kOSbunEu

Entwicklung und Untersuchung einer **k**onsequent **O**pen-**S**ource **b**asierten **u**nd **n**achhaltigen **E**ntwicklungs**u**mgebung in Anlehnung an die Infrastruktur der Informatikstudiengänge der Hochschule Bremerhaven

Lamis Aiche, Franjo Gießel, Laura Halbeck, Julian Nehring, Silja Rehmke, Jan Ole Seutter, Lennart Steffen, Alexey Spitsyn, Maxim Ziegler und Oliver Radfelder

Hochschule Bremerhaven 1975 – 2025

Zielsetzung

Im Informatik- und Wirtschaftsinformatikstudium in Bremerhaven werden Studierende von Anfang an in eine eigene Linux-Server-Infrastruktur eingeführt. Diese Infrastruktur ist über SSH für alle Studierenden zugänglich und soll einen nachhaltigen und fertig-konfigurierten sowie gemeinsam nutzbaren Unix-Workspace bieten. Das Ziel besteht darin, einen niedrigschwelligen Zugang zu einer Lehr- und Lernumgebung zu schaffen, die von allen frei genutzt werden kann.

Ziel des Bachelor-Projekts ist es, zu hinterfragen, ob die gegebene Infrastruktur wirklich nachhaltiger ist, als ein Setup, in dem auf jedem Endgerät ein eigener Workspace mit Entwicklungsumgebung und Webserver genutzt würde.

Im Rahmen des Studiums erfolgt die Einführung in den Umgang mit dem Terminaleditor Vim und VSCodium, einen browserbasierten, erweiterbaren Open-Source-Texteditor, der u.A. in GitHub-codespaces oder gitlab verwendet wird.

Für das Projekt haben wir uns dafür entschieden, einen Teilaspekt dieser Infrastruktur zu untersuchen. Im Speziellen analysieren wir dabei die Unterschiede zwischen VSCodium mit und ohne Erweiterungen sowie Vim, insbesondere im Hinblick auf deren Energieeffizienz und Nachhaltigkeit.

Ausgehend von der oben formulierten Arbeitshypothese, messen wir den Energiebedarf des System-under-Test (SuT).

Darüberhinaus haben wir uns weitergehend mit Nachhaltigkeit in der IT, Open-Source und Green Coding auseinandergesetzt.

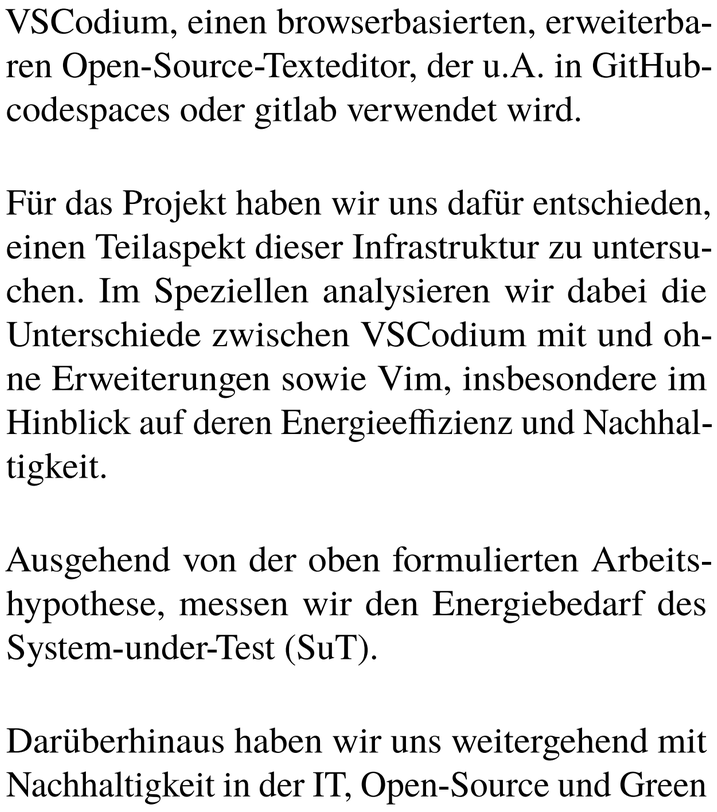

Abbildung 1: Vereinfachte Darstellung der Test-Infrastruktur

Aufbau der Test-Infrastruktur

Die Test-Infrastruktur besteht aus einem Automation- und einem Workspace-Server mit jeweils dazugehörigen Containern, siehe Abbildung 1.

Der Automation-Server steuert über die Automation-Container die systematische Entwicklung der Webanwendung. Diese Container kommunizieren dabei mit den Workspace-Containern über ein privates Netzwerk.

Der Workspace-Container ist eine Entwicklungsumgebung mit Texteditor, Compiler, Web- und Tomcat-Server sowie reverse Proxy. Der Energiebedarf des Workspace-Servers wird mittels eines Messgeräts ermittelt und in einer Datenbank protokolliert.

