Ausgabe 02/2026

Java aktuell

Praxis. Wissen. Networking. Das Magazin für Entwicklerinnen und Entwickler

D: 7,95 EUR | CH: 11,99 CHF | A: 9,00 EUR | Benelux: 8,50 EUR

4 191978 307959 02

Java aktuell

iJUG member
java-aktuell.eu

Updates im Detail
JUnit 6, Maven 4

Fediverse
Die Alternative zu gängigen sozialen Netzwerken

Women in Tech
Status quo und Perspektiven

JavaLand

VERPASST?

Mit dem **On-demand-Ticket**
bestes Java-Wissen nach Hause holen
und **Vortragsaufzeichnungen** und
-unterlagen für mindestens
ein Jahr genießen!

#JavaLand | www.javaland.eu

Präsentiert von: iJUG Verbund | heise medien | DOAG

Veranstalter: JavaLand

Liebe Leserinnen und Leser,

schön, dass ihr wieder bei einer neuen Ausgabe dabei seid! Wir haben euch diesmal eine bunte Mischung aus Technik, Community Themen und Denkanstößen zusammengestellt – alles, was die Java Welt und das Drumherum gerade spannend macht. Details zu aktuellen Geschehnissen in der Java-Community findet ihr wie immer ganz am Anfang im Java-Tagebuch. Zum Auftakt von Andreas Monschaus neuer Reihe „wie du deine Softwareprojekte zuverlässig ruinierst" behandelt er das Thema technische Schulden.

Los geht's mit Michael Thomas, der euch zeigt, wie *JUnit 6* das altbewährte Testframework modernisiert und welche Neuerungen euren Projektalltag wirklich weiterbringen. Matthias Bünger knüpft daran an und schaut im dritten Teil seiner Reihe zu *Maven 4* auf die kleinen, aber wichtigen Minor Changes des neuen Releases.

Dann wird's gesellschaftlich technisch: Jörg Liedl nimmt euch mit ins Fediverse und erklärt, warum dezentrale Social Media Plattformen gerade so viel Aufmerksamkeit bekommen. Direkt danach zeigt Marcus Fihlon, wie Mastodon, PeerTube und Mobilizon vom iJUG bereits heute konkret genutzt werden – und wie sie die Java Community digital stärken.

Auch Machine Learning hat in dieser Ausgabe seinen Platz: Eldar Sultanow und Cornelius May geben euch einen frischen Überblick über moderne ML Anwendungsfälle und die riesige Toollandschaft, die mittlerweile kaum noch jemand überschaut. Mit Rico Komenda lernt ihr anschließend, worauf ihr beim *Vibe Coding* achten müsst – denn KI gestützte Softwareentwicklung bringt nicht nur Komfort, sondern auch neue Sicherheitsrisiken.

Ein Thema, das viele kennen, aber wenige so richtig angehen: Software Dokumentation. Liam Bergh zeigt euch, wie sie nicht zur Last, sondern zum echten Werkzeug wird. Und wer Tests liebt (oder liebt, sie zu hassen), bekommt mit Sebastian Lauth neuen Stoff: Er macht deutlich, wie *BDD mit dem Spock Framework* Tests nicht nur besser, sondern regelrecht elegant macht.

Mit technischen Schulden räumt Christian Seifert auf. Er zeigt, warum Technical Debt keine Ausrede ist, sondern eine Entscheidung – und warum es sich lohnt, bewusst damit umzugehen. Gesellschaftlich wichtig wird es dann in Ildikó Tárkányis Beitrag über den „undichten Karriereweg" von Frauen in der IT. Sie zeigt, wo Strukturen brechen und wie sich mehr Diversität nachhaltig erreichen lässt.

Für euren Beratungs und Projektalltag gibt Julia Pedak wertvolle Einblicke: Wie setzt ihr gesunde Grenzen? Welche psychologischen Muster spielen eine Rolle? Und wie gelingt eine Kundenbeziehung, die beiden Seiten guttut?

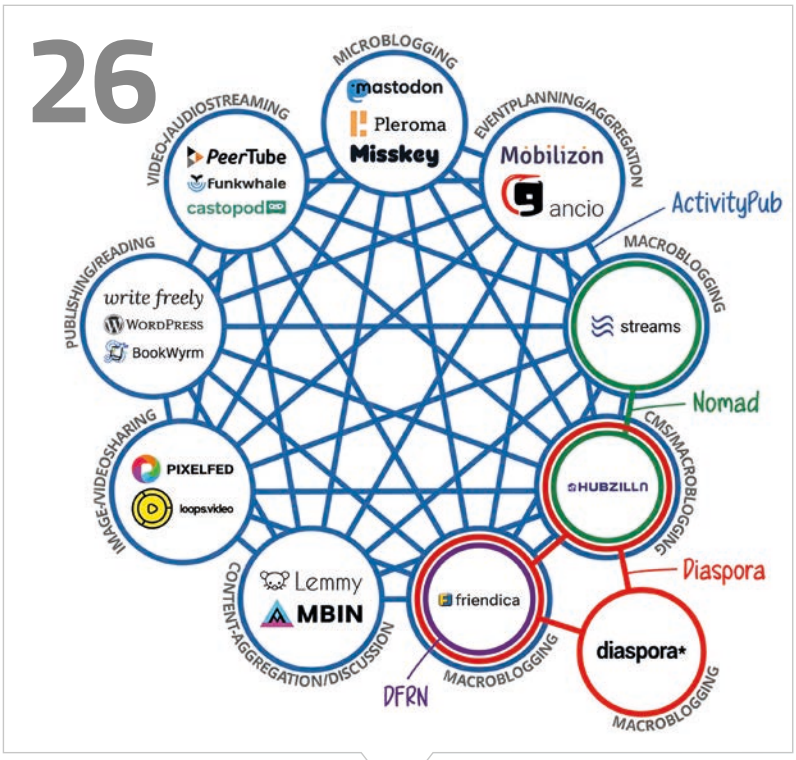
Zum Abschluss lädt Berend Semke zum Umdenken ein: Er stellt die gängige Idee von Motivation infrage und zeigt, warum wir Menschen nicht vorschnell „Schuld" zuschreiben sollten – genauso wenig wie einer Avocado, die einfach zu viel Wasser bekommt.

Wir wünschen euch viel Freude beim Lesen und viele Impulse, die euch inspirieren, weiterbringen und vielleicht auch zum Nachdenken anregen.

Lisa Damerow
Redaktionsleitung Java aktuell

INHALT

JUnit 6:
Was gibt es Neues?



Fediverse: Die Alternative zu den gängigen, zentralen Social-Media-Plattformen

Vibe Coding? Aber sicher!

Frauen in der IT-Branche: Warum der Karriereweg Talente verliert – und wie er sich abdichten lässt.

Andreas Badelt, stellv. Leiter der DOAG Cloud Native Community

JAVA TAGEBUCH

8. Dezember 2025

IBM kauft Confluent

IBM kauft Confluent – genau, nicht das Wiki („Confluence"), sondern die Firma hinter der gleichnamigen Datenstreaming-Plattform auf Basis von Apache Kafka (das sicher eines der bedeutendsten Java-Open-Source-Projekte ist, aber weit über das Java-Ökosystem hinaus Verwendung findet). Kostenpunkt: Magere 11 Milliarden US-Dollar – was im Vergleich mit den gerade üblichen KI-Deals geradezu Peanuts sind. Und um die datenhungrige KI soll es hier hauptsächlich gehen: „With the acquisition of Confluent, IBM will provide the smart data platform for enterprise IT, purpose-built for AI." Ob es spürbare Auswirkungen auf das Java-Ökosystem hat, wird sich zeigen. Aber es wird zumindest die Bedeutung von Java nicht schwächen – wenn schon nicht als wesentliche Grundlage für die „intelligenten Maschinen", dann zumindest für die ebenso wichtige „Treibstoffversorgung" *[1]*.

10. Dezember 2025

Und Azul kauft Payara

Bleiben wir doch gleich bei Firmenkäufen – Kontextwechsel kosten ja unnötig Zeit. Diesmal kauft Azul, der „JVM Vendor", den langjährigen Partner Payara, der auf der Basis von Glassfish den Jakarta EE-kompatiblen Application Server baut (und mit Payara Micro eine MicroProfile-optimierte Runtime). Zwei weitere Firmen, die rund um Open-Source ein erfolgreiches Geschäftsmodell gebaut haben. Die Zahlen zum Kauf sind hier im Gegensatz zu Confluent aber unter Verschluss. Das Ziel dahinter, so Azul selbst, sei der Aufbau eines durchgängigen, kommerziell unterstützten Open Source Stacks, von der JVM bis zum Application Server beziehungsweise Microservice-Framework. Bereits in den vergangenen Jahren waren die Angebote der Firmen miteinander verzahnt, insofern ist das konsequent. Beide Unternehmen waren bislang auch stark in die Arbeit an der Java-Basis eingebunden, Azul naturgemäß eher beim OpenJDK, Payara bei Jakarta EE und MicroProfile. Ich vermute, dass sich daran nicht viel ändern wird, auch wenn sie jetzt ein gemeinsames Unternehmen sind *[2]*.

2. Januar 2026

User CPU Time jetzt 95 % günstiger

Dieser Artikel eines schwedischen Forschers ist für Java-Nerds eine interessante Lektüre: Jonas Norlinder hat sich mit der „User CPU Time"-Messung in der JVM unter Linux beschäftigt, und herausgefunden, dass der aktuelle Ansatz darin besteht, „ein Spreadsheet auf Papier zu drucken, es dann mit einer Kamera einzuscannen, um das Foto dann wieder in ein Spreadsheet zu konvertieren und abschließend einen einzelnen Wert auszulesen." Also sinnbildlich, nicht wörtlich zu verstehen.

Bislang wurden diese Metriken (entsprechend der Linux-Philosophie „alles ist ein File") aus virtuellen „Dateien" unter /proc ausgelesen. Norlinder stieß bei seinen Analysen darauf, dass es schon seit vielen Jahren einen deutlich einfacheren beziehungsweise schnelleren Weg gibt, mit dem sich die Metrik 95 % schneller ermitteln lässt, und hat diesen als Patch ins OpenJDK eingebracht (geplant für Release 26). Für die meisten Anwendungen in Produktion wird es keinen nennenswerten Unterschied machen, aber wenn es um detaillierte Performance-Messungen geht (nicht nur zur Test-/Tuning-Zeit), schon eher. Und wer weiß, so Norlinder, welche ähnlichen Performanceblocker noch in der JVM schlummern *[3]*?

5. Januar 2026

MicroProfile – Finanzen und Ziele für 2026

MicroProfile hat kürzlich seine strategischen Ziele für 2026 verabschiedet. Jetzt war der Finanzplan dran. Er ist einstimmig, aber eher halbherzig verabschiedet worden: die Hälfte der 10 stimmberechtigten Steering Committee Mitglieder hat ihre Zustimmung gegeben, der Rest hat sich gar nicht geäußert. Das liegt vielleicht auch daran, dass die einzige wirkliche Einnahmequelle die „Working Group Participation Fees" sind. Und die sind von 66,5 auf 50,5 Tausend Dollar im Jahr 2026 gesunken, weil Primeton und Red Hat ihre Mitgliedschaft nicht erneuert haben. Primeton war ein paar Jahre dabei, und sieht die Mitgliedschaft scheinbar nicht mehr als strategisch an. Red Hat gehört schon seit Jahren zu IBM, und da „Big Blue" bereits (durch Emily Jiang) vertreten ist, spart man sich jetzt die „Zweitvertretung" (John Clingan) und erhöht den Konzerngewinn um 10.000 Dollar vor Steuern (oder ungefähr 0,0001 % nach meiner Rechnung).

Aber wir schweifen ab, eigentlich wollte ich mehr über die strategischen Ziele reden: Da gibt es schon noch einige wichtige für 2026. Unter anderem sollen die Abstimmungen zu einem „Alignment" mit Jakarte EE abgeschlossen werden. Dazu passen diese technischen Ziele: Die „JWT Bridge" soll als potenzielle neue Spezifikation

evaluiert werden – als „Alignment" zwischen MicroProfile JWT und Jakarta Security – und zugleich als Blaupause für weitere Harmonisierungsbemühungen dienen: zum Beispiel für die „Config"-Spezifikation, die auch in beiden Mantelspezifikationen („Umbrellas") vorhanden ist und mit Jakarta abgestimmt werden muss. MicroProfile AI steht auch auf der Kandidaten-Liste (natürlich; „consider renaming" steht da auch noch); unter anderem geht es dabei um eine CDI-Erweiterung zur Integration von Langchain4J *[4]*. Darüber hinaus soll die Identifizierung von „fehlenden" Features im Vergleich zu Spring weitergehen (nicht nur zum Thema AI).

8. Januar 2026

Die Abschlusstabelle der Saison 2025

Den Tiobe-Index habe ich glaube ich schon lange nicht mehr angeschaut – zum Jahresbeginn gönne ich mir mal die Auswertung von 2025. „Beliebteste" Sprache (anhand der von Tiobe ausgewerteten Suchmaschinen-Anfragen) ist wie schon im Vorjahr Python – dem AI-Boom geschuldet. Dahinter steht zwischen C und C++ Java wieder auf dem 3. Rang. C und C++ haben dabei die Plätze getauscht. Von den Top 4 haben bis auf C alle ein bisschen verloren. Der größte (und einer der wenigen) Gewinner zum Vorjahr ist C#, dicht hinter C++. JavaScript bleibt direkt hinter dem Treppchen. Größter Verlierer des Jahres ist Go, das aus den Top 10 herausgefallen ist.

Aber egal wie spannend der Anblick der Tabelle ist, der ähnlich wie beim Fußball auch die Emotionen der „Fangruppen" anspricht: Die Methodik stand ja schon immer in der Kritik ob ihrer Aussagekraft. Und wenn ich heute etwas zu Fortran wissen muss, dann google oder binge ich das nicht, sondern frage Claude, Perplexity, Langdock, oder gleich den freundlichen Assistenten in der IDE. Das taucht dann nach meinem Verständnis nicht in der TIOBE-Statistik auf, wäre aber ein interessantes Upgrade der Methode. Ich stelle mir allerdings gerade die Anfrage bei AI-Provider XY vor: „Hey, wir sind von TIOBE, das steht für The Importance Of Being Earnest', und wir hätten gerne eine umfassende Auswertung eurer Nutzer-Queries in Bezug auf Programmiersprachen." Klingt nach einer Challenge.

10. Januar 2026

Was bringt 2026 für das JDK?

Zunächst mal ganz banal: Java 26 am 17. März. Mit Release 26 wird die Applet-API entfernt (wenn du die kennst, weißt du, dass du zum alten Eisen gehörst). Das hatte ich neben den „richtigen" Features aber schon beim letzten Mal erwähnt. Aber es kam tatsächlich noch ein JDK Enhancement Proposal (JEP Nr. 500) dazu, mit dem schönen Namen „Prepare to Make Final Mean Final". Ohne weitere semantische Diskussion: Es geht darum, zu verhindern, dass ein „final" Feld durch „deep reflection" doch nicht so ganz final ist. Erstmal hagelt es mit Java 26 nur Warnungen, mit einem zukünftigen Release soll das dann tatsächlich unterbunden werden – und müsste beim Startup explizit aktiviert werden. Und den hier hatte ich auch noch nicht erwähnt: JEP 516 „Ahead-of-Time Object Caching with Any GC". Damit sollen die Startzeitverbesserungen durch „Ahead-of-Time Class Loading & Linking" (JEP 483 aus Release 24) auch mit dem ZGC nutzbar sein.

Sonst ist von 2026 vermutlich kein Feature-Feuerwerk zu erwarten, aber in den bereits seit vielen Jahren laufenden Projekte dürften einige Features „heranreifen", die bislang maximal als Preview vorliegen: Angefangen mit Projekt Leyden (Startup Times, ein großes Cloud-Thema); das hatten wir ja auch gerade schon, aber über die GC-Kompatibilität hinaus wird es noch weitere Verbesserungen geben. Dann Projekt Loom – das bereits Großartiges geliefert hat (Virtual Threads!), und hoffentlich dieses Jahr endlich die Structured-Concurrency-API finalisiert. Valhalla (Value Types) wird vermutlich nicht vor Java 28 (also 2027) eine stabile Version liefern, arbeitet aber parallel an weiteren Performance-Verbesserungen (zum Beispiel „Nullness Marker"). Projekt Panama wird mit der Vector-API weitere Preview-Runden drehen und auf die Fertigstellung der Value Types warten. Babylon arbeitet weiter an Unterstützung für High-Performance- und GPU-nahe Workloads – mittels Arbeiten an Code Reflection/„Code as Data" usw. Angeblich soll es bald in eine Inkubationsphase gehen.

Und last but not least Projekt Amber, das sich um die vielen kleineren Produktivitätsverbesserungen kümmert: Hier wird an der Finalisierung des Pattern Matchings gearbeitet, aber vorerst wird es wohl nur weitere Previews geben. Die String Templates sind zuvor komplett zurückgezogen worden; auch an ihnen wird weitergearbeitet, aber wohl erst mal nichts geliefert.

17. Januar 2026

1 Milliarde Zeilen

Im JEP Café #25, einer OpenJDK-Show von José Paumard, in der aktuelle JEPs beleuchtet werden, widmet Paumard sich der vor einiger Zeit viral gegangenen „1 Billion Rows Challenge" (1BRC) von Gunnar Morling. Die Fragestellung ist, wie man so große Datenmengen effizient mit der JVM verarbeiten kann; aber nicht mit Optimierung um jeden Preis, sondern sauber und „ohne Tricks".

Paumard nutzt dazu die Memory-API von „Project Panama" (seit dem JDK 22) und zeigt Schritt für Schritt, wie man „off-heap" MemorySegments, MemoryLayouts und VarHandles nutzt, um Dateien jenseits der Gigabyte-Grenze im Speicher zu bearbeiten. Damit zeigt er eindrücklich, wie viel Performance heute auch ohne „Unsafe-Tricks" oder JNI-Code in Java steckt *[5]*.

6. Februar 2026

Der neue Glassfish ist da

Glassfish 8 ist gestern freigegeben worden. Er bietet jetzt unter anderem Unterstützung für Virtual Threads in HTTP-Thread-Pools und Managed Executors zur besseren Skalierbarkeit bei vielen parallelen Requests. Außerdem werden Jakarta Data Repositories unterstützt (JPA und NoSQL), und es gibt wichtige Sicherheits-Upgrades: Insbesondere wird MicroProfile JWT 2.1 integriert und mit Jakarta Security 4.0 verzahnt. Eckdaten: Unterstützt Jakarta EE 11 (Plattform) und benötigt mindestens Java 21 (getestet wurde auf Java 21 und 25).

12. Februar 2026

Java 27

Für das JDK 27 im Herbst gibt es ein erstes „Enhancement Proposal": JEP 527 „Post Quantum Hybrid Key Exchange (TLS 1.3)". Damit soll – mit neuen Verschlüsselungsalgorithmen – künftigen Angriffen mittels Quantencomputern vorgebeugt werden. Bis es so weit ist, sollen nach den diversen Voraussagen zwar noch 5 bis 20 Jahre vergehen, aber es wird schon länger davor gewarnt, dass Angreifer bereits jetzt „interessant" erscheinende Daten auf Vorrat abgreifen, um sie dann zukünftig einfach entschlüsseln zu können *[6]*.

20. Februar 2026

VSCode Plug-in: Java zu Kotlin

JetBrains hat gestern einen „Java to Kotlin Converter" für Visual Studio Code in dessen Marktplatz veröffentlicht, um mehr Entwicklerinnen und Entwickler zur syntaktisch schlankeren Java-Alternative zu bewegen *[7]*.

Batterien sind nicht inkludiert. Der Konverter hat wohl nicht viel mit dem klassischen Konverter aus IntelliJ zu tun, sondern arbeitet mit einem LLM, wie das heute halt üblich ist (und ein bisschen Spaß – sprich Non-Determinismus – reinbringt). Der Zugriff auf das LLM muss entsprechend noch konfiguriert werden; es kann aber ein lokales Modell sein (ollama, LM Studio …).

22. Februar 2026

Embabel – Agentische KI mit Spring

Rod Johnson, der „Vater" von Spring, hat etwas Neues auf die Beine gestellt: das „Embabel Agent Framework". Hinter der bescheidenen biblischen Referenz auf den Turmbau zu Babel steht eine Gruppe von Projekten zum Aufbau von dynamischen Agenten, die auf ein formal modelliertes „Goal Oriented Action Planning" basierend auf typsicheren Domänenmodellen setzen. Das Planen wird dabei nicht als (statischer) Workflow verstanden, sondern als regelbasierte Zustandsraum-Suche über Ziele, Preconditions und Effekte mit systematischem Replanning nach jeder Aktion. Der Ansatz ist nicht komplett anders, aber ein bisschen systematischer und deterministischer als das, was LangGraph und ähnliche Frameworks als Default bieten, und vermutlich passt das auch besser zur „Enterprise Java Welt".

Die Komponenten lassen sich natürlich austauschen. Durch Ersetzen des „GOAP Planners" mit dem auch im Framework enthaltenen „Supervisor Planner" kann man zum Beispiel auch auf freie LLM-Orchestrierung à la LangGraph setzen.

Implementiert ist Embabel im Wesentlichen in Kotlin und in kleineren Teilen in Java, und ist wie zu erwarten tief in Spring Boot integriert, mit neuen Annotationen wie @EnableAgents. Es ist aber kein kein offizielles Spring-Projekt. Neben dem Framework gibt es unter anderem Starter-Module, um schnell eigene Agenten-Flows zu erstellen und RAG-Werkzeuge. Ein besonderer Schwerpunkt des Projekts liegt darin, Agenten-Speicher und -Gedächtnis nicht als etwas Neues oder Separates zu denken, sondern als Vernetzung oder Projektion der Daten, die in existierenden (Enterprise-) Systemen bereits in Massen und verschiedenen Formen vorliegen, bevor die erste User-Interaktion mit einem Agenten passiert. Dazu gibt es ein eigenes Unterprojekt namens DICE („Domain Integrated Context Engineering") *[8]*.

[1] *https://newsroom.ibm.com/2025-12-08-ibm-to-acquire-confluent-to-create-smart-data-platform-for-enterprise-generative-ai*
[2] *https://www.azul.com/company/payara-acquisition/*
[3] *https://norlinder.nu/posts/User-CPU-Time-JVM*
[4] *https://github.com/langchain4j/langchain4j-cdi*
[5] *https://inside.java/2026/01/17/jepcafe25*
[6] *https://openjdk.org/jeps/527*
[7] *https://marketplace.visualstudio.com/items?itemName=JetBrains.j2k-vscode*
[8] *https://github.com/embabel*

Andreas Badelt
stellv. Leiter der DOAG Cloud Native Community
andreas.badelt@doag.org

Andreas Badelt ist seit 2001 ehrenamtlich im DOAG e.V. aktiv und hat dort inzwischen seine Heimat in der Cloud Native Community gefunden, wobei ihn das Java-Ökosystem bis heute fasziniert. Beruflich hat er von Ende des vorigen Jahrtausends an als Entwickler und später auch Architekt für deutsche und globale IT-Beratungsunternehmen gearbeitet. Seit 2016 ist er als freiberuflicher Software-Architekt unterwegs („www.badelt.it").

»DEV LAND«

THE NEXT GENERATION OF DEVELOPERS

VERPASST?

ON-DEMAND-TICKET SICHERN UND EIN JAHR LANG DEVELOPMENT-EXPERTISE MIT VORTRÄGEN UND UNTERLAGEN ABRUFEN.

#DEVLAND WWW.DEVLAND.EU

ARCHITECTURE AVENUE
AI VALLEY
CLOUD CLIFFS
DATA DOCKS
CAREER COAST

Wie du deine Softwareprojekte zuverlässig ruinierst – Ein Drama in mehreren Akten.

Teil 1: Lasse technische Schulden wachsen

Andreas Monschau, Haeger Consulting

Stetig bin ich auf der abenteuerlichen Suche nach ihnen – anwendbare Regeln oder Antipattern, mit denen man erfolgreich Softwareprojekte gegen die Wand fahren kann.

Diese Regeln und Anti-Pattern entspringen nicht meiner bunten Fantasie, sie finden sich in vielen Projekten, Unternehmen, Organisationen und Verfahren wieder – mit dieser Sammlung möchte ich zum einen erheitern, und zum anderen zur Reflexion anregen: Wie sieht es denn in deinem Projekt aus? Hast du weitere Beispiele für Regeln und möchtest sie mit mir und anderen teilen? Dann kontaktiere mich gerne!

Im ersten Akt unseres Dramas geht es um das illustre Thema „technische Schulden". Jeder kennt sie, denn sie sind unter anderem das, was passiert, wenn man heute „nur mal schnell" eine Abkürzung nimmt, und ab dann täglich über den umgestürzten Code stolpert, während alle behaupten, das sei „historisch gewachsen".

Lasse technische Schulden wachsen – sie schaffen Bindung

Aber eigentlich sind technische Schulden ja gar kein Problem, sondern ein stetiges Zeichen des Fortschritts: Wer heute sauber baut, hat ja morgen nichts mehr zu tun. Deshalb gilt stets: Lieber schnell liefern als richtig, Refactoring kann warten, denn der nächste Release kommt mit an Sicherheit grenzender Wahrscheinlichkeit.

Sauberer Code ist doch nur was für Greenfield-Projekte, in der Realität zählt doch nur Geschwindigkeit. Und wenn vielleicht irgendwann niemand mehr versteht, wie das gesamte System funktioniert, dann hat man endlich erreicht, was jedes Projekt anstreben sollte: Veränderungsresistenz durch Angst!

Konsequent könnt ihr dieses Anti-Pattern wie folgt leben:

- „Das fixen wir später" wird als valide Architekturentscheidung anerkannt
- To-do-Kommentare sind eine Form der Dokumentation
- Copy & Paste (von Code aus anderen Klassen des Projekts, aus dem Internet oder von einer KI erzeugt) spart Denkzeit
- Alte Workarounds werden nicht entfernt, denn sie könnten ja später noch wichtig sein.

Effektiv wird es ganz besonders, wenn die technischen Schulden zwar identifiziert werden, aber unsichtbar bleiben: Keine entsprechenden Tickets, keine Zeit für Refactoring, kein Ownership.

Wenn ihr in eurem Projekt die technischen Schulden wachsen lasst, werde ihr sehr bald vermehrt diverse Symptome feststellen: Zum Bei-

spiel fasst niemand mehr bestimmte Module an, da etwaige Nebenwirkungen nicht absehbar sind. Oder Änderungen dauern länger als geplant, egal, wie klein sie sind. Darüber hinaus brauchen neue Entwickler mehrere Monate, bis sie sich ein wenig sicher im Code fühlen – wenn überhaupt. Wenn Bugs behoben werden, erzeugen sie wieder neue Bugs, und Releases fühlen sich jedes Mal wie russisches Roulette an. Und trotzdem heißt es: „Jetzt ist leider nicht der richtige Zeitpunkt für Refactoring." Und hier spoilere ich gerne: Er kommt niemals.

Aber wie sieht das ganz konkret aus?

