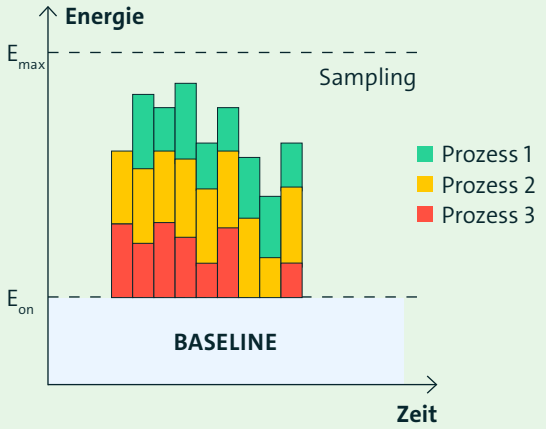
Eine Schwachstelle der derzeit verfügbaren Mechanismen ist, dass sich die Messungen oder Schätzungen nur auf einen Teil der Hardware, z. B. CPU, Speicher oder Grafikkarte beziehen. Der Energieverbrauch anderer Rechnerkomponenten und der Peripherie muss entweder anders bestimmt oder über verfügbare Verbrauchswerte extrapoliert werden.

Weiterer Schwachpunkt der vorhandenen Interfaces ist, dass sie – genau wie physikalische Messungen mittels Spezialhardware – keinen direkten Bezug zur Softwareaktivität herstellen. Stattdessen diskretisieren sie den Energieverbrauch des Rechners oder einzelner Komponenten wie CPU-Cores über kurze Zeitintervalle (»Sampling«):



Dieses Sampling von Energiewerten als Basis für eine kleinteilige Beurteilung des Energieverbrauchs von Software stößt derzeit aufgrund mangelnder Integration in die Betriebssysteme auf zwei hauptsächliche Grenzen:

- Die Sampling-Frequenz ist aus Sicherheitsgründen, d. h. um ein Ausspähen der Code-Aktivität zu verhindern, künstlich limitiert. Das behindert eine ausreichend feingranulare Analyse.
- Moderne Rechner arbeiten parallel viele verschiedene Programme ab. Aus der Energiemessung z. B. eines Cores oder ganzer CPU-Packages geht nicht hervor, welche Codeteile welchen Anteil des Energieverbrauchs verantworten. Dazu müssen derzeit mittels modellbasierter Schätzungen Informationen des Betriebssystems über Thread- oder Prozessaktivitäten mit den Energie-Messwerten in Einklang gebracht werden (s. Abbildung unten).



Dass diese Integration von Energiemessung und Softwareaktivität noch nicht hochauflösend in den Betriebssystemen unterstützt wird, verhindert präzise Energieschätzungen für kleinteilige Software-Funktionseinheiten R (z. B. die sogenannte Unit-Ebene). Dennoch erlauben Anwendungen und Frameworks wie JoularJX, Scaphandre, Green Metrics Tool, Red Hat Kepler und andere schon heute ein gutes Monitoring auf Modul-Ebene.

6 Betrieb

Die Betriebsphase des Softwareentwicklungsprozesses weist signifikante Unterschiede zur reinen Entwicklungsphase auf. Wie bereits erläutert, resultieren die Betriebskosten und der Ressourcenverbrauch im Realbetrieb aus den in der Entwicklung getroffenen Entscheidungen. Obwohl bestimmte Aspekte in der Entwicklungs- und Betriebsphase unterschiedliches Gewicht haben, ermöglichen Tests und Monitoring während der Entwicklung fundierte Prognosen zum späteren Ressourcenverbrauch. Die Empfehlungen der Betriebsphase ähneln daher oft denen der Entwicklungsphase, werden aber um neue ergänzt.