Tabelle 1: Testfälle

Fall	System		Workspace		Editor		
	Server	Laptop	remote	local	vim	codium light	codium full
1	✓		✓		✓		
2	✓		✓			✓	
3	✓		✓				✓
4		✓	✓		✓		
5		✓	✓			✓	
6		✓	✓				✓
7		✓		✓	✓		
8		✓		✓		✓	
9		✓		✓			✓

Methodik

Zur Evaluierung der Energieeffizienz werden unterschiedliche Nutzungsszenarien entsprechend der Tabelle 1 definiert und getestet, um einen Vergleich des Energiebedarfs zwischen lokalen und serverbasiertem Arbeiten zu ermittlen.

Für die folgenden Tests werden Geräte mit folgenden Sezifikationen genutzt:

- Workspace-Server: Intel Xeon Silver 4214 @ 12x2.20 GHz, 64 GB RAM, 2x 500 GB HDD
- Laptop: Intel Core i5-6300U @ 2x2.40 GHz, 8 GB RAM, 256 GB SSD

Damit in den genannten Nutzungsszenarien vergleichbare Entwicklungsprozesse abgebildet werden können, wurde die Entwicklung einer Web-App automatisiert. Hierfür wurde eine VSCodium-Extension und, für das Terminal, ein entsprechendes Programm in Python entwickelt.

Testdurchführung

Der Entwicklungsprozess soll möglichst realitätsnah abgebildet werden, mit Denkpausen, Fehlern, wiederholten Compile-Aufrufen und dem Deployment auf einem Tomcat-Server. Es wird somit die Arbeit von Studierenden mit Vim und VSCodium auf dem Laptop und dem Server simuliert.

Durch anwendungsspezifische Erweiterungen kann VSCodium von einem Texteditor in der Anwendung und Funktionsumfang vergleichbar zu einer vollständigen IDE aufgewertet werden. Hierzu gehört vor allem die Verwendung eines Language-Servers. In der Testdurchführung wird zwischen einem VSCodium ohne und mit Erweiterungen unterschieden, ersteres als VSCodium Light, letzteres als VSCodium Full bezeichnet.

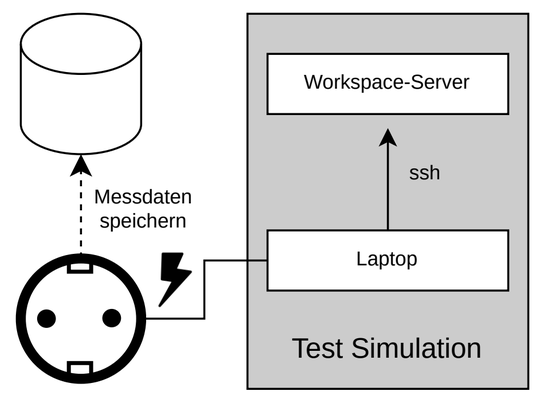

Abbildung 2: Testaufbau der Laptopmessung

Im ersten Aufbau, siehe Abbildung 2, wird der Energiebedarf des Laptops gemessen, dieser ist an einem Messgerät angeschlossen. Eine direkte Verbindung zum Workspace-Server vom Laptop wird nur in den Testfällen 4 bis 6 hergestellt.

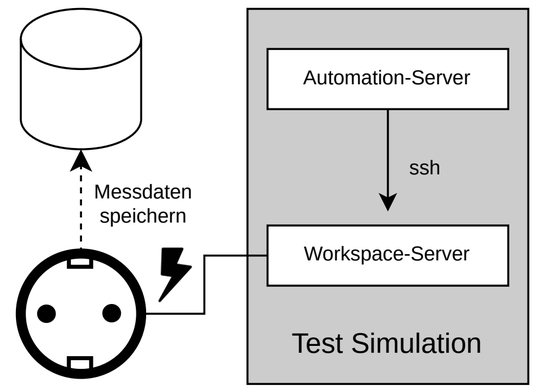

Abbildung 3: Testaufbau der Servermessung

Im zweiten Aufbau, siehe Abbildung 3, wird der an das Messgerät angeschlossene Workspace-Server gemessen.

In den Testfällen 1 bis 3 startet der Automation-Server 16 Automation-Container, denen jeweils ein Container auf dem Workspace-Server zugeordnet ist. Der Test wird im Automation-Container gestartet, dann wird eine SSH-Verbindung zwischen Automation- und Workspace-Container aufgebaut. Danach führt der Automation-Container den Test systematisch auf dem Workspace-Container aus.