Wie kann man nun für technische Schulden im Code sorgen? Wir haben hier ein kleines Beispiel einer klassisches Service-Klasse, die ursprünglich ganz harmlos gewesen ist *(siehe Listing 1)*.

```
public void processOrder(Order order) {
  validate(order);
  save(order);
  notifyCustomer(order);
}
```

Listing 1: Beispiel einer klassischen Service-Klasse

Klingt sauber? Ist sauber. War sauber. Aber mit jedem Sprint kommt „nur mal schnell" noch etwas dazu *(siehe Listing 2)*.

```
public void processOrder(Order order) {
  if(order == null) {
        return;
  }
  if(!featureToggleService.isEnabled("NEW_ORDER_FLOW")) {
        legacyProcess(order);
        return;
  }
  if(order.getCustomer() != null &&
     order.getCustomer().getType() == CustomerType.SPECIAL) {
        applySpecialDiscount(order);
  }
  try {
        validate(order);
        enrichFromExternalSystem(order);
        save(order);
  } catch(Exception e) {
        log.warn("Order processing failed", e);
  }
  if(order.getAmount().doubleValue() > 10000) {
        notifySales(order);
  }
  notifyCustomer(order);
}
```

Listing 2: Schnelle Erweiterung oder auch: Entstehung von technischen Schulden

Dies ist im Grunde die perfekte technische Schuld in Codeform: Business-Logik, Fehlerbehandlung und Feature-Toggles werden wild gemischt, es ist nicht wirklich gut testbar, und wenn, dann sind sie entweder riesig oder fehlen ganz. Am Ende traut sich niemand mehr, die Methode anzufassen, weil man den Code nicht kaputtmachen möchte, oder weil er nicht wirklich verstanden werden kann, was aus jedem Refactoring-Versuch ein gewaltiges Risiko macht. Also gerne nachmachen!

Aber…

Eskalierende technische Schulden

Technische Schulden vermehren sich im Laufe der Zeit und führen dazu, dass sich das Entscheidungsverhalten eines Teams ändert: Denn das Team beginnt, sinnvolle Verbesserungen zu vermeiden, da jede Änderung oder Anpassung als massives Risiko wahrgenommen wird. Dadurch werden Innovationen sehr schnell als Gefahr wahrgenommen, die Entwickler beginnen, nur noch irgendwie Features umzusetzen und zu hoffen, dass das System nicht irgendwann in der Produktion vollends zusammenbricht. Aus technischen Problemen wird daher über kurz oder lang ein kulturelles Muster.

Allerdings sind technische Schulden nicht grundsätzlich falsch, sie werden allerdings dann besonders gefährlich, wenn niemand mehr weiß:

- wo sie liegen,
- warum sie existieren,
- wer sie verantwortet.

Oder zusammengefasst: Nicht die technischen Schulden ruinieren das Projekt, sondern die kollektive Entscheidung, sie zu ignorieren. Schaut gemeinsam drauf.

Wie sieht es in deinem Projekt aus?

Andreas Monschau
amonschau@haeger-consulting
www.haeger-consulting.de

Andreas Monschau ist im zweiten Jahrzehnt als Senior IT-Consultant bei der Haeger Consulting GmbH in Bonn tätig. Neben seinen Projekten leitet er das umfangreiche Traineeprogramm des Unternehmens und ist als Sprecher und Autor unterwegs.

JUG SAXONY DAY 2026

Jetzt Tickets sichern!

Sei dabei auf der größten IT-Community-Konferenz in Sachsen am 25. September 2026 in Radebeul!

Frühbuchertickets und Freitickets für Studierende und Auszubildende unter **jugsaxony.day** erhältlich!

Alle Infos zum JUG Saxony Day 2026 findest du unter **jugsaxony.day**

JUG SAXONY DAY

JUnit 6 – Evolution eines Testklassikers

Michael Thomas, CGI Deutschland

Die Menge an Aphorismen und Zitaten über das Testen von Software ist ähnlich groß wie die Liste an Frameworks, die dafür genutzt werden. Viele sind im Laufe der Zeit zu bloßen Floskeln geworden: Gemeinplätze, zwar oft mit Unterhaltungswert, aber ohne rechten Erkenntnisgewinn. Im Gegensatz dazu bilden Testframeworks nach wie vor, sofern richtig eingesetzt, das solide Fundament professioneller Software-Entwicklung.

Eines der ersten und bekanntesten überhaupt ist JUnit. Aus einem während eines Fluges von Kent Beck und Erich Gamma entwickelten Prototyp *[1]* hervorgegangen, etablierte es sich früh als leichtgewichtiges Framework für automatisierte Unit-Tests in Java und wurde rasch zum De-facto-Standard. Mit JUnit 4 erfolgte der Übergang zu annotierungsbasierten Tests statt Namenskonventionen, was die Nutzung vereinfachte und die Integration in moderne Build- und IDE-Werkzeuge verbesserte. Zusätzlich wurde mit dieser Version ein einheitlicher Testlebenszyklus eingeführt. JUnit 5 markierte schließlich eine grundlegende Neuausrichtung mit modularer Architektur, klarer Trennung zwischen Test-API und Laufzeitumgebung, erweiterbarem Ausführungsmodell sowie der Einführung von parametrisierbaren Tests und der Trennung von Assumptions und Assertions.

Heute ist JUnit nach wie vor das Rückgrat für automatisiertes Testen im Java-Ökosystem *[2]*, fungiert als Standard-Testframework in Spring, wird in so gut wie jeder Java-IDE unterstützt und bildet die Basis für weitere Frameworks wie Cucumber, Spock, JGiven oder Gauge, welche entweder direkt darauf aufbauen oder sich über die JUnit-Laufzeit-Mechanismen in die Testausführung integrieren.

Mit dem zunehmenden Einsatz KI-gestützter Werkzeuge zur Code-Generierung und -Transformation, durch die Implementierungen automatisiert entstehen oder sich in hoher Frequenz verändern, sind mehr denn je Werkzeuge mit klaren, deterministischen Rückmeldungen über das tatsächliche Laufzeitverhalten gefragt. Vor diesem Hintergrund stellt sich weniger die Frage nach einem weiteren grundlegenden Neuanfang, sondern vielmehr, wie JUnit seine etablierte Architektur an veränderte Plattform- und Projektanforderungen anpasst. Genau hier setzt JUnit 6 an, das am 30. September 2025 veröffentlicht wurde: Grund genug, sich einmal intensiver mit den Neuerungen zu beschäftigen. Der vollständige Beispielcode zu diesem Beitrag ist auf GitHub *[10]* veröffentlicht.

Zentrale Änderungen

Waren die Schritte von JUnit 3 zu JUnit 4 und weiter zu JUnit 5 von tiefgreifenden Veränderungen geprägt, stellt der Übergang zu JUnit 6 eher eine Evolution als eine Revolution dar. Als erstes Major-Release seit neun Jahren bringt es dennoch eine Reihe nennenswerter Neuerungen mit sich *[3]*:

- **Neue Mindestversionen:** JUnit 6 setzt Java 17 als Mindestversion voraus und beendet damit die Unterstützung für Java 8 und 11, was zur LTS-Strategie vieler Unternehmen passt. Für Kotlin gilt ein Minimum von Version 2.1, für Maven Surefire und Failsafe von 3.0.0.
- **Einheitliche Versionierung:** Mit JUnit 6 werden die Artefakte für Platform, Jupiter und Vintage endlich einheitlich in der Versionsnummer 6.0.x veröffentlicht, was das Abhängigkeitsmanagement vereinfacht und Inkonsistenzen reduziert.
- **Cancellation-API und Fail-Fast-Ausführung:** Mit der Cancellation-API führt JUnit 6 erstmals einen kooperativen Abbruchmechanismus für laufende Tests ein, der über Assumptions und Timeouts hinausgeht. Sie ermöglicht Fail-Fast-Strategien und einen geordneten Abbruch ganzer Testläufe, ohne Tests fälschlicherweise als fehlgeschlagen zu markieren.
- **Native Unterstützung für Kotlin-Suspend-Funktionen:** JUnit 6 erlaubt es erstmals, Testmethoden direkt als `suspend`-Funktionen zu deklarieren, wodurch asynchrone Kotlin-Tests ohne zusätzlichen Wrapper idiomatischer, lesbarer und besser in Cancellation-Mechanismen integriert werden.
- **Erweiterte String-zu-Objekt-Konvertierung in parametrisierten Tests:** JUnit 6 unterstützt bei Fallback-Konvertierungen nun neben Factory-Methoden und -Konstruktoren mit String auch solche mit `CharSequence`, was die API-Flexibilität erhöht, unnötige String-Allokationen vermeidet und die Integration mit textverarbeitenden APIs erleichtert.
- **Aufgeräumte Stacktraces:** JUnit 6 kürzt Stacktraces standardmäßig bis zur jeweiligen Test- oder Lifecycle-Methode, blendet Framework-interne Frames aus und erleichtert so die schnelle Lokalisierung der eigentlichen Fehlerursache, insbesondere in CI-Logs und großen Test-Suiten.
- **Migration der CSV-Unterstützung zu FastCSV:** Der Wechsel auf eine aktiv gepflegte und leistungsfähigere Bibliothek reduziert Edge-Case-Probleme und macht CSV-basierte parametrisierte Tests robuster und langfristig stabiler. Das grundlegende Parsing-Verhalten bleibt konsistent, bei fehlerhaftem CSV-Input können sich allerdings Meldungen geworfener Exceptions ändern und gegebenenfalls vereinzelt Anpassungen an bestehenden Tests und benutzerdefinierten Fehlerbehandlungen erforderlich machen.
- **API-Bereinigung und Entfernen von Altlasten:** Lange als veraltet markierte APIs wurden entfernt, was zu einem aufgeräumteren Kern führt, aber Anpassungen an bestehenden Erweiterungen und kundenspezifischen Integrationen bedingen kann. Alle JUnit-Module verwenden nun JSpecify-Nullability-Annotationen, um kenntlich zu machen, welche Methodenparameter, Rückgabewerte usw. „null" sein können. Ebenso wurde die Kommandozeilen-Struktur und das Exit-Code-Verhalten des `ConsoleLaunchers` vereinheitlicht, um ihn in automatisierten Umgebungen verlässlicher einsetzen zu können. JUnit Vintage als Engine zur Ausführung von Tests auf Basis von JUnit 3.8 und 4 ist nun deprecated.

Einige dieser Änderungen sind vor allem struktureller Natur, andere wirken sich unmittelbar auf den Alltag in Testprojekten aus. Im Folgenden werden einige Aspekte näher betrachtet, die praktisch besonders relevant sind.

JUnit 6 ins Projekt einbinden

Um JUnit ins eigene Projekt einzubinden, genügt die Referenz im entsprechenden Dependencies-Block der Build-Konfiguration. Wie bereits angesprochen, genügt hier nun eine einzige Definition der Versionsnummer für alle JUnit-Artefakte. Für die native Unterstützung von Kotlin-Suspend-Funktionen wird außerdem noch Kotlins Reflection-Bibliothek vorausgesetzt. *Listing 1* zeigt die entsprechende Maven-Konfiguration, *Listing 2* und 3 die für Gradle.

Damit sind die Voraussetzungen für die Nutzung von JUnit 6 geschaffen. Der nächste Abschnitt beleuchtet näher, wie sich das neue Release auf die Steuerung und Semantik der Testausführung auswirkt.

```
<properties>
      [...]
      <junit.version>6.0.2</junit.version>
      <kotlin.version>2.1.0</kotlin.version>
      [...]
</properties>

<dependencies>
      [...]
      <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter</artifactId>
            <version>${junit.version}</version>
            <scope>test</scope>
      </dependency>
      <dependency>
            <groupId>org.junit.platform</groupId>
            <artifactId>junit-platform-launcher</artifactId>
            <version>${junit.version}</version>
            <scope>test</scope>
      </dependency>
      <dependency>
            <groupId>org.jetbrains.kotlin</groupId>
            <artifactId>kotlin-reflect</artifactId>
            <version>${kotlin.version}</version>
            <scope>test</scope>
      </dependency>
      [...]
</dependencies>
```

Listing 1: pom.xml

```
[versions]
junit = "6.0.2"
kotlin = "2.1.0"

[libraries]
junit-jupiter = { module = "org.junit.jupiter:junit-jupiter", version.ref = "junit" }
junit-platform-launcher = { module = "org.junit.platform:junit-platform-launcher", version.ref = "junit" }
kotlin-reflect = { module = "org.jetbrains.kotlin:kotlin-reflect", version.ref = "kotlin" }
```

Listing 2: libs.versions.toml

Cancellation-API und Fail-Fast-Ausführung

Treten fehlgeschlagene Testfälle auf, kann es sinnvoll sein, die Testausführung frühzeitig abzubrechen. Im Unit-Test-Kontext spart dies vor allem Zeit und verkürzt die Roundtrip-Zeiten im Code-/Compile-/Test-Zyklus, bei Integrations-Testsuiten (JUnit ist trotz seines Namens schließlich nicht nur ein Unit-Testframework *[4]*), insbesondere im Cloud-Umfeld vermeidet ein solches Fail-Fast-Vorgehen den unnötigen Verbrauch von Ressourcen und damit letztlich auch Kosten.

Existierte Fail-Fast bislang lediglich als Option auf Build-Tool- und IDE-Ebene, unterstützt JUnit 6 dieses Prinzip erstmals explizit durch seine neue Cancellation-API. Sie ermöglicht einen kontrollierten und kooperativen Abbruch der Testausführung, berücksichtigt auch parallellaufende Tests und stellt die Ausführung von Cleanup-Logiken sowie ein sauberes Herunterfahren der für die Tests verwendeten JVM sicher. Gleichzeitig werden abgebrochene beziehungsweise gecancelte Tests semantisch klar von tatsächlich fehlgeschlagenen Testfällen abgegrenzt.

Kernstück des Mechanismus ist das `CancellationToken` *[5]*, das typischerweise vom Testrunner erzeugt, verwaltet und genutzt wird, in der Praxis also dem eingesetzten Build-Tool, CI-Server oder der IDE. JUnits `ConsoleLauncher` bietet über den Kommandozeilenparameter `--fail-fast` bereits entsprechenden Support, Maven Surefire unterstützt das Feature seit Version 3.5.4 *[6]* und für Gradle ist dies für Version 9.4.0 geplant *[7]*.

Die grundlegende Funktionsweise lässt sich anhand eines einfachen Testrunners nachvollziehen, in dem ein `CancellationToken` erzeugt und dann einem speziellen `TestExecutionListener` übergeben wird *(siehe Listing 4)*. In diesem wird nach Fehlschlag eines Tests das Token als `cancelled` markiert *(siehe Listing 5)* und folgende einfache Test-Klasse ausgeführt *(siehe Listing 6)*. So ist zu beobachten,

```
dependencies {
      [...]
      testImplementation(libs.junit.jupiter)
      testImplementation(libs.kotlin.reflect)
      testRuntimeOnly(libs.junit.platform.launcher)
      [...]
}
```

Listing 3: build.gradle.kts

```
public class CancellationApiExample {
	public static void main(String[] args) {
		CancellationToken token = CancellationToken.create();

		LauncherExecutionRequest request = LauncherDiscoveryRequestBuilder.request()
				.selectors(selectClass(CancellationTest.class))
				.forExecution()
				.cancellationToken(token)
				.listeners(new CancellationListener(token))
				.build();

		LauncherFactory.openSession().getLauncher().execute(request);
	}
}
```

Listing 4: CancellationApiExample.java

```
class CancellationListener implements TestExecutionListener {
	private final CancellationToken token;

	CancellationListener(CancellationToken token) {
		this.token = token;
	}

	public void executionFinished(TestIdentifier id, TestExecutionResult result) {
		if (id.isTest()) {
			System.out.println(id.getDisplayName() + " - " + result.getStatus());
			if (result.getStatus() == TestExecutionResult.Status.FAILED) {
				token.cancel();
			}
		}
	}

	public void executionSkipped(TestIdentifier id, String reason) {
		if (id.isTest()) {
			System.out.println(id.getDisplayName() + " - SKIPPED");
		}
	}
}
```

Listing 5: CancellationListener.java

```
public class CancellationTest {

	@BeforeAll
	static void beforeAll() {
		System.out.println("Setting up test class");
	}

	@AfterAll
	static void afterAll() {
		System.out.println("Tearing down test class");
	}

	@Test
	void testFails() {
		fail("Trigger cancellation.");
	}

	@Test
	void testWouldSucceed() {
		assertTrue(true);
	}
}
```

Listing 6: CancellationTest.java

```
Setting up test class
testFails() - FAILED
testWouldSucceed() - SKIPPED
Tearing down test class
```

Listing 7: Ausgabe der Testausführung

```
class SuspendFunctionExample {

	suspend fun invokeAddition(a: Int, b: Int): Int {
		delay(100L)
		return a + b
	}

	fun calculate(a: Int, b: Int): Int = runBlocking {
		invokeAddition(a, b)
	}
}
```

Listing 8: SuspendFunctionExample.kt

dass die Testausführung abgebrochen wird, aber trotzdem alle Aufräumschritte erfolgen *(siehe Listing 7)*.

Ein Abbruch laufender Testmethoden durch Prüfung des Token-Status innerhalb der Methode erscheint insbesondere bei langlaufenden Tests verlockend, ist jedoch nicht möglich, da die Test-API keinen Zugriff auf ein solches Token bietet. Als Alternative bietet sich eine Strukturierung in kleinere, mittels `@TestMethodOrder` logisch aufeinander aufbauende Testmethoden an.

Die Ausrichtung von JUnit 6 auf moderne Ausführungsmodelle zeigt ebenso das nachfolgende Feature.

Native Unterstützung für Kotlin-Suspend-Funktionen

Kotlins `suspend`-Funktionen sind das zentrale Sprachelement zur Unterstützung von Coroutines und ermöglichen asynchrone, nicht-blockierende Programmierung in einer sequenziell lesbaren Form. Eine als `suspend` deklarierte Funktion kann ihre Ausführung an bestimmten Punkten unterbrechen, ohne dabei einen Thread zu blockieren, und später an genau dieser Stelle fortgesetzt werden. Für Entwickler ist dies weitestgehend transparent und wird durch den Compiler gewährleistet; sie können asynchronen Code ähnlich wie synchronen Code schreiben. `suspend`-Funktionen dürfen dabei nur aus anderen `suspend` -Funktionen oder aus einem Coroutine-Kontext heraus aufgerufen werden und bilden damit die Grundlage für strukturierte Nebenläufigkeit in Kotlin *[8]*.

Im einfachen Beispiel in *Listing 8* ruft die Funktion `calculate(...)` die `suspend` -Funktion `invokeAddition(...)` auf, die mit `delay(...)` eine asynchrone, nicht-blockierende Verarbeitung simuliert, bevor sie die Summe der beiden Parameter zurückgibt.

In JUnit 5 sind Entwickler zum Testen von `suspend`-Funktionen auf Hilfskonstruktionen angewiesen, wobei typischerweise `runTest` aus der `kotlinx-coroutines-test`-Bibliothek zum Einsatz kommt *(siehe Listing 9)*.

Zu den Nachteilen dieses Ansatzes gehört neben der expliziten Syntax, die speziell in komplexeren Testsetups zu einer Vermischung von Testlogik und Infrastruktur-Code führen kann, auch die fehlende Framework-Integration, da JUnit keine Kenntnis davon hat, dass es sich um eine `suspend`-Funktion handelt und so Lifecycle-Hooks, Timeouts und Cancellation-Mechanismen nicht nahtlos mit dem Coroutine-Kontext zusammenarbeiten. Zusätzlich nutzt `runTest` einen eigenen Test-Dispatcher, der Zeit und Scheduling simuliert, was zwar schnelle und deterministische Tests ermöglicht, jedoch in asynchronen Szenarien mit Timeouts oder Cancellation vom Laufzeitverhalten in Produktion abweicht. Hinzu kommt die Abhängigkeit zur Kotlin-Testbibliothek.

JUnit 6 beseitigt diese Limitierungen durch die native Unterstützung von `suspend`-Funktionen in Test-Methoden *(siehe Listing 10)*. Der idiomatische Kotlin-Code verbessert die Lesbarkeit, gleichzeitig ist die Coroutine-Ausführung nun fest in das Framework integriert: Abgebrochene Tests beenden ihre Coroutines korrekt und `@Timeout` unterbricht die Ausführung kooperativ im Einklang mit der Coroutine-Semantik. Da JUnit selbst den Coroutine-Kontext bereitstellt, entfällt zudem der Einsatz spezieller Test-Dispatcher und externer Wrapper, wodurch sich das Timing-Verhalten besser am Produktivkontext orientiert.

JUnit 6 enthält darüber hinaus eine Reihe kleinerer, aber im Alltag spürbarer Verbesserungen. Dazu ein Beispiel im folgenden Abschnitt.

```
class SuspendFunctionExampleTest {

      @Test
      fun testInvokeAddition() = runTest {
            val example = SuspendFunctionExample()
            val result = example.invokeAddition(10, 20)
            assertEquals(30, result)
      }
}
```

Listing 9: SuspendFunctionJUnit5ExampleTest.kt

```
class SuspendFunctionExampleTest {

      @Test
      suspend fun testInvokeAddition() {
            val example = SuspendFunctionExample()
            val result = example.invokeAddition(10, 20)
            assertEquals(30, result)
      }
}
```

Listing 10: SuspendFunctionJUnit6ExampleTest.kt

```
public class Identifier {
      private final String value;

      private Identifier(String value) {
            this.value = value;
      }

      public String getValue() {
            return value;
      }

      public static Identifier of(String input) {
            return new Identifier(input.toString().toUpperCase());
      }
}
```

Listing 11: Identifier.java

```
class IdentifierTest {

     @ParameterizedTest
     @ValueSource(strings = {"abc", "def", "xyz"})
     void testUppercaseConversion(Identifier id) {
        assertEquals(id.getValue(), id.getValue().toUpperCase());
     }
}
```

Listing 12: IdentifierTest.java

```
public static Identifier of(CharSequence input) {
     return new Identifier(input.toString().toUpperCase());
}
```

Listing 13: Factory-Methode mit „CharSequence"

Erweiterte String-zu-Objekt-Konvertierung in parametrisierten Tests

Ein mächtiges JUnit-Feature sind parametrisierte Tests: Sie erlauben es, eine einzelne Testmethode mehrfach mit unterschiedlichen Eingabewerten auszuführen, was redundanten Testcode vermeidet. Häufig werden Domänenobjekte dabei als Stringrepräsentationen definiert, die durch JUnit mittels automatischen Aufrufs einer entsprechenden Factory-Methode in den Zieltyp konvertiert werden *[9]*.

Listing 11 zeigt eine solche einfache Klasse, die eine Factory-Methode `of(String)` bereitstellt.

Diese wird im Folgenden im mittels `@ValueSource` parametrisierten Test zur Umwandlung von Text in den Typ `Identifier` aufgerufen *(siehe Listing 12)*.

Oftmals sinnvoller ist allerdings eine Factory-Methode, die `CharSequence` nutzt, statt auf den konkreten Typ `String` festgelegt zu sein: Diese akzeptiert auch Alternativen wie `StringBuilder`, `StringBuffer` oder `CharBuffer`, was String-Allokationen gerade bei großen Zeichenketten unnötig machen kann und so Speicher spart und mehr Flexibilität bei der Nutzung zeichenbasierter APIs erlaubt *(siehe Listing 13)*.

Während in JUnit 5 eine solche Methode nicht nutzbar ist, wird in JUnit 6 nun auch diese Methodensignatur bei der automatischen String-zu-Objekt-Konvertierung berücksichtigt, wodurch Implementierungen von Domänenobjekten ihre Factory-Methoden allgemeiner und technikneutraler gestaltet werden können, ohne sich an den speziellen Bedarf von Tests anpassen zu müssen.

Zentrale Änderungen

Angesichts dieser Neuerungen stellt sich für bestehende Projekte die Frage, wie ein Umstieg auf JUnit 6 sinnvoll vorbereitet und durchgeführt werden kann. Der folgende Überblick skizziert dabei einen groben Leitfaden ohne Anspruch auf Vollständigkeit:

1. **Prüfung der Voraussetzungen:** Sicherstellen, dass die eingesetzte Java-Version (mindestens 17), gegebenenfalls Kotlin (mindestens 2.1) sowie Build-Tools und IDEs JUnit 6 vollständig unterstützen. Bevor dies nicht gewährleistet ist, sollte unbedingt von einer Migration abgesehen werden.
2. **Update der Dependencies:** Umstellen aller JUnit-Abhängigkeiten auf Version 6.0.x sowie konsequentes Entfernen oder Ersetzen nicht mehr unterstützter oder veralteter Module.
3. **Validierung:** Überprüfen und gegebenenfalls Anpassen kundenspezifischer Extensions und genutzter APIs sowie parametrisierter Tests und CSV-basierter Eingaben.

Fazit und Ausblick

JUnit 6 steht nicht für neue Testkonzepte, sondern für die konsequente Ausrichtung eines etablierten Testframeworks auf moderne Java-Plattformen, polyglotte Entwicklungsteams und langfristige Wartbarkeit bei überschaubaren Risiken, die primär aus der erhöhten Mindestanforderung an Java, Kotlin und Build-Tools resultieren.

Durch die Festlegung auf Java 17 als Mindestversion zwingt das Framework Projekte dazu, ihre Plattformentscheidungen zu überprüfen und technische Schulden in dieser Hinsicht offen zu adressieren, statt sie über Abwärtskompatibilität zu kaschieren. Gleichzeitig bleibt der Anpassungsaufwand für bestehenden JUnit-5-Testcode überschaubar. Ein Wechsel auf JUnit 6 kann somit auch gezielt als Hebel für eine technische Konsolidierung genutzt werden.

Auf Ebene der Testarchitektur stärkt JUnit 6 die Aussagekraft von Testergebnissen: Mit der Cancellation-API werden fehlgeschlagene, übersprungene und bewusst abgebrochene Tests klar unterschieden, wodurch große Test-Suiten gezielter gesteuert und ressourcenschonender ausgeführt werden können. Die native Unterstützung von Kotlin-suspend-Funktionen integriert asynchrone Tests sauber in das Framework und rückt deren Verhalten näher an den Produktivkontext. Insgesamt verschiebt sich der Fokus von impliziten Konventionen hin zu expliziten, architektonisch nachvollziehbaren Entscheidungen im Testdesign.

Im Zeitalter KI-gestützter Softwareentwicklung gewinnen automatisierte Tests zudem eine neue Bedeutung: Sie fungieren als Referenzsystem, an dem sich automatisch erzeugter oder veränderter Code messen lassen muss. Je weniger Entwickler jede Codezeile selbst schreiben oder prüfen, desto wichtiger wird eine präzise Testsemantik, die Fehlverhalten, Abbrüche und bewusste Einschränkungen klar unterscheidet. JUnit 6 verzichtet auf KI-spezifische Erweiterungen und gerade diese Zurückhaltung macht das Framework zu einer stabilen Grundlage für Entwicklungsprozesse,

in denen KI zwar Code erzeugt, die Verantwortung für Korrektheit und Wartbarkeit jedoch weiterhin beim Team liegt.

JUnit dürfte somit auch künftig weniger durch spektakuläre Einzel-Features auffallen, sondern durch kontinuierliche Evolution weiterhin das verlässliche Rückgrat und Fundament für automatisiertes Testen im Java-Ökosystem bleiben.