Wartbarkeit & Stabilität

Im operativen Betrieb sind Schwächen in der Wartbarkeit deutlicher als in der Entwicklungsphase zu erkennen. Obwohl Programmierer den Quellcode analysieren können, offenbart sich die tatsächliche Qualität einer Software erst durch die Nutzung, also wie häufig bestimmte Codestellen durchlaufen, wie oft Algorithmen ausgeführt und welche Datenmengen erfasst werden. Wartbarkeit zeigt sich zudem in der Systemstabilität und der Reaktionsfähigkeit der Entwicklerteams, wenn Probleme auftreten. Für die Ressourceneffizienz bedeutet das, dass Teams mit schlecht wartbarer Software kaum in der Lage sind, die Ursachen von Ressourcenverschwendung zu identifizieren oder entsprechende Gegenmaßnahmen zu ergreifen. Stattdessen sind sie oft gezwungen, ihre Ressourcen auf die Stabilisierung der Software zu konzentrieren, statt sie zu optimieren.

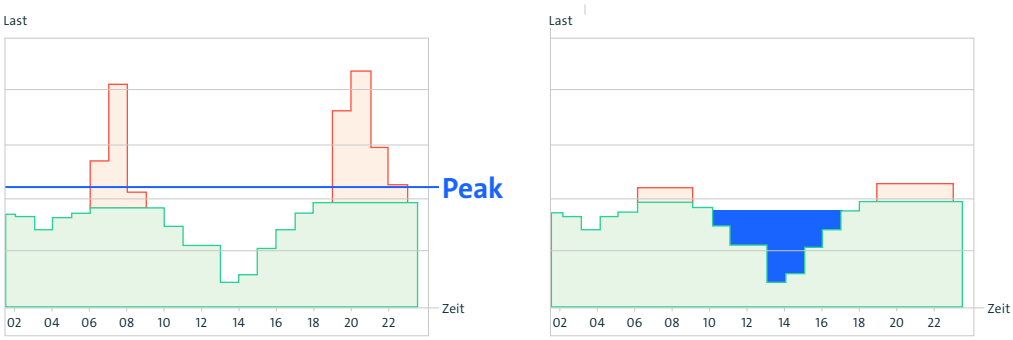
Monitoring & Auslastung

Das Monitoring des Gesamtsystems sowie die Betrachtung der Auslastung aller eingesetzten Betriebsmittel stellen die wichtigsten Werkzeuge dar, um reale Einsparpotenziale offenzulegen. Durch Konsolidierung können Server zusammengelegt und der Energiebedarf gesenkt werden. Zudem trägt die automatisierte Skalierung der Infrastruktur dazu bei, Energieverbrauch und Hardwarebedarf zu senken, indem nur die tatsächlich benötigte Kapazität genutzt wird und Ressourcen nicht unnötig überdimensioniert werden. Eine weitere wirkungsvolle Maßnahme zur Energieeinsparung sind Sleep-Modi für Server, um in Phasen geringer Nutzung Ressourcen zu sparen.

Peak-Shaving & Load-Shifting

Peak-Shaving und Load-Shifting sind strategische Ansätze, um im Softwarebereich den Ressourcenverbrauch zu optimieren und den CO2-Ausstoß zu reduzieren. Peak-Shaving minimiert Lastspitzen des Systems oder einzelner Systembestandteile durch Ansätze wie Caching, Loadbalan-

cing und dynamische Skalierung. Beim Caching werden Lasten im Backend vermieden, indem häufig abgefragte Daten aus einem schnellen Cache-Speicher bereitgestellt werden, statt bei jeder Anfrage neu berechnet zu werden. So sinkt die Rechenlast und es werden höhere Reaktionszeiten erreicht, was zugleich den Ressourcenverbrauch senkt. Loadbalancing verteilt die Last gleichmäßig auf mehrere Serverinstanzen, welche durch dynamische Skalierung auf Basis des tatsächlich notwendigen Bedarfs bereitgestellt werden.



Load-Shifting bezeichnet Verfahren, mit denen Last zeitlich oder räumlich verschoben wird. Statt aufwändige Berechnungen zu einer Zeit durchzuführen, in der das System von vielen Endnutzern verwendet wird, können diese Berechnungen ggf. auch in der Nacht vorgenommen werden, wenn das System insgesamt einer geringeren Last unterliegt. Im Kontext der Nachhaltigkeit kann dieses Vorgehen noch weitergedacht werden. Aufwändige Berechnungen können zum Beispiel in Regionen ausgelagert werden, die einen sehr hohen Anteil CO2-neutraler Energiequellen verwenden, was wiederum den CO2-Fußabdruck der Berechnungen reduziert. Ein Load-Shifting-Verfahren sollte im besten Fall bereits im Design berücksichtigt werden (↗ Band 1).

