GREEN CODING

DER ENTSCHEIDENDE SCHRITT FÜR NACHHALTIGE TECHNOLOGIE, DEN JEDE:R ENTWICKLER:IN GEHEN SOLLTE

UNSERE AUTOREN

JUTTA
SENIOR MANAGING CONSULTANT

MICHI
SENIOR SOFTWARE ENGINEER

Der Klimawandel, CO2-Einsparungen und Nachhaltigkeit sind derzeit in aller Munde. In der IT-Branche sind derzeit hauptsächlich Betreiber von Rechenzentren oder die Nutzungsdauer von Hardware im Fokus. Als Softwareentwickler sehen wir meist nicht direkt, welchen Impact unsere Arbeit und unsere Produkte auf den Energieverbrauch und CO2-Emissionen haben. Dabei ist dieser deutlich größer als zunächst gedacht. Der Einfluss, der in der Softwareentwicklung möglich ist und der im Sinne „guter“ Software wahrgenommen werden sollte, soll im Folgenden aufgezeigt werden.

Was passiert bisher in der IT-Branche?

Wenn es um die Umwelteinflüsse der IT-Branche geht, dann fallen zwei Bereiche sofort auf: Die Hardware, die genutzt wird, und der Strom, den sie verbraucht. Als nächster Gedanke kommt oft noch die Abwärme der Rechenzentren, die ja rein intuitiv direkt in die Klimaerwärmung einfließt.

Was also passiert im Bereich der Hardware? Während es früher fast schon zum guten Ton gehörte, sich alle zwei Jahre ein neues Handy zuzulegen und das auch beispielsweise von Mobilfunk-Anbietern stark subventioniert wurde, stellen sich mittlerweile immer mehr Menschen die Frage, ob sie das wirklich brauchen und was das neue Gerät tatsächlich mehr kann. Ähnlich gehen auch die Betreiber von Rechenzentren und Firmen mit den Geräten ihrer Mitarbeiter vor. Dass neue Hardware ein Kostenfaktor ist, bekräftigt dieses Vorgehen zusätzlich. Neue Trend-Technologien werden gefühlt wieder länger und geduldiger begutachtet, bevor man sich eine Unmenge von Geräten zulegt, die dann in der Ecke liegen, weil sich der Use Case nicht etabliert hat.

Ein Ansatz, der sich zum Teil mit dem Inhalt dieses Artikels überschneidet, ist die Virtualisierung. Diese setzt darauf, mehr und teilweise unterschiedliche Software auf der gleichen Hardware laufen zu lassen. Dadurch wird weniger physische Hardware benötigt und der Ressourcenverbrauch eingedämmt. Wie sich diese Technologie auf den Energieverbrauch der Software auswirkt, wird später noch betrachtet.

Wenden wir uns dem Thema Energieverbrauch zu: Jeder einzelne RZ-Betreiber hat diesen als erheblichen Kostenfaktor auf dem Schirm. So werden Komponenten entwickelt und eingekauft, die weniger Strom verbrauchen. Diese Technologien finden noch in einem anderen Bereich Anwendung, der uns allen immer wieder begegnet: Egal ob Smartphone oder Laptop, wenn der Akku länger hält, werden Geräte besser bewertet.

Der dritte Punkt, der gerne auch nachrichtenwirksam inszeniert wird, ist das Thema Kühlung. So gibt es inzwischen Rechenzentren, deren Abwärme zur Beheizung von Schwimmbädern genutzt oder ins Fernwärmenetz eingespeist wird. Zusätzlich versucht man, den Bedarf an Kühlung zu reduzieren, sodass ein sicherer Betrieb auch bei höheren Temperaturen möglich ist und eine passive Kühlung ausreicht.

Im Bereich von Hardware und Rechenzentren ist also bereits einiges in Bewegung geraten. Gesetzliche Vorgaben sind bereits vorhanden und wer möchte, kann sein Rechenzentrum auch als „grün“ oder „klimaneutral“ zertifizieren lassen.

Macht die Software-Branche denn nichts?

Dass der Stromverbrauch von Hardware sehr direkt von der darauf betriebenen Software abhängt, benötigt keinerlei Erklärung. Allerdings ist es in manchen Fällen einfacher als in anderen, zu beeinflussen oder abzuwägen, welche Maßnahmen zu priorisieren sind.

Geht es um Anwendungen, die eigenständig auf einem Computer installiert sind, so ist das Gesamtsystem relativ einfach und auch die Messung reproduzierbar auf einem Standardsystem. Doch wenn wir uns bereits den Energieverbrauch einer Webseite anschauen, kann die genutzte Umgebung der Endanwender erheblichen Einfluss auf den Energieverbrauch haben. Nutze ich das Angebot als App oder in einem Browser? Und wenn wir einen Browser nutzen, welchen?

Die wenigsten Anwendungen, die heutige Entwickler bearbeiten, sind Standalone-Anwendungen. Und damit wächst die Komplexität. Das macht es auf der einen Seite einfacher, wenigstens irgendetwas zu tun. Auf der anderen Seite erschwert es auch die Priorisierung, welcher Ansatz die größte Wirkung entfaltet. Zusätzlich verringert sich die Übertragbarkeit von Ergebnissen, weil schon kleine Änderungen am Gesamt-Setting diese zunichte machen können.

Anwendern ist das Argument, Energie einzusparen, in der Regel egal, wenn sie gerne ein neues Feature haben wollen. Aber brauchen wir das Argument der Energieeinsparung überhaupt? Haben Maßnahmen, die den Energieverbrauch reduzieren nicht vielleicht noch andere Effekte, die unseren Lösungen generell weiterhelfen?

Bevor wir diese Frage zum Ende des Artikels beantworten, widmen wir uns den Ansatzpunkten, die wir in der Software-Entwicklung haben, um Energie einzusparen.

Was kostet bei Software Energie? Und was spart bei Software Energie?

Es ist relativ simpel und auch intuitiv: Jeder Rechenschritt und jeder Speicher- bzw. Ladevorgang braucht Strom. Auch die Übertragung von Daten über ein Netzwerk braucht Strom. Jedes aktivierte Gerät und die Zeit, die dieses benötigen wird, ist ein Faktor.