Quellen

[1] *https://martinfowler.com/bliki/Xunit.html*
[2] *https://devecosystem-2025.jetbrains.com/#methodology-and-data*
[3] *https://docs.junit.org/6.0.2/release-notes.html#v6.0.0*
[4] *https://dzone.com/articles/is-junit-only-for-unit-testing-what-about-system-t*
[5] *https://docs.junit.org/6.0.2/advanced-topics/launcher-api.html#launcher-cancellation*
[6] *https://marcphilipp.de/blog/2025/12/28/stf-milestone-7-safe-cancellation/*
[7] *https://github.com/gradle/gradle/issues/34184*
[8] *https://kotlinlang.org/docs/coroutines-basics.html#suspending-functions*
[9] *https://docs.junit.org/6.0.2/writing-tests/parameterized-classes-and-tests.html*
[10] *https://github.com/mithomas/java-aktuell-junit6*

Michael Thomas
CGI Deutschland
mi.thomas@cgi.com

Michael Thomas ist bei der CGI als Software-Architekt und Director Consulting Expert tätig. Mit mehr als zwei Jahrzehnten Erfahrung gestaltet und verantwortet er komplexe Softwaresysteme. Ganz gleich, ob KI oder nicht: Sein Schwerpunkt liegt auf klaren Architekturen, automatisierten Entwicklungs- und Delivery-Prozessen sowie nachhaltiger Softwarequalität. Dabei verbindet er technische Tiefe mit pragmatischem Blick für Machbarkeit und Betrieb.

Maven 4, Teil 3: Major Changes

Matthias Bünger

Im zweiten Teil der Artikelreihe haben wir die Major Changes von Maven 4 betrachtet. In diesem Teil werfen wir einen Blick auf weitere Minor Changes der neuen Version.

Properties für das Wurzelverzeichnis

Im vorherigen Teil haben wir gelernt, dass es in Maven 4 möglich ist, das Wurzelverzeichnis eines Maven-Projekts durch das neu geschaffene root-Attribut in der pom.xml oder die Existenz eines `.mvn`-Verzeichnisses zu definieren. Ist dieses definiert, kann man sich zwei neue `rootDirectory`-Properties zu Nutze machen, zum Beispiel für Pfadangaben auf Konfigurationsdateien. Die beiden Properties heißen `${project.rootDirectory}` und `${session.rootDirectory}`. Sie unterscheiden sich in ihrem Variablenscope: `project`-Properties beziehen sich auf die grundsätzliche Projektdefinition (quasi die pom.xml) und sind während der Validierungsphase verfügbar. Die `session`-Properties sind hingegen nur während des tatsächlichen Build-Prozesses verfügbar.

In *Listing 1* wird die `${project.rootDirectory}`-Property verwendet, um auf eine Exclude-Datei für das Maven-PMD-Plug-in, die im Wurzelverzeichnis des Projektes abgelegt ist, zu zeigen.

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-pmd-plugin</artifactId>
  <configuration>
     <excludeFromFailureFile>${project.rootDirectory}/
exclude-pmd.properties</excludeFromFailureFile>
  </configuration>
</plugin>
```

Listing 1

Man sollte bedenken, dass die beiden Properties keinen Wert haben, wenn das Wurzelverzeichnis nicht definiert ist. Neben den beiden `rootDirectory`-Properties gibt es die `${session.topDirectory}`-Property. Da dies eine Session-Property ist und somit nur während des tatsächlichen Build-Prozesses existiert, hat sie immer einen Wert. Dieser ist abhängig davon, wie man den Maven-Prozess aufgerufen hat. Führt man einen Build im gleichen Verzeichnis wie die pom.xml aus, so ist der Wert das aktuelle Verzeichnis. Nutzt man den `--file`-Parameter, entspricht der Wert dem Verzeichnis, in dem die angegebene pom.xml liegt.

Die bereits in Maven 3 bekannte `${basedir}`-Property ist auch in Maven 4 verfügbar und zeigt auf das Verzeichnis eines jeden einzelnen Projektes. Sie unterscheidet sich also bei Projekten mit mehreren Subprojekten für jedes einzelne Subprojekt, während die `${project.rootDirectory}`-Property immer den gleichen Wert hat. Zum Abschluss eine Warnung: Vor Maven 4 sind einige, schon immer für den internen Gebraucht gedachte Verzeichnis-Properties nicht mehr verfügbar. Wenn also beispielsweise `${executionRootDirectory}`, `${multiModuleprojectDirectory}` oder ähnliche Properties in der Konfiguration verwendet werden, so müssen diese für die Nutzung von Maven 4 angepasst werden.

Änderungen am Lifecycle

Der Maven-Lifecycle definiert verschiedene Phasen, in denen Plug-ins Aktionen ausführen können, zum Beispiel die `compile`-Phase. In Maven 3 haben einige wenige dieser „Hauptphasen" zudem Zusatzphasen, die vor beziehungsweise nach der eigentlichen Phase durchlaufen werden. Dabei ist jedoch weder einheitlich, welche Hauptphasen solche Zusatzphasen haben, noch sind sie einheitlich benannt. So hat beispielsweise die `test`-Phase keine Zusatzphasen, während zur `test-compile`-Phase eine `process-test-classes`-Zusatzphase existiert, die im Nachgang zu `test-compile` ausgeführt wird. Die `integration-test`-Phase hat sogar zwei Zusatzphasen, namentlich `pre-integration-test` und `post-integration-test`, von denen die `pre`-Phase vorher und die `post`-Phase im Nachgang ausgeführt wird. Diese verwirrende Situation hat zur Folge, dass man teilweise die Ausführung eines Plug-ins an die nachgelagerte Zusatzphase der vorherigen Hauptphase gebunden hat, da die eigentlich „semantisch korrekte" Hauptphase keine vorgeschaltete Phase besitzt.

In Maven 4 wird dieses Chaos ausgeräumt und der Lifecycle vereinheitlicht. Zukünftig hat jede Phase im Lifecycle eine `before:`- und eine `after:`-Phase. So haben Entwickler die Möglichkeit, in klar definierte Zusatzphasen Plug-ins auszuführen. Außerdem ist es mit Maven 4 möglich, eindeutig zu definieren, in welcher Reihenfolge verschiedene Plug-in-Ausführungen („executions") innerhalb der gleichen Phase ausgeführt werden. Die Reihenfolge kann mit Ganzzahlwerten in eckigen Klammern am Ende des Phasennamens angegeben werden. Ein höherer Wert bedeutet eine spätere Ausführung.

Das Beispiel in *Listing 2* zeigt die Änderungen exemplarisch. Dort sind drei Ausführungen für das `echo-maven-plugin` konfiguriert. Zum einen (Ausführung mit der ID `msg1`) wird nach der clean-Hauptphase die Meldung `after:clean phase` ausgegeben. Die anderen beiden Ausführungen definieren jeweils eine Meldung während der `compile`-Phase. Anhand der Reihenfolge (100 bei `msg2` beziehungsweise 200 bei `msg3`) ist erkennbar, dass die Meldung `compile phase` vor der Meldung `again compile phase` ausgegeben wird.

Die bekannten Lifecycle-Phasen gelten innerhalb jeden einzelnen Projekts. In Maven 4 werden zusätzlich Phasen bereitgestellt, um die Ausführung von Plug-ins vor beziehungsweise nach allem und/oder jedem Projekt durchzuführen. Angelegt an bekannte Test-Frameworks heißen diese `all` und `each`. Die zuvor beschriebenen `before:`- und `after:`-Phasen existieren auch hier.

Die `all`- und `each`-Phasen werden für jedes Projekt und Subprojekt ausgeführt, aber ihr Scope ist unterschiedlich:

- Die `each`-Phase kapselt die bekannten Phasen des Lifecycles (`validate`, `compile`, etc.) für jedes Projekt oder Subprojekt. Es dient also dazu, das Verhalten des Builds eines einzelnen (Sub-) Projekts zu spezifizieren.
- Die `all`-Phase umfasst den gesamten Build eines Projektes einschließlich seiner eigenen `each`-Phase und allen Phasen sämtlicher Subprojekte. Mit der `all`-Phase lässt sich also elegant eine

```
<plugin>
  <groupId>com.github.ekryd.echo-maven-plugin</groupId>
  <artifactId>echo-maven-plugin</artifactId>
  <executions>
     <execution>
      <id>msg1</id>
        <phase>after:clean</phase>
        <goals>
           <goal>echo</goal>
        </goals>
        <configuration>
           <message>after:clean phase</message>
        </configuration>
     </execution>
     <execution>
        <id>msg3</id>
        <phase>compile[200]</phase>
        <goals>
           <goal>echo</goal>
        </goals>
        <configuration>
           <message>again compile phase</message>
        </configuration>
     </execution>
     <execution>
        <id>msg2</id>
        <phase>compile[100]</phase>
        <goals>
           <goal>echo</goal>
        </goals>
        <configuration>
           <message>compile phase</message>
        </configuration>
     </execution>
  </executions>
</plugin>
```

Listing 2

Konfiguration erstellen, die genau einmal pro Projekt unabhängig der Hierarchie von Subprojekten ausgeführt wird.

Dies klingt sehr kryptisch, ist aber sehr intuitiv, wenn man es mit bekannten Test-Frameworks vergleicht:

- `before:all` wird vor jeder anderen Phase innerhalb des aktuellen Projekts ausgeführt. In einem Build mit mehreren Projekten wird die `before:all`-Phase des Elternprojektes vor irgendeiner anderen Phase eines Subprojektes ausgeführt.
- `after:all` wird ganz am Ende des Builds ausgeführt. In einem Build mit mehreren Projekten wird die `after:all`-Phase der Subprojekte vor der des Elternprojektes ausgeführt.
- `before:each` wird vor jeder Standard-Phase von jedem Projekt ausgeführt, während `after:each` nach jeder Standard-Phase ausgeführt wird. Sie eignen sich also sehr gut für vorbereitende beziehungsweise nachbereitende Aktionen (vergleichbar mit `SetUp` und `Teardown` bei Test-Frameworks).

Man sieht, dass mit den neuen Phasen der Build von Projekten beliebiger Komplexität zielgenau gesteuert werden kann.

Während in Maven 3 ein Build nur in einem Fehlerfall abbricht, bringt der neu eingeführten `--fail-on-severity`-Parameter (kurz `-fos`) die Möglichkeit, das Log-Level von Meldungen festzulegen, bei der ein Build abbrechen soll. Ruft man in Maven 4 seinen Build unter Verwendung von `-fos WARN` auf, so schlägt der Build fehl, wenn irgendwo im Build-Log eine Warnmeldung auftritt. Dies ist besonders hilfreich, um frühzeitig Probleme festzustellen, frei nach dem Motto „*A warning is an error in the making.*"

Mehr Flexibilität bringt zudem eine Neuerung bei der Verwendung von Profilen. Ruft man in Maven 3 den Build mit einem nicht definierten Profil auf, so schlägt der Build fehl *(siehe Listing 3)*.

Im Maven 4 Build gibt es nun die Möglichkeit, ein Profil als optional zu definieren, indem man ein Fragezeichen bei der Angabe hinzufügt. Ist das Profil nun nicht definiert, schlägt der Build nicht mehr fehl, sondern es wird ein eine Information am Anfang und am Ende des Builds ausgegeben *(siehe Listing 4)*.

Verbesserungen bei Projekten mit Subprojekten

Der Reactor ist die Komponente in Maven, die dafür sorgt, dass Subprojekte gefunden und in der richtigen Reihenfolge gebaut werden. Bei der Nutzung von Maven 3 gibt es jedoch einen Bug in dieser Komponente, der vermutlich jeden Entwickler schon einmal verwirrt hat. Der Bug tritt auf, wenn in einem Projekt mit mehreren Subprojekten, nennen wir sie A und B, bei dem eines der Subprojekte (B) das andere (A) als projekteigene Dependency verwendet. Außerdem ist der Build des benötigten Subprojektes (A) erfolgreich, wohingegen der Build des nutzenden Subprojektes fehlschlägt. In dieser Situation empfiehlt Maven dem Nutzer, dass er den Build (nach Behebung des Fehlers) fortsetzen kann, indem der Befehl `--resume-from :B` verwendet wird. Doch versucht man dies, schlägt der Build direkt wieder fehl, weil der Reactor das Subprojekt A nicht mehr findet. Selbst der explizite Hinweis, dass A benötigt wird (`--also-make A`) hilft in dieser Situation nicht. Dieser Bug ist in Maven 4 endlich behoben, sodass die Kombination der beiden Befehle nun korrekt den Build von der Stelle, an der er fehlgeschlagen ist, erfolgreich fortgesetzt werden kann. Um dem Entwickler das Leben weiter zu vereinfachen, kann man in Maven 4 anstelle von `--resume-from B` auch einfach `--resume` (oder noch kürzer `-r`) als Parameter nutzen, um das gleiche Ergebnis zu erreichen. In parallelen Builds werden in Maven 4 dabei auch alle Subprojekte übersprungen, die bereits erfolgreich gebaut wurden.

Eine weitere grundlegende Verbesserung ist, dass der Reactor selbstständig Subprojekte erkennen und analysieren kann, wenn das Wurzelverzeichnis des Projektes definiert ist (siehe oben). Dies ist besonders in zwei Szenarien hilfreich:

- Beim Ausführen eines Plug-ins oder Tools (zum Beispiel *jetty* zum lokalen Starten eines Webservers) im Verzeichnis eines Subprojekts, das Abhängigkeiten auf andere Subprojekte hat.

```
> mvn compile -Pnonexistent
[ERROR] The requested profiles [nonexistent] could not be activated or deactivated because they do not exist.
```

Listing 3: Fehlschlagen des Builds