Einige erneuerbare Energiequellen wie Wind und Solar führen außerdem zeitlich begrenzt zu einem sehr hohen Angebot von Energie, welches durch Stromnetz und Verbraucher ausgeglichen werden muss. Das senkt temporär den Strompreis, was durch geschicktes Load-Shifting ausgenutzt werden kann, um die Betriebskosten zu senken[10].

Durch die Kombination dieser Ansätze können Unternehmen sowohl finanzielle Vorteile nutzen als auch ihren ökologischen Fußabdruck verringern. Dafür muss jedoch bereits in der Entwurfsphase ein entsprechend dynamisches System entworfen werden. Weiterhin lassen sich die Verfahren meist eher im Rahmen einer Cloud-Strategie realisieren, da sie die dafür notwendigen Grundvoraussetzungen bietet.

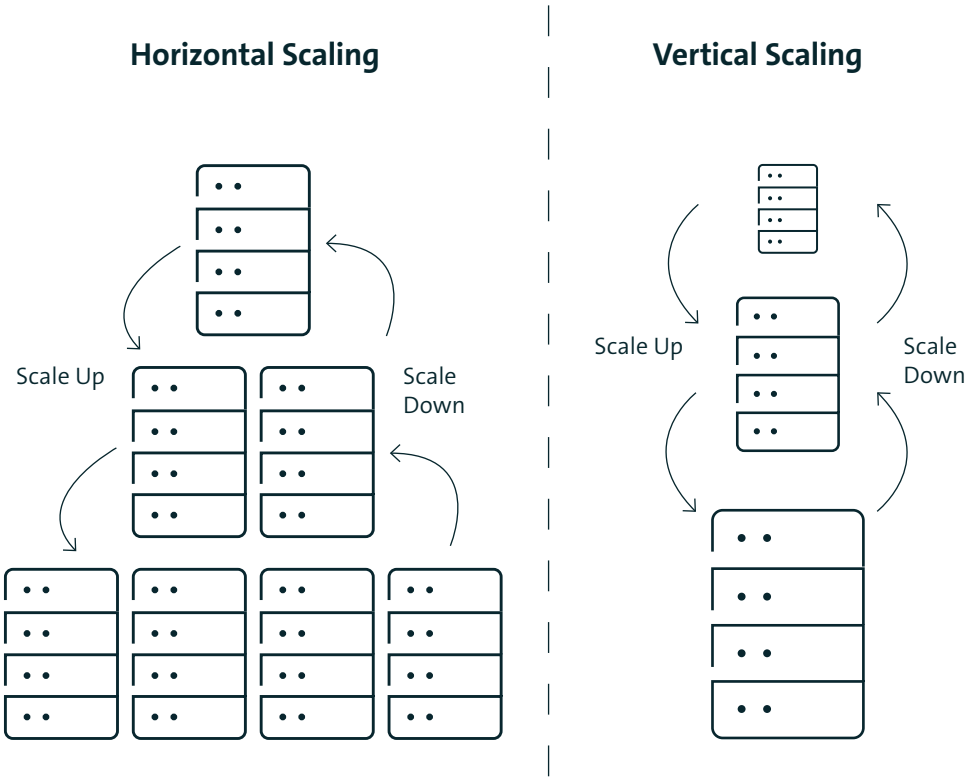
Skalierbarkeit & Ausfallsicherheit

Wie sich im vorherigen Kapitel gezeigt hat, kommt der Skalierung von Softwaresystemen eine zentrale Rolle bei der Ressourceneffizienz zu. Eine gezielte Skalierungsstrategie ermöglicht es, die

10 ↗ https://www.sciencedirect.com/science/article/pii/S0306261923019268

benötigten Ressourcen an die tatsächliche Nutzung anzupassen, wodurch die Auslastung optimiert werden kann.