Abgesehen von Datenübertragungen kommt man zu einer einfachen Faustregel: Jeder Rechenschritt braucht eine gewisse Menge Zeit und Energie. Habe ich viele Rechenschritte, brauche ich damit viel Zeit und Energie. Senke ich die Ausführungszeit einer Aufgabe, spare ich diese. Der Zusammenhang lässt

sich in einen einzigen Begriff zusammenfassen: Performance. Dass Hardware auch im Leerlauf Strom verbraucht, ist dabei ein Nebenaspekt, den wir im Hinterkopf behalten. [1]

Mit diesen einfachen Grundgedanken möchte ich nun verschiedene Themengebiete beleuchten. Leider ist nicht in allen Bereichen die Forschungslage gleich fortgeschritten, doch soweit möglich werden die vorliegenden Ergebnisse verlinkt.

Thema 1: Fachliche Ansatzpunkte

Unter fachlichen Ansatzpunkten verstehen wir die Business- und Anwenderperspektive. Diesbezüglich sind drei Stellschrauben zu identifizieren: der Prozess, die Automatisierung und die Anzahl an Features, also auch die Komplexität der Anwendung.

Das Schöne an fachlichen Ansätzen ist, dass sie in der Regel gegenüber Anwendern gut zu argumentieren sind. Wenn ich einem Anwender vorgeschlagen habe, einen Prozess einfacher zu gestalten, Fehlerquellen zu beseitigen oder lästige Tätigkeiten zu automatisieren wurde ich bisher nur noch gefragt, in welcher Reihenfolge oder wie schnell diese Maßnahmen umsetzbar sind.

Auch kann man sich hier schrittweise an die Umsetzung begeben. Ein Prozess muss meist nicht in einem Schritt vom silogedachten Papierformular zur supereinfachen Komplettlösung werden.

Wie kann ein guter Prozess Energie sparen?

Denken wir einmal an einen schlechten Prozess. Was stört die Nutzer, wenn sie mit einer Lösung nicht zufrieden sind? Es sind einfache Dinge: Wenn Daten mehrmals eingegeben werden müssen, wenn sie denken, der Prozess sei abgeschlossen, um eine Weile später doch noch Informationen nachreichen zu müssen, oder wenn Benutzeroberflächen nicht intuitiv bedienbar und unnötig kompliziert sind.

PROZESSGESTALTUNG

Jeder Kontakt des Nutzers mit einer Oberfläche benötigt neben der Rechenleistung auch noch weitere Geräte: den Bildschirm und die Eingabegeräte. Diese müssen nicht nur während der eigentlichen Rechentätigkeit betrieben werden, sondern die gesamte Zeit, die der Nutzer zur Arbeit im Prozess benötigt. Und im Vergleich zu reinen Rechenprozessen sind Nutzer wirklich langsam!

Wir sollten also anstreben, dass möglichst wenige Nutzer manuell im Prozess tätig werden müssen und dass diese in wenigen und schnellen Schritten fertig sind.

Damit ergeben sich für die Prozessgestaltung ein paar Richtwerte, die auch die Anwender gut finden werden:

1. Prozesse so gestalten, dass jede einzubindende Partei so selten wie möglich tätig werden muss.

2. Fehlerquellen reduzieren, da Fehler in Prozessschritten wieder korrigiert werden müssen – in der Regel durch Anwender, womit wir wieder bei Punkt 1 wären.

3. Die Benutzeroberflächen intuitiv und effizient gestalten: Wenn ein Anwender direkt versteht, was sie von ihm wollen und er weniger Einzelschritte dafür benötigt, ist er schneller fertig. Zusätzlich zahlt dies auch auf Punkt 2 ein.

4. Manuelle Dateneingaben reduzieren: Diese Maßnahme stellt bereits einen Übergang in Richtung Automatisierung dar. Der Effekt ist einfach: Wiederverwendete Daten reduzieren die Eingabezeit, sind oft bereits validiert und damit weniger fehlerträchtig. Zusätzlich erzielt der Anwender schnelleren Fortschritt im Prozess und empfindet ihn angenehmer.

5. Die Anwender von Anfang an in die Prozessgestaltung mit einbinden: Allzu oft haben Product Owner und Entwickler eine Vorstellung davon, wie Anwender ein System benutzen (sollen). Wir haben noch nicht erleben dürfen, dass dies genau den Bedarf der Anwender trifft. Symptomatisch sind folgende Aussagen: „Die Anwender verstehen … nicht“ oder „Die Anwender benutzen das System falsch“. Nur, wenn ich in der Gestaltung meiner Applikation tatsächlich die Sichtweise und Bedarfe der Anwender erfrage und umsetze, kann ich Mehraufwände in der Umsetzung und Probleme in der Nutzung des Systems vermeiden. Das Wiederum zahlt direkt auf die vorgenannten Punkte ein. Oft wird argumentiert, dass die Einbindung der Anwender zu viel Aufwand bedeutet. Allerdings hat die Erfahrung hier gezeigt, dass dieser sich in der Lebensdauer einer Lösung deutlich amortisiert.

AUTOMATISIERUNG

Jeder Prozessschritt benötigt seine Zeit und damit Rechenleistung. Ich habe selten einen Prozess erlebt, in dem nicht ein paar Schritte vereinfacht oder ganz weggelassen werden konnten. Wenn ich Aufgaben ganz vom Anwender an die Software übergeben kann, spare ich all die Geräte, die die Anwender sonst benötigen und zusätzlich zumeist noch etwas Zeit.

Die meisten Menschen empfinden hauptsächlich die Aspekte einer Aufgabe als lästig, die wenig Intelligenz erfordern. Einfache Regeln zu befolgen, Daten zwischen zwei Formularen abzugleichen o. ä. sind als menschliche Aufgaben sehr fehleranfällig. Hier können wir mit Automatisierung einen sinnvollen Beitrag leisten. Bei steigender Komplexität einer Aufgabe wird die Automatisierung allerdings schwieriger in der Umsetzung und ab einem gewissen Punkt vom Menschen wieder besser erledigt. Spätestens, wenn es eine Reihe von

Sonderfällen oder Interpretations- und Handlungsspielräumen dazu kommen, darf der Sinn der Automatisierung hinterfragt werden.