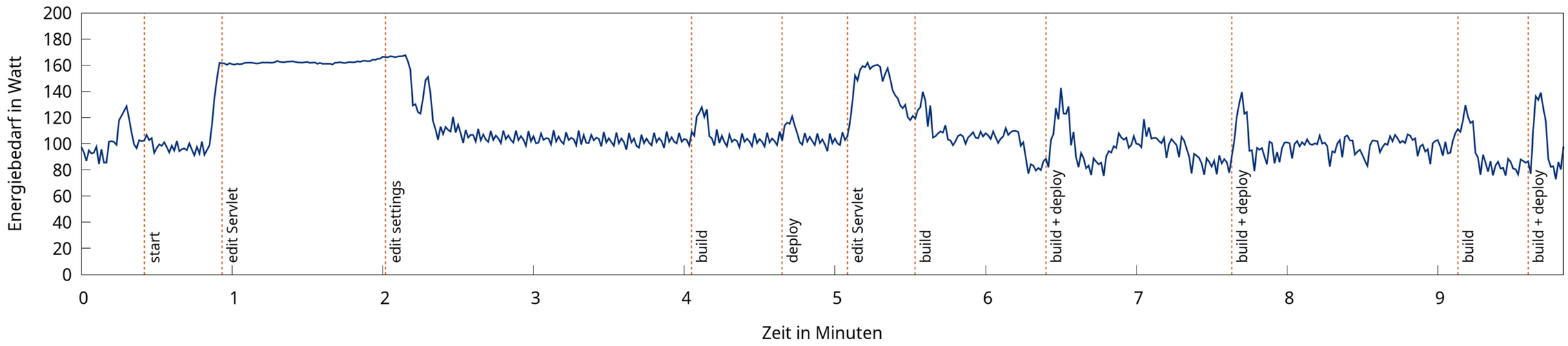

Abbildung 4: Energiebedarf des Workspace-Servers bei 16 Containern mit VSCodium Full (Fall 3)

Ergebnisse

Für die Auswertung der Testergebnisse wird beispielhaft der Graph von Testfall 3 erläutert. Außerdem wird der durchschnittliche Energiebedarf der unterschiedlichen Testfälle verglichen.

Der Graph (siehe Abb. 4) beschreibt den Energiebedarf vom Workspace-Server bei 16 parallel laufenden VSCodium Full-Tests. Die erste (Minute eins) und die zweitgrößte Verbrauchsspitze (Minute fünf) sind auf den Language-Server zurückzuführen. Bei Minute eins werden durch das Öffnen der ersten Java-Dateien die Language-Server initialisiert. Bei Minute fünf werden die Java-Dateien geöffnet und analysiert.

In der fünften Minute werden nacheinander die Compile-Aufrufe und Deployments getrennt ausgeführt. In der siebten Minute gibt es eine erneute Verbrauchsspitze, hier finden simultane Compile-Aufrufe und Deployments in allen 16 Workspace-Container statt.

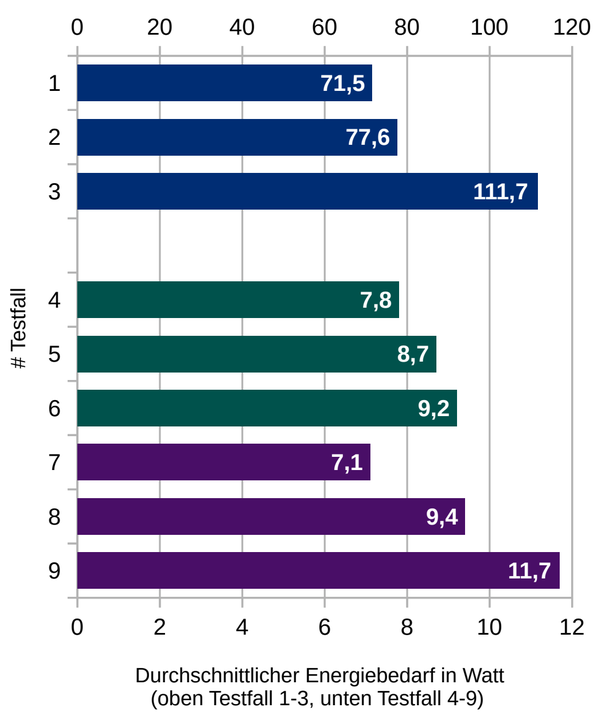

Abbildung 5: Durchschnittlicher Energiebedarf der Testfälle

Nach Durchführung der ersten Testläufe zeigen sich bereits deutliche Unterschiede in den Messergebnissen. Die Werte sind in der oben dargestellten Grafik aufgeführt und unterliegen einer Messungenauigkeit von $\pm 2\%$.

Beim Vergleich von Vim und VSCodium wird deutlich, dass die Vim-Tests (Fall 1, 4 und 7) weniger Energie benötigen als die VSCodium Light-Tests (Fall 2, 5 und 8). Im Durchschnitt sind diese etwa 18% energieeffizienter als die entsprechenden VSCodium Light-Tests (Fall 2, 5 und 8).

Noch auffälliger wird der Unterschied, wenn VSCodium Light (Fall 2, 5 und 8) mit VSCodium Full (Fall 3, 6 und 9) verglichen wird, dabei liegt der Unterschied im Durchschnitt bei etwa 25%.

Deutung

Ausgehend von der Arbeitshypothese, eines geteilten Workspace-Servers als nachhaltigere Alternative zu Endgeräten mit individuellem Workspace, lassen sich folgende konkrete Fragestellung ableiten:

1. Wie viel Energie wird auf einem Laptop eingespart, wenn der Workspace auf einem Server liegt?

2. Welcher Energiebedarf entsteht auf dem Server im Vergleich zur eingesparten Energie auf dem Laptop?

Für die Beantwortung der ersten Fragestellung wird der Energiebedarf vom Laptop bei Remote-Tests den lokalen Tests gegenüber gestellt. Bei der Betrachtung der Testfälle 5 und 8 sowie 6 und 9 bestätigt sich die Annahme, dass lokales Arbeiten auf dem Laptop einen höheren Energiebedarf zufolge hat. Werden hingegen die Tests 4 und 7 verglichen zeigt sich ein unerwartetes Ergebnis. Scheinbar ist der Energiebedarf des Laptops beim Vim Remote-Test niedriger als beim lokalen Test, obwohl die Compile-Aufrufe und der Webserver remote laufen. Die Begründung hierfür bedarf weiterer Untersuchungen.