Hierbei wird zwischen vertikaler und horizontaler Skalierung unterschieden: Während bei vertikaler Skalierung die Kapazität bestehender Systeme erhöht wird, werden bei horizontaler Skalierung zusätzliche Systeme hinzugefügt. Beide Ansätze sollten sorgfältig geprüft werden, um Energieverbrauch und Hardwareanforderungen im Gleichgewicht zu halten.



Wichtig ist auch der Umgang mit Reservekapazitäten. In vielen Systemen werden Ressourcen vorgehalten, um Lastspitzen abzudecken oder unerwartete Erhöhungen im Datenaufkommen zu bewältigen. Diese Reserven können jedoch zu ineffizienter Ressourcennutzung führen, wenn sie dauerhaft bereitgestellt, aber selten genutzt werden. Der Einsatz dynamischer, bedarfsgerechter Skalierungsmechanismen – beispielsweise durch Cloud-Services mit automatischer Anpassung der Ressourcen – kann dabei helfen, diesen Überprovisionierungseffekt zu minimieren. Eine bedarfsorientierte Skalierung reduziert nicht nur die Betriebskosten, sondern auch den ökologischen Fußabdruck, indem der Energiebedarf der Infrastruktur auf das notwendige Minimum beschränkt wird.

Skalierung und Reserve sollten deswegen hinsichtlich ihrer Nachhaltigkeit betrachtet werden. Die genaue Analyse der Nutzungsmuster und eine darauf basierende Vorhersage zukünftiger Anforderungen ermöglichen eine präzise Ressourcenplanung. Durch die Vermeidung von Überprovisionierung und die intelligente Nutzung von Cloud-Technologien lässt sich sowohl die Effizienz steigern als auch der Ressourcenverbrauch nachhaltig senken.

Datensparsamkeit & Bereinigung

Wie auch in früheren Phasen des Entwicklungsprozesses spielt der Umgang mit Daten eine entscheidende Rolle für die Ressourceneffizienz. Das umfasst die regelmäßige Datenbereinigung, bei der überflüssige Daten eliminiert werden und dadurch Speicherplatz freigegeben wird. Sie führt nicht nur zu einer Reduktion des Speicherbedarfs, sondern auch zu einem geringeren Energieverbrauch der Speicherlösungen. Datenkomprimierung ist ebenfalls ein wichtiger Faktor, da durch sie der benötigte Speicherplatz und somit der Ressourcenverbrauch verringert wird.

Besonders hervorzuheben sind sogenannte *Dark Data*. Diese unstrukturierten und häufig ungenutzten Daten werden zwar erfasst und gespeichert, aber selten weiter verwendet. Dazu gehören neben Klassikern wie Spam-E-Mails auch Sensordaten von Hardwarekomponenten, Logdateien und ähnlich groß angelegte Datenbestände. Die Bereinigung dieser Dark Data birgt neben Ressourceneinsparungen auch Sicherheitsvorteile, da potenziell sensitive Informationen enthalten sein können, die möglichen Angreifern tiefe Einblicke in die Systeme gewähren können.

Procurement

Das Procurement beschreibt die operative Beschaffung von Betriebsmitteln. Hierbei sollten Produkte und Dienstleistungen von Anbietern, die selbst nachhaltige Prinzipien einhalten, bevorzugt werden. Das ist insbesondere in Bezug auf Berichtspflichten zur Nachhaltigkeit eines Digitalprodukts wichtig, wie sie beispielsweise von der Europäischen Union im Rahmen der Corporate Sustainability Reporting Directive (CSRD[11]) gefordert werden. Diese gewinnt in den letzten Jahren vor allem in Unternehmen mit mehr als 250 Angestellten an Bedeutung.

Zudem kann eine Lebenszyklusanalyse der Software durchgeführt werden, um die Umweltauswirkungen im gesamten Lebenszyklus zu bewerten und die relevanten Stellen zu optimieren. Diese Analyse ist auch Voraussetzung für Siegel wie den Blauen Engel[12]. Auch die Kooperation mit anderen Unternehmen und Organisationen ist wichtig, um gemeinsam nachhaltige Lösungen zu entwickeln und voneinander zu lernen.