Hinzu kommt die Frage, wie oft eine Aufgabe ausgeführt werden muss. Mit steigender Anzahl von Durchläufen, am Ende sogar von verschiedenen Personen, steigt die Wahrscheinlichkeit, dass eine Automatisierung sinnvoll ist.

In Bezug auf die Energiebilanz dieser Optimierung können wir uns ebenfalls auf die Aspekte Laufzeit und Ausführungshäufigkeit beziehen. Muss meine Automatisierung länger rechnen, als ein Mensch für das Erledigen dieser Aufgabe braucht, sollte ich mir den Aufwand sparen.

Eine Sonderform der Automatisierung stellen die Algorithmen aus den Bereichen Machine Learning & Data Science – landläufig auch als „künstliche Intelligenz“ bezeichnet – dar. Ob die gewonnene Leistung den Energiebedarf beim Training der Modelle und in der Anwendung rechtfertigt, ist zu hinterfragen. Dies muss allerdings im Einzelfall entschieden werden.

Am Ende muss im Bereich der Automatisierung der Anwendungsfall betrachtet werden, ob diese das richtige Mittel ist, um den Energiebedarf einer Software zu senken. Ein Weg, um schrittweise den Energiebedarf zu senken und sinnvolle Verbesserungen zu erzielen, ist es, nicht von Anfang an alles umzusetzen, was einem so einfällt. Oft zeigt sich erst im Laufe der Zeit, welches die wirklich sinnvollen Automatisierungen sind und welche man sich vielleicht lieber spart.

FACHLICHE FEATURES

Auf den ersten Blick könnte man meinen, dass zusätzliche Features auch den Energiebedarf einer Software steigern. Tatsächlich kommt es allerdings sehr auf die Art und Weise der Umsetzung an. Komplexität, Datenmengen und Software-Architektur spielen eine große Rolle. Zumeist gibt es mehrere Optionen, wie ein Feature technisch umgesetzt werden kann. Im Rahmen der Konzeption wird daher in der Regel auch der Aspekt von Performance beleuchtet, was direkt in die Energieeffizienz einzahlt.

Es ist aber auf jeden Fall sinnvoll, sich hin und wieder zu fragen, welche der vielen Features tatsächlich genutzt werden und welche man vielleicht wieder ausbauen könnte. Dazu dann mehr im Thema Maintenance.

UI/UX DESIGN

Die Gestaltung einer User Experience geht eng einher mit der Gestaltung von Prozessen und der Interaktion der Anwender mit der Applikation.
Es gibt darüber hinaus allerdings ein paar Punkte, die auch hier für eine bessere Energieeffizienz sorgen können. Diese betreffen hauptsächlich die Gestaltung der User Interfaces.

Gestalten Sie die Oberflächen nach dem Prinzip „Mobile first“. Die Technologien für mobile Applikationen sind aufgrund von begrenzten Akku-Kapazitäten darauf ausgelegt, wenig Energie zu benötigen. Auch wenn eine Anwendung nicht auf einem mobilen Gerät aufgerufen wird, kann sie von diesen Technologien profitieren.

Thema 2: Architektur

In der IT ist es wie mit Bauwerken: Stimmt das Fundament nicht, kann der Dachdecker das Gebäude auch nicht mehr retten. Und so ist die Architektur einer der wichtigsten Ansatzpunkte, um den Energieverbrauch von Software zu reduzieren. Stimmt die Architektur nicht, kann ein Entwickler mit Feinjustierungen im Code den Energieverbrauch nicht so einfach maßgeblich reduzieren.

Auch in diesem Gebiet lassen sich einige grundsätzliche Stellschrauben identifizieren, mit denen die Energieeffizienz einer Applikation beeinflusst werden kann:

RESSOURCENBEDARF ALLGEMEIN

Benötigt meine Applikation viele Ressourcen im Sinne von Speicher- oder Rechenkapazität, so verbrauchen diese Kapazitäten, egal ob physisch oder virtualisiert, Strom. Besonders, wenn Ressourcen bereitgestellt aber nicht genutzt werden, ist dies oft vermeidbar.

Ein häufiger Auslöser für größere Dimensionierung ist die Verarbeitung von Lastspitzen. Habe ich eine kontinuierliche Last auf meinem System, so kann ich recht genau kalkulieren, welche Kapazitäten ich benötige. Habe ich dagegen ein System, das einmal im Monat für Planungsvorgänge genutzt wird und den restlichen Monat nicht, so richtet sich der Bedarf nach der zu erwartenden Maximallast. Den Rest des Monats steht die Kapazität ungenutzt bereit. Entsprechend sind Maßnahmen, die die Verteilung der Last über die Zeit ausgleichen durchaus sinnvoll.

Darüber hinaus gibt es eine weitere Maßnahme, die dazu beiträgt, ungenutzte Ressourcen zu reduzieren. Diese besteht in der Zerlegung des Gesamtsystems. Dadurch können flexibel Ressourcen hinzugenommen werden, wenn sie benötigt werden und wieder abgeschaltet, wenn dies nicht der Fall ist.

MODULARISIERUNG

Im Grunde genommen geht es bei der Modularisierung darum, die Software so aufzubauen, dass ich eine Anzahl kleinerer, klar getrennter Teile habe. Dabei ist es wichtig, die Trennung so zu vollziehen und auch schon im Vorfeld so zu planen, dass die einzelnen Module möglichst unabhängig voneinander arbeiten können. Das betrifft sowohl die Logik als auch die Datenhaltung. Die Arbeit eines Softwareteils darf nicht dazu führen, dass der andere seine

Daten nicht lesen oder schreiben kann.

Konsequente Modularisierung hilft auch bei der (Weiter-)Entwicklung. Hier können nun mehrere Entwicklungen parallel vorangetrieben werden, ggf. sogar von getrennten Teams. Die Veränderungen an Software wirken sich nur lokal in einem Modul aus und können mit weniger Risiko umgesetzt werden. Auch die Komplexität meiner Software kann sich reduzieren, wenn ich sie in kleinere voneinander unabhängige Teile zerlege. Das wiederum erleichtert nicht nur die einzelnen Entwicklungen sondern auch die Einarbeitung für neue Entwickler.