Für die Beantwortung der zweiten Fragestellung wird der durchschnittliche Energiebedarf eines Remote-Workspace-Containers $\varnothing P_{Remote}$ mit der eingesparten Energie auf dem Laptop $\varnothing P_{Lokal}$ verglichen. $\varnothing P_{Remote}$ ist die Differenz aus dem Gesamt- und Grundverbrauch des Servers auf einen einzelnen Container heruntergerechnet. $\varnothing P_{Local}$ ist die Differenz aus einem lokalen und dem entsprechenden remote Test.

Der ermittelte Grundverbrauch, der Energiebedarf wenn keine Tests durchgeführt werden, beträgt auf dem Server $63,3W$ und auf dem Laptop $5,0W$.

$$\begin{aligned} \varnothing P_{Remote} &= \frac{(\varnothing P_{Fall3} - \varnothing P_{Idle})}{16} \\ &= \frac{(111,7W - 63,3W)}{16} \\ &\approx 3,0W \end{aligned} \tag{1}$$

$$\begin{aligned} \varnothing P_{Local} &= \varnothing P_{Fall9} - \varnothing P_{Fall6} \\ &= 11,7W - 9,2W \\ &= 2,5W \end{aligned} \tag{2}$$

Aus der Berechnung für Fall 3 ergibt sich ein Energiebedarf von $3,0W$ pro Workspace-Container. Das entspricht ungefähr der Energie von $2,5W$, die auf dem Laptop eingespart wird. Das Verschieben der Last auf den Workspace-Server erzeugt somit einen nicht signifikant höheren Ressourcenverbrauch.

Grenzen der Deutung

Bei der Analyse der Testergebnisse sind Einschränkungen zu berücksichtigen, die sich durch den bisherigen Testaufbau ergeben.

Bis zum aktuellen Zeitpunkt wurden lediglich wenige Tests durchgeführt, wodurch die Aussagekraft der Ergebnisse eingeschränkt ist.

Darüber hinaus wurde mit lediglich 16 Workspace-Containern nur eine begrenzte Anzahl von Nutzer:innen gleichzeitig simuliert. Bei nur wenigen Nutzer:innen ist der Grundverbrauch des Servers mit $63,3W$ im Vergleich zum Verbrauch der einzelnen Container relativ hoch. Diesen Grundverbrauch nicht mitzuberechnen, ist eine Vereinfachung des Testvorgangs, die im weiteren Verlauf überdacht werden muss.

Die im Testdurchlauf entwickelte Web-App umfasst lediglich ein Servlet, einen Filter und einen Listener und besteht somit aus einer kleinen Anzahl an Grundkomponenten. Daraus ergibt sich die Fragestellung, ob der Energiebedarf eines Language-Servers bei einer größeren Anwendung ansteigt, da die Anzahl der zu analysierenden Dateien wächst.

Ausblick

Nach ersten Erkenntnissen tragen die untersuchten Aspekte der Informatik-Infrastruktur zu einer nachhaltigen Praxis bei. Diese Schlussfolgerung basiert auf den durchgeführten Messungen und deren Auswertung.

Das Verschieben der Rechenlast von einem Laptop auf einen Server wirkt sich positiv auf den Energiebedarf des Laptops aus. Dadurch können die Anforderungen an die Leistungsfähigkeit der Endgeräte reduziert und die tatsächliche Akkulaufzeit verlängert werden, was eine Verlängerung des Lebenszyklus älterer Laptops und Akkus ermöglicht und somit zur Ressourcenschonung beiträgt.

Die Effizienz des Servers hängt von der Anzahl der zeitgleichen Nutzer:innen ab. Insbesondere dann, wenn ein Server ausgelastet ist, ist die Energieeffizienz hoch.

Es wäre daher sinnvoll, die Tests mit einer erhöhten Anzahl simulierter Nutzer:innen zu wiederholen. Um die Aussagekraft der Testergebnisse zu steigern, sollte zudem die Gesamtzahl der durchgeführten Tests erhöht werden. Dies würde ermöglichen, Mittelwerte aus einer größeren Datenbasis zu berechnen und damit zuverlässigere Schlussfolgerungen zu ziehen.

Nachhaltiges Handeln in der Informatik

Nachhaltigkeitsaspekte und Open-Source sind definierte Schlüsselkonzepte des Projekts. Eine Gelegenheit, diese zu vertiefen, bot die FIfF-Kon2024, die im Oktober 2024 in Bremerhaven tagte und bei der wir uns mit einem Workshop beteiligt haben. Insbesondere durch die Diskussionen und Reflexionen im und rund um den Workshop haben wir uns zudem intensiv mit dem Thema *Green Coding* auseinandergesetzt.

In der Informatik-Infrastruktur wird, wenn möglich und sinnvoll, gebrauchte und refurbished Hardware eingesetzt. Um über diesen Aspekt fundierte Aussagen treffen zu können, haben wir uns in einer ausführlichen Literatur-Recherche mit dem Stand der wissenschaftlichen Forschung zum Thema refurbished und gebrauchter Hardware im Endgeräte- und Server-Bereich befasst.