11 ↗ https://eur-lex.europa.eu/legal-content/en/TXT/?uri=CELEX:32023R2772
12 ↗ https://www.blauer-engel.de/de/produktwelt/software

Mögliche Maßnahmen und Ansätze

Im Folgenden werden Maßnahmen und Ansätze aufgeführt, die darauf abzielen, den Ressourcenverbrauch zu reduzieren. Es wird empfohlen, diese sowohl in der Konzeptarbeit als auch im technischen Entwurf zu berücksichtigen, damit diese ihre volle Wirkung entfalten. (↗ Band 1)

Planung

Capacity Planning: Der Ressourceneinsatz der Infrastruktur sollte gezielt geplant werden. Dafür eignen sich etwa historische Daten, um Lastspitzen zu prognostizieren.

Gezielte Testplanung: Auch im Betrieb ergibt es Sinn, Tests durchzuführen, um Rückschlüsse auf das Verhalten des Gesamtsystems zu ziehen und frühzeitig seine realen Belastungsgrenzen zu erkennen.

Standortwahl: Je nach Standort stehen unterschiedlich viele erneuerbare Energien im Energiemix zur Verfügung. Hinzu kommt, dass der Standort des Rechenzentrums auch die Entfernung zu den Endgeräten der User verkürzen kann.

Monitoring

Log-Analyse: Log-Analyse-Tools sollten regelmäßig genutzt werden, um Fehler und ungewöhnliche Ereignisse zu identifizieren.

Performance Monitoring: Zumindest aufwändige und wichtige Prozesse sollten kontinuierlich überwacht werden, um Engpässe und Leistungseinbrüche gezielt und frühzeitig zu erkennen.

Optimierung

Autoscaling: Wenn nötig, sollte sich das Softwaresystem dynamisch an schwankende Lasten anpassen können.

Caching: Häufig abgerufene Daten sollten in einem Cache nahe der Abrufenden gespeichert werden, um die Belastung des Backends und den Datentransport zu reduzieren.

Cloud: Im Vergleich zu typischen Server-Lösungen erlauben es Cloud-Lösungen, den Betrieb eines Softwaresystems zu flexibilisieren und Ressourcen gezielter zu nutzen.

Daten bereinigen: Geringere Datenbestände erhöhen die Effizienz der Datenverarbeitung. Das optimiert die Hardwareauslastung, die Betriebskosten und die Gesamtleistung des Systems. Daher sollten Datenbestände regelmäßig von Altlasten befreit und Archivierungsverfahren genutzt werden.

Edge Computing: Daten sollten möglichst nah an ihrem Entstehungsort vorverarbeitet werden, um nur die Datenmenge zu übertragen und zentral zu speichern, die wirklich langfristig benötigt wird.

Energieeffiziente Kühlung: Zentralisierte und optimierte Kühltechniken können den Energieverbrauch senken und Abwärme – zum Beispiel als Fernwärme – weiter verwenden.

Bedarfsgerechte Auslastung: Rechenzentren und die Cloud können dank Virtualisierung Hardware-Ressourcen besser auslasten.

Redundanz: Redundante Systeme mit einer heißen Reserve sollten nur da aufgebaut werden, wo Ausfälle unbedingt abgefangen werden müssen.

Zeitversetzte Verarbeitung: Nicht zeitkritische Aufgaben sollten asynchron und außerhalb von Spitzenlasten verarbeitet werden.

Lebenszyklus

Software-Updates: Nachträgliche Updates können die Ressourcennutzung optimieren, da sie Veränderungen der Laufzeitumgebung berücksichtigen.

Langlebigkeit: Neu entwickelte Software startet mit einem negativen CO2-Fußabdruck, der während der Entwicklungsphase entsteht. Je länger eine Software genutzt wird, desto mehr verteilt sich der CO2-Fußabdruck auf die gesamte Lebensdauer. Dadurch sinkt er relativ pro Nutzungsjahr.

Abwärtskompatibilität: Ist eine Software abwärtskompatibel, verträgt sie sich auch mit älteren Versionen. Das erhöht die Lebenszeit der Software und vermeidet vorzeitige Obsoleszenz bei der verbundenen Hardware.