Ein weiterer Aspekt der Modularisierung ist die erleichterte Austauschbarkeit von Software-Anteilen. Hat man es, bei allen Bemühungen um gute Wartbarkeit nicht geschafft, Software-Anteile nutzbar zu halten, so kann in einer gut geschnittenen Lösung eine Teil-Ablösung statt eines kompletten Neubaus erfolgen. Auch die Einführung einer neu gebauten Nachfolgelösung kann so sukzessive gestaltet werden, ohne den Anwendern einen Big Bang zumuten zu müssen.

Betrachtet man weitere Aspekte der Modularisierung, die in Form von Microservices längst State of the art sind, so fällt auch die verbesserte Wiederverwendbarkeit einzelner Funktionen auf. Im Falle von Microservices wird sogar ein unabhängiger Betrieb von Modulen möglich. Es können einzelne Softwareteile abgeschaltet, wenn sie nicht genutzt werden, und Ressourcen dynamisch angefordert werden. Das System muss nicht auf die erwartbare Maximallast dimensioniert werden. Dies ist besonders dann von Vorteil, wenn sich Module mit stark unterschiedlichen Hardwareanforderungen getrennt skalieren lassen.

Auch im Rahmen der Modularisierung muss man nun wieder die Balance wahren: Jede Kommunikation zwischen einzelnen Softwareteilen erzeugt einen Overhead. Sind die Module zu klein, kann der Overhead den Mehrwert überwiegen. Sind die Module zu groß, so kann ich die o.g. Vorteile nicht ausschöpfen. Dieser Overhead ist bei internen Modulen im Monolithen gering, aber gerade bei Microservices mit Netzwerklatenz spürbar.

Wir möchten an dieser Stelle anmerken, dass Microservices nicht jedem Anwendungsfall die beste Lösung darstellen. Wie bei anderen Architektur-Entscheidungen sollte für den Anwendungsfall der sinnvollste Ansatz gewählt werden.

DATENHALTUNG

Der Speicher verursacht neben den Aufwänden für das Lesen und Schreiben auch weniger transparente Energieverbräuche, da mit Backups und Archivierungen große Datenmengen von einer Instanz auf die nächste übertragen werden. Maßnahmen, die die Menge des benötigten Speichers reduzieren, entfalten also durchaus ebenfalls Wirksamkeit. Auf der anderen Seite profi-

tiert durch ein regelmäßiges Housekeeping von nicht mehr benötigten Daten auch die Performance der Gesamtapplikation, da Suchvorgänge in kleinen Datenmengen schneller sind.

Die Reduktion der insgesamt gespeicherten Daten ist aus Gründen der Datensparsamkeit im Sinne von Datenschutz und IT-Sicherheit unabhängig vom Energieverbrauch einer Applikation gegeben und gängige Praxis.

In manchen Anwendungen werden Berechnungen mit den Daten durchgeführt. Hier ist abzuwägen, wann es sinnvoller ist, Daten einmalig zu berechnen und dann zu speichern gegenüber einer wiederholten Ad-Hoc-Berechnung. Je höher der Aufwand zur Berechnung, desto wahrscheinlicher ist es sinnvoll, die Ergebnisse zu speichern. Hinzu kommt, dass man berücksichtigen muss, ob sich die Berechnungsgrundlage verändert, was zu Inkonsistenz der gespeicherten Ergebnisse führen kann.

DATENSTRUKTUR

Die Art und Weise, wie Daten abgelegt werden, ist eine recht frühe Betrachtung im Projekt bzgl. der Architektur. Hier spielt zum einen die passende Technik eine Rolle (relationale, hierarchische oder andere Datenbanken), die man sich einmal überlegt und nur mit wirklich großen Kosten ändern kann. Zum anderen ist die Art und Weise der Daten-Strukturierung relevant, welche die Modularisierung meiner Lösung bestmöglich unterstützen sollte. Alles Weitere ist stark von meiner gesamten Architektur abhängig und muss immer im Zusammenspiel betrachtet werden.

Jenseits der reinen Architektur-Entscheidung liegt dagegen die Frage, wie Zugriffe technisch gestaltet sind. Hier konnten wir in bisherigen Projekten schon Refactorings erleben, die 80 Prozent Zeitreduktion einbrachten. Was früher, als die Ressourcen knapp waren, ein ganzer Tätigkeitsbereich namens Softwaretuning war, wird heute leider allzu oft erst als letzte Möglichkeit betrachtet. Mehr dazu im Abschnitt Coding und Design.

EINGESETZTE PROGRAMMIERSPRACHEN UND FRAMEWORKS

Vor einiger Zeit lief ein Studienergebnis durch das Netz, in dem der Energieverbrauch einer Referenzaufgabe bei Umsetzung in verschiedenen Programmiersprachen verglichen wurde. Er wies einen Spreizungsfaktor von 1:79 auf. [2] Die gleiche Aufgabe in Perl umgesetzt benötigte 79-mal so viel Energie wie eine Umsetzung in C. Doch sind wir einmal ehrlich: Wie viele Fälle gibt es, in denen beide Sprachen gleichermaßen für eine Aufgabe in Betracht gezogen werden? Das Szenario ist zumeist, dass wir eine Aufgabe haben, sich dafür bestimmte Sprachen anbieten und im Kontext von verfügbaren Entwicklern nur noch wenige Optionen übrig bleiben. Sofern es mindestens zwei Sprachen sind, lohnt sich ein Blick in diese Liste, vorher nicht.

Allerdings hatte die Veröffentlichung dieser Ergebnisse einen anderen positi-

ven Effekt: Das Codon-Projekt am MIT hat einen verbesserten Python-Compiler entwickelt, der verbesserte Geschwindigkeit und damit auch Energieeffizienz aufweist. [3]

Die Entscheidung für oder gegen eine Programmiersprache wird sich weiterhin zuerst nach dem Anwendungsfalls und erst später nach der Energieeffizienz richten.