Open-Source und FLOSS

Es gibt zahlreiche Gründe, insbesondere im Hochschulkontext, ausschließlich auf Open-Source-Technologie zu setzen. Für das vorliegende Projekt ist Open-Source im Wesentlichen ein Mittel, um nachhaltige Konzepte zu realisieren.

Wir verstehen *Nachhaltigkeit* als ein vielschichtiges, mehrdimensionales Konzept. Daher diskutieren wir Nachhaltigkeit und Open-Source nicht exklusiv im Kontext von ökologischen, sondern auch sozialen Gesichtspunkten.

Das Paradigma hinter Free/Libre and Open-Source Software (FLOSS) basiert auf den Prinzipien Offenheit, Zusammenarbeit und Gemeingutbildung. FLOSS versteht Software als ein öffentliches Gut, das allen zugänglich sein sollte, um Innovationen zu fördern und technologische Abhängigkeiten zu verringern.

Die Open-Source Initiative (2024) definiert FLOSS als Software, deren Quellcode einsehbar, veränderbar und weiter verteilbar ist [1].

Es gibt gute Belege dafür, dass FLOSS einen wichtigen Beitrag zu ökologischerem Technolgieeinsatz beiträgt [2]. Exemplarisch hervorzuheben ist hier das Betriebssystem *Linux*, womit z.B. die Lebensdauer von Hardware erheblich verlängert werden kann. Unter, im weiteren Sinne, sozialen und wirtschaftlichen Gesichtspunkten ist FLOSS ein Instrument, das technologische Abhängigkeiten reduziert und die digitale Souveränität stärkt.

Refurbished Hardware

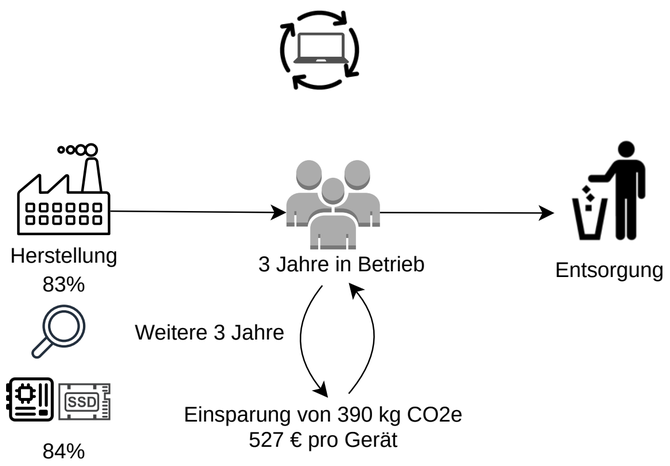

Abbildung 6: Lebenszyklus von Endgeräten

Ein großer Hebel für die nachhaltige Konfiguration einer IT-Infrastruktur ist die richtige Auswahl der Hardware. Für den vorliegenden Anwendungszweck (HS-Informatik-Infrastruktur) argumentieren wir für gebrauchte, bzw. refurbished Hardware, sowohl für End- als auch Server-Geräte.

Der Lebenszyklus von Hardware lässt sich in drei Phasen einteilen: Herstellung, Betrieb und Entsorgung. Insbesondere bei Endgeräten fällt der größte Teil der CO2-äquivalenten Emissionen (CO2e) in der Herstellungsphase und dort überwiegend für die Produktion der mikroelektronischen Komponenten an [3]. Die Verlängerung der Lebenszyklen eines Gerätes reduziert somit nicht nur die Investitionskosten, sondern insbesondere auch das anfallenden CO2e.

Für Server muss diese Bilanz differenzierter betrachtet werden, da hier weitere Aspekte wie Sekundärnutzung der Abwärme, Auslastung, Anwendungszweck etc. berücksichtigt werden müssen. Allerdings zeigen Bashroush et. al. [4], dass der Einsatz von refurbished Servern sich nicht nur aus ökologischer, sondern auch aus wirtschaftlicher Sicht lohnt. Das zugrunde liegende Kalkül ist hier, dass das historische Wachstum der Energieeffizienz durch Moore's Law in den letzten Jahren deutlich verlangsamt wurde.

Die Autoren analysieren verschiedene Szenarien zur Erneuerung von Servern und zeigen, dass der Austausch von Servern, die älter sind als 5 Jahre durch refurbished Geräte wirtschaftlich und ökologisch sinnvoll ist. Für jüngere Server empfiehlt sich eine gezielte Optimierung, wie z.B. durch Speicheraufrüstung.