```
> mvn compile -P?nonexistent
[INFO] The requested optional profiles [nonexistent] could not be activated or deactivated because they do not exist.
[...]
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------------------------------------------------------
[INFO] Total time:  0.746 s
[INFO] Finished at: 2024-12-14T13:24:15+01:00
[INFO] ------------------------------------------------------------------------------------------------------------------------
[INFO] The requested optional profiles [nonexistent] could not be activated or deactivated because they do not exist.
```

Listing 4

Diese werden in Maven 3 nicht gefunden und die Ausführung schlägt fehlt.

- Wenn man ausgehend vom Wurzelverzeichnis analog zum ersten Beispiel ein Plugin oder Tool in einem Subprojekt ausführt und dabei die pom.XML mit dem File-Parameter (`--file`) referenziert. In Maven 4 werden die benötigten Subprojekte nun endlich, so wie man es erwartet, gefunden.

Im Blog von Maven-PMC-Mitglied Maarten Mulders *[1]* sind die diversen Verbesserungen nochmals ausführlich mit Beispielen beschrieben.

Mit Maven 4 kommen zwei weitere kleine, aber feine Verbesserungen im Zusammenhang mit solchen Projekten, die das Verhalten von Maven endlich so ändern, wie man es intuitiv erwarten würde:

- Der Zeitstempel in allen Artefakten ist nun einheitlich über alle Subprojekte.
- Alle Subprojekte werden nun erst am Ende des Builds in das lokale beziehungsweise remote Repository übertragen, da die Default-Werte der Konfigurationswerte `installAtEnd` (Install-Plugins) und `deployAtEnd` (Deploy-Plug-in) in Maven 4 auf `true` gesetzt wurden. Dieses Verhalten kann in Maven 3 durch manuelle Anpassung der Konfigurationswerte ebenfalls erzielt werden.

Alternative Syntax für die POM-Datei

Immer mal wieder sieht sich das Maven-Team mit der Forderung verschiedener Nutzer konfrontiert, anstelle von XML ein anderes Format für die Projektdefintion zu verwenden. Manche Nutzer wünschen sich JSON, andere YAML und wieder andere noch andere Formate. Mit Maven 4 geht das Maven-Team einen Schritt auf diese Anwender zu. Zwar wird sich standardmäßig an der Verwendung von XML für die POM-Datei nichts ändern, aber es wird ein ModelParser Service Provider Interface (SPI) bereitgestellt. Dieses kann von Nutzern verwendet werden, um die anderen Dateiformate zu unterstützen. Die von einigen Maven-Maintainern erstellte „Mason"-Extension *[2]* ist eine der ersten Extensions, die dieses Feature nutzt und es Anwendern erlaubt, die Projektdefintion in YMAL, JSON5, TOML und HOCON vorzunehmen.

Erster Blick auf Maven 4.1: Mixins

Während Maven 4.0 sich in der Release-Candidate-Phase befindet, wird bereits an Features für Maven 4.1 gearbeitet. Eines davon wird die Einführung sogenannter „Mixins" *[3]* sein. Dieses Feature erlaubt es, die Projektkonfiguration aus verschiedenen Dateien zusammenzufügen oder umgangssprachlich ausgedrückt: Es wird möglich sein, mehrerer Parent-POMs zu nutzen. So wird es leichter sein, Teilkonfigurationen auszulagern und in verschiedenen Projekten wiederzuverwenden.

Fazit

Bis zum finalen Release von Maven 4 freut sich das Maven-Team über jedes Feedback von Nutzern, die die Vorabversionen von Maven 4 sowie das Migrationstool *[4]* testen. Die Übersicht über sämtliche Änderungen in Maven 4 *[5]* wird kontinuierlich aktualisiert.

Neben den großen Änderungen (siehe vorheriger Teil der Artikelreihe), wie der Einführung einer Consumer-POM und eine neue Modelversion, bringt Maven 4 auch viele Bugfixes und kleinere Verbesserungen, die den Nutzern das Entwicklerleben leichter machen. Dieser Weg wird in Maven 4.1 mit der Einführung von Mixins fortgesetzt. Die Zukunft von Maven sieht also so aus, wie die kommende Jahreszeit: sonnig 😊

Quellen

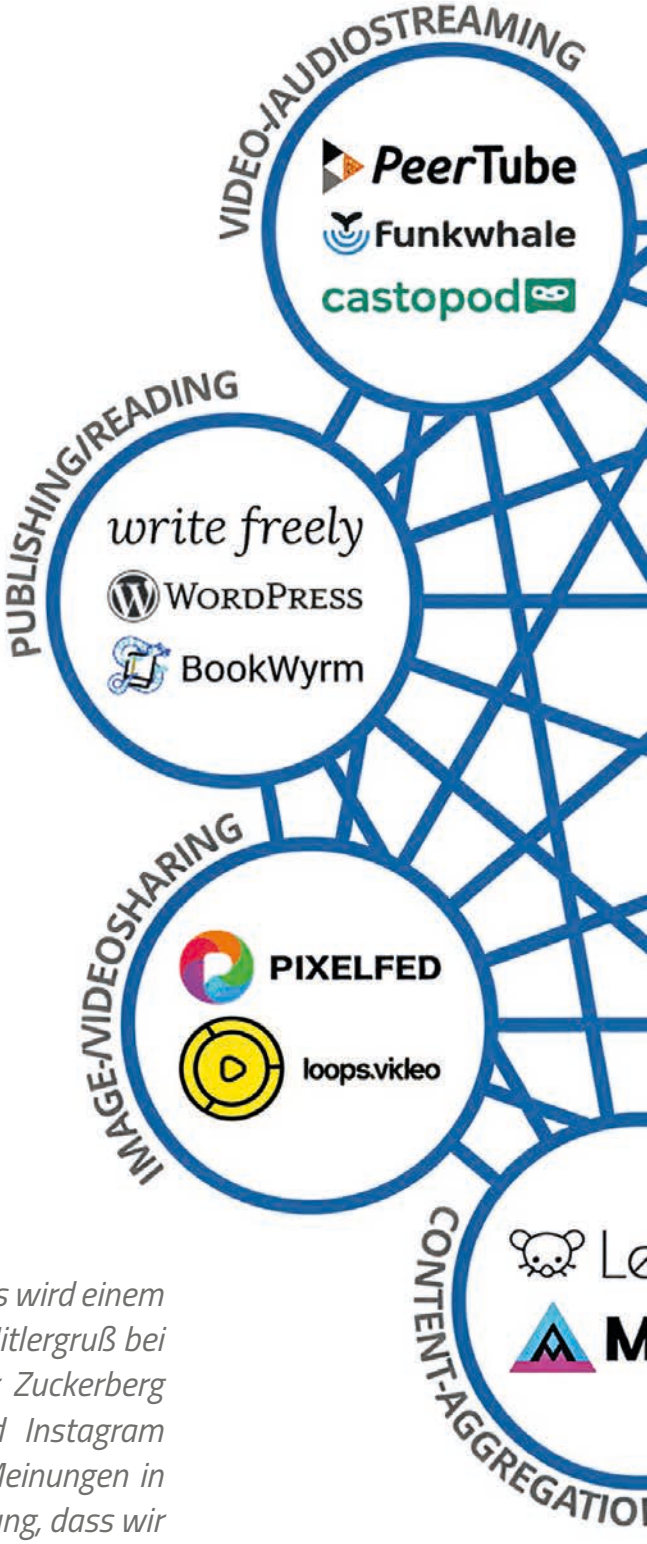
[1] *https://maarten.mulders.it/2020/11/whats-new-in-maven-4/*
[2] *https://github.com/maveniverse/mason*
[3] *https://maven.apache.org/guides/mini/guide-mixins.html*
[4] *https://maven.apache.org/guides/mini/guide-migration-to-mvn4.html*
[5] *https://maven.apache.org/whatsnewinmaven4.html*

Matthias Bünger
mbuenger@apache.org

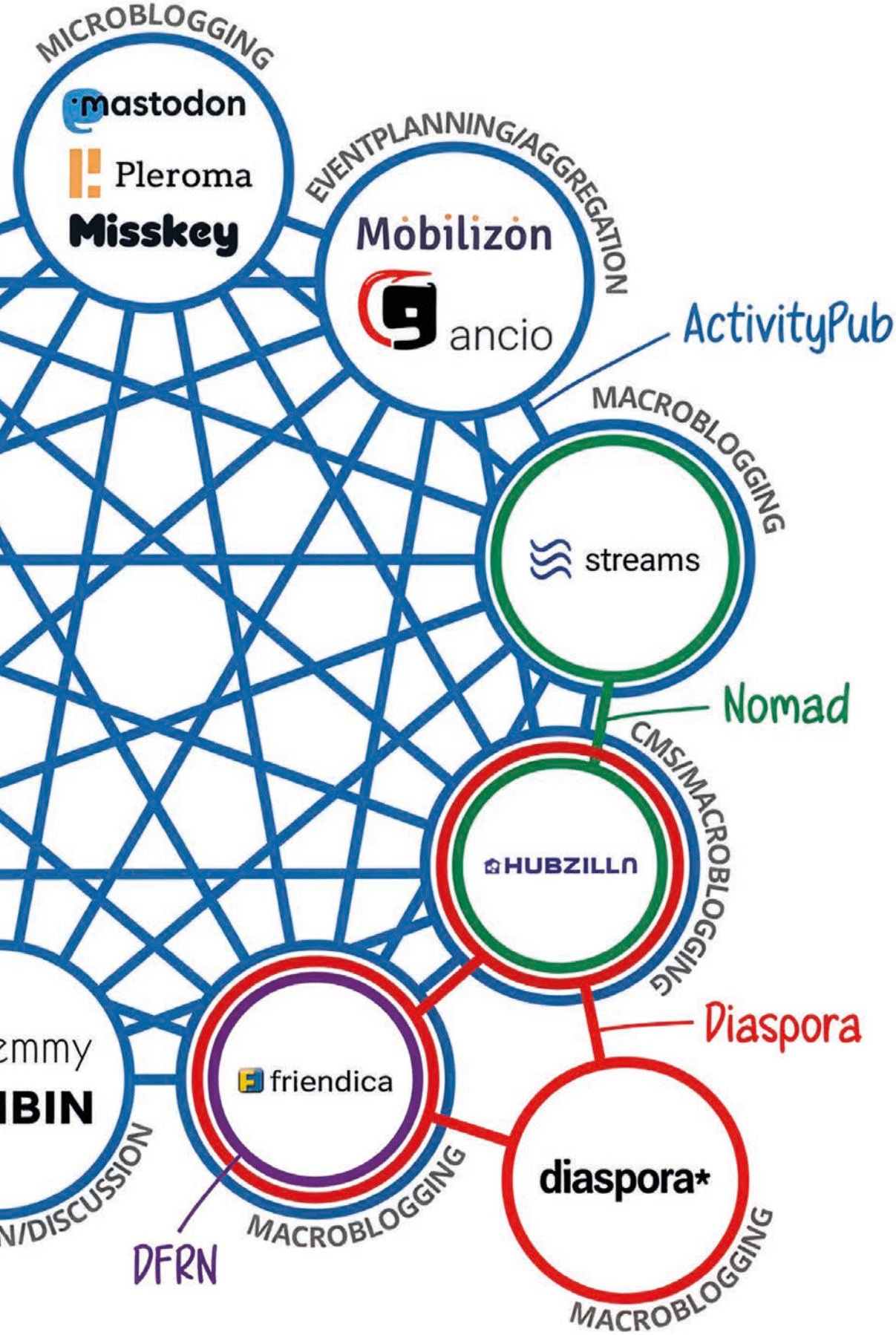
Matthias Bünger ist Softwarearchitekt und Teamleiter mit einer Vorliebe für Java und Testen. In seiner Freizeit unterstützt er Open-Source und ist Teammitglied von Apache Maven.

Fediverse – Social Media, dezentral und Community-driven

Jörg Liedl, SpringerNature AG



Wir alle lieben – mehr oder weniger – soziale Medien. Allerdings wird einem dies durch Macher wie Musk (X), der sich unter anderem mit Hitlergruß bei der Amtseinführung von Trump auf eine Bühne stellte oder Zuckerberg (Meta), der das Faktencheck-Programm bei Facebook und Instagram beendete, vermiest. Die Tatsache, dass immer mehr rechte Meinungen in die Timeline gespielt werden, stärkt immer mehr die Überzeugung, dass wir mehr freie Alternativen brauchen.


MICROBLOGGING
mastodon
Pleroma
Misskey
EVENTPLANNING/AGGREGATION
Mobilizon
ancio
ActivityPub
MACROBLOGGING
streams
Nomad
CMS/MACROBLOGGING
HUBZILLA
Diaspora
friendica
diaspora*
MACROBLOGGING
MACROBLOGGING
DFRN
emmy
IBIN
N/DISCUSSION

Digitale Souveränität und der Di.day

„Wir müssen unabhängig von den US-Tech-Firmen werden", hört man derzeit überall in den Nachrichten. Digitale Souveränität wird gefordert. Meist geht es bei den Diskussionen eher um Unternehmen und Verwaltung. Aber auch Einzelpersonen haben einiges, was sie noch ändern können. Marc-Uwe Kling, den viele als Autor der Känguru-Chroniken kennen, hat zusammen mit Save-Social, dem Chaos Computer Club (CCC) und einigen weiteren NGOs und Institutionen auf dem 39C3, dem jährlichen Treffen des CCC in Hamburg, den Digital Independence Day ausgerufen. Es gibt dazu eine Website *[1]*, auf der viele sogenannte Rezepte aufgeführt sind, die Alternativen aufzeigen, zu denen man wechseln kann: von Amazon zum lokalen Buchhändler, von WhatsApp zu Signal, von Microsoft Office zu LibreOffice und noch viele weitere.

Der Teil, den ich heute hier betrachten möchte, ist der Wechsel von Twitter/X zu Mastodon. Allerdings geht es hier im Artikel nicht nur um Mastodon, sondern um das Fediverse. Alle Möglichkeiten im Fediverse kann ich nicht aufzählen, das würde den Rahmen sprengen, aber es soll euch einen kleinen Überblick geben.

Abbildung 1: Fediverse Logo

Das Fediverse – Unterschiede zu Big-Tech Social Media

Bei den klassischen Sozialen Medien, wie Instagram, Facebook oder Twitter/X seid ihr in dem jeweiligen Netzwerk eingeschlossen. Ihr habt keine Möglichkeit, von Twitter zu Facebook zu kommentieren, oder bei Instagram mit einem Facebook-Account zu chatten. Jedes Netzwerk ist für sich abgeschlossen und das ist auch das Ziel, damit ihr dieses nicht verlasst. So werden gerne mal Posts, die andere soziale Netzwerke bewerben, mit weniger Reichweite bestraft, mit einem sogenannten Shadow-Ban.

Im Fediverse *(siehe Abbildung 1)*, dem **FED**erated Un**IVERSE**, ist es anders. Es ist egal, ob ihr Mastodon, Pixelfed, Lemmy, PeerTube, Loops, oder eine andere Software benutzt, alles ist dank des offenen, dezentralen ActivityPub-Protokolls vernetzt. Ihr könnt euch das ähnlich wie bei E-Mails vorstellen. Da ist es ja auch egal, ob Person A eine Posteon.de-, Person B eine Web.de- und Person C eine GMX-E-Mail-Adresse hat, ihr könnt alle dank der SMTP/POP3/IMAP-Protokolle untereinander E-Mails versenden und empfangen. Genauso verhält es sich im Fediverse, hier wird alles durch das ActivityPub-Protokoll übertragen. Das Protokoll ist ein W3C-Standard *[2]*.

Das ganze Fediverse ist dezentral aufgebaut. Es gibt also nicht nur einen Mastodon-Server, oder einen Pixelfed-Server, sondern es sind tausende kleine Server, sogenannte Instanzen. Es gibt kleine Instanzen mit nur einem Account, oder die größte Instanz, die Haupt-Mastodon-Instanz mit über drei Millionen Accounts *[3]*. Gerade diese Auswahl macht es vielen Leuten schwer, den richtigen Server und die richtige Plattform zu wählen. Das Fediverse ist unkommerziell, es gehört keinen Tech-Bros, sondern die Server werden von der Community fast immer kostenlos zur Verfügung gestellt, insgesamt gibt es derzeit über 42.000 verschiedene Fediverse-Instanzen. Es gibt kein Tracking, kein Algorithmus treibt die User:innen in eine Abhängigkeit und die Daten der User:innen gehören ihnen selbst. Man ist auch nicht auf seinem Server eingesperrt, wenn einem der Server aus welchen Gründen auch immer, nicht mehr gefällt, ist es möglich, einfach zu wechseln. So kann man von Mastodon.social ganz einfach zu Mastodon.de wechseln und umgekehrt. Das Gute dabei ist, dass man alle seine Follower mitnehmen kann, man fängt also nicht wieder bei null an.

Es gibt auch Bemühungen, dass man von einer Fediverse-Plattform zu einer anderen wechseln kann, aber das ist noch nicht wirklich überall funktionsfähig. Man braucht auch für die unten vorgestellten Plattformen nicht jeweils einen separaten Account. Es genügt eigentlich ein Account, am besten Mastodon, um Accounts von den jeweiligen Plattformen folgen zu können. Selbst das Kommentieren geht meistens von Mastodon aus, lediglich, wenn ihr selbst Videos zu Loops oder PeerTube hochladen möchtet, braucht ihr dort jeweils einen separaten Account. Die unten vorgestellte Fediverse-Software ist nur eine Auswahl. Eine kom-

Mastodon – joinmastodon.org

Geschrieben in Ruby + TypeScript, 991 Contributors

Mastodon *(siehe Abbildung 2)* ist die im Fediverse am meisten verwendete Software. Hierbei handelt es sich um einen Microblogging-Dienst, ähnlich wie Twitter oder Threads. Ihr habt die Möglichkeit, einen Text mit standardmäßigen 500 Zeichen abzusetzen. Außerdem könnt ihr Links, Bilder oder Videos in eurem Post mit anderen teilen. Diese Posts können von anderen User:innen gelikt, kommentiert, normal oder als Zitat geteilt werden. Alles ist so, wie ihr es bei Twitter geliebt habt, bevor es unbenutzbar wurde. Mastodon ist nicht nur die größte Plattform, sondern ist auch am weitesten im Mainstream angekommen. Beispielsweise ARD (ard.social), ZDF (zdf.social) und noch viele weitere haben schon ihre eigenen Instanzen. Der Bund *(social.bund.de)* bietet für die eigenen Institutionen wie Zoll, den Deutschen Wetterdienst, dem BSI, verschiedene Landtage und viele mehr sogar eine eigene Plattform. So haben die Bürger:innen eine weitere Möglichkeit, diesen Behörden zu folgen.

Abbildung 2: Mastodon-Logo

Pixelfed – pixelfed.org

Geschrieben in PHP + VUE.js, 181 Contributors

Pixelfed *(siehe Abbildung 3)* von Daniel Supernault ist 2018 angetreten, um das Instagram des Fediverse zu werden. Mittlerweile ist Pixelfed die am zweithäufigsten genutzte Software im Fediverse und bringt es auf mit fast einer Millionen Accounts auf 1/9 der Mastodon-Nutzer:innen. Ähnlich wie bei Instagram seht ihr in eurer Timeline nur Bilder oder Videos und keine reinen Textnachrichten von Mastodon. Es gibt jedoch die Überlegung, dies zu ändern und eine extra Timeline anzubieten. Also einmal die normale, aktuelle, mit Bildern und Videos und einmal eine Timeline, die alle Inhalte anzeigt, denen man folgt. Dies wird oft kritisiert. Das Besondere an Instagram seien die Storys, die 24 Stunden online bleiben und nur für Follower sichtbar sind. Pixelfed hat das auch (sehr rudimentär) umgesetzt. Allerdings könnt ihr in den Storys noch keine Leute markieren und was aktuell noch entscheidend gegen die Storys spricht: Sie werden nicht zu Mastodon oder anderer Fediverse-Software föderiert, das heißt, sie sind nur innerhalb von Pixelfed zu sehen.

Abbildung 3: Pixelfed-Logo

Loops – joinloops.org

Geschrieben in PHP + VUE.js, 15 Contributors

Loops *(siehe Abbildung 4)* ist das jüngste Produkt des Pixelfed-Gründers Daniel Supernault. Es ist derzeit noch in der Betaversion, hat aber schon sehr viele Funktionen. Derzeit wird noch das Hinzufügen von Musik aus freien Quellen zu den Videos implementiert. Hier könnt ihr, ähnlich wie bei TikTok, Videos hochladen oder anschauen. Die Videos sind recht kurz.

Zum nächsten Video kommt ihr, indem ihr das Video nach oben schiebt. Ihr könnt Videos liken oder kommentieren, genau wie in den anderen Fediverse-Anwendungen. Ebenso könnt ihr die Musik aus anderen Videos zu eurem eigenen hinzufügen und daraus ein neues kreieren, ganz so wie ihr es von TikTok kennt. Einen Nachteil hat Loops: Es gibt keine Möglichkeit, Posts, die nicht Loops basiert sind, zu sehen. Allerdings könnt ihr in anderen Fediverse-Plattformen Loops-Accounts ganz normal folgen und seht die Videos dann in eurer Timeline.

Abbildung 4: Loops-Logo

PeerTube – joinpeertube.org

Geschrieben in TypeScript, 515 Contributors

PeerTube *(siehe Abbildung 5)* wird von einer französischen, gemeinnützigen Organisation programmiert. Ähnlich wie bei YouTube könnt ihr hier Videos anschauen, kommentieren, Daumen rauf oder runter geben und eigene Playlists erstellen. Oder ihr könnt, statt nur zu konsumieren, eigene Videos hochladen. Ihr könnt mit eurem PeerTube, aber zum Beispiel auch mit eurem Mastodon-Account, kommentieren. Auch hier gilt, ähnlich wie bei Loops: ihr könnt von Loops aus nur andere PeerTube-Videos sehen, aber keine Mastodon-Posts lesen.

Abbildung 5: PeerTube-Logo

Lemmy – join-lemmy.org

Geschrieben in Rust, 256 Contributors

Lemmy *(siehe Abbildung 6)*, unter anderem nach dem Motörhead Sänger Lemmy Kilmister benannt, ist der Fediverse-Ersatz für Reddit. Ähnlich wie bei dem Original könnt ihr hier im guten alten Foren-Stil über Sachen diskutieren sowie Antworten mit Pfeil rauf und runter bewerten. Wenn ihr also an einem bestimmten Topic in Lemmy Interesse habt, könnt ihr dem Topic ganz einfach von Mastodon aus folgen. Das Kommunizieren von Mastodon aus ist aber bis jetzt nur sehr begrenzt möglich. Wenn man an einer Lemmy-Diskussion teilnehmen möchte, benötigt man einen Lemmy-Account.

Abbildung 6: Lemmy-Logo

Gancio - gancio.org

Geschrieben in TypeScript, Vue, 77 Contributors

Mobilizon - mobilizon.org

eschrieben in Elixir, Vue, Tpyescript, 254 Contributors

Gancio *(siehe Abbildung 7)* und Mobilizon *(siehe Abbildung 8)* versuchen beide dasselbe Problem zu lösen, weswegen ich sie hier zusammenfasse: Beide versuchen, einen Terminkalender für das Fediverse anzubieten. Es werden Events angelegt und je nach Einstellung können sie automatisch im Fediverse verteilt werden. Bei Gancio ist es sogar Standard, dass Events ohne einen Login angelegt werden können, sie benötigen dann aber noch eine Freischaltung. Jedes Event bei Gancio benötigt einen Flyer, weswegen es sich super für Konzertseiten anbietet und in dem Kontext auch oft verwendet wird. Bei Mobilizon ist das ganze eher wie bei Meetup.com. Ihr könnt bei Events teilnehmen, um es so in euren Kalender zu übertragen. Leider unterstützt Mastodon dies nicht, aber andere Services, die einen Kalender anbieten (Hubzilla, Friendica etc., im Artikel nicht vorgestellt), können damit interagieren. So haben Veranstalter einen Überblick oder können einstellen, dass nur eingetragene Teilnehmer auch wirklich teilnehmen dürfen, wie es bei vielen JUG-Meetups der Fall ist. Der iJUG verwendet auch Mobilizon, mehr dazu im Artikel von Marcus Filhon.

Abbildung 7: Gancio-Logo

Abbildung 8: Mobilizon-Logo

plette Auswahl findet ihr ebenfalls auf der oben genannten Website *[3]* sowie auf *[4]*.

Blick als Developer auf die Software

Die Software für die oben genannten, wie auch für alle anderen mir bekannten Fediverse-Plattformen sind Open-Source. Allein anhand der oben genannten Contributor-Zahlen ist schon zu sehen, dass Mastodon wesentlich weiter ausgereift ist als die anderen Projekte und ein großes Team dahintersteht.

Im Gegensatz dazu sind Pixelfed und Loops weniger ausgereift. Beide wurden von Daniel Supernault ins Leben gerufen, aber die Anzahl an Personen, die dauerhaft etwas zu den Projekten beitragen, ist sehr gering. Auch Supernaults Tag hat nur 24 Stunden – und wenn er sich intensiv um sein Loops-Projekt kümmert, bleibt natürlich nur sehr wenig Zeit für Pixelfed. Leider hat er sehr viele Ideen und entwickelt noch nebenher sehr viele großartige Fediverse-Projekte. Als User und Betreiber einer Pixelfed-Instanz ist das frustrierend. Also bleibt einem als Developer nur die Möglichkeit, selbst Hand an den Code zu legen.

Zuallererst schaue ich mir hierzu im Normalfall bei Projekten, bei denen ich einen Bug fixen oder eine Änderung vornehmen will, immer die Tests an. Allerdings ist bei Loops der Test-Ordner komplett leer. Bei Pixelfed sind die Test-Ordner nicht komplett leer, aber so gut wie. Dafür ist es schon sehr erstaunlich und bewundernswert, wie gut alles funktioniert. Für Java-/Kotlin-Entwickler ohne PHP-Background macht dies das Einarbeiten und das Unterstützen der Projekte durch Pull-Requests wesentlich schwieriger.

Man kann auch jederzeit selbst eine Instanz hosten. Dank Open-Source und einer mal mehr mal weniger guten Dokumentation ist dies im Handumdrehen möglich. Dies wird auch schon vom iJUG-Infrastructure-Team mit einigen Plattformen gemacht, dies könnt ihr am besten im zweiten Teil des Artikels von Marcus Fihlon nachlesen. P. S.: Ich habe damals das Angebot für meinen ersten Java-aktuell-Artikel auch von Marcus Fihlon über Mastodon erhalten, mit dem ich jetzt diese Artikel über das Fediverse schreibe.

Quellen

[1] *https://di.day*
[2] *https://www.w3.org/TR/activitypub/*
[3] *https://fedidb.com*
[4] *https://fediverse.party/*

Jörg Liedl
SpringerNature AG
joerg.liedl@rocket-design.de

Jörg Liedl arbeitet als Senior-Software-Entwickler bei dem Wissenschaftsverlag SpringerNature AG. In seiner Freizeit setzt er sich in der Musikszene für eine stärkere Verbreitung und Akzeptanz des Fediverse ein. Er stellt auch eine eigene Pixelfed-Instanz zur Verfügung und gab schon mit anderen zusammen einen Fediverse-Workshop. Ihr erreicht ihn im Fediverse unter @joergi@chaos.social.

Mit dem iJUG ins Fediverse: Digitale Unabhängigkeit mit Mastodon, PeerTube und Mobilizon

Marcus Fihlon