SCHNITTSTELLEN

Kein System steht heute mehr allein auf weiter Flur. Erst durch die Verbindung zwischen verschiedenen Systemen erreicht die Digitalisierung ihren vollen Mehrwert. Der Austausch von Daten zwischen verschiedenen Systemen erspart Anwendern mehrfache Dateneingaben, sorgt dafür, dass wir mit hochwertigen Daten arbeiten können und aus der Vernetzung verschiedener Informationen Mehrwert generieren können.

Auf Seiten des Energiebedarfs lohnt es sich deshalb, genauer zu betrachten, welche Datenaustauschprozesse stattfinden. Denn jeder Datenaustausch benötigt, neben den Prozessen zur Bereitstellung und für die Verarbeitung aus Datenübertragungen bei versendendem und empfangenden System, auch noch Energie bei der Übermittlung. Daher lohnt sich im Bereich der Schnittstellen ein genauer Blick auf die Fragen: Was? Wann? Wie? Wie oft? Und an wen? Bzw. von wem?

Fangen wir beim größten Hebel an: Die Partnersysteme.

Trennen wir hier einmal nach den beiden Seiten „Empfang von Daten“ und „Weitergabe von Daten“.

Befinden wir uns auf der Empfängerseite, so können wir relativ gut selbst der Frage nachgehen, woher und welche Daten wir empfangen und wofür sie verwendet werden. Allzu oft wird diese Frage leider nicht regelmäßig gestellt. Ähnlich wie bei Refactorings oder dem Blick auf eventuell nicht mehr verwendete Features lohnt es sich, hier einmal hinzuschauen und zu hinterfragen, was aus den empfangenen Daten wird.

Für Daten, die wir an andere Systeme weitergeben ist die Frage meist schwerer zu beantworten. Im besten Fall haben wir eine aktuelle Dokumentation zu unseren Zielsystemen, deren Ansprechpartner und Verwendungszwecken unserer Daten. Doch auch die Empfängersysteme entwickeln sich weiter und selbst im Zielsystem ist nicht immer klar, was dort mit den vorhandenen Daten passiert. Spätestens wenn Daten an ein oder mehrere Datawarehouses weitergegeben werden, verliert man die Kontrolle. Ein Vorgehen, das uns in der Systementwicklung allzu oft begegnet, ist die fachliche Anforderung eines Datenempfängers, „das Gleiche“ wie ein anderer Empfänger zu bekommen. Es ist deutlich schneller umgesetzt, auch dem zweiten Ziel, die gleichen Daten in der gleichen Frequenz usw. zukommen zu lassen. Doch im

Sinne der Qualität und Kontrolle über die Daten unseres Systems, lohnt sich die Frage, ob „das Gleiche“ wirklich die fachlich und ökologisch nachhaltigere Lösung ist.

Ist die Frage geklärt, ob Daten übertragen werden sollen, ist es möglich, tiefer ins Detail zu gehen: Welche Daten sollen übertragen werden? Wie oft? Und welche Technologie ist angemessen?

Die Technologie stellt einen entscheidenden Faktor dar, um zu beurteilen, ob eine Reduktion der Häufigkeit oder eine Reduktion der Datenmengen mehr Einfluss hat. Auch die Strukturierung der Daten spielt an dieser Stelle eine erhebliche Rolle. Jede Übertragung verursacht neben den eigentlichen Nutzdaten auch eine gewisse Menge Overhead, z. B. um was für Daten es sich handelt und welchen Stand sie haben. Können wir diesen Overhead reduzieren, indem wir Daten bündeln, so reduzieren wir den Energieverbrauch pro Datensatz. Die Bündelung der Daten hängt dabei von verschiedenen Faktoren ab: Benötigen wir Daten in Echtzeit, auch wenn sie nur wenige Änderungen enthalten, so können wir nicht warten, bis eine gewisse Menge von Änderungen zusammengekommen ist. Haben wir dagegen stabile Daten mit einer Toleranz für Verzögerungen, so ist eine Bündelung möglich.

Die Technologie-Entscheidung je Schnittstelle ist dabei recht früh zu treffen und unterliegt daher auch Mutmaßungen, die sich später als falsch herausstellen können.

Oft sind Schnittstellen mit dafür verantwortlich, wenn ein System größer ausgelegt werden muss. Entsprechend sollte auch überlegt werden, ob Schnittstellen genau dann versorgt werden müssen, wenn die Systemlast am größten ist.

Sind die Technologie und Bündelung geeignet gestaltet, bleibt als letzter Punkt noch die Klärung, welche Einzeldaten tatsächlich versorgt werden müssen. Allerdings spielt dies – sofern nicht im großen Stil reduziert wird – eine untergeordnete Rolle.

FAZIT ARCHITEKTUR

Im Sinne einer ressourcenschonenden Architektur ist es sinnvoll, Aspekte der Energieeffizienz neben Kriterien wie technisch sinnvollen und praktikablen Lösungen mit in die Entscheidung einzubeziehen. Diese einmal getroffenen Entscheidungen beeinflussen den Energiebedarf nachhaltig und werden die Effizienz dauerhaft verbessern. Dazu gehören natürlich auch Techniken, die hier nicht explizit erwähnt wurden, wie beispielsweise das Reduzieren von Polling, Verwenden von Lazy Loading, Caching von Daten und vieles mehr.

Thema 3: Coding und Design

Bevor wir diesen Bereich genauer betrachten, soll an dieser Stelle eine deut-

liche Warnung ausgesprochen werden: Im Bereich Coding und Design geht es um das tägliche Arbeiten und die Weiterentwicklung. Hier gibt es zwar diverse Ansätze, doch es ist extrem wichtig, sich nicht in vielen kleinen Mikro-Optimierungen zu verlieren. Verlangen wir im Rahmen der Entwicklung eine Entscheidung zwischen einem guten Umsetzungsprozess mit lesbarem Code und der Nachhaltigkeit, so wird nicht nur für den Moment die Nachhaltigkeit verlieren, auch die Anerkennung für das gesamte Thema leidet. Solange wir noch nicht die großen Stellschrauben bewegt haben, sollte daher auf jede Form der Mikro-Optimierung verzichtet werden!