Green Coding

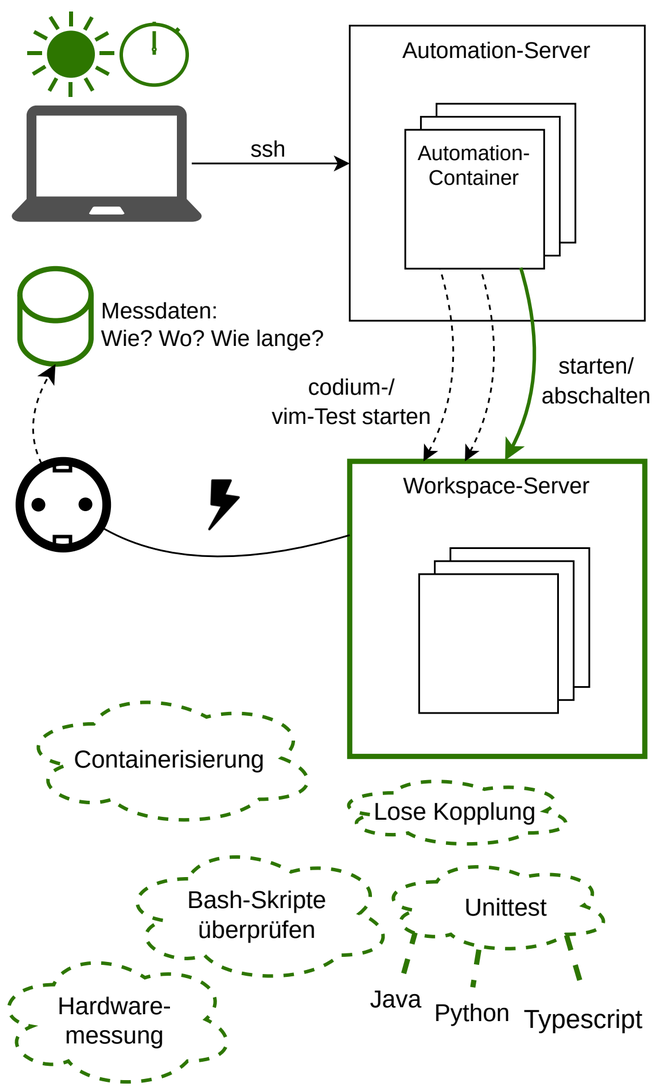

Abbildung 7: Green Coding Aspekte in unserer Architektur

Green Coding bezeichnet eine Herangehensweise in der Softwareentwicklung, die darauf abzielt, den Energieverbrauch und die Umweltbelastung von Software zu minimieren. Dies beinhaltet die Optimierung von Code, Algorithmen und Softwarearchitekturen, um den Ressourcenverbrauch, wie Prozessorleistung, Speicherplatz und Netzwerkbandbreite, zu reduzieren [5]. Darüber hinaus beinhaltet Green Coding eine bewusste Praxis und Haltung in Bezug auf den alltäglichen Entwicklungsprozess in der IT.

Ausgehend von den Diskussionen um die FIfFKon2024 und auf Grundlage des Booklets „Green Coding", herausgegeben vom BMUV, haben wir das vorliegende Projekt als Untersuchungsgegenstand herangezogen, um dieses auf Green-Coding-Aspekte zu überprüfen oder sie ggf. zu implementieren.

In der Green-Coding-Praxis lassen sich zwei Kategorien von Maßnahmen unterscheiden: Erstens, solche, die den Entwicklungsprozess und das Umfeld betreffen und zweitens jene, die darauf abzielen qualitativ hochwertige und leistungseffiziente Software zu entwickeln.

Unter die handlungsorientierten Ansätze fallen:

1. *Verbrauch sichtbar machen:*
   Der Verbrauch von Ressourcen wie CPU, RAM und Energie wird durch Monitoring-Tools analysiert und visualisiert, um Optimierungspotenziale zu erkennen und die Effizienz zu steigern.
2. *Dokumentation für bessere Wartbarkeit:*
   Eine klare und umfassende Dokumentation der Architektur, Abhängigkeiten und Prozesse sichert die langfristige Wartbarkeit und erleichtert die Weiterentwicklung der Anwendung.
3. *Schulung der Mitwirkenden:*
   Weiterbildung durch Fachliteratur, Online-Kurse und interne Schulungen, um nachhaltige Software und effiziente Prozesse zu implementieren.
4. *Nutzungszeiten definieren:*
   Systeme werden bei Bedarf hoch- oder heruntergefahren, um Leerlaufzeiten zu vermeiden und die Ressourcen optimal zu nutzen.
5. *Lastspitzen auf „grüne" Tageszeiten legen:*
   Es gibt Zeitfenster, in denen erneuerbare Energien regelmäßig besser verfügbar sind (z.B. Testen zur Mittagszeit).
6. *Realen Verbrauch der Test-Infrastruktur messen:*
   Der Ressourcenverbrauch von Testsystemen wird erfasst und dokumentiert, um einen Überblick über den realen Verbrauch zu erhalten.
7. *Daten-Speicherung hinterfragen:*
   Es wird geprüft, welche Daten gespeichert werden müssen, in welchem Format und wie lange sie aufbewahrt werden sollten.
8. *Modular entwickeln, modular testen:*
   Die Softwarearchitektur wird modular aufgebaut und Komponenten können und werden einzeln getestet.
9. *Code-Reviews:*
   Code wird optimiert, z.B. durch das Erkennen von ineffizienten Algorithmen, *Tree Shaking* oder Peer-Reviews.

Hauptsächlich produktorientierte Ansätze sind hingegen:

1. *Echtzeitverarbeitung von Daten und dynamische Inhalte:*
   Es wird hinterfragt, ob Echtzeitverarbeitung notwendig ist. Das Verarbeiten in größeren Intervallen kann viel Energie sparen. Dynamische Inhalte sollten sparsam verwendet werden.