Die Java-Community lebt vom Austausch: über neue Technologien, über Erfahrungen aus Projekten, über Meetups und Konferenzen. Genau dafür brauchen wir heute digitale Räume, die nicht von einzelnen Konzernen kontrolliert werden, sondern der Community selbst gehören. Der vorhergehende Grundlagenartikel in dieser Ausgabe zeigt, was das Fediverse ausmacht und warum dezentrale, offene Plattformen an Bedeutung gewinnen. Das Infrastruktur-Team des iJUG betreibt seit einiger Zeit eigene Dienste im Fediverse und stellt sie den Java User Groups und ihren Mitgliedern kostenlos zur Verfügung. In diesem Artikel zeige ich, wie Mastodon, PeerTube und Mobilizon konkret genutzt werden können, um Informationen zu verbreiten, Events zu organisieren und die Java-Community über JUG-Grenzen hinweg besser zu vernetzen.

Warum der iJUG im Fediverse aktiv ist

Jörg Liedl beschreibt in seinem Grundlagenartikel in dieser Ausgabe, was das Fediverse ausmacht und warum immer mehr Menschen sich nach Alternativen zu zentralisierten Plattformen der großen Tech-Konzerne umsehen. Dieser Artikel knüpft daran an und zeigt die iJUG-Dienste im praktischen Einsatz. Für den iJUG *[1]* sind dabei drei Eigenschaften besonders relevant: Dezentralität, Offenheit und Unabhängigkeit.

Diese Prinzipien passen sehr gut zur Kultur der Java User Groups. JUGs sind keine zentral gesteuerten Organisationen, sondern lokal verankerte Communities, die sich selbst organisieren, eigene Schwerpunkte setzen und eigenständig Veranstaltungen durchführen. Der iJUG versteht sich als Dachorganisation, die diese Vielfalt unterstützt, vernetzt und mit gemeinsamer Infrastruktur entlastet, ohne den JUGs ihre Eigenständigkeit zu nehmen.

Das Fediverse spiegelt genau dieses Modell wider. Es gibt keine zentrale Plattform, die Regeln und Reichweite vorgibt. Stattdessen existieren viele unabhängige Instanzen, die miteinander föderieren und dennoch eigenständig bleiben. Offene Standards und Open-Source-Software sorgen dafür, dass die Kontrolle über Inhalte, Daten und technische Weiterentwicklung nicht bei einzelnen Anbietern liegt, sondern bei den Communities selbst.

Für den iJUG ist das kein ideologisches Statement, sondern eine bewusste Infrastruktur-Entscheidung. Wer als Community für Offenheit, Wissensaustausch und Unabhängigkeit steht, sollte diese Werte auch in den genutzten Kommunikations- und Publikationsplattformen widerspiegeln. Mastodon, PeerTube und Mobilizon bieten dafür eine tragfähige Grundlage.

Das Angebot des iJUG-Infrastruktur-Teams

Das Infrastruktur-Team des iJUG *[2]* stellt den Java User Groups und ihren Mitgliedern verschiedene digitale Dienste zur Verfügung, die gemeinschaftlich betrieben werden und auf Open-Source-Software basieren. Ziel ist es, den JUGs praxistaugliche Werkzeuge an die Hand zu geben, ohne sie in kommerzielle Plattformen oder Abhängigkeiten zu drängen.
Im Kontext dieses Artikels stehen drei Angebote im Fokus:

- **Mastodon** als Kommunikationskanal für Ankündigungen, Austausch und Vernetzung
- **PeerTube** als Videoplattform für Aufzeichnungen von Vorträgen und Community-Inhalten
- **Mobilizon** als Eventplattform zur Planung und Veröffentlichung von Veranstaltungen

Diese Dienste können von den Java User Groups genutzt werden, ohne selbst Infrastruktur betreiben zu müssen. Für JUGs und ihre Mitglieder sind sie kostenfrei verfügbar. Betrieb und Wartung erfolgen gemeinschaftlich durch das Infrastruktur-Team des iJUG.

Daneben betreibt der iJUG noch weitere Dienste, etwa für Cloud-Storage und Video-Meetings. Auf diese Angebote wird in Folge-Artikeln genauer eingegangen. Das Infrastruktur-Team ist zudem offen für neue Vorschläge aus der Community. Aktuell ist beispielsweise der Aufbau einer eigenen GitHub-Alternative auf Basis von Forgejo in Planung. Einen Überblick über Forgejo und die dahinterstehende Idee gibt der Artikel „Softwareprojekte selbst hosten mit Forgejo" in Ausgabe 4/25. *[3]*

Mastodon: Reichweite für JUGs und Mitglieder

Mastodon *[4]* eignet sich für Java User Groups und ihre Mitglieder als niederschwelliger Kommunikationskanal im Fediverse, vergleichbar mit einem offenen, dezentralen Microblogging-Dienst. Inhalte lassen sich schnell veröffentlichen und weiterverbreiten sowie mit anderen Instanzen im Fediverse teilen. Dadurch entsteht Reichweite auch über die eigene JUG hinaus *(siehe Abbildung 1 und 2)*.

Nutzung durch Java User Groups

Für Java User Groups eignet sich Mastodon als zentraler Kommunikations- und Vernetzungskanal. Veranstaltungen wie Meetups, Workshops oder Konferenzen lassen sich unkompliziert ankündigen, ebenso Call for Papers oder Calls for Participation. Auch die Kontaktaufnahme mit potenziellen Sprecherinnen und Sprechern funktioniert niedrigschwellig über direkte Interaktionen im Netzwerk. Kurzfristige Informationen, etwa bei Änderungen von Ort, Zeit oder bei Absagen, erreichen die Community schnell und ohne Umwege.

Beiträge lassen sich einfach teilen und werden von anderen Accounts im Fediverse weiterverbreitet. Dadurch beschränkt sich die Reichweite nicht auf die eigenen Follower. Ankündigungen werden auch von Interessierten außerhalb der lokalen Community wahrgenommen und weitergetragen.

Nutzung durch Mitglieder

Für Mitglieder bietet Mastodon eine einfache Möglichkeit, im Java-Umfeld auf dem Laufenden zu bleiben. Über abonnierte Accounts lassen sich Java-News, Ankündigungen aus der Community sowie Beiträge von Sprecherinnen und Sprechern verfolgen. Gleichzeitig bleibt man mit der eigenen Java User Group in Kontakt und kann auch anderen JUGs oder thematisch passenden Projekten folgen.

Durch die Teilnahme an fachlichen Diskussionen entsteht ein direkter Austausch über Themen, Projekte und Erfahrungen aus der Praxis. Wer Mastodon aktiv nutzt, erhält Einblicke in Entwicklungen und Veranstaltungen aus der gesamten Java-Community und nicht nur aus der eigenen lokalen JUG.

Zugang zum Mastodon-Server des iJUG

Der iJUG betreibt eine eigene Mastodon-Instanz für Java User Groups und ihre Mitglieder. Die Registrierung ist bewusst nicht öffentlich. Ziel ist es, die Plattform als Community-Angebot zu betreiben und nicht als allgemeines soziales Netzwerk für beliebige Dritte. Die iJUG-Instanz ist dabei vollständig in das Fediverse eingebunden. Inhalte können uneingeschränkt mit Accounts auf anderen Mastodon-Instanzen und mit Nutzenden anderer Fediverse-Dienste ausgetauscht werden.

Mitglieder einer Java User Group erhalten automatisch einen Mastodon-Account, sobald sie sich erstmals mit ihrer iJUG ID auf der iJUG-Instanz anmelden. Eine separate Registrierung ist nicht nötig. So wird sichergestellt, dass die Plattform der Community vorbehalten bleibt und gleichzeitig der administrative Aufwand gering bleibt.

Für offizielle Accounts von Java User Groups stellt das Infrastruktur-Team des iJUG auf Anfrage spezielle Einladungslinks zur Verfügung.

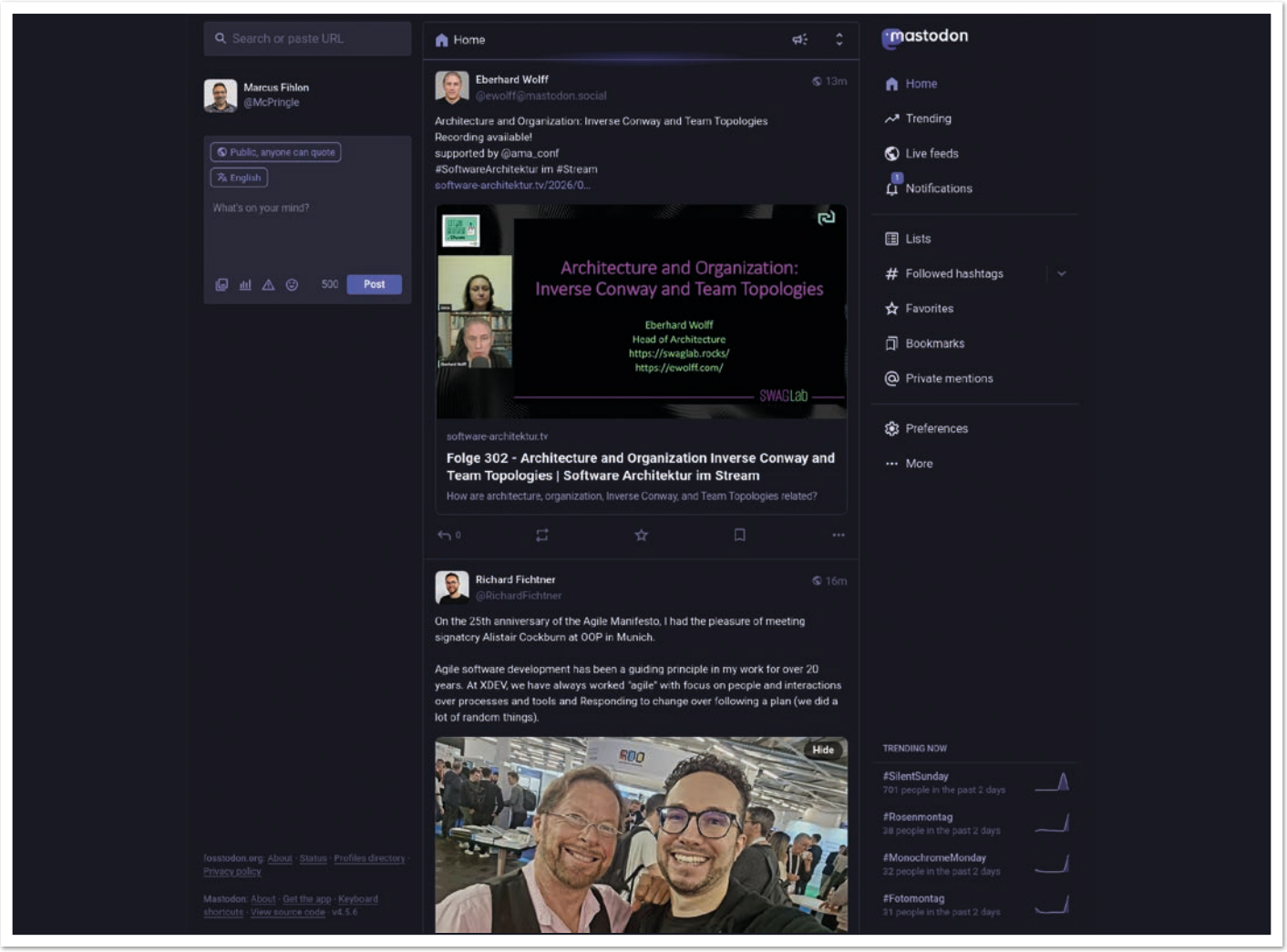


Abbildung 1: Startseite von Mastodon mit persönlichem Feed

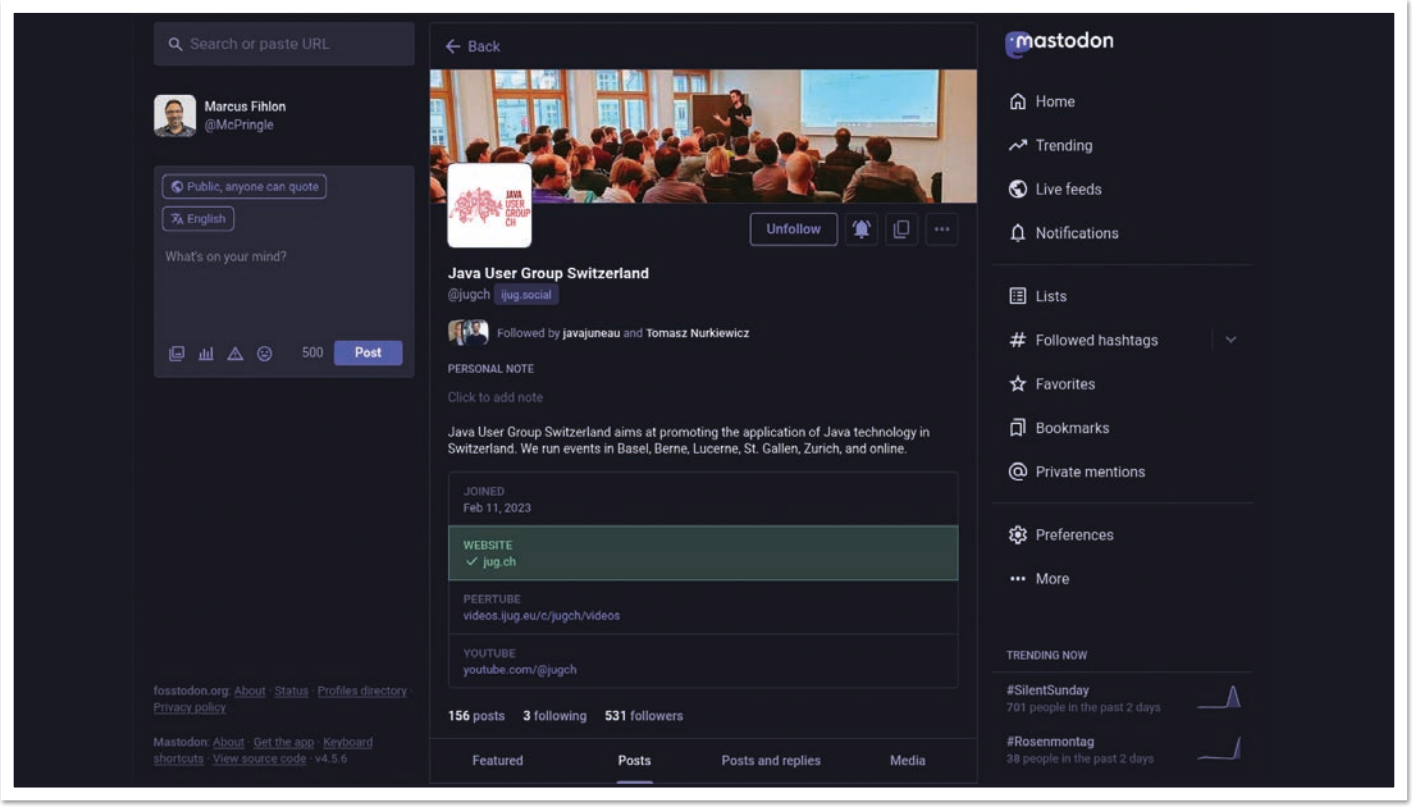


Abbildung 2: Ansicht eines Profils in Mastodon mit einer verifizierten Webseite (das gibt Sicherheit, dass dieses Profil „echt" ist)

Damit können JUGs ihre öffentlichen Profile für Ankündigungen und Community-Kommunikation einrichten.

PeerTube: Videos ohne YouTube-Abhängigkeit

Der iJUG betreibt mit *KaffeeFilm* eine eigene PeerTube-Instanz *[5]* als gemeinsame Video-on-Demand-Plattform der Java User Groups im iJUG für den Wissensaustausch im Fediverse. Sie dient als gemeinschaftliche Videoplattform für die Java-Community, auf der Vorträge, Talks und Tutorials unabhängig von kommerziellen Videoplattformen veröffentlicht und langfristig verfügbar gemacht werden.

PeerTube ist vollständig in das Fediverse eingebunden. Kanäle und einzelne Videos können instanzübergreifend abonniert und geteilt werden. Inhalte, die auf *KaffeeFilm* veröffentlicht werden, sind damit auch für Nutzende auf anderen PeerTube-Instanzen und im weiteren Fediverse sichtbar.

Nutzung durch Java User Groups

Java User Groups können PeerTube als zentrale Plattform für ihre Video-Inhalte einsetzen. Für die eigene JUG lassen sich eigene Kanäle betreiben, auf denen Vorträge, Talks oder Mitschnitte von Meetups veröffentlicht werden. So entsteht nach und nach ein gemeinsames Archiv von Inhalten, das langfristig verfügbar bleibt und nicht an einzelne kommerzielle Anbieter gebunden ist.

Nutzung durch Mitglieder

Auch für einzelne Mitglieder bietet PeerTube vielfältige Nutzungsmöglichkeiten. Kanäle auf beliebigen PeerTube-Instanzen lassen sich abonnieren, Vorträge und Talks können zeitunabhängig nachgeschaut und Inhalte mit anderen geteilt werden. Wer selbst Videos veröffentlichen möchte, kann dies ebenfalls über die Plattform tun. Die Nutzung ist dabei nicht auf die iJUG-Instanz beschränkt. Abonnements und Interaktionen funktionieren instanzübergreifend, sodass Inhalte aus dem gesamten PeerTube-Netzwerk verfolgt werden können.

Zugang zum PeerTube-Server des iJUG

Jedes Mitglied einer Java User Group im iJUG kann sich mit der eigenen iJUG-ID auf dem PeerTube-Server des iJUG anmelden. Existiert noch kein Benutzerkonto, wird dieses bei der ersten Anmeldung automatisch erstellt. Danach steht sofort der volle Funktionsumfang der Plattform zur Verfügung.

Angemeldete Nutzerinnen und Nutzer können eigene Kanäle anlegen und dort Videos veröffentlichen *(siehe Abbildung 3)*. So können beispielsweise Vertreterinnen und Vertreter einer JUG selbstständig einen Kanal für ihre Java User Group erstellen und direkt mit dem Veröffentlichen von Inhalten beginnen.

Als reine Konsumentinnen und Konsumenten von Videos ist kein Benutzerkonto erforderlich. Alle veröffentlichten Videos können öffentlich und ohne Anmeldung angeschaut werden. Kanäle, zum Beispiel von Java User Groups, lassen sich zudem mit anderen Fediverse-Konten abonnieren, etwa mit einem Mastodon-Konto. Neu veröffentlichte Videos erscheinen dann automatisch im eigenen Mastodon-Feed.

Zusätzlich unterstützt der PeerTube-Server des iJUG das Abonnieren von Kanälen über Podcast-Clients und RSS-Reader *(siehe Abbildung 4)*. Als Zuschauerin oder Zuschauer kann man so selbst entscheiden, über welchen Kanal und in welchem Werkzeug man neue Videos verfolgen und konsumieren möchte.

Um den Speicherverbrauch auf dem Server kontrollierbar zu halten und den stetig wachsenden Anforderungen gerecht zu werden, gelten pro Benutzeraccount ein Gesamtlimit von 50 GB für hochgeladene Videos sowie ein Upload-Limit von 5 GB pro 24 Stunden. Für

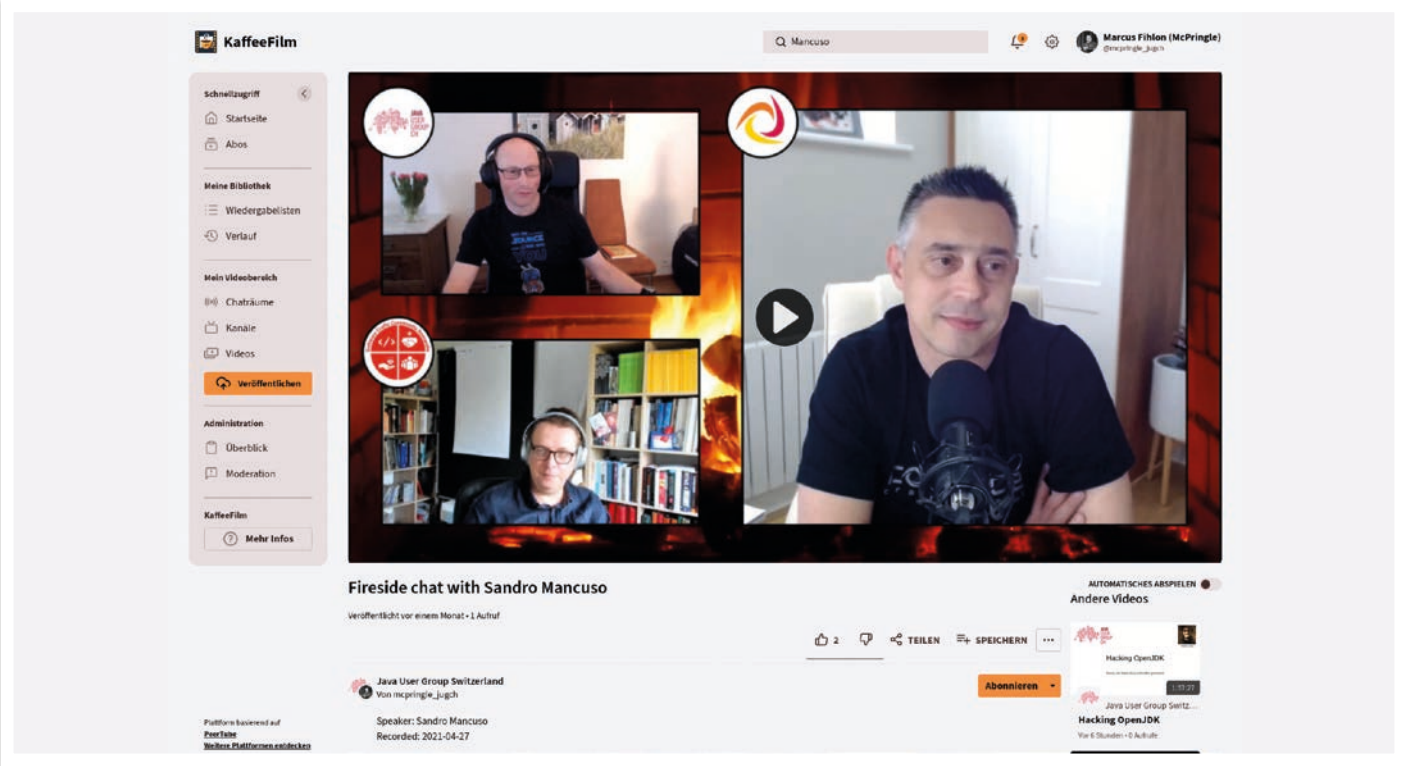


Abbildung 3: Ein Video auf der PeerTube-Instanz des iJUG

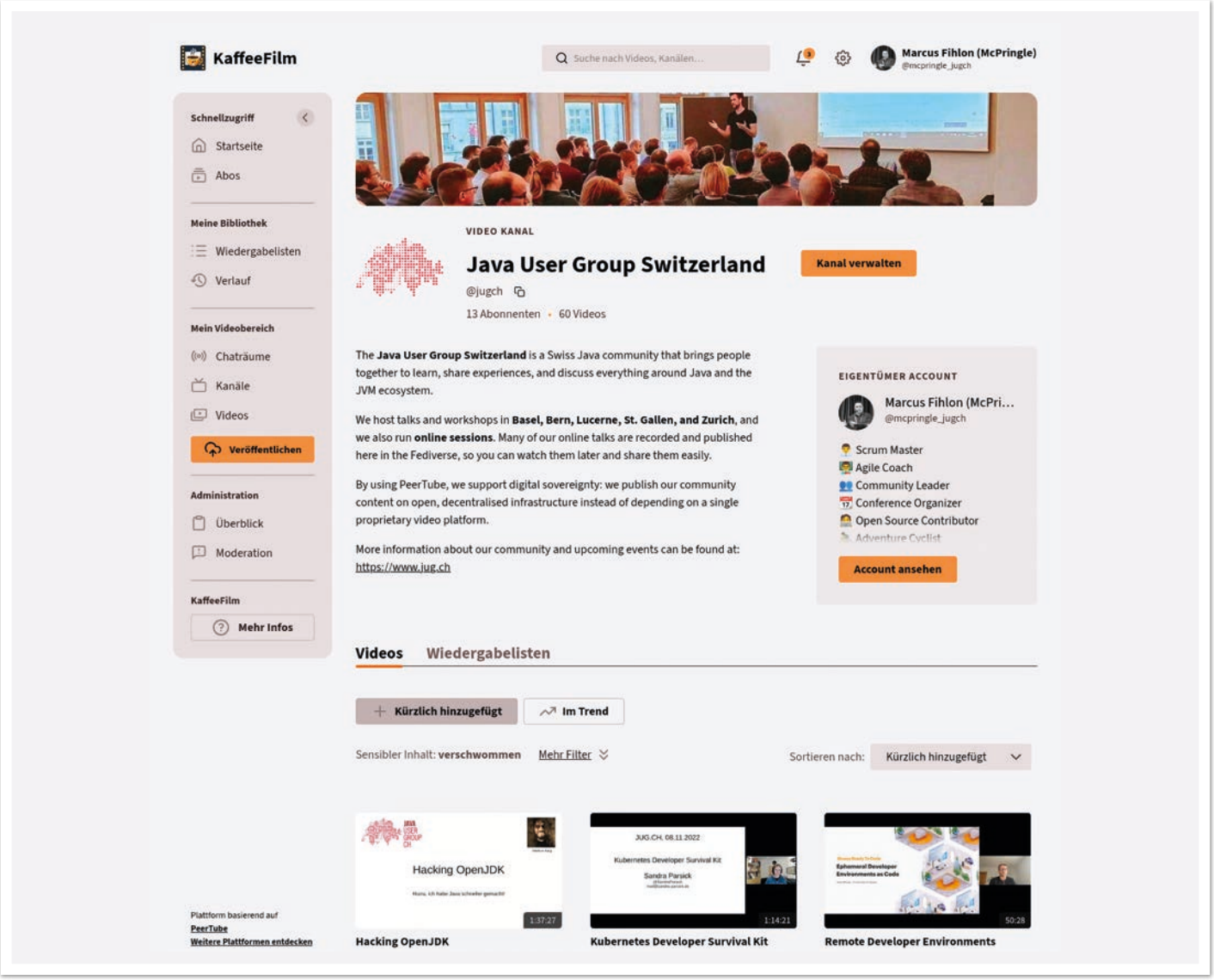


Abbildung 4: Ein Video-Kanal auf einer PeerTube-Instanz mit Beschreibung, weiteren Informationen und aktuellen Videos

das Anschauen und Abonnieren von Videos gibt es selbstverständlich keine Einschränkungen oder Limits. Reichen die Upload-Limits für einen Kanal nicht aus, können sie auf Anfrage beim Infrastruktur-Team des iJUG angepasst werden.

Außerdem bietet PeerTube umfangreiche Filtermöglichkeiten, um genau die Videos zu finden, die man sucht *(siehe Abbildung 5)*.

Mobilizon: Events im Fediverse planen

Mobilizon *[6]* eignet sich für Java User Groups als Plattform zur Planung, Verwaltung und Veröffentlichung von Veranstaltungen *(siehe Abbildung 6)*. Events lassen sich föderiert im Fediverse verbreiten und sind damit nicht nur auf eine einzelne Plattform beschränkt sichtbar. Für Mitglieder bietet Mobilizon eine einfache Möglichkeit, Veranstaltungen zu entdecken und in die eigene Planung zu integrieren.

Nutzung durch Java User Groups

Java User Groups können Mobilizon für die Organisation und Veröffentlichung ihrer Veranstaltungen einsetzen *(siehe Abbildung 7)*. Events lassen sich anlegen und verwalten, Teilnehmerzahlen im Blick behalten oder bei Bedarf begrenzen. Gruppen können ihre Mitglieder selbst verwalten und Veranstaltungen gezielt für bestimmte Zielgruppen freigeben. Durch die föderierte Struktur sind Events nicht nur auf der eigenen Instanz sichtbar, sondern können auch über andere Mobilizon-Instanzen und im weiteren Fediverse wahrgenommen werden.

Nutzung durch Mitglieder

Für Mitglieder bietet Mobilizon eine einfache Möglichkeit, Veranstaltungen zu entdecken und die eigene Teilnahme zu organisieren. Zusagen und Absagen lassen sich direkt über die Plattform verwalten. Termine können zudem in den eigenen Kalender integriert werden, sodass Veranstaltungen der eigenen JUG oder anderer Gruppen direkt in der persönlichen Planung erscheinen.

Zugang zum Mobilizon-Server des iJUG

Wie bei den anderen Diensten des iJUG ist auch bei Mobilizon die Anmeldung mit der iJUG-ID möglich. Nach der Anmeldung kann man Gruppen auf der iJUG-Instanz sowie auf allen anderen Mobilizon-Instanzen folgen. Zusätzlich ist es möglich, Gruppen beizutreten. Beim Folgen einer Gruppe erhält man Benachrichtigungen über neu angekündigte Veranstaltungen. Wer einer Gruppe beitritt, gilt als Mitglied und kann an Veranstaltungen teilnehmen, die nur für Gruppenmitglieder vorgesehen sind.

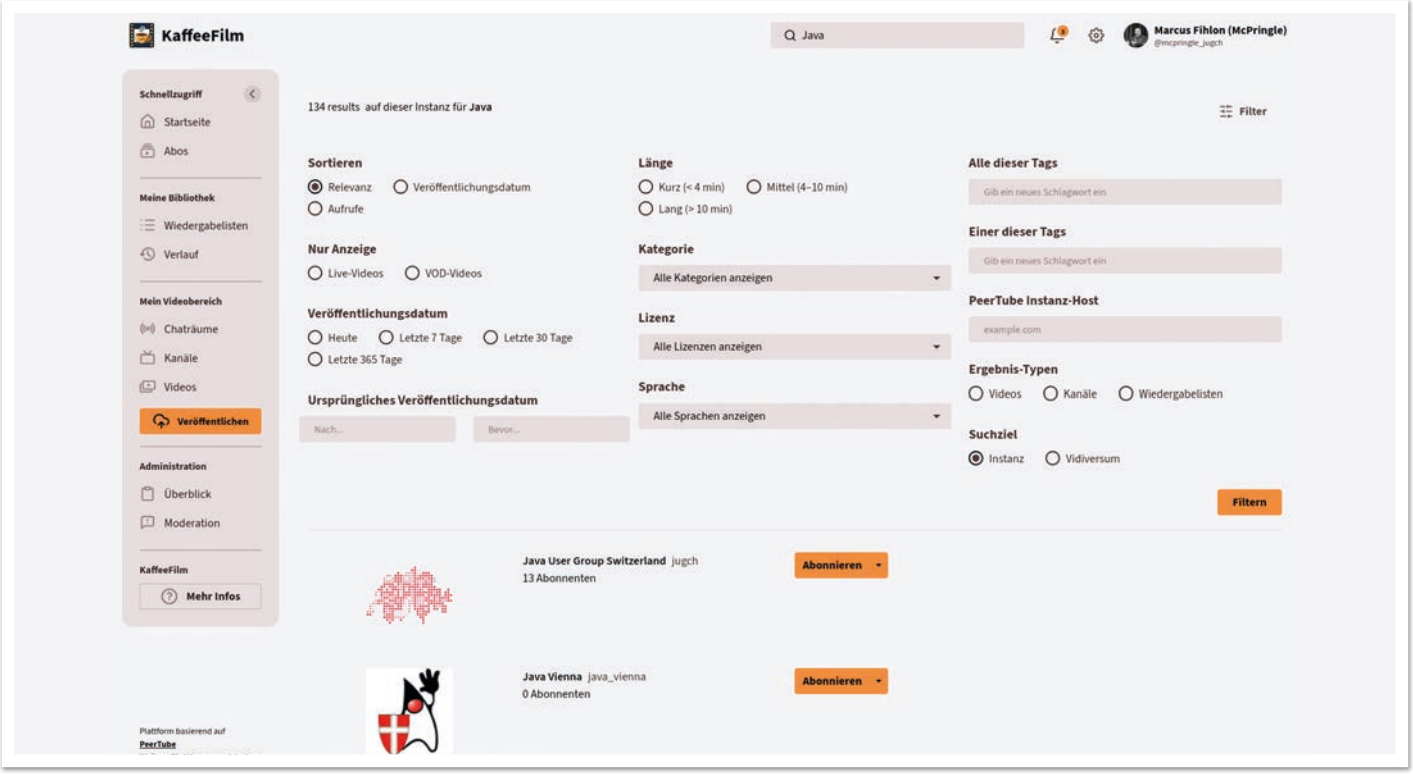


Abbildung 5: PeerTube bietet umfangreiche Filtermöglichkeiten, um genau die Videos zu finden, die man sucht

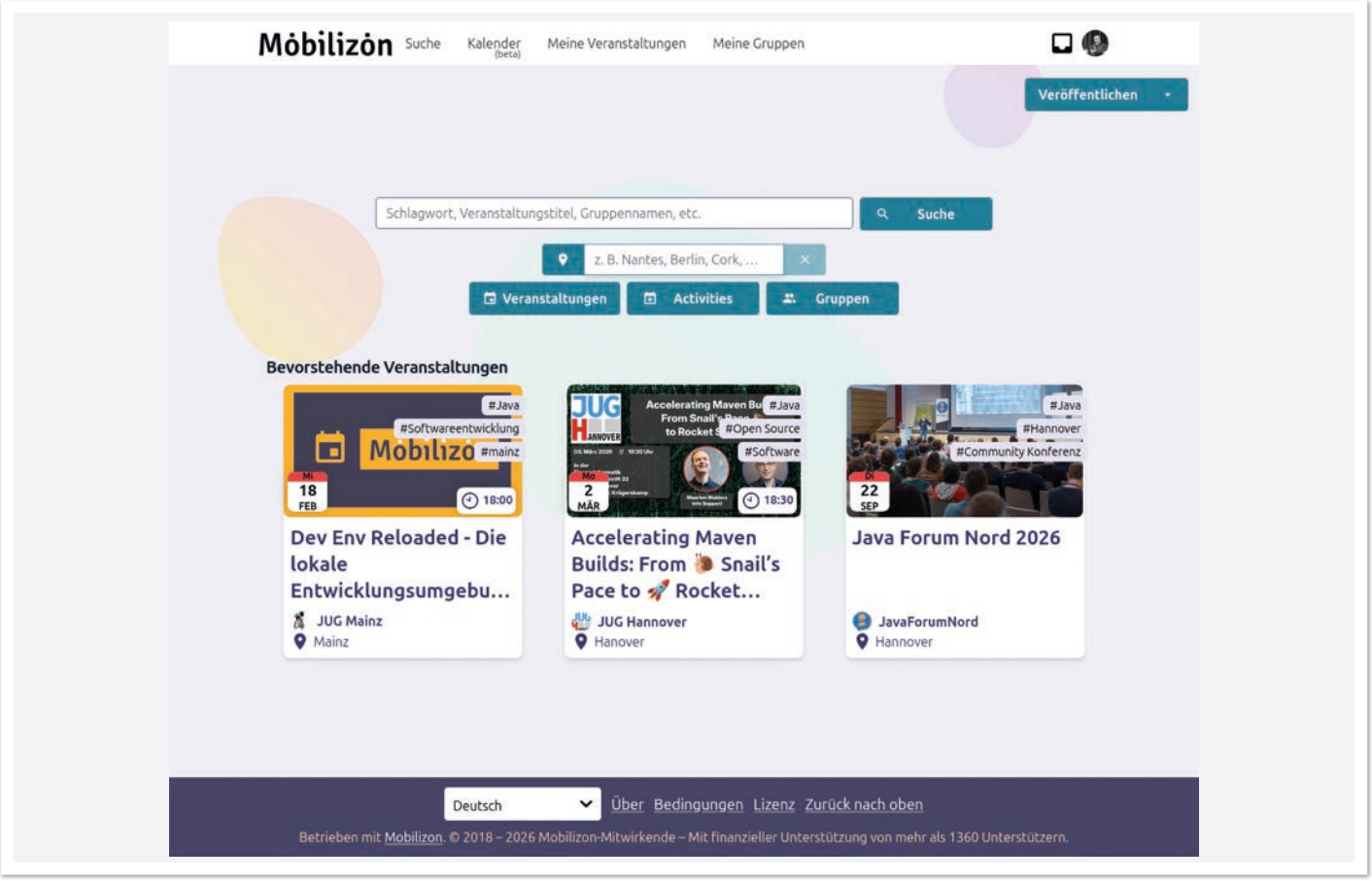


Abbildung 6: Startseite von Mobilizon mit Suchmöglichkeiten und den nächsten Veranstaltungen

Die Registrierung auf der Mobilizon-Instanz des iJUG ist bewusst offengehalten. Jede Person kann sich ein Benutzerkonto erstellen, Gruppen folgen und ihnen beitreten, auch ohne eigene iJUG-ID oder Mitgliedschaft in einer Java User Group. Dadurch können Veranstaltungen nicht nur von iJUG-Mitgliedern, sondern auch von Interessierten außerhalb der JUG-Community verfolgt und besucht werden.

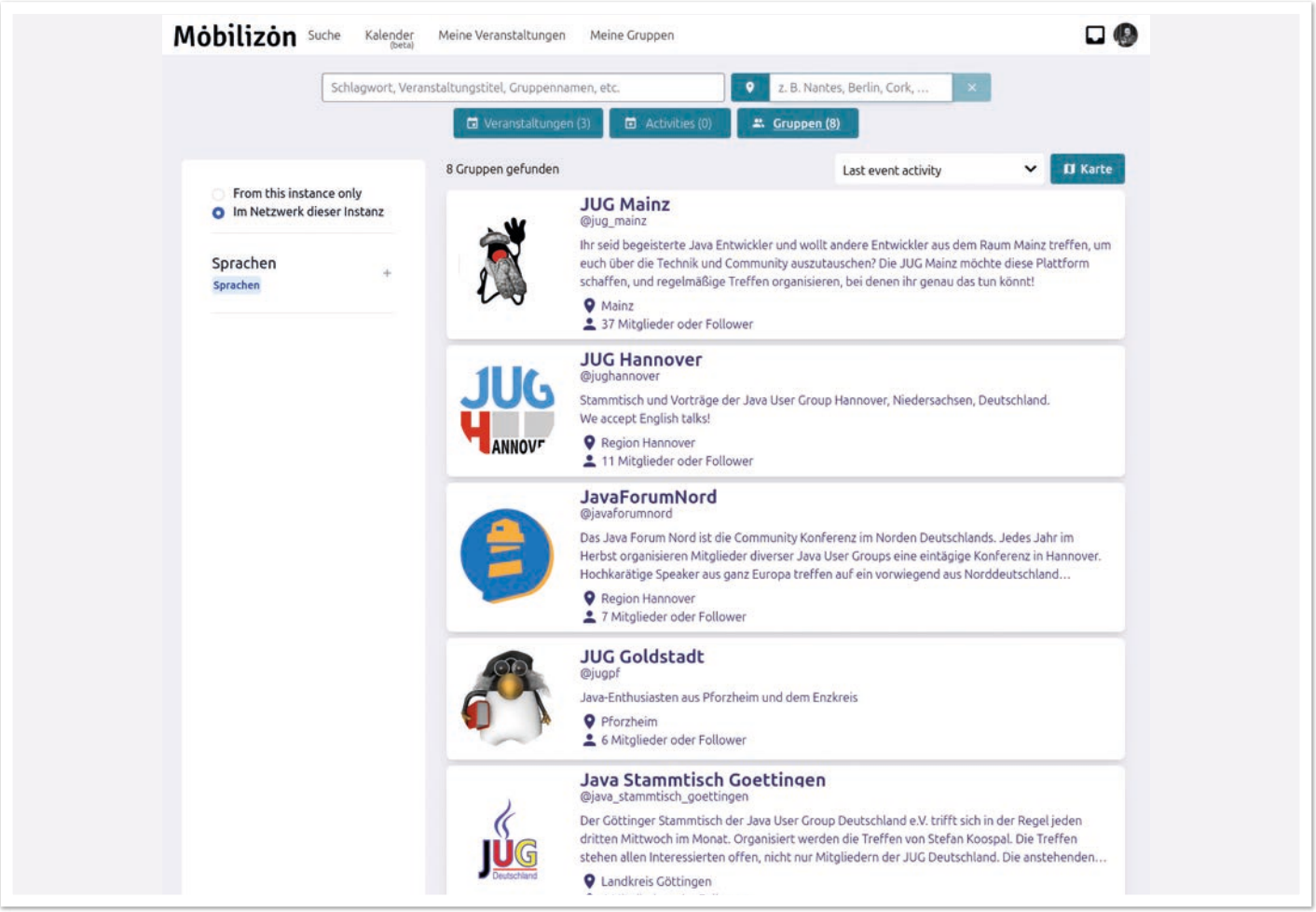


Abbildung 7: Auf Mobilizon kann man gezielt nach den Java User Groups des iJUG und ihren Veranstaltungen suchen

Gruppen auf Mobilizon lassen sich zudem mit anderen Fediverse-Accounts abonnieren, beispielsweise mit einem Mastodon-Konto. Neu angekündigte Veranstaltungen erscheinen dann automatisch im eigenen Mastodon-Feed. Zusätzlich können Gruppen über RSS- oder Atom-Feeds verfolgt oder über einen ICS- beziehungsweise Webcal-Feed direkt in den eigenen Kalender eingebunden werden. So kann man selbst entscheiden, über welchen Kanal man neue Events verfolgen möchte.

Die iJUG-ID als zentraler Zugang

Die iJUG-ID ist ein zentrales Login für verschiedene Dienste, die der iJUG den Java User Groups und ihren Mitgliedern zur Verfügung stellt. Technisch basiert dieses Single-Sign-On auf einer vom iJUG selbst betriebenen Keycloak-Instanz. Damit bleibt die Kontrolle über Identitäten und Zugriffsdaten vollständig in der Community und bei der Organisation selbst.

Jede Java User Group im iJUG verfügt in dieser Keycloak-Instanz über einen eigenen Identity Provider mit eigener Benutzerverwaltung. Vertreterinnen und Vertreter der jeweiligen JUG haben darin Administrationsrechte. Das lässt sich am ehesten mit einem eigenen Mandanten vergleichen. Dadurch bleibt der administrative Aufwand zentral gering, während die JUGs ihre Benutzerkonten dezentral selbst verwalten können.

Ein Benutzerkonto auf Mastodon, PeerTube oder Mobilizon wird dabei nicht im Voraus angelegt, sondern jeweils erst dann, wenn sich ein Mitglied zum ersten Mal mit der iJUG-ID beim jeweiligen Dienst anmeldet.

In der Praxis bedeutet das: JUGs vergeben die iJUG-IDs für ihre Mitglieder im Self-Service. Wer eine iJUG-ID benötigt, wendet sich an die Vertreter der eigenen Java User Group. Eine Übersicht aller dem iJUG angeschlossenen Java User Groups sowie deren Ansprechpartner ist auf der iJUG-Webseite zu finden *[7]*. Die iJUG-ID kann anschließend für die Anmeldung bei verschiedenen Diensten des iJUG genutzt werden, die den Mitgliedern kostenfrei zur Verfügung stehen.

Eine iJUG-ID erhalten Mitglieder von Java User Groups, die selbst Mitglied im iJUG sind. Diese JUGs zahlen einen geringen Jahresbeitrag von aktuell 90 Euro, unabhängig von der Anzahl ihrer Mitglieder. In vielen Fällen sind dies rechtlich eigenständige Vereine, in Deutschland oft am Zusatz „e. V." (eingetragener Verein) erkennbar.

Daneben gibt es Java User Groups, die direkt als Teil des iJUG gegründet wurden und keine eigenständigen Vereine sind. Diese Gruppen profitieren davon, organisatorisch Teil des iJUG e. V. zu sein, etwa durch den Versicherungsschutz bei der Durchführung von Veranstaltungen. Wer Mitglied einer solchen Gruppe werden möchte, zahlt einen geringen Jahresbeitrag von aktuell 15 Euro und erhält damit Zugang zu den Angeboten und Vorteilen des iJUG, einschließlich der iJUG-ID. Weitere Informationen zu Mitgliedschaftsmodellen und Vorteilen sind auf der iJUG-Webseite zu finden *[8]*.

Benutzernamen in den angebundenen Diensten

Bei der Einrichtung der iJUG-ID kann jedes Mitglied einen Benutzernamen frei wählen. Dieser ist innerhalb des Identity Providers der eigenen Java User Group eindeutig. Für die angebundenen Dienste des iJUG wie Mastodon, PeerTube oder Mobilizon wird jedoch ein über alle Java User Groups hinweg eindeutiger Benutzername benötigt.

Aus diesem Grund wird der gewählte Benutzername bei der automatischen Anlage der Benutzerkonten in den jeweiligen Diensten des iJUG um das Kürzel der eigenen JUG ergänzt. Wählt ein Mitglied beispielsweise den Benutzernamen *foo* und ist Mitglied der JUG mit dem Kürzel *bar*, so lautet der Benutzername in den angebundenen Systemen *foo_bar*.

Wer betreibt das und was kostet das?

Da es sich bei Mastodon, PeerTube und Mobilizon um Open-Source-Software auf Basis offener Standards handelt, kann grundsätzlich jede Organisation diese Dienste selbst betreiben. In der Praxis erfordert der Betrieb jedoch technisches Know-how, kontinuierliche Wartung sowie die laufende Überwachung öffentlich erreichbarer Systeme. Das bindet Zeit und Ressourcen und setzt eine verlässliche personelle Abdeckung voraus.

Viele Java User Groups können diesen Aufwand mangels aktiver Mitglieder, verfügbarer Zeit oder finanzieller Mittel nicht leisten. Der iJUG e. V. übernimmt diese Aufgabe daher für seine Mitglieds-JUGs. Ein eigenes Infrastruktur-Team, aktuell bestehend aus Vertreterinnen und Vertretern von fünf Java User Groups, kümmert sich um Einrichtung, Betrieb und Weiterentwicklung der genannten Systeme.

Die anfallenden Kosten für Hosting, Betrieb und Wartung werden über die Mitgliedsbeiträge des Vereins gedeckt. Für die Java User Groups und ihre Mitglieder ist die Nutzung der angebotenen Dienste dadurch kostenfrei.