Befasst man sich mit Coding und Design, so handelt es sich um Entscheidungen, die mit jedem Umsetzungsschritt erneut getroffen werden müssen. So wie sich der Code jeden Tag verändert, kann jede Änderung zugleich auch Einfluss auf den Energieverbrauch der gesamten Applikation haben. Dabei sind manche Effekte zunächst nicht spürbar und erst die Summe der Einzelteile führt dazu, dass ein Handlungsbedarf entsteht. Hier hilft es, neben den einzelnen Entscheidungen, die zu treffen sind, immer wieder auch die Aufmerksamkeit auf das Thema Energieeffizienz oder – und hier ist es wichtig, die Sprache der Entwickler zu sprechen – Performance zu lenken. Die Abstraktion auf die Themen Performance oder Ressourcenbedarf ist weder schadhaft noch widersprüchlich. An manchen Punkten hilft es auch, die Arbeitserleichterung für die Entwickler in den Fokus zu rücken. Ziel sollte auf jeden Fall sein, dass Entwickler die verschiedenen Aspekte kennen und jeden Tag aufs Neue in ihre Umsetzungsentscheidungen einbeziehen.

UMSETZUNGSPRAKTIKEN

Ziel beim Entwickeln ist es immer, sauberen Code zu schreiben. Die Qualität und Wartbarkeit der Anwendung soll möglichst hoch sein. Auch wenn unter sogenanntem „Clean Code“ eher Richtlinien als strikte Definitionen existieren, hilft es, sich an derartigen Ansätzen zu orientieren. Kleine Verbesserungen für verständlicheren Code können sich langfristig auswirken.

Da wir fast immer in einem Team arbeiten, ist es sinnvoll, in den Austausch mit Kollegen zu gehen. Häufig sind wir in der Lage, Anforderungen allein umzusetzen, und wollen ungern die Zeit anderer beanspruchen. Allerdings kommt man im Team – zum Beispiel mittels Pair Programming – oft schneller zu einem Ergebnis, das qualitativ hochwertiger ist, da man gemeinsam besser auf Codestruktur und Wartbarkeit achten kann. Auch Fehleranalysen lassen sich gemeinsam besser bewältigen, solange sich die Zusammenarbeit in einem sinnvollen Rahmen bewegt.

Ein weiterer Beitrag für sauberen Code stellen Code Reviews dar. Im Code Review wechselt der Fokus vom Erzeugen von Code zur Betrachtung der gemeinsamen Coding-Richtlinien und der Harmonisierung aller geleisteten Entwicklungen. Die Entwickler profitieren davon, voneinander zu lernen und können sich verbessern und weiterentwickeln. In komplexeren Anwendungen hat man zusätzlich einen indirekten Wissensaustausch.

Initial mögen Code Reviews und Pair Programming nach einer großen Aufwandserhöhung aussehen, tatsächlich ist es bei weitem keine Verdopplung. Nachhaltig profitiert das Gesamtprojekt stärker von der erhöhten Code-Qualität und der daraus resultierenden verbesserten Wartbarkeit.

Einen großen Beitrag zur Steigerung der Softwarequalität kann die Methode des Test Driven Development darstellen. Der Effekt davon liegt nicht nur darin, dass Fehler früher bzw. noch während des Entwickelns entdeckt und damit vermieden werden, sondern auch, dass die Software in testbaren und besser wartbaren Einheiten strukturiert wird.

LOGGING

Im Betrieb und für das Debugging stellt Logging ein unverzichtbares Werkzeug dar. Es wird sichtbar gemacht, wann und wo Fehler auftreten. Voraussetzung für die Nutzbarkeit ist dabei, dass die Konfiguration sinnvoll durchgeführt wurde und stetig weiterentwickelt wird. Stehen zu viele Informationen im Log, sucht man buchstäblich die Nadel im Heuhaufen. Erhält man zu wenige, so bringt einen das Log nicht näher zum Ziel. Wer einmal in einer Applikation das Logging komplett deaktiviert hat, weiß, welche Auswirkungen es auf Performance und damit auch auf den Energiebedarf hat.

Trotz gängiger Maßnahmen wie Log-Rotation kann der Ressourcen-Bedarf für Logging erheblich sein. Schreibt man außerdem allzu viele Daten in seinen Log, so kann dies eine vorher schon schlechte Performance noch weiter ruinieren. Daher gilt für Energieeffizienz beim Logging das Gleiche, das auch in vielen anderen Bereichen gilt: So viel wie nötig, so wenig wie möglich.

TESTING

Einer der wichtigsten Schritte auf dem Weg zu hochwertiger Software ist das Testing. Wie bei fast allen Produkten kommt auch in der Softwareentwicklung die sogenannte 10er-Regel der Fehlerkosten zum Tragen: Mit jeder Produktionsstufe, in der wir einen Fehler später entdecken, steigen die Kosten für dessen Beseitigung um den Faktor zehn. [4] Entsprechend sollte auch in der Softwareentwicklung vom Konzept bis zum Deployment auf die Validierung des Erreichten und den Test der zur Auslieferung vorgesehenen Inkremente Wert gelegt werden.

Man darf zwar nicht vernachlässigen, dass auch Testing an sich den Energiebedarf steigert, allerdings kann dies durch eine entsprechend sinnvoll gestaltete Testkonzeption im Rahmen gehalten werden. In diesem Fall überwiegt die Einsparung späterer Mehraufwände, den Invest zur Umsetzung von automatisierten Tests. Weiterhin machen automatisierte Tests meine Applikation resilient gegenüber späteren Anpassungen, z. B. bei Fehlerbehebungen oder Überarbeitungen im Code (Refactoring).

Testing kann zusätzlich einen weiteren Nebeneffekt bieten: Dadurch dass im

automatisierten Testing mit standardisierten Testfällen gearbeitet wird, können diese verwendet werden, um Messungen zur Performance-Entwicklung durchzuführen.