2. *Datenbank-Indizes und Explain-Plans:*
   Der Einsatz von Indizes beschleunigt Datenbankabfragen, während Explain-Plans genutzt werden, um Abfragen zu analysieren und deren Effizienz zu optimieren.
3. *Dynamische Skalierung:*
   Systeme passen ihre zugeteilten Ressourcen automatisch an die aktuelle Auslastung an, um Ressourcenverschwendung zu vermeiden.
4. *Lose Kopplung von Komponenten:*
   Die Architektur wird modular aufgebaut, um Redundanzen zu vermeiden, die Flexibilität zu erhöhen und Systeme unabhängig voneinander skalierbar zu machen.
5. *Importe und Abhängigkeiten kennen:*
   Nur wer die eigenen Abhängigkeiten kennt, kann die eigene Anwendung schlank halten.
6. *Reaktives Programmieren:*
   Software wird so entwickelt, dass sie auf Ereignisse und Veränderungen reagiert (anstelle von Polling).
7. *Effiziente Programmierung mit guten Algorithmen:*
   Durch den Einsatz optimaler Algorithmen wird der Code ressourcenschonend gestaltet.
8. *Komprimierte Netzwerkkommunikation:*
   Datenübertragungen zwischen Systemen werden minimiert und komprimiert, um den Energieverbrauch in der Kommunikation zu senken.
9. *Container-basierte Plattform:*
   Anwendungen laufen in isolierten Containern, die unabhängig voneinander betrieben werden können, im Sinne von Portabilität und Skalierbarkeit. Container sind gegenüber VMs deutlich effizienter.

Wir konnten einige Aspekte identifizieren, die von Beginn an Teil unserer Arbeitspraxis waren. Unter diesen sind zu nennen *Schulung der Mitwirkenden*, *Containerisierung* der Projektinfrastruktur oder die *Modularität* und *Lose Kopplung* der Komponenten. Letzteres ist konzeptionell angelehnt an das Prinzip der *Orthogonalität* in der Unix-Philosophie [6].

Weiterhin haben wir Aspekte identifiziert, die wir bewusst in das Projekt integrieren (vgl. Abb. 7). Als besonders sparsame Methode, um die automatisierten Tests zu beobachten, hat sich früh das Protokoll *VNC* etabliert. Voraussetzung dafür ist ein laufender *X-Server*, in dem *Automation-Container*. Über VNC wird dann (vergleichbar zu SSH) der entfernte X-Server mit einem Client auf dem Endgerät verbunden. Das Protokoll ermöglicht, durch einen Bildbuffer, eine visuelle Übertragung ohne einen kontinuierlichen Videostream übertragen zu müssen.

Die Aspekte Datenerhebung, -Verarbeitung und -Speicherung erfordern zum einen, sorgfältige und vorausschauende Planung, zum anderen auch eine fortwährende Überprüfung: Wir haben uns dafür entschieden, die Messdaten für den Zeitraum des Projekts anstelle von Text-Dateien auf einem Datenbank-Server (DB-Server) abzulegen, um strukturiert auf die gesamten Testdaten zugreifen zu können. Dieses Vorgehen hat sich bereits mehrfach bewährt. Aber insbesondere bei der Erhebung und Abspeicherung der Daten sehen wir Optimierungsbedarf: Anstatt die sekündlich erfassten Daten direkt an die Datenbank zu übertragen, wäre es effizienter, die DB-Verbindung über einen längeren Zeitraum offen zu halten, die Daten z.B. in einem Buffer zwischenzuspeichern und minütlich an den DB-Server zu übertragen. Ein Best-Practice bei der Verarbeitung großer Datenmengen in einer relationalen Datenbank, ist die Verwendung von Indizes. Dies beschleunigt den Abruf von Daten nicht nur erheblich, sondern wirkt sich auch günstig auf den Ressourcenverbrauch des DB-Servers aus.

Bezüglich *effizienter* und *modularer Programmierung* sei hier die Automatisierung der App-Programmierung mittels eines eigens entwickelten Parsers erwähnt. Ein Parser erleichtert die Wartbarkeit der Software, indem in definierten Schnittstellen gedacht und gearbeitet wird. Eine gute Praxis ist ebenfalls einen systematischen *Review*-Prozess für Code und Skripte zu etablieren, um die Qualität von einzelnen Software-Komponenten zu erhöhen.

Weiterhin sei hier auch wieder auf den Open-Source-Gedanken verwiesen, da Quelloffenheit eine Voraussetzung für Anpassbarkeit und Überprüfbarkeit von Code ist. Um valide Aussagen über die Qualität der eigenen Software treffen zu können, müssen insbesondere externe Abhängigkeiten und Importe wohl bekannt und dokumentiert sein. *Abhängigkeiten und Importe* liegen bei uns besonders in den Container-Images und der Web-App vor. Abhängigkeiten sollten sorgfältig ausgewählt werden und, ebenso wie die Images, auf lokalen Maschinen verfügbar sein, sodass kein vermeidbarer Netzwerkverkehr anfällt.

Übernommen haben wir ebenfalls die Praxis unsere Test-Infrastruktur (etwa $60W$ Grundlast) außerhalb der Nutzungszeiten abzuschalten, für andere Projekte bietet sich an Zeitpläne festzulegen.