Klare Empfehlung für den Einstieg in das Fediverse

Wie im Grundlagenartikel von Jörg Liedl in dieser Ausgabe beschrieben, genügt für den Einstieg ins Fediverse in der Regel ein einzelner Account. Wer in erster Linie Informationen konsumieren möchte, fährt mit einem Mastodon-Account am einfachsten. Mastodon eignet sich gut als zentrale Anlaufstelle, um Inhalte aus verschiedenen Quellen gebündelt zu verfolgen, ohne sich bei mehreren Plattformen separat anmelden zu müssen.

Über Mastodon lassen sich nicht nur Beiträge von Java User Groups und anderen Community-Accounts abonnieren. Auch Video-Kanäle auf PeerTube sowie Veranstaltungen von Java User Groups auf Mobilizon können darüber verfolgt werden. Neue Videos und neu angekündigte Events erscheinen dann automatisch im eigenen Mastodon-Feed.

So landen Informationen aus unterschiedlichen Diensten an einem Ort. Man behält den Überblick über Neuigkeiten aus der Java-Community, ohne mehrere Plattformen parallel im Blick behalten zu müssen. Mastodon dient damit als einfacher Einstiegspunkt in das Fediverse und als zentrale Schnittstelle für Inhalte aus verschiedenen Community-Angeboten.

Keine Bindung an eine Instanz

Ein Mastodon-Account ist nicht an einen Server gebunden. Man kann jederzeit zu einer anderen Instanz umziehen und den Account mitsamt Followern mitnehmen. Wer bereits auf einer anderen Mastodon-Instanz aktiv ist und zur iJUG-Instanz wechseln möchte, ist ausdrücklich willkommen. Umgekehrt ist ein Wechsel ebenso möglich, falls jemand mit dem Angebot des iJUG nicht zufrieden sein sollte.

Fazit: Fediverse als Community-Infrastruktur

Mit Mastodon, PeerTube und Mobilizon stellt der iJUG seiner Community eine eigenständige, offene und unabhängige Kommunikations- und Publikationsinfrastruktur zur Verfügung. Java User Groups gewinnen damit Sichtbarkeit über die eigene Region hinaus und behalten gleichzeitig die Kontrolle über ihre Inhalte und Kanäle. Mitglieder profitieren von transparenten Informationsflüssen, einfachen Zugängen zu Veranstaltungen und einem gemeinsamen digitalen Raum für Austausch und Wissenstransfer.

Diese Infrastruktur entfaltet ihren Wert jedoch nur, wenn sie auch genutzt wird. Verwaiste Accounts, leere Kanäle und ungenutzte Eventplattformen bringen der Community keinen Mehrwert. Der Nutzen entsteht durch aktive Beteiligung, regelmäßiges Teilen von Inhalte und sichtbare Präsenz der Java User Groups im Fediverse.

Der Betrieb der Dienste liegt beim Infrastruktur-Team des iJUG. Dieses lebt von Engagement aus den JUGs selbst. Wer sich technisch einbringen möchte oder Interesse an der Weiterentwicklung der Community-Infrastruktur hat, ist ausdrücklich eingeladen, sich zu beteiligen.

Referenzen

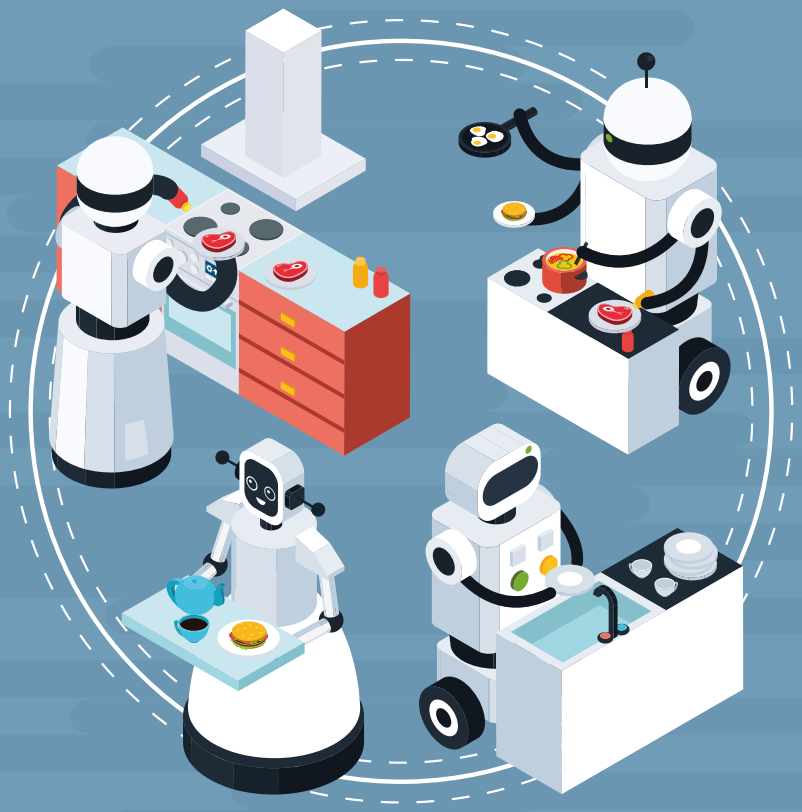
[1] Webseite des iJUG: *https://www.ijug.eu/*
[2] Infrastruktur-Team: *https://www.ijug.eu/infrastruktur*
[3] Softwareprojekte selbst hosten mit Forgejo: *Java aktuell 04/25 S. 50-61*
[4] Mastodon-Instanz des iJUG: *https://ijug.social/*
[5] PeerTube-Instanz des iJUG: *https://videos.ijug.eu/*
[6] Mobilizon-Instanz des iJUG: *https://events.ijug.eu/*
[7] Java User Groups im iJUG: *https://www.ijug.eu/verein/gruppen*
[8] Mitgliedschaft im iJUG: *https://www.ijug.eu/mitgliedschaft*

Marcus Fihlon
www.fihlon.swiss

Marcus Fihlon arbeitet als Scrum Master und Agile Coach in der Schweiz. Er engagiert sich seit vielen Jahren für freie Software, nutzt seit über 30 Jahren Linux und programmiert fast genauso lange in Java. Als Vorstandsmitglied der Java User Group Switzerland und des iJUG setzt er sich für die deutschsprachige Java-Community ein. Er organisiert Konferenzen und Community-Events für Software-Entwickler und hält regelmäßig Vorträge und Workshops. In seiner Freizeit entwickelt er Open-Source-Software und findet seinen Ausgleich auf ausgedehnten Veloreisen auf der ganzen Welt.

Anwendungsfälle und Werkzeuge für Machine Learning

Eldar Sultanow, Cornelius May, Capgemini GmbH



Zu den typischen Anwendungsfällen für Machine Learning, die mittlerweile fast jeder kennt, zählen vor allem autonomes Fahren, Warenkorbanalyse und Spam-Detection. Aber Machine Learning kann noch vieles mehr und ist mittlerweile in den verschiedensten Industrien angekommen. Neben den üblich bekannten Tools oder Frameworks wie Tensorflow, SAS, Tableau und Weka gibt es mittlerweile zig weitere und kaum jemand kann all diese noch sinnvoll voneinander unterscheiden oder einordnen. Dieser Artikel liefert einen Überblick über die Werkzeugvielfalt und verschiedenen Anwendungsfälle für Machine Learning.

Neben Python und R spielt Java im industriellen Machine-Learning-Umfeld weiterhin eine zentrale Rolle. Insbesondere in großen Unternehmen, im Finanzsektor, in der Telekommunikation sowie in sicherheitskritischen Anwendungen ist Java oft die dominante Laufzeitumgebung. Gründe dafür sind die hohe Stabilität der JVM, ausgereifte Toolchains, langfristige Wartbarkeit sowie eine enge Integration in bestehende Enterprise-Architekturen.

Während Python häufig für Prototyping, Exploration und Forschung genutzt wird, kommt Java vor allem dort zum Einsatz, wo Machine-Learning-Modelle produktiv, skalierbar und robust betrieben werden müssen.

Im Java-Ökosystem existieren mehrere etablierte Frameworks, die produktive ML-Anwendungen ermöglichen. **Apache Spark MLlib** ist eine skalierbare Machine-Learning-Bibliothek für verteilte Datenverarbeitung. Sie eignet sich besonders für große Datenmengen, Streaming-Daten und Batch-Analytics. Typische Anwendungsfälle sind Betrugserkennung, Recommendation Engines und Predictive Maintenance. **DeepLearning4j (DL4J)** ist ein Deep-Learning-Framework für die JVM und lässt sich nahtlos in bestehende Java- und Scala-Systeme integrieren. Es wird häufig in regulierten Umgebungen eingesetzt, etwa im Finanz- oder Gesundheitswesen, wo On-Premises-Betrieb und Auditierbarkeit entscheidend sind. **Weka** ist eine der ältesten und bekanntesten ML-Toolkits im Java-Umfeld. Es eignet sich besonders für Lehre, Forschung sowie schnelle Evaluation klassischer ML-Verfahren wie Entscheidungsbäume, SVMs oder Clustering. Und dann gibt es noch die Java-Integration von Python-Modellen. In der Praxis werden ML-Modelle häufig in Python trainiert und anschließend in Java-Anwendungen integriert, etwa über REST-Services, PMML/ONNX oder durch direkte JVM-Bindings. So lassen sich Data-Science-Workflows und Enterprise-IT sauber trennen.

Maschinelles Lernen gilt als eines der ältesten Gebiete der KI. Es soll IT-Systeme in die Lage versetzen, auf Basis vorhandener Daten und Algorithmen Muster und Gesetzmäßigkeiten zu erkennen und Lösungen zu entwickeln. So entsteht künstliches Wissen aus Erfahrungen. Die gewonnenen Erkenntnisse sind in der Regel allgemeiner Art und daher für neue Einsatzfelder oder für die Analyse bisher unbekannter Daten verwendbar.

Ganz so einfach wie sich KI eingrenzen lässt, verhält es sich beim Machine Learning leider nicht. Hier gibt es unterschiedliche Erklärungsversuche und verschiedene Arten, Verfahren und Algorithmen.

Maschinelles Lernen versucht, anhand von Daten, Gesetzmäßigkeiten oder Zusammenhänge (das können lineare oder nichtlineare Zusammenhänge sein) zu erkennen. Es kann dabei passieren, dass man den Zusammenhang zu schwach (zu wenig) erkennt – man spricht dann von „Unteranpassung" (*Underfitting*) oder man übertreibt – dann spricht man von „Überanpassung" (*Overfitting*). *Abbildung 1* illustriert das.

Die oben dargestellte Regressionsmethode ist eine sogenannte Eager-Learning-Methode – jedes Mal, wenn neue Daten hinzukommen, wird das gesamte Modell (recht aufwendig) neu erstellt, die

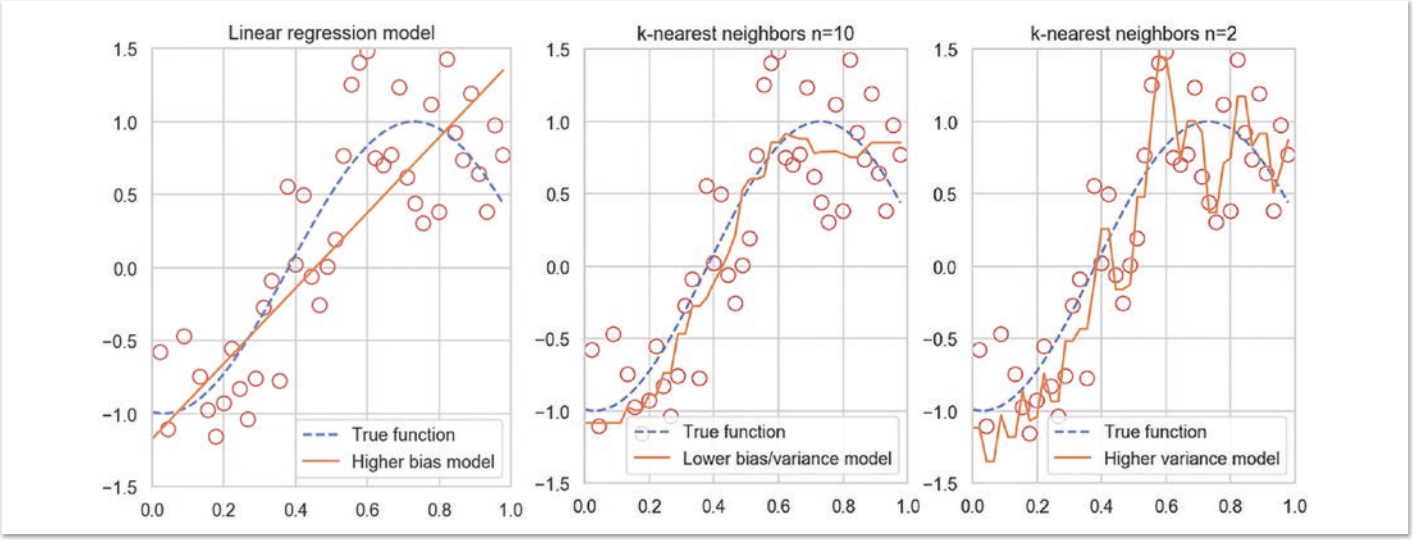


Abbildung 1: Bei der Unteranpassung (links) ist die Gerade ein zu schwacher Zusammenhang in den Datenpunkten. Die Kurve in der Mitte beschreibt den Zusammenhang der Punkte ideal. Die Kurve rechts ist eine Überanpassung – sie „übertreibt". [MuMa2025]

Vorhersage dafür geht schnell. Demgegenüber stehen sogenannte Lazy-Learning-Modelle, die etwa einmal täglich aktualisiert werden. Eine der verbreitetsten Lazy-Learning-Methoden ist die sogenannte k-Nächste-Nachbarn-Klassifikation (k-Nearest-Neighbor-Methode, kurz KNN) *(siehe Abbildung 2)*.

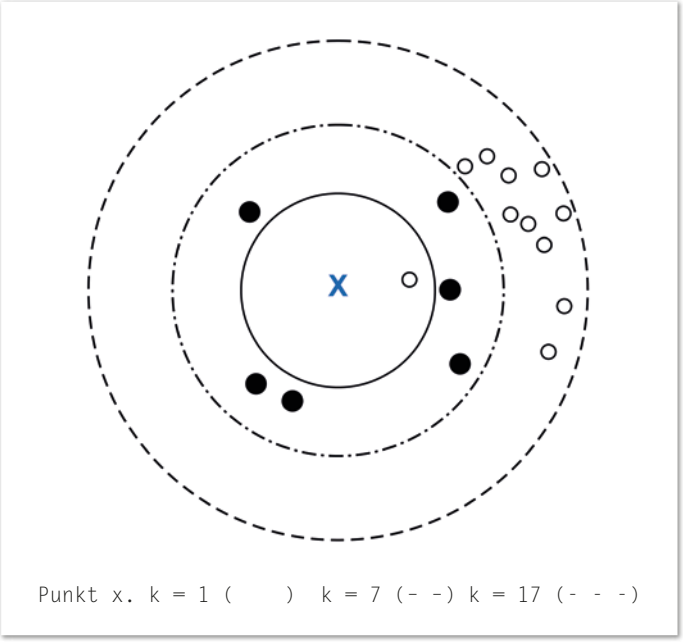


Abbildung 2: Anhand zunehmender Distanzen wird die Zugehörigkeit eines Punktes x zu den k nächstgelegenen Punkten aus den Lerndaten untersucht. Die unter den k-nächsten Nachbarn am häufigsten vertretene Klasse wird ausgewählt. Eine geeignete Wahl von k balanciert Über- und Unteranpassung aus.

Beim Eager Learning (eifriges Lernen) dagegen ist die Lernphase aufwendig, die Anwendung auf neue Beispiele dafür jedoch äußerst effizient. Passende Anwendungsfälle für diese Technik sind die Assoziationsanalyse, die beispielsweise für die Warenkorbanalyse eingesetzt wird, oder Entscheidungsbäume, die etwa zum Einschätzen von Kreditrisiken eingesetzt werden. Verbreitete Eager-Learning-Methoden sind die lineare Regression. Lineare Regression versucht, einen linearen Zusammenhang zwischen einer unabhängigen und abhängigen Variablen zu erklären, zum Beispiel:

- In Berlin soll im Auftrag des Polizeipräsidenten untersucht werden, ob eine höhere Präsenz der Polizei die Kriminalitätsrate senkt.
- Der Schulleiter einer Privatschule ist daran interessiert, ob und in welchem Maße die Zahl der Anmeldungen für die erste Klasse sinken würden, wenn er 1.000 Euro mehr an jährlichen Gebühren verlangt.
- Ein Winzer möchte überprüfen, inwiefern das Wetter einen Einfluss auf die Qualität des gewonnenen Weines hat.
- Eine Supermarktkette möchte herausfinden, wie viel ein Haushalt mit durchschnittlichem Einkommen pro Monat für Lebensmittel ausgibt.
- Wie nimmt das Körpergewicht von Personen mit steigender Körpergröße zu?

Hierzu wird eine bestmögliche Gerade durch die Datenpunkte gelegt, wie hier im Beispiel des Körpergewicht-/Größe-Beispiels zu sehen ist *(siehe Abbildung 3)*.

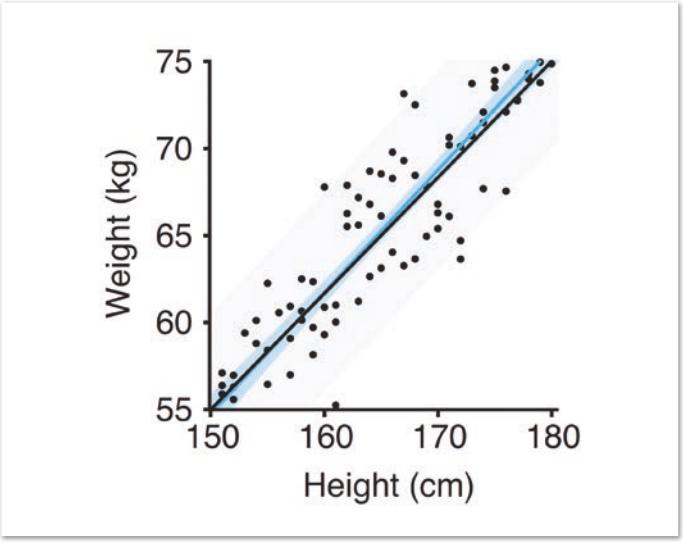


Abbildung 3: Lineare Regression zwischen Körpergröße und Körpergewicht [AlKr2015].

Die Übersicht in *Tabelle 1* zeigt, dass Java weniger im Bereich explorativer Datenanalyse eingesetzt wird, dafür aber eine wichtige Rolle bei **Modellbereitstellung**, **Integration** und **skalierbarer Verarbeitung** spielt. In produktiven Systemen wird Machine Learning häufig nicht als isolierte Data-Science-Aufgabe verstanden, sondern als Bestandteil einer bestehenden Software-Architektur – ein Szenario, in dem Java traditionell seine Stärken ausspielt.

Anwendungsfälle

Vorhersagende Polizeiarbeit (Predictive Policing)

Beim Predictive Policing geht es darum, potenzielle Verbrechen im Vorfeld zu verhindern oder auch vorherzusagen, wann und wo ein Täter erneut zuschlagen wird. Als Grundlage werden Datensätze zu polizeilich registrierter Kriminalität verwendet. Diese werden mit sozialräumlichen, kalendarischen und meteorologischen Daten ergänzt. Bereits vor 20 Jahren wurden derartige Daten – in deutlich geringerem Ausmaß – in Excel-Tabellen erfasst und ausgewertet. Dieser Vorgang ist zeitaufwendig und auf eine maximale Komplexität beschränkt. Durch den Einsatz von ML-basierter Software kann die Masse an verfügbaren Daten effizient ausgewertet werden, wodurch sich schnellere und bessere Schlüsse ziehen lassen können. Auf Basis dieser Schlüsse können Streifen bevorzugt in Gegenden geschickt werden, in denen eine erhöhte Wahrscheinlichkeit für eine bevorstehende Straftat besteht [Kras2018, Krem2018].

Recht und Ordnung

Einige der Bereiche, in denen KI Rechtsverfahren verbessern kann, umfassen eine verbesserte Ermittlung und Analyse basierend auf der Rechtsprechung und die Formulierung von rechtlichen Argumenten basierend auf der Identifizierung relevanter Beweise [PWC2017].

Anomalie-Erkennung

Im Bereich der Cybersicherheit wird seit vielen Jahren an der Verbesserung von Angriffserkennungssystemen gearbeitet. Ziel ist es immer, Angriffe gegen Computersysteme oder Rechnernetze zu erkennen. Der Ansatz der Anomalie-Erkennung basiert darauf, dass Angreifer sich

		Daten-erfassung	Daten-erfassung ETL	Visuali-sierung	Feature-Engineering	Model-Engineering	Optimierung/ Bewertung	Bereit-stellung	kollaboratives Arbeiten	AutoML
1	Python	x	x	x	x	x	x	x		
2	RapidMiner	x	x	x	x	x	x	x		x
3	R	x	x	x	x	x	x	x		
4	Excel	x	x	x		pyXLL	pyXLL	pyXLL		
5	Anaconda	x	x	x	x	x	X	x		Anaconda Enterprise
6	SQL	x	x							
7	Tensorflow					x	X	x		
8	Keras					x	X			
9	scikit-learn					x	x			
10	Tableau	x	x	x	x			x		x
11	Apache Spark	x	x	x	x	x	x	x		
12	Microsoft Power BI		x		x					x
13	XGBoost					x	x	x		
14	Java					x	x	x		
15	PyTorch					x	x			
16	Microsoft SQL Server					x				
17	KNIME	x	x	x	x	x	x	x	KNIME Server	x
18	Unix shell/awk					x	x	x		
19	H2O	x	x	x	x	x	x	x		
20	C/C++					x	x	x		
21	Javascript					x	x	x		
22	Weka	x	x	x	x	x	x	x		
23	MATLAB	x	x	x	x	x	x	x		
24	Apache Kafka	x								
25	SAS Base	x	x	x	x	x	x	x	x	x
26	IBM SPSS Statistics/ Modeler		x	x						
27	Google Bigquery		x	x		x				
28	Microsoft Azure Machine Learning	x	x	x	x	x	x	x	x	x
29	Alteryx	x	x	x	x	x	x	x		Datarobot
30	QlikView	x	x	x	x					
31	Scala		x			x	x	x		
32	SAS Enterprise Miner	x	x	x	x	x	x	x	x	x
33	MLlib					x	x	x		
34	Orange	x	x	x	x	x	x	x		
35	LightGBM					x				
36	Amazon SageMaker	x	x	x	x	x	x	x		
37	Databricks	x		x		x	Mlib		x	
38	BigML	x	x	x	x	x	x	x	x	x
39	DeepLearning4J					x	x	x		
40	fastai library					x	x	x		
41	Dataiku	x	x	x	x	x	x	x	x	x
42	Apache Strom	x	x		x					
43	IBM Watson Studio	x	x	x	x	x	x	x	x	
44	Gnu Octave					x	x	x		
45	Pentaho	x	x	x	x	x	x	x	x	
46	CatBoost					x				
47	Apache MXnet					x	x	x		
48	Julia					x	x	x		
49	Teradata	x	x	x		x	x	x		x
50	Microsoft Cognitive Toolkit					x	x	x		
51	Theano					x				

52	*IBM Cognos*	x	x	x	x					
53	*Stata*	x	x	x						
54	*DataRobot*	x	x		x	x	x	x		x
55	*Rattle Data Mining*		x	x	x					
56	*Google Cloud AutoML*	x	x	x	x	x	x	x		x
57	*Perl*					x	x	x		
58	*Microsoft Machine Learning Server*	x	x	x	x	x	x	x		
59	*TIBCO Spotfire*	x	x	x	x	x	x	x	x	
60	*SAP Analytics/Predictive Analytics + SAP HANA*					x				
61	*Cloudera Data Science Workbench*	x	x	x	x	x	x	x		
62	*Minitab/Salford Predictive Modeler*	x	x		x	x	x	x	x	
63	*Mathematica*	x	x	x	x	x	x	x		
64	*C4.5/C5.0/See5*					x	x	x		
65	*Iodide*		x	x	x					
66	*NVidia*					x	x	x		
67	*Chainer*					x	x	x		
68	*DSSTNE*					x	x	x		
69	*DyNet*					x	x	x		
70	*Gluon*					x	x	x		
71	*Paddle*					x	x	x		
72	*ML.Net*					x	x	x		x
73	*Matillion*		x							

Tabelle 1: Übersicht einiger Frameworks für Machine Learning, siehe vollständige Tabelle in [Grum2020]

deutlich anders verhalten als berechtigte Nutzer. Bei einer Vielzahl von Nutzern und abweichendem Verhalten zwischen diesen, ist es ohne effiziente Softwarelösungen nahezu unmöglich, ein Muster im Verhalten berechtigter Nutzer zu erkennen. Damit ist es dann noch schwieriger zu erkennen, wann eine Anomalie und somit ein Angriff vorliegen. Durch den Einsatz von ML können die zugrundeliegenden Muster und dadurch potenzielle Angriffe erkannt werden [OmNJ2013].

Im Bereich Verteidigung und Sicherheit dient die Erkennung von anomalem Verhalten dem Schutz von Infrastrukturen wie Flughäfen, Kraftwerken und Wirtschaftssektoren, die anfällig für Angriffe sind. Das Erkennen von anomalem Verhalten von Individuen sowie die Vorhersage von Infrastrukturstörungen aufgrund natürlicher oder künstlicher Ursache wird dabei durch die Verwendung von verteilten Sensoren und Mustererkennung ermöglicht.

In sicherheitskritischen Umgebungen werden ML-Modelle zur Anomalie-Erkennung häufig in Java-basierten Backend-Systemen betrieben. Streaming-Plattformen wie Kafka in Kombination mit Spark oder JVM-basierten Inferenzdiensten ermöglichen Echtzeit-Erkennung bei hoher Last und klar definierten Latenzanforderungen.

Katastrophenmanagement und Notfallplan

KIs zeigen ein bemerkenswertes Potenzial bei der Unterstützung von Kontroll- und Abhilfemaßnahmen nach Umweltkatastrophen oder von Menschen verursachten Katastrophen auf. Nach derartigen Ereignissen steigt normalerweise der Kommunikationsbedarf und Netzwerke können überlastet werden. Um die Kontinuität des Netzwerkdienstes inmitten von katastrophalen Ereignissen zu gewährleisten, kann eine KI bei der Optimierung von Mobilfunknetzen und einer intelligenten Breitbandzuweisung helfen.

Unbemannte Drohnen und Satellitenübertragungen in Kombination mit Bildverarbeitung und -erkennung können zur Bewertung und Vorhersage von Infrastrukturschäden verwendet werden. Katastrophenteams können auf der Grundlage dieser Informationen strukturiert und aktiv gesteuert werden. Eine KI kann auch für die Analyse von sozialen Medien verwendet werden. Hierbei werden ortsspezifische Dringlichkeiten gemessen und gezielte Warnungen gesendet, um somit den Verlust von Leben und Eigentum zu minimieren.

Medizin und Gesundheitswesen

Evidenzbasierte Behandlung und medikamentöse Behandlung erfordern ein Maß an Präzision, das den Patienten hilft, Vertrauen in ihren Arzt zu entwickeln. In diesem Bereich können die manuellen und auf Erfahrungen beruhenden Beurteilungen des Arztes durch KI ergänzt werden.

Aufgrund der Vielzahl an Informationsverarbeitungsmöglichkeiten, die in den Bereichen wie Bioinformatik erforderlich sind, ist der Einsatz von KI-basierten Algorithmen und Lösungen unumgänglich. KI-Lösungen im Gesundheitswesen, in der Medizin und in der Biotechnologie umfassen unterstützenden Systeme zur Identifizierung genetischer Risiken aus groß angelegten Genomstudien, die Vorhersage der Sicherheit und Wirksamkeit neu eingeführter Medikamente zur Entscheidungsunterstützung bei medizinischen Beurteilungen sowie die Anpassung der Arzneimittelverabreichung an den einzelnen Patienten.

KI für Computerpathologie: Die CAMELYON Grand Challenge 2016 zur Erkennung von Krebsmetastasen ergab eine Fehlerrate von 7,5 % für

KI-Systeme, eine Fehlerrate von 3,5 % für Ärzte und eine Fehlerrate von 0,5 % für Ärzte, die durch KI unterstützt wurden. Dies zeigt das Potenzial für die Zusammenarbeit zwischen Medizinern und KI [NSTC2016].

Substrukturvorhersage von Metaboliten (Zwischenprodukte bei Stoffwechselweg)