DATENZUGRIFFE

Beim Laden von Daten aus einer Datenbank gibt es mehrere Optimierungsmöglichkeiten. Es sollte darauf geachtet werden, dass unnötige wiederholte Zugriffe auf die gleichen Daten vermieden werden. Selbst wenn eine Query gecached wurde, besteht ein Overhead durch den Zugriff auf die Datenbank beziehungsweise das verwendete Datenbank-Framework. Wenn unterschiedliche Daten geladen werden, können manche Datenbankabfragen trotzdem so zusammengefasst werden, dass insgesamt weniger Aufrufe nötig sind.

Ein weiterer Faktor ist die Performance individueller Queries. Gerade bei komplexeren Abfragen über mehrere Tabellen oder beim Laden großer Datenmengen kann es zu Performanceeinbußen kommen, auch wenn diese auf den ersten Blick nicht ersichtlich sind. Oft muss die Struktur einer betroffenen Abfrage überarbeitet werden, es kann aber auch Komplikationen mit Datenbanken oder dem verwendeten Framework geben.

Wenn mit den geladenen Daten einfache Berechnungen erfolgen, so lassen sich diese manchmal als Teil der Query bereits auf der Datenbank ausführen. Das kann bei der Performance helfen, weil in der Anwendung weniger Arbeitsschritte notwendig sind.

Thema 4: Maintenance

Mit Software ist es wie mit so vielen anderen Produkten auch: Wenn wir die regelmäßige Wartung ernst nehmen und regelmäßig Verschleißteile austauschen, hält sie länger. Ein wesentlicher Punkt, der bei der Entwicklung von Software einen massiven Mehrverbrauch an Energie bedeutet, ist die Zeit, in der sie entwickelt, aber noch nicht genutzt wird. In dieser Zeit werden bereits Ressourcen genutzt, die CO2-Emissionen erzeugen, ohne den produktiven Mehrwert zu erhalten. Entsprechend sollten wir uns gut darum kümmern, dass unsere Software aktuell bleibt und möglichst nie durch eine Neu-Entwicklung abgelöst werden muss. Denn gerade die Phase, in der eine Applikation neu entwickelt wird, während die Alt-Lösung noch betrieben wird, kann so vermieden werden.
TECHNICAL DEBT

Es steht bei der Entwicklung nicht die Zeit zur Verfügung, für jedes Problem eine ideale Lösung umzusetzen. Es gibt Kompromisse, die die Umsetzung einer Anforderung ermöglichen, auch wenn diese im Vergleich zu einer aufwändigeren Alternative als „unschön“ empfunden werden. Selbst bei perfekter Ausarbeitung funktionieren neue Anforderungen und Erkenntnisse nicht immer mit der bestehenden Umsetzung. Mit der Zeit altert also die Software, es entsteht eine technische Schuld, die die zukünftige Entwicklung

ausbremst. Um dieses Technical Debt abzubauen und der Alterung der Software entgegenzuwirken – ohne eine komplette Neu-Entwicklung – macht es Sinn, Abhängigkeiten aktuell zu halten und den bestehenden Code wartbar zu halten.

REFACTORING

Zum Teil geschieht das Überarbeiten von altem Code aus der Anforderung heraus, die Performance der Anwendung zu verbessern, was in der Regel auch die Energieeffizienz verbessert. In Bezug auf Technical Debt sind Refactorings zum Verbessern von Lesbarkeit und Wartbarkeit des Codes ebenfalls sinnvoll. Dies verbessert nicht zwingend die Effizienz oder Performance der Anwendung, wirkt sich aber positiv auf zukünftige Aufwände beim Arbeiten an den betroffenen Stellen aus.

Wichtig beim Refactoring ist, dass man sich Gedanken darüber macht, an welchen Stellen Code einfacher und besser gestaltet werden kann und mit wie viel Aufwand. Der Ausbau von ungenutztem Code oder ungenutzten Bibliotheken ist teils recht einfach und kann Komplexität reduzieren, die Verständlichkeit verbessern und eventuell die Kompilierung beschleunigen. Verglichen damit wäre die Umstrukturierung einer komplexen Berechnung zwar mit mehr Aufwand und Risiko verbunden, kann aber langfristig Arbeitsaufwände reduzieren und zu flexiblerem Code führen. Entsprechend sollte die Überarbeitung von Code vor allem dann angegangen werden, wenn sich die Performance, spätere Entwicklungs- oder Kompilierungsaufwände dadurch signifikant verbessern.

Manche Software wird über Jahre oder Jahrzehnte betrieben. Es ist dann nicht nur die Aktualität des Codes zu berücksichtigen, sondern auch die der Architektur an sich. Denn gerade, wenn wir eine Anwendung lange weiterentwickeln, können sich im Laufe der Zeit grundlegende Anforderungen und Use Cases verändern. Es mag zwar zunächst nach einem erheblichen Aufwand klingen, in einer Anwendung grundlegende Architektur-Elemente umzugestalten, allerdings gilt es hier zu berücksichtigen, dass eine Applikation nur mit einer zukunftsfähigen Basis dauerhaft weiterbetrieben werden kann. Um also zu verhindern, dass allzu früh die Applikation genau so, nur moderner neu gebaut wird, sind auch größere Änderungen sinnvoll und wichtig zu bedenken.

AKTUALISIERUNGEN

Neuere Versionen von Abhängigkeiten in einer Anwendung sind oft mit zusätzlichen Funktionalitäten ausgestattet, die bei der Entwicklung oder eventuell auch bei der Laufzeit der Anwendung helfen können. Vor allem werden mit der Zeit aber mehr Sicherheitslücken in älteren Abhängigkeiten aufgedeckt, welche ein Risiko darstellen. Regelmäßige Aktualisierungen von Abhängigkeiten sind deshalb wichtig und können aus Gründen der Sicherheit und Wartbarkeit bzw. Support normalerweise nicht ausgelassen, sondern

höchstens verzögert werden. Entsprechend bleibt hier nur die Suche nach vergleichbaren Alternativen, die mit einer besseren Performance punkten. Da ein Wechsel der genutzten Funktionalitäten meist deutlich mehr Aufwand als ein einfaches Update verursacht, sollte dies nur bei einer deutlich erwartbaren Verbesserung angegangen werden.

Thema 5: Betriebsmodus

Der Betrieb einer Anwendung kostet neben den oben genannten Architektur- und Code-Entscheidungen immer Energie und kann auf verschiedene Arten ebenfalls effizienter gestaltet werden.