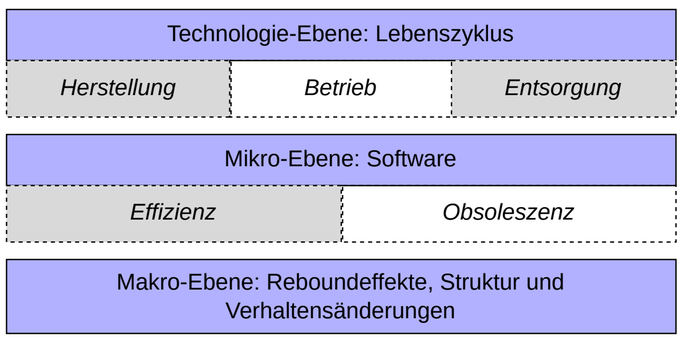

Abbildung 8: Effektebenen der Digitalisierung hinsichtlich Umweltaspekte

Zum Schluss sind auch einige, bisher ungelöste, Schwachstellen zu nennen: Zwar ist die Messung der Test-Infrastruktur essenzieller Bestandteil des Projekts, allerdings ist die Verbrauchsermittlung der gesamten, für das Projekt verwendeten, Infrastruktur schwierig. Insbesondere betrifft dies die Messung der ausgelagerten Komponenten (VM, DB-Server) auf anderen Geräten. Zudem stellt die Ermittlung in Bezug auf die real anfallende Netzwerkkommunikation und deren Energiebedarf eine gewisse Hürde dar.

Wir stellen fest, dass Green Coding i.d.R. bedeutet, qualitativ hochwertige und langlebige Software zu bauen. Viele Aspekte, die hier unter dem Schlagwort Green Coding fallen, sind gängige Methoden in der Qualitätssicherung, dies kann und sollte ein Anknüpfungspunkt in der Softwareentwicklung sein. Insbesondere sollte Green Coding als Haltung und Alltagspraxis verstanden werden, in der Handlungen und etablierte Methoden regelmäßig hinterfragt und Entscheidungen anhand fundierter Kenntnisse abgewogen werden können.

Diskussion

Im vorliegenden Projekt haben wir zwei Hauptthemen untersucht: 1. Wie kann eine nachhaltige IT-Infrastruktur gestaltet werden und 2. Welche Auswirkungen hat ein Remote-Workspace auf den Energieverbrauch? Für die zweite Fragestellung haben wir uns im Besonderen mit *VSCodium* auseinandergesetzt, da dies eine Konfiguration ist, die zunehmend in Anwendungen wie *GitHub-codespaces* u.ä. verwendet wird.

Wir haben versucht uns dem Konzept Nachhaltigkeit ganzheitlich zu nähern und dafür verschiedene Ebenen untersucht (vgl. Abb. 8) [7]. Auf der Technologie-Ebene gibt es gute Indizien für das Potenzial, älterer Endgeräte und refurbished Server, dies konnten wir im vorangegangenen Abschnitt plausibel darlegen.

Die durchgeführten Experimente mit *Vim* und *VSCodium* sind auf der Mikroebene anzusiedeln. Die vorliegenden Ergebnisse der ersten Testläufe liefern Hinweise auf die Energieeffizienz serverbasierter Entwicklungsumgebungen. Hier ist, wie diskutiert, weiterer Forschungsbedarf gegeben. Für die praktische Anwendung bieten die Erkenntnisse aus dem Themenbereich *Green Coding* wertvolle Einsichten. Eine große Herausforderung für langlebige Hardware ist häufig das Thema rund um *obsoleszente* Soft- und Hardware, dieses könnte durch *Open-Source* adressiert werden.

Sowohl Open-Source als auch Green Coding sind Aspekte, die auf die Makro-Ebene wirken und systemische Strukturen und Verhaltensänderungen adressieren. Bisher unberücksichtigt bleiben weitere Makro-Aspekte wie *Rebound-Effekte*, die in vielen Bereichen Effizienzgewinne aufwiegen.

Referenzen

[1] Open Source Initiative, besucht am 23. Jan. 2025. Adresse: https://opensource.org/osd.

[2] D. Viduka und A. Baić, „Impact of Open Source software on the environmental protection,“ *Computational Ecology and Software*, Jg. 5, S. 113–118, 2015.

[3] S. Prakash, F. Antony, A. Köhler und R. Liu, *Ökologische und ökonomische Aspekte beim Vergleich von Arbeitsplatzcomputern für den Einsatz in Behörden unter Einbeziehung des Nutzerverhaltens*. 2016.

[4] R. Bashroush, N. Rteil, R. Kenny und A. Wynne, „Optimizing Server Refresh Cycles: The Case for Circular Economy With an Aging Moore's Law,“ 2020.

[5] n. S. u. V. Bundesministerium für Umwelt Naturschutz. „Green Coding - Booklet zur Workshopreihe.“

[6] E. Raymond, *Art of Unix Programming, the*, 1st edition. Addison-Wesley Professional, 2003, 1 Online-Ressource (560 S.), ISBN: 0131429019 ; 9780131429017.

[7] ECO Austria - Institut für Wirtschaftsforschung, besucht am 24. Jan. 2025. Adresse: https://ecoaustria.ac.at/wp-content/uploads/2021/05/Studie-Digitalisierung-Emissionen.pdf.