Aktuell ist die Gaschromatographie gekoppelt mit Massenspektrometrie (GC-MS) eine der weltweit meistverbreiteten Routinetechnologien, mit der Biomarker erkannt werden können. Biomarker sind messbare biologische Prozesse, die eine pro- oder diagnostische Aussagekraft haben und als Indikator für Krankheiten herangezogen werden können. Hierbei wird mit dem Gaschromatograph das zu untersuchende Stoffgemisch aufgetrennt und mit dem Massenspektrometer die einzelnen Komponenten identifiziert. Da es jedoch nicht genug authentifizierte Referenzsubstanzen (MST) gibt, die für eine Identifikation nötig sind, wird mit Hilfe von überwachten maschinellen Lernansätzen versucht, weitere MSTs anhand schon vorliegender zu identifizieren [Hummel].

Malaria-Ausbruchsprognose

Malaria ist eines der größten Probleme der öffentlichen Gesundheit in Indien und die weltweit häufigste Infektionskrankheit. Die Verbreitung der Krankheit wird durch Faktoren wie Temperatur, Niederschlag, Flut oder Dürre, aber auch durch menschliche Migration und Bautätigkeiten beeinflusst. Die frühzeitige Vorhersage eines Malaria-Ausbruchs ist der Schlüssel zur Kontrolle der Ausbreitung und damit der Reduzierung von Infizierten und Toten. Forscher aus Indien haben es geschafft, mit dem Klassifikationsalgorithmus „Support Vector Machine" den Ausbruch von Malaria bereits 15 bis 20 Tage im Voraus vorherzusagen [SKP+2015].

Selbstmordvorhersage

Um Warnung zu geben und rechtzeitig psychiatrische Maßnahmen ergreifen zu können, um das Schlimmste zu verhindern, wird versucht, eine Suizidgefahr rechtzeitig zu erkennen. Mit Hilfe von maschinellem Lernen kann die Selbstmordabsicht eines Patienten recht genau vorhergesagt werden. Anhand von Datensätzen, die unter anderem Alter, Geschlecht, Medikamentierung aus Krankenakten enthalten, kann mit 80 % Genauigkeit vorhergesagt werden, ob ein Patient innerhalb der nächsten zwei Jahre Selbstmord begeht [SWT], [GoldhilI].

Wissenschaft und Technologie

KI kann die wissenschaftliche Forschung und das Experimentieren verbessern, indem sie Wissenschaftler und Ingenieure dabei unterstützt, Publikationen und Patente zu lesen, Hypothesen zu formulieren und diese mit Hilfe von Robotersystemen zu testen.

Bildung

In weiten Teilen des Landes herrscht ein Mangel an Akademikern und Lehrern, wenn es darum geht, den Schülern einen effektiven Unterricht mit unterschiedlichsten sozialen und kognitiven Fähigkeiten zu ermöglichen. KI-Lösungen können durch adaptive Betreuung, basierend auf der Aufnahmebereitschaft der Schüler und einer genauen Einschätzung der Entwicklung der Schüler, unterstützend eingreifen und dadurch den Lehrer im Klassenzimmer komplementieren.

Kommunikation und soziale Medien

In Indiens mehrsprachiger und multikultureller Gesellschaft kann Neuro-Linguistisches Programmieren und Übersetzen ein potenziell wertvolles Gebiet für die Anwendung von Algorithmen für maschinelles Lernen sein. Einige indische Start-ups beginnen mit der Entwicklung von Chat-Bots, Spracherkennungs-Tools und intelligenten digitalen Assistenten, die über die Kanäle von sozialen Medien aufgebaut werden sollen. Auch kann KI eine effizientere Nutzung von Bandbreite und Speicher, verbesserte Filterung, Websuche und Sprachübersetzung im Bereich von Kommunikation und sozialen Medien schaffen.

Echtzeit-Versicherungsprämien

Klassischerweise beruhen Versicherungsprämien auf der Risikobewertung historischer Daten. Dabei können aktuelle Trends entweder gar nicht oder nur durch menschliche Arbeit berücksichtigt werden. In Zeiten von immer größerer Datenverfügbarkeit ist es möglich, Versicherungsprämien in Echtzeit auf Basis einer Vielzahl von Parametern anzupassen. So kann zum Beispiel der Beitrag einer Gesundheitsversicherung steigen, wenn man eine Wildwasserrafting-Tour geplant hat oder sinken, wenn man sich für einen Yoga-Kurs anmeldet. Beschränkungen für Echtzeit-Versicherungsprämien existieren derzeit primär beim Datenschutz und der Akzeptanz von Kunden, technisch können ML-basierte Algorithmen bei entsprechenden Rahmenbedingungen Versicherungsprämien heutzutage bereits in Echtzeit berechnen [IBM2015, Mies2018].

Intelligente Haarbürste

Um Tipps für ein gesünderes Haar bieten zu können, wird mit einer Intelligenten Haarbürste das Kämm-Verhalten des Nutzers aufgezeichnet. Mit Hilfe verschiedener Sensoren wie einem Gyroskop, Mikrofon oder Leifähigkeitssensor werden Charakteristiken bestimmt und ausgewertet werden, sodass die Pflegegewohnheit verbessert und helfende Produkte empfohlen werden können [L'Oréal].

Kartierung der globalen Fischereitätigkeiten

Die Fischbestände einiger Arten sind bereits um 90 % gesunken. Da aktuell die Ernährung von über einer Milliarde Menschen von wild gefangenem Fisch abhängig ist, ist es essenziell wichtig, illegales Fischen zu unterbinden und Regelungen für eine sinnvolle Menge von Fischerei aufzustellen. Da jedes Schiff seine Positionsdaten an das „Automatic Identification System" schickt, kann mit Hilfe von maschinellem Lernen der Schiffstyp und das Fanggerät und wo sie aufgrund ihrer Bewegungsmuster fischen, identifiziert werden [Sullivan].

Verbesserte Reinigung in der Lebensmittelindustrie

Im Bereich der Lebensmittelindustrie werden viele Geräte, Werkzeuge und Gefäße zur Verarbeitung und Aufbewahrung von Lebensmitteln genutzt und müssen gereinigt werden. Dabei könnte es sich zum Beispiel um Besteck oder auch leere Flaschen handeln. Klassischerweise landet alles in Geschirrspülern oder ähnlichen Geräten. Mit dem Ziel, dass nach der Reinigung sämtliche Verschmutzungen beseitigt sind, laufen die Geräte auf hoher Stufe, um auch starke Verschmutzungen effektiv zu reinigen. Dabei wird deutlich mehr Zeit, Energie und Wasser benötigt, als es für manche Verschmutzungen nötig wäre. Stattet man die Reinigungsgeräte mit Sensoren aus, die den Grad der Verschmutzung scannen, lässt sich die Reinigungszeit laut Forschern der Universität Nottingham um bis zu 50 % reduzieren und somit die gesamte Produktivität erhöhen [Walk2018].

Vorausschauende Wartung (Predictive Maintenance)
Um Ausfällen vorzubeugen, werden Produktionsmaschinen in regelmäßigen Abständen gewartet. Dabei gilt es, die Personalkosten für die Wartung sowie gegebenenfalls die Kosten für den Stillstand der Maschine während der Wartung gegen die Kosten bei Ausfall der Maschine abzuwägen. Um diese Entscheidung zu optimieren und genau dann Wartungen durchzuführen, wenn ein Ausfall bevorsteht, hat beispielsweise Thyssenkrupp eine Software entwickelt, die die Daten aller Aufzüge des Unternehmens analysiert und Muster erkennt, wann ein Ausfall bevorsteht. Eine solche Software kann auch bei Produktionsmaschinen im Bereich Industrie 4.0 Ausfälle verhindern und Kosten sparen. Dort sind jedoch viele Ausfälle auf menschliche Fehler zurückzuführen, die derzeit nicht ausreichend im Algorithmus berücksichtigt werden (können) [Müll2018].

Gerade im industriellen Umfeld sind Produktionssysteme oft Java-basiert. ML-Modelle zur vorausschauenden Wartung werden hier direkt in bestehende Leitsysteme, Microservices oder Edge-Komponenten integriert, was den Einsatz von Java-basierten ML-Stacks begünstigt.

Produktfertigung und Logistik
Durch die Unterstützung von Produktionsbetrieben mit KI-Lösungen können sich zuverlässigere Bedarfsprognosen, flexiblere und reaktionsschnellere Lieferketten, schnellere Betriebsänderungen, genauere Planungen sowie Bestandsoptimierungen einstellen. Weitere Vorteile sind die Schaffung intelligenterer, schnellerer und umweltfreundlicherer Prozesse. Auch können diese Verbesserungen die Produktivität und Qualität erhöhen, Kosten senken sowie die Gesundheit und Sicherheit des Unternehmens erhöhen.

Ein Schlüsselbereich der KI-Anwendung in der logistischen Abwicklung ist die adaptive Planung von Lieferungen und die Routenplanung der Fahrzeuge. Fortschrittliche Logistik- und Lieferketten werden mit Hilfe von Expertensysteme zur Entscheidungsunterstützung erstellt. Bestellungen können durch optische Fahrerassistenz- und automatisierte/robotische Systeme effizienter transportiert werden. Dies hat den Transport weniger anfällig für Störungen durch Wetter, Verkehr und unnatürliche Ereignisse gemacht.

Financial Services
Zu den wichtigsten Anwendungsbereichen von KI im Bank- und Finanzdienstleistungssektor gehören die Früherkennung von Finanzrisiken und Systemfehlern sowie die Automatisierung zur Reduzierung böswilliger Absichten in Finanzsystemen wie Marktmanipulation, Betrug, anomales Trading und die Reduzierung von Marktvolatilität und Handelskosten.

Banken und Versicherungen setzen Machine Learning häufig in Java-Anwendungen ein, um regulatorische Anforderungen, Nachvollziehbarkeit und langfristige Wartbarkeit sicherzustellen. Scoring-Modelle, Betrugserkennung und Risikobewertung laufen dabei oft in JVM-basierten Systemen mit klar versionierten Modellen.

Landwirtschaft
Die Landwirtschaft ist ein weiterer Sektor, der von intelligenten Lösungen profitieren kann, indem er intelligentere Produktions-, Verarbeitungs-, Lager-, Verteilungs- und Verbrauchsmechanismen nutzt. Des Weiteren können KI-Lösungen, durch standortspezifische und aktuelle Daten über Pflanzen, dazu beitragen, dass die Anwendung geeigneter Substanzen wie Düngemitteln und Chemikalien optimiert wird [PWC2017].

Konsumgüter und Dienstleistungen
Konsumgüter und Dienstleistungen waren einer der ersten Bereiche der Einführung von KI in Indien und machen derzeit einen erheblichen Teil der Anwendung im privaten Sektor aus. Um es den Verbrauchern zu ermöglichen, bessere Produkte zu niedrigen Preisen zu finden, werden Algorithmen für maschinelles Lernen eingesetzt, um das Angebot besser an die Nachfrage der Verbraucher anzupassen.

Online-Shopping-Portale nutzen in großem Umfang Prognosefunktionen, um das Interesse der Verbraucher an Produkten zu messen, indem sie ein gezieltes Verständnis der Präferenzen durch die Erfassung von Browsing- und Click-Stream-Daten aufbauen und Kunden über einen mehrkanaligen Ansatz effektiv ansprechen und binden.

Audit Services
Unternehmen setzen KI ein, um die Aufgabe, Stapel von Dokumenten zu Durchsuchen und Schlüsselbegriffe identifizieren, weitestgehend zu automatisieren. Dies war bisher ein zeitaufwändiger manueller Prozess. Die maschinelle Verarbeitung natürlicher Sprachen liest und versteht wichtige Punkte in den Dokumenten. Die Technologie des maschinellen Lernens ermöglicht es, dieses System anhand einer Reihe von Beispieldokumenten zu trainieren, damit es lernt, wie man Informationen automatisiert identifiziert und extrahiert.

Reisen und Transport
KI kann die Effizienz des Betriebs im Reise- und Transportsektor verbessern, indem es durch Systeme zur Strukturüberwachung und Infrastrukturdatenmanagement für mehr Sicherheit sorgt. Dies kann sich in Form von reduzierten Reparatur- und Rekonstruktionskosten oder Echtzeitinformationen in der Routenberechnung auszahlen, wodurch die Transparenz steigt, und der Energieverbrauch und Emissionen reduziert werden.

Augmented Reality (AR) Emojis
Normale Smartphone-Emojis sind statisch oder haben eine fest definierte Animation. AR-Emojis werden erstellt, indem der Gesichtsausdruck des Nutzers in ein animiertes 3D-Modell gespiegelt wird [Pal2019].

Fazit
Machine Learning ist heute kein rein experimentelles Werkzeug mehr, sondern integraler Bestandteil produktiver Softwaresysteme. Während Python die Innovationsgeschwindigkeit vorantreibt, sorgt Java dafür, dass Modelle zuverlässig und skalierbar sind sowie langfristig betrieben werden können. In der Praxis ergänzen sich beide Welten – vom Notebook bis zur Enterprise-Plattform.

Quellen

[MuMa2025] John Paul Mueller & Luca Massaron (2025): Machine Learning For Dummies, 3rd Edition, John Wiley & Sons, ISBN 978-1-394-37323-9).

[AlKr2015] Naomi Altman & Martin Krzywinski (2015): Simple linear regression. Nat Methods 12, 999–1000 (2015). *https://doi.org/10.1038/nmeth.3627)*

[IBM2015] IBM Corporation (2015): Harnessing the power of data and analytics for insurance. IBM Analytics.

https://www.the-digital-insurer.com/wp-content/uploads/2016/06/715-IMW14672USEN.pdf [abgerufen am 01.02.2019].

[Kras2018] Prof. Dr. Susanne Krasmann (2018): Forschungsprojekt – Predictive Policing. Universität Hamburg. https://www.wiso.uni-hamburg.de/fachbereich sowi/professuren/hentschel/forschung/predictive policing.html [abgerufen am 01.02.2019].

[Krem2018] Stefan Krempl (2018): Predictive Policing: Die Polizei arbeitet verstärkt wie ein Geheimdienst. heise online. https://www.wiso.uni-hamburg.de/fachbereich-sowi/professuren/hentschel/forschung/predictive-policing.html [abgerufen am 01.02.2019].

[Mies2018] Daniel Miessler (2018): The Future of Insurance is Real-time Auto-Adjusted Premiums. https://danielmiessler.com/blog/the-future-of-insurance-is-real-time-auto-adjusted-premiums [abgerufen am 01.02.2019].

[Müll2018] Bernd Müller (2018): Vorausschauende Wartung in der Industrie 4.0 – Der größte Störfaktor ist der Mensch. WirtschaftsWoche. https://www.wiwo.de/unternehmen/mittelstand/hannovermesse/vorausschauende-wartung-in-der-industrie-4-0-der-groesste-stoerfaktor-ist-der-mensch/21166908.html [abgerufen am 01.02.2019].

[NSTC2016] National Science and Technology Council (2016): The National Artificial Intelligence Research and Development Strategic Plan. https://www.nitrd.gov/PUBS/national_ai_rd_strategic_plan.pdf [abgerufen am 01.02.2019].

[OmNJ2013] Salima Omar, Asri Ngadi, Hamid H. Jebu (2013): Machine Learning Techniques for Anomaly Detection – An Overview. International Journal of Computer Applications (0975–8887), Volume 79 No. 2, October 2013. https://pdfs.semanticscholar.org/0278/bbaf1db5df036f02393679d485260b1daeb7.pdf [abgerufen am 01.02.2019].

[Pal2019] Palladino Tommy, Mobile AR News, https://mobile-ar.reality.news/news/here-are-all-things-you-can-do-augmented-reality-with-samsung-galaxy-s9-s9-0183131/ [abgerufen am 01.02.2019].

[PWC2017] Artificial Intelligence and Robotics – 2017: Leveraging artificial intelligence and robotics for sustainable growth.

[SKP+2015] Vijeta Sharma, Ajai Kumar, Lakshmi Panat, Dr. Ganesh Karajkhede, Anuradha lele (2015): Malaria Outbreak Prediction Model Using Machine Learning. International Journal of Advanced Research in Computer Engineering & Technology (IJARCET). Volume 4 Issue 12, December 2015. https://pdfs.semanticscholar.org/2ce3/631949498f6f40cf3b9ab4096c082f1d0047.pdf [abgerufen am 01.02.2019].

[Walk2018] Jon Walker (2018): AI in Food Processing – Use Cases and Applications That Matter. Emerj. https://emerj.com/ai-sector-overviews/ai-in-food-processing [abgerufen am 08.02.2019].

[Hummel] Hummel, J., Strehmel, N., Selbig, J. et al. Metabolomics (2010): Decision tree supported substructure prediction of metabolites from GC-MS profiles. Metabolomics, Vol. 6, Issue 2,. 322–333. DOI: https://doi.org/10.1007/s11306-010-0198-7

[SWT] Social Work Today: How Artificial Intelligence Will Save Lives in the 21st Century. https://www.socialworktoday.com/news/enews_0417_2.shtml

[Goldhill] Goldhill, O. (2018). Machines know when someone's about to attempt suicide. How should we use that information? https://qz.com/1367197/machines-know-when-someones-about-to-attempt-suicide-how-should-we-use-that-information/

[L'Oréal] L'Oréal (2017). Kérastase and Withings Unveil World's First Smart Hairbrush at CES 2017. https://www.loreal.com/media/press-releases/2017/jan/kerastase-and-withings-unveil-worlds-first-smart-hairbrush-at-ces-2017

[Sullivan] Sullivan, B. (2016). Mapping global fishing activity with machine learning. https://blog.google/products/maps/mapping-global-fishing-activity-machine-learning/

[Grum2020] Marcus Grum, Eldar Sultanow, Daniel Friedmann, André Ullrich, Norbert Gronau (2020): "Tools des Maschinellen Lernens: Marktstudie, Anwendungsbereiche & Lösungen der Künstlichen Intelligenz" (Buch), GITO, https://doi.org/10.30844/grum_2020

Eldar Sultanow
Capgemini Deutschland GmbH
eldar.sultanow@capgemini.com

Dr. Eldar Sultanow ist IT-Strategе bei Capgemini, Lehrperson an der Universität Potsdam und Gutachter für das Deutsche Zentrum für Luft- und Raumfahrt (DLR). Der promovierte Wirtschaftsinformatiker blickt auf eine mehr als zwanzigjährige Erfahrung in der Softwareentwicklung zurück: von der Programmierung bis hin zum KI-Design.

Cornelius May
Capgemini Deutschland GmbH
cornelius.may@capgemini.com

Cornelius May ist als DevOps Engineer bei Capgemini tätig. Sein Schwerpunkt liegt auf der Konzeption hochperformanter Cloud-Native-Infrastrukturen und der Architektur skalierbarer Machine Learning Umgebungen für komplexe Enterprise-Szenarien.

Java aktuell

NEWSLETTER

Anmeldung

Der Java aktuell Newsletter informiert dich alle vier Wochen über das aktuelle Geschehen in der Java-Welt.

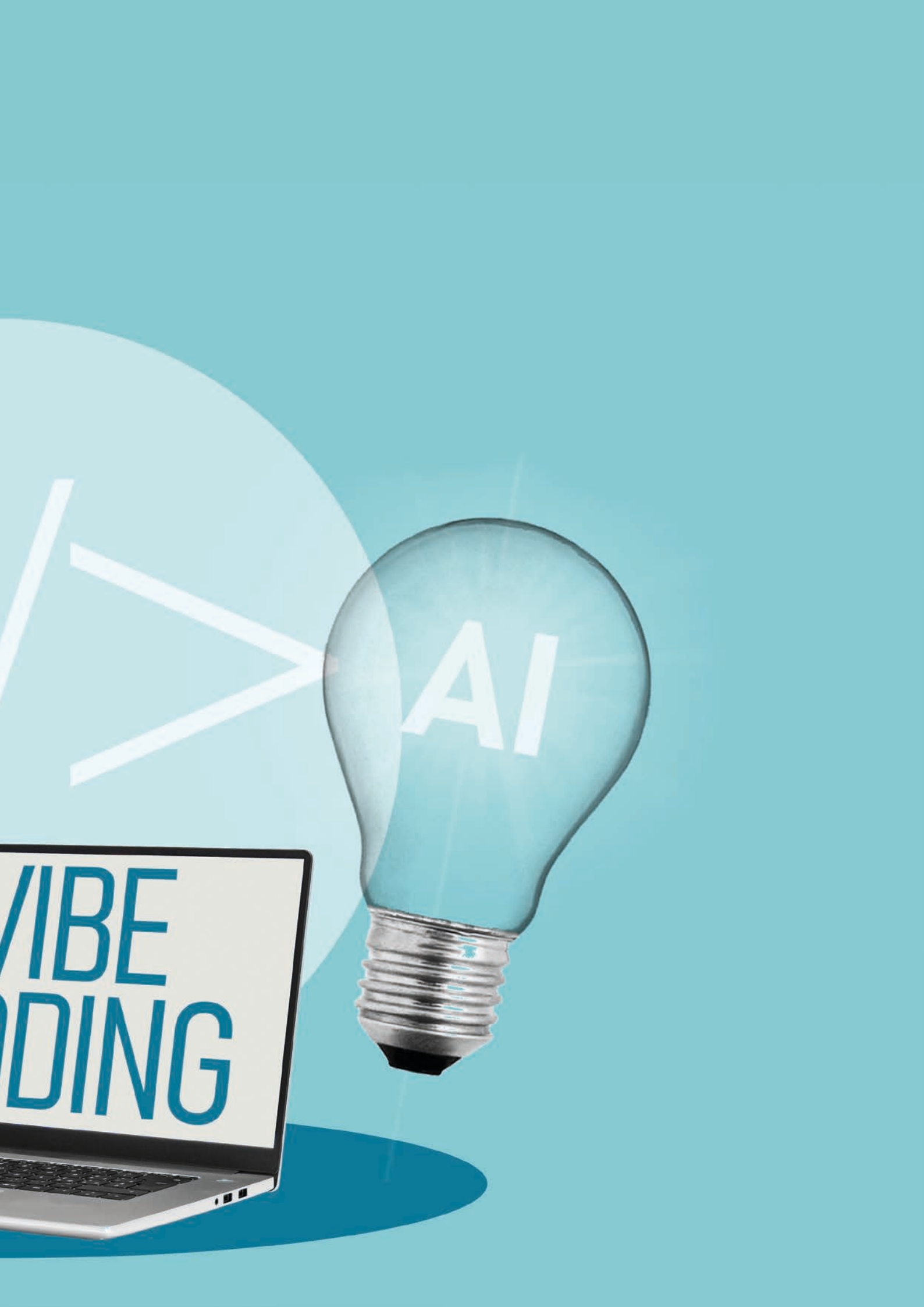
Abonniere ihn kostenfrei und bleibe immer auf dem Laufenden!

https://www.java-aktuell.eu/go/nlabonnieren

oder nach dem Login mit euren Zugangsdaten direkt über euer Profil abonnieren.

Secure Vibe Coding: Sicherheit im Zeitalter der KI-gestützten Softwareentwicklung

Rico Komenda, adesso SE


AI
VIBE
DDING

Vibe Coding, bei dem KI-gestützt Anwendungen aus natürlicher Sprache generiert werden, gewinnt zunehmend an Bedeutung. Diese Technologie beschleunigt die Entwicklung und senkt Barrieren für Nicht-Programmierer, birgt jedoch neue Sicherheitsrisiken. Der Artikel zeigt praxisnahe Risiken, Best Practices und Strategien für sichere KI-gestützte Softwareentwicklung.

Mit den leistungsfähigen Large Language Models (LLMs) hat sich eine neue Methode in der Softwareentwicklung etabliert: das sogenannte Vibe Coding. Nutzer formulieren Softwareanforderungen in natürlicher Sprache und überlassen es einem KI-System, daraus lauffähigen Programmcode zu generieren. Dieser Ansatz beschleunigt Entwicklungsprozessen deutlich und vereinfacht die Softwareerstellung, da auch Personen ohne tiefgehende Programmierkenntnisse lauffähige Anwendungen umsetzen können. Gleichzeitig entsteht jedoch ein Spannungsfeld zwischen Geschwindigkeit und Sicherheit.

Das Konzept des Vibe Codings und seine Besonderheiten

Sobald der Mensch seine Anforderungen in natürlicher Sprache formuliert hat, generiert der KI-Agent beim Vibe Coding daraufhin Quellcode, der iterativ angepasst und erweitert wird. Der Entwicklungsprozess ist dabei stark explorativ und häufig nicht streng formalisiert.