Hardwareauswahl und -Nutzung

Energieeffiziente Hardware: Bei der Auswahl von Hardwarekomponenten wie Rechnern, Routern etc. gibt es deutliche Unterschiede in der Energieeffizienz, die beim Einkauf berücksichtigt werden sollten.

Auslastung optimieren: Eine hohe Auslastung der Hardware reduziert den Ressourcenverbrauch pro verarbeiteter Einheit.

Initiale Emissionen berücksichtigen: Nicht nur die Nutzung, sondern auch die Herstellung von Hardware verursacht CO2-Emissionen. Diese initialen Emissionen sollten in der Beschaffung berücksichtigt werden.

Arbeitsgestaltung

Automatisierung: Wiederkehrende Aufgaben wie Tests, Deployments und Monitoring sollten automatisiert werden. Das schafft gegebenenfalls Freiräume für andere Tätigkeiten und erlaubt umfangreichere Untersuchungen.

Beschaffung: Berücksichtigt man Nachhaltigkeit in der Beschaffung, können Einsparungen erreicht werden, zum Beispiel durch die Externalisierung von Aufwänden an Zulieferer.

DevOps: Durch eine enge Zusammenarbeit zwischen Entwicklung und Betrieb können sich Erkenntnisse verbessern und schneller angewendet werden.

Versionierung: Versionskontrollsysteme können auch im Betrieb genutzt werden, um Änderungen nachzuvollziehen und bei Bedarf Rollbacks zu ermöglichen.

7 Zusammenfassung

Nachhaltigkeit und Ressourceneffizienz sind essenzielle, aber auch komplexe Themen. In der Softwareentwicklung ist noch viel Grundlagenforschung erforderlich, um wissenschaftlich fundierte Empfehlungen zu geben. Dennoch können viele etablierte Best Practices aus Software-Entwurf, -Entwicklung und -Betrieb auch im Kontext ökologischer Ziele bereits heute angewendet werden. Durch ökologische Ziele ergeben sich neue Perspektiven. Dadurch lassen sich neue Einsparpotenziale aufdecken, die nicht nur ökologisch, sondern auch ökonomisch sinnvoll sein können.

Dabei ist es wichtig, Entscheidungen im Kontext zu sehen und zu analysieren, wie sie sich auf den Verbrauch bei der Übertragung, Verarbeitung und Speicherung von Daten auswirken. Es muss außerdem bedacht werden, dass die Entwicklung von Software oft weniger Ressourcen verbraucht als ihr Betrieb. Die während der Konzeption und Umsetzung getroffenen Entscheidungen werden also durch die späteren Anwender multipliziert.

Die Softwareentwicklung allein kann aber nur eingeschränkt nachhaltig wirken. Vielmehr braucht es eine allgemeine Nachhaltigkeitsstrategie der Business-Architektur, an der sich alle beteiligten Personen orientieren können.

Bitkom vertritt mehr als 2.200 Mitgliedsunternehmen aus der digitalen Wirtschaft. Sie generieren in Deutschland gut 200 Milliarden Euro Umsatz mit digitalen Technologien und Lösungen und beschäftigen mehr als 2 Millionen Menschen. Zu den Mitgliedern zählen mehr als 1.000 Mittelständler, über 500 Startups und nahezu alle Global Player. Sie bieten Software, IT-Services, Telekommunikations- oder Internetdienste an, stellen Geräte und Bauteile her, sind im Bereich der digitalen Medien tätig, kreieren Content, bieten Plattformen an oder sind in anderer Weise Teil der digitalen Wirtschaft. 82 Prozent der im Bitkom engagierten Unternehmen haben ihren Hauptsitz in Deutschland, weitere 8 Prozent kommen aus dem restlichen Europa und 7 Prozent aus den USA. 3 Prozent stammen aus anderen Regionen der Welt. Bitkom fördert und treibt die digitale Transformation der deutschen Wirtschaft und setzt sich für eine breite gesellschaftliche Teilhabe an den digitalen Entwicklungen ein. Ziel ist es, Deutschland zu einem leistungsfähigen und souveränen Digitalstandort zu machen.

Bitkom e.V.

Albrechtstraße 10
10117 Berlin
T 030 27576-0
bitkom@bitkom.org

bitkom.org

bitkom