VIRTUALISIERUNG UND CLOUD

Mithilfe von Virtualisierung ist es möglich, mehrere Anwendungen getrennt aber auf der gleichen Hardware zu verwalten, diese Konsolidierung kann insgesamt die Energiekosten verringern. Dies liegt oft in der Hand der Betreiber eines Rechenzentrums. Für die eigene Anwendung ist hauptsächlich interessant, wie viel Speicher und Rechenleistung zur Verfügung steht. Veränderungen an diesen Parametern ist dank Virtualisierung ebenfalls einfacher, was es erlaubt, (Energie-)Kosten zu optimieren, indem nur so viel Speicher und Rechenleistung wie nötig reserviert werden.

Diese Skalierung von Ressourcen ist einer der großen Vorteile von Cloud-Servern, zu denen dann die verschiedenen Servicemodelle kommen, die das Auslagern bestimmter Aufgaben erlauben. Es kann, wie bei traditionellen Rechenzentren, alles auf der virtuellen Maschine selbst verwaltet werden (Stichwort: Infrastructure as a service). Alternativ können vom Cloud-Betreiber vordefinierte Lösungen für Anwendungsserver oder sogar Anwendungen angeboten werden.

Für die Energieeffizienz kann dies einen Vorteil sein, wenn insgesamt weniger individuelle Lösungen erforderlich sind und mehr Anwendungen auf zentralen, optimierten Lösungen basieren – egal, ob es sich dabei um Cloud-Lösungen handelt oder nicht. Die Sinnhaftigkeit und der Aufwand bzw. das Geld bleiben aber eher der ausschlaggebende Faktor für die Wahl des Servicemodells.

VERFÜGBARKEIT

Wenn eine Anwendung stets erreichbar sein soll, auch im Falle von technischen Ausfällen, ist Redundanz eine gängige Lösung. Wenn die Software damit umgehen kann, lassen sich parallel mehrere Instanzen verschiedener Bestandteile betreiben, was natürlich die Energiekosten erhöht. Bei der Wahl der Verfügbarkeit ist daher zu überlegen, ob eine 99,9-prozentige Verfügbarkeit wirklich notwendig ist oder in welchem Umfang Ausfälle verkraftbar sind.

Wenn neben dem Produktivsystem auch noch ein oder mehrere – nur spo-

radisch verwendete – Testsysteme laufen, kann Energie und oft auch Geld gespart werden, wenn diese nicht immer verfügbar sind, sondern nur bei Bedarf. Sind die Verwendungszeiten regelmäßig und vorhersehbar, dann können diese Systeme entsprechend betrieben werden. Optimierungen sind auch bei Pipelines für continuous integration / continuous deployment möglich, sodass ein Deployment nur dann ausgeführt wird, wenn auch tatsächlich Änderungen vorliegen.

Fazit

In allen Lebensbereichen halten ein Label, Zertifizierungen und Initiativen Einzug. So auch im Bereich der Green IT. Für Rechenzentren kann es daher durchaus sinnvoll sein, deren Energieeffizienz zu testen und ggf. auch Maßnahmen zu verlangen. Der Vorteil: Die Systeme sind hierbei standardisierbar. Gleichermaßen gibt es mit dem „Blauen Engel“ auch eine Zertifizierung für Software. Allerdings gilt dieser nur für Einzelapplikationen, die auf einem PC installiert werden. Auch hier befinde ich mich auf einer standardisierbaren Umgebung. In der Mitte klafft eine Lücke: Komplexe Applikationen, die über Vernetzungen und Schnittstellen betrieben werden, sind nicht standardisierbar. Entsprechend ist hier keine Zertifizierung möglich.

Es gibt mittlerweile eine ganze Reihe von Initiativen, die sich mit Green Coding beschäftigen. Bei BetterCallPaul haben wir uns entschieden, keiner dieser Initiativen beizutreten. Das liegt vor allem daran, dass es sich zumeist um reine Absichtserklärungen ohne direkte Handlungserfordernis handelt. Auch wenn es ein ehrenhaftes Ziel ist, so ist es nur eine Frage der Zeit, bis in diesen Fällen ein Missbrauch in Gestalt von Greenwashing stattfindet. Das bedeutet allerdings nicht, dass es nichts zu tun gäbe oder wir uns nicht trotzdem konkrete Ziele setzen.

Es ist nicht immer eindeutig, dass Qualität auch direkt zu einem reduzierten Energiebedarf führt. Aber spätestens, wenn wir verhindern können, dass eine Software vollständig neu entwickelt werden muss, haben wir einen relevanten Unterschied erzeugt. Wie in vielen anderen Lebensbereichen auch, gilt es, sich nicht in den kleinen Einzelmaßnahmen zu verlieren, sondern die großen Räder zu drehen. Es wird hier von außen keine sinnvollen Impulse oder Anreize geben, damit unsere Software grüner wird. Dadurch wird es auch keinem Auftraggeber zu vermitteln sein, wenn die Entwicklung teurer wird, wenn es dafür keinen Kostenvorteil gibt. Aber dass Qualität mehr kosten darf, ist Allgemeinwissen und darüber auch für (grünere) Software vermittelbar. Lasst uns also unsere Jobs mit Qualitätsanspruch ausführen, dann erreichen wir auch für grüne Software eine Menge!

Wir vertreten die Position, dass es weder einer Absichtserklärung noch einer Zertifizierung bedarf, wenn man sich voll darauf committed, gute Software zu bauen, die die Bedürfnisse der Benutzer trifft. Wie in diesem Artikel zu lesen ist, haben wir keinen Punkt gefunden, an dem gute Software einen Widerspruch zu grüner Software darstellt, auch wenn in diesem Rahmen einige

Maßnahmen für grüne Software noch nicht automatisch mit bedacht sind. Gleichzeitig gibt es in jeder noch so guten Software Möglichkeiten, die Qualität und auch die Energieeffizienz zu verbessern. Wir sollten also auf jeden Fall das Bewusstsein für diese Möglichkeiten schärfen und gefundene Verbesserungen auch umsetzen. Denn es gibt keinen Grund, nichts zu tun!