Ein besonderer Mehrwert von Vibe Coding zeigt sich in klar abgegrenzten, gut testbaren Szenarien. Besonders geeignet ist der Ansatz bei Migrationen bestehender Anwendungen auf neue Frameworks, Plattformen oder Programmiersprachen, sofern bereits eine solide Testbasis vorhanden ist. Bestehende Unit- und Integrationstests können dabei als funktionale Referenz dienen: Der KI-generierte Code gilt als akzeptabel, wenn das erwartete Verhalten unverändert erhalten bleibt. In solchen Fällen eignet sich Vibe Coding hervorragend, um Boilerplate-Code zu modernisieren, APIs zu aktualisieren oder Architekturänderungen umzusetzen, ohne die fachliche Logik der Anwendung neu zu modellieren.

Im Unterschied zu klassischen Low-Code- oder No-Code-Plattformen verzichtet Vibe Coding meist auf vordefinierte Sicherheitsmechanismen oder eingeschränkte Baukästen. Die KI kann nahezu beliebige Bibliotheken, Frameworks oder Sprachkonstrukte verwenden. Diese Freiheit ist einerseits ein Vorteil, andererseits erhöht sie das Risiko, dass unsichere oder veraltete Patterns übernommen werden. Besonders kritisch ist, dass Anwender den generierten Code häufig nicht vollständig verstehen und daher Sicherheitslücken nicht erkennen.

Der Verzicht auf klassische Entwicklungsartefakte wie detaillierte Architekturspezifikationen, explizite Sicherheitskonzepte oder manuelle Code-Reviews führt dazu, dass Sicherheitsaspekte häufig erst spät oder gar nicht berücksichtigt werden. Fachbeiträge der Cloud Security Alliance (CSA) *[1]* sowie praktische Analysen aus dem Databricks-Umfeld *[2]* zeigen, dass KI-generierter Code zwar funktional korrekt sein kann, jedoch regelmäßig grundlegende Sicherheitsprinzipien verletzt. Dieser Fachartikel analysiert die sicherheitsrelevanten Herausforderungen des Vibe Codings und zeigt, wie etablierte Security-Best-Practices an dieses neue Entwicklungsparadigma angepasst werden können.

Warum Sicherheit im Vibe Coding entscheidend ist

Studien zeigen, dass KI-generierter Code häufiger Sicherheitslücken enthält als manuell entwickelter Code *[3]*. Die funktionale Korrektheit allein garantiert keine Sicherheit: Fehlende Eingabevalidierung, unzureichende Berechtigungsprüfungen oder unsichere Fehlerbehandlung sind typische Probleme.

Das Databricks Red Team *[2]* konnte demonstrieren, dass selbst einfache KI-generierte Anwendungen kritische Schwachstellen enthalten können, etwa durch unsichere Serialisierung, unkontrollierte Speicherzugriffe oder ungeprüfte Datenformate. Diese Schwachstellen entstehen nicht absichtlich, sondern weil KI-Modelle bekannte Standardlösungen reproduzieren, die funktional, aber unsicher sind. Ohne aktive Lenkung wird Sicherheit häufig vernachlässigt.

Sicherheitsfundamente im Vibe Coding

Die Cloud Security Alliance und OWASP *[4]* empfiehlt, bewährte Sicherheitsgrundlagen bewusst in den Entwicklungsprozess zu integrieren. Dabei sollten folgende Aspekte besonders beachtet werden:

1. *Vermeidung harter Kodierung sensibler Daten*

Ein häufiger Fehler KI-generierter Anwendungen ist das Einfügen von Zugangsdaten, API-Schlüsseln oder Passwörtern direkt in den Code. Dies stellt ein erhebliches Sicherheitsrisiko dar, da solche Informationen leicht ausgelesen oder versehentlich öffentlich zugänglich gemacht werden können.
Sicherheitsmaßnahmen:

- Nutzung von Umgebungsvariablen
- Speicherung in Secret-Management-Systemen
- Zugriff über Cloud-native Dienste mit Zugriffssteuerung

Beim Vibe Coding sollte bereits im Prompt klar angegeben werden, dass sensible Daten nicht hartcodiert werden oder in die Logs geschrieben werden dürfen.

2. *Endpoint-Sicherheit und Authentifizierung*

APIs bilden eine zentrale Angriffsfläche moderner Anwendungen. KI-Modelle neigen dazu, einfache oder unvollständige Authentifizierungsmechanismen zu generieren. Best Practices:

- Implementierung/Nutzung starker Authentifizierung (z. B. OAuth2, JWT)
- Rollenbasierte Zugriffskontrollen
- Prüfung der Authentifizierungslogik durch Entwickler

Databricks zeigt, dass fehlende oder fehlerhafte Authentifizierung zu den häufigsten Problemen in KI-generierten Services zählt *[2]*.

3. *Eingabevalidierung und Sanitizing*

Unzureichend validierte Eingaben führen zu SQL-Injection, Command-Injection, Cross-Site-Scripting oder Buffer Overflows. Auch wenn KI-Modelle oft Basisvalidierungen liefern, sind diese meist nicht kontextspezifisch oder vollständig. Entwickler sollten daher etablierte Validierungsbibliotheken verwenden und Eingaben konsequent überprüfen.

4. *HTTPS und Transportverschlüsselung*

Alle Datenübertragungen zwischen Clients und Servern müssen verschlüsselt erfolgen. TLS ist inzwischen Standard, sollte aber nicht als selbstverständlich betrachtet werden. Unsichere Protokolle müssen deaktiviert und Zertifikate korrekt verwaltet werden

Erweiterte Sicherheitsmaßnahmen im Lebenszyklus der Entwicklung

Secure Coding endet nicht nach der „normalen" Entwicklung. Moderne Entwicklungsprozesse integrieren Sicherheitsprüfungen in den gesamten Lebenszyklus. Dazu gehören:

1. *Code-Reviews und KI-gestützte Analysen*
   Regelmäßige Code-Reviews sind essenziell. KI-Werkzeuge wie statische Code-Analyse oder spezialisierte Sicherheitsscanner (zum Beispiel Snyk, Checkmarx) helfen dabei, Schwachstellen automatisch zu erkennen. Menschliche Überprüfung bleibt jedoch unerlässlich, insbesondere bei sicherheitskritischen Komponenten.

2. *Sicherheits-Scanning im CI/CD*
   Automatisierte Sicherheits-Scans in der Continuous Integration/Continuous Delivery-Pipeline stellen sicher, dass bei jedem Commit geprüft wird, ob neue Schwachstellen eingeführt werden. Tools zur dynamischen Anwendungssicherheit (DAST) und statischen Analyse decken unterschiedliche Klassen von Problemen ab.

3. *Geheimnisprotokollierung und Fehlerverarbeitung*
   Fehlerausgaben dürfen keine internen Systemdetails an Endnutzer preisgeben. Stattdessen sollten sie in Logs gesammelt werden, die nur autorisierten Entwicklern zugänglich sind und nach sicherheitsrelevanten Indikatoren durchsucht werden.

API- und Repository-Sicherheit

APIs und Repository-Hosting stellen häufige Angriffsziele dar. OWASP-Richtlinien bilden auch hier ein robustes Fundament.

1. *Authentifizierung & Rate-Limiting*

Neben standardisierter Authentifizierung sollten API-Gateways Rate-Limiting und Throttling einbauen, um Missbrauch wie DDoS zu begrenzen.

2. *Repositoryhärtung*

Git-Repositorys müssen geschützt werden – etwa durch Zwei-Faktor-Authentifizierung, Zugriffskontrollen, Abhängigkeitsscans (Dependabot) und die Vermeidung harter Umgebungsdateien im Repository.

Schutz von Datenbanken und persistenter Speicherung

Datensicherheit ist zentraler Bestandteil moderner Anwendungen. Eingesetzte Strategien umfassen:

- Parameterized Queries zur Verhinderung von SQL-Injections
- Verschlüsselung ruhender Daten mit robusten Algorithmen (zum Beispiel AES-256)
- Least Privilege-Zugriffskonzepte für Datenbankbenutzer
- Regelmäßige Backups und Überprüfung von Restore-Prozessen

Spezifische Risiken beim Einsatz von LLMs in der Software-Architektur

Die OWASP LLM Top 10 für 2025 *[5]* identifiziert neue Gefährdungsklassen, die beim Einsatz generativer KI-Modelle auftreten können:

- Prompt Injection: Manipulation der KI-Ausgabe durch bösartige Eingaben
- Unbeabsichtigte Leaks sensibler Daten durch das Modell
- Supply Chain Risiken und Modellvergiftung
- Übermäßige Agentenautonomie

Jede dieser Klassen erfordert spezifische Gegenmaßnahmen, einschließlich Input-/Output-Validierung, strenger Zugangskontrolle und definierter Grenzen zur Modellnutzung.

Deployment und Cloud-Betrieb

Sicheres Deployment bietet zusätzliche Schutzschichten:

- DDoS- und Firewall-Funktionen des Providers nutzen
- Automatische SSL/TLS-Zertifikate für HTTPS-Sicherheit
- Zugangssteuerungen und Log-Überwachung

Diese Maßnahmen ergänzen den sicheren Code durch eine robuste Infrastrukturhärtung.

Mitigationen bei KI Code Generatoren

Mit dem zunehmenden Einsatz von KI-gestützten Entwicklungswerkzeugen bietet sich nicht nur die Möglichkeit, Software schneller zu entwickeln, es entstehen auch neue Angriffsflächen und Risiken, die über klassische Sicherheitsprobleme hinausgehen. Zahlreiche Sicherheitsforscher und Anbieter haben deshalb Methoden entwickelt, um diese Risiken frühzeitig zu erkennen und zu mindern. Drei besonders relevante Ansätze sind **Cursor Rules**, **Rules Files** und sichere Konfigurationspraktiken für KI-IDE-Umgebungen wie Cursor – sie lassen sich als systematische Mitigationen für KI-generierten Code zusammenfassen.

R.A.I.L.G.U.A.R.D.: Ein Framework für sichere Code Generierung

Die Cloud Security Alliance schlägt mit dem **R.A.I.L.G.U.A.R.D. Framework** *[6]* einen strukturierten Ansatz vor, um KI Agenten sicheren Code generieren zu lassen. Das Framework besteht aus folgenden Komponenten:

- **R: Risk First (Risiko zuerst):** Definiere das Sicherheitsziel und die damit verbundenen Risiken bereits auf der Ebene der Prompt-Regeln. Dadurch hältst du die KI dazu an, mögliche An-

griffsvektoren zu erkennen und entsprechende Sicherungsmaßnahmen vorzuschlagen.

- **A: Attached Constraints (angehängte Zwänge):** Legt unverhandelbare Sicherheitsgrenzen fest, z. B. „Keine harten Secrets im Code", „kein unsicheres Krypto".
- **I: Interpretative Framing (sichere Interpretationsrahmen):** Bestimmt, wie die KI die Anforderungen sicher interpretieren soll – selbst wenn die ursprüngliche Prompt-Formulierung unspezifisch ist.
- **L: Local Defaults (lokale sichere Voreinstellungen):** Definiert projektspezifische Standardannahmen wie z. B. „TLS immer aktiv", „Verwendung sicherer Bibliotheken", die auch bei fehlenden Angaben gelten.
- **G: Generative Path Checks (generative Pfadprüfungen):** Legt einen nachvollziehbaren Prozess fest, den die KI prüfen muss, bevor Code erzeugt wird – etwa Eingabevalidierung oder Authentifizierungschecks.
- **U: Uncertainty Disclosure (Unsicherheitsdarstellung):** Regelt, wie die KI mit Unsicherheiten umgehen soll – z. B., indem sie Rückfragen stellt, statt unsicheren Code zu erzeugen.
- **A: Auditability (Auditierbarkeit):** Definiert Markierungen oder Kommentare, die im generierten Code Hinweise auf sicherheitsrelevante Entscheidungen geben, um die Überprüfung zu erleichtern.
- **R+D: Revision + Dialogue (Überprüfung & Dialog):** Beschreibt Mechanismen, wie Entwickler mit der KI über unsichere Ergebnisse kommunizieren und diese überarbeiten können.

Dieses Framework hilft, die KI schon bei der Generierung auf sichere Pfade zu lenken und liefert Mechanismen, um die Zusammenarbeit zwischen Entwickler und Modell zu strukturieren – ein Schritt weg vom reinen Funktionsfokus hin zu einem *Security First Design*.

Rules Files: Standardisierte Sicherheitsrichtlinien für KI Code

Parallel dazu setzen Sicherheitsforscher und Entwickler auf Rules Files – vordefinierte Richtliniendateien, die als Sicherheits-Regelsatz für KI-Code-Generatoren fungieren [7]. Diese Dateien (zum Beispiel Cursor Rules, Copilot Custom Instructions, Claude.md) lassen sich in vielen modernen KI IDE Tools einbinden und enthalten klare Vorgaben, wie die KI sich verhalten soll.

Die zentrale Idee ist, dass Regeln nicht nur als freie Prompts formuliert werden, sondern als formal strukturierter Leitfaden, der Sicherheitsaspekte wie die folgenden berücksichtigt:

- **Klarheit und Aktionierbarkeit:** Regeln sollten konkret, präzise und umsetzbar sein – und nicht nur allgemeine Hinweise enthalten.
- **Kontextspezifische Vorgaben:** Regeln können auf bestimmte Sprachen, Frameworks oder Bibliotheken zugeschnitten werden.
- **Kombinierbarkeit und Modularität:** Komplexe Sicherheitsrichtlinien können aus kleineren, wiederverwendbaren Regeln zusammengesetzt werden.

Beispiele für solche Sicherheitsregeln umfassen Anweisungen wie „Vermeide unsichere Funktionen wie eval", „Validiere alle Eingaben kontextsensitiv", „Nutze nur Bibliotheken aus verifizierten Quellen", oder „Speichere keine Secrets im Code".
Diese Regeln können dann direkt in die Arbeitsumgebung der KI geladen werden, sodass der Agent diese Leitplanken automatisch berücksichtigt.

Zudem werden mittlerweile offene Sammlungen von Rules Files bereitgestellt, die als Baseline für verschiedene Sprachen und Frameworks dienen und von Teams weiter angepasst werden können.

Risiken und Pflege von Rules Files

Das Arbeiten mit Rules Files birgt jedoch auch neue Risiken: Da diese Dateien die KI Entscheidungsprozesse beeinflussen, können kompromittierte oder falsch formulierte Regeln dazu führen, dass die KI unsicheres oder sogar schädliches Verhalten reproduziert. Beispielsweise könnten Angreifer durch Manipulation von Rules Files versteckte Anweisungen einschleusen, die dann unbemerkt in generierten Code „mitgeschrieben" werden.
Um diesen Risiken zu begegnen, gelten folgende Praktiken als Best Practices:

- **Strenge Reviews:** Behandle Rules Files wie kritischen Code und unterziehe sie einem formellen Review Prozess.
- **Automatisierte Validierung:** Nutze Tools zur automatischen Analyse von Rules Files auf mögliche Manipulationen oder Schwächen.
- **Zugriffskontrolle:** Speichere Rules Files in sicheren, versionierten Repositories und beschränke Änderungen auf autorisierte Personen.
- **Regelmäßige Audits:** Überprüfe regelmäßig, ob die Regeln noch aktuellen Sicherheitsstandards entsprechen.

Diese Maßnahmen minimieren das Risiko, dass diese Mechanik, die eigentlich zur Sicherheit beitragen soll, selbst zur Schwachstelle wird.

Sichere Konfigurationen für KI IDEs und lokale Sicherheit

Neben der Codegenerierung spielt auch die *Sicherung der Entwicklungsumgebung* eine wichtige Rolle, wenn Entwickler mit KI IDEs wie Cursor arbeiten. Insbesondere Systeme, die nicht nur Textvorschläge machen, sondern auch Befehle ausführen oder Dateien verändern können, eröffnen neue Angriffsvektoren [8].
Best Practices umfassen hier unter anderem:

- **Deaktivierung automatischer Ausführungsmodi:** Schalte sicherheitskritische Funktionen wie Auto Run oder YOLO Modes ab, damit KI Vorschläge nicht ohne Überprüfung ausgeführt werden.
- **Minimale Allow Lists:** Erlaube nur explizit geprüfte und sichere Befehle, um das Risiko von Command Injection oder Datenexfiltration zu reduzieren.
- **Schutz von Konfigurationsdateien:** Schütze Hidden Dateien (.dotfiles) vor KI Änderungen, da sie laufende Prozesse beeinflussen können.
- **OS Level Schutz:** Nutze zusätzliche Sandboxing Maßnahmen, Containerisierung oder Betriebssystem Berechtigungen, sie erhöhen die Barriere gegen unerwartete KI Operationen.

Darüber hinaus spielen **Secrets Management und Versionierung**

eine herausragende Rolle: API Keys oder Zugangsdaten gehören nicht in den Code oder in lokale Dateien, sondern in dedizierte Vaults und Tools zur Geheimnisverwaltung, die vor automatischen KI Zugriffen geschützt sind.

Der menschliche Faktor und kontinuierlicher Lernprozess

Auch wenn KI-Agenten viele Aufgaben automatisieren, bleibt menschliche Expertise unverzichtbar. Security-Profis sollten in Teams eingebunden werden, um Reviews, Audits und Penetrationstests durchzuführen. Kontinuierliche Weiterbildung in Sicherheitspraktiken ist ebenfalls entscheidend, da die Bedrohungslandschaft ständig im Wandel ist.

Darüber hinaus ist Vibe Coding nicht als zufälliger oder rein intuitiver Prozess zu verstehen, sondern als deterministische Interaktionsform, die – ähnlich wie eine Programmiersprache – gezielt erlernt werden kann und muss. Mit wachsender Erfahrung entsteht ein Verständnis dafür, wie Prompts formuliert sein müssen, um reproduzierbare, konsistente und kontrollierbare Ergebnisse zu erzielen. In diesem Sinne stellt Vibe Coding eine weitere Abstraktionsebene in der historischen Entwicklung der Softwareentwicklung dar: von Binärcode über Assembler, Low-Level- und High-Level-Programmiersprachen bis hin zur natürlichen Sprache. Mit jeder dieser Ebenen wird ein Teil der unmittelbaren Kontrolle gegen höhere Produktivität und schnellere Entwicklung eingetauscht. Vibe Coding lässt sich daher als konsequente Fortsetzung dieser Entwicklung begreifen – vorausgesetzt, die zugrunde liegenden Mechanismen, Grenzen und Risiken werden bewusst verstanden und reflektiert eingesetzt.

Fazit

Vibe Coding eröffnet neue Horizonte in der Softwareentwicklung, indem es Barrieren senkt und Kreativität beschleunigt. Doch Geschwindigkeit darf nicht auf Kosten der Sicherheit und Qualität gehen. Durch die konsequente Integration von Sicherheitsprinzipien – von Eingabeverifikation bis hin zu API-Härtung, Repository-Schutz und Cloud-Security – lässt sich eine sichere und vertrauenswürdige Entwicklungsumgebung schaffen. Nur durch die Kombination von automatisierten Tools und menschlicher Expertise können Applikationen, die auf KI-generiertem Code beruhen, auch den Anforderungen heutiger Sicherheitsstandards genügen.

Quellen

[1] Cloud Security Alliance (2025): Secure Vibe Coding Guide. Cloud Security Alliance, *https://cloudsecurityalliance.org/blog/2025/04/09/secure-vibe-coding-guide*.

[2] Databricks (2025): Passing the Security Vibe Check: The Dangers of Vibe Coding. Databricks, *https://www.databricks.com/blog/passing-security-vibe-check-dangers-vibe-coding*.

[3] Pearce, H. et al. (2021): Asleep at the Keyboard? Assessing the Security of GitHub Copilot's Code Contributions. arXiv, *https://arxiv.org/abs/2108.09293*.

[4] OWASP Foundation (2023): OWASP Secure Coding Practices – Quick Reference Guide. OWASP, *https://owasp.org/www-project-secure-coding-practices-quick-reference-guide/*.

[5] OWASP Foundation (2025): OWASP Top 10 for Large Language Model Applications. OWASP, *https://owasp.org/www-project-top-10-for-large-language-model-applications/*.

[6] Cloud Security Alliance (2025): Secure Vibe Coding Level Up with Cursor Rules and the R.A.I.L.G.U.A.R.D. Framework. Cloud Security Alliance, *https://cloudsecurityalliance.org/blog/2025/05/06/secure-vibe-coding-level-up-with-cursor-rules-and-the-r-a-i-l-g-u-a-r-d-framework*.

[7] Wiz (2025): Safer Vibe Coding: Rules Files. Wiz, *https://www.wiz.io/blog/safer-vibe-coding-rules-files*.

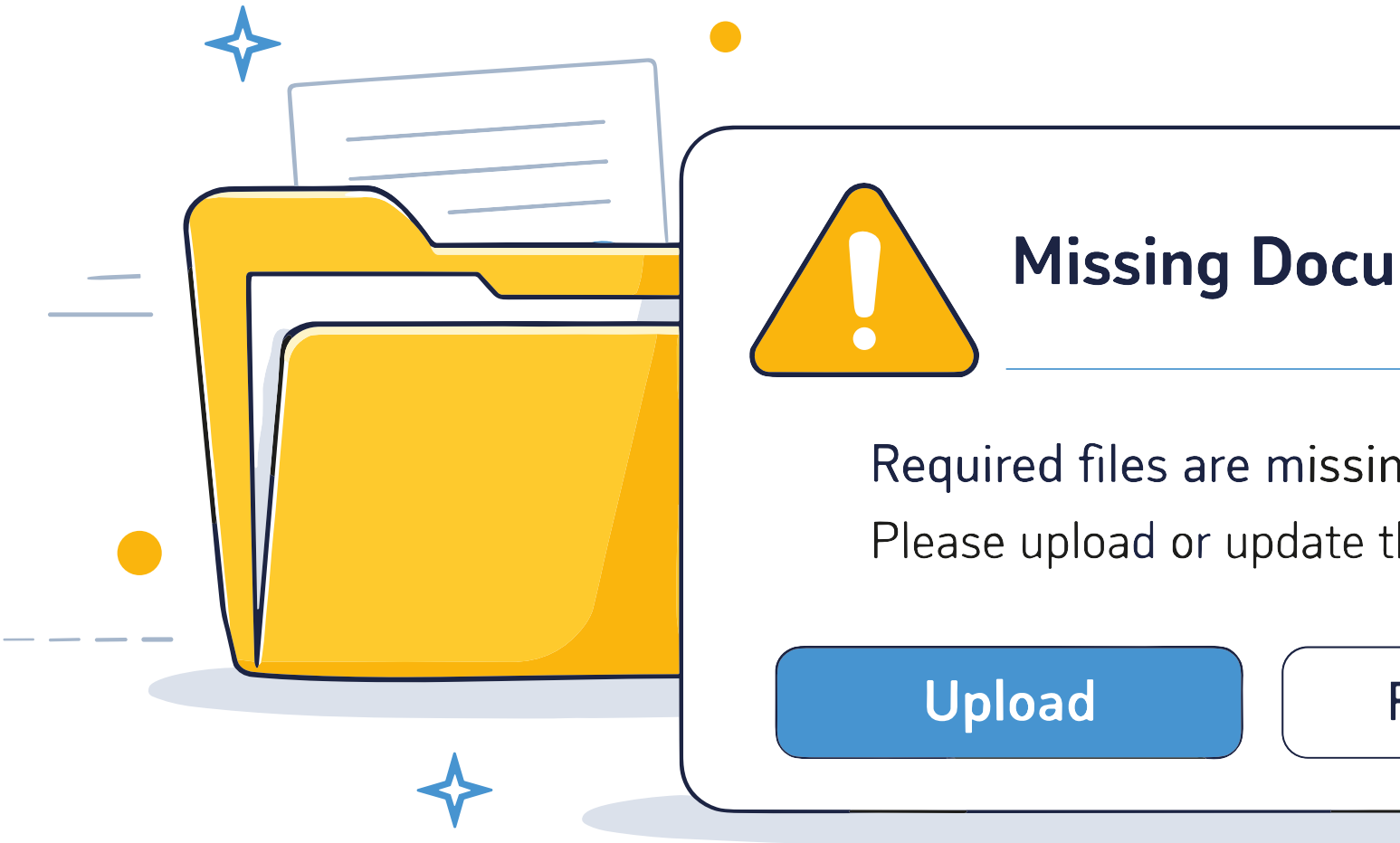
[8] Embrace The Red (2025): Wrapping Up: The Month of AI Bugs. Embrace The Red Blog, *https://embracethered.com/blog/posts/2025/wrapping-up-month-of-ai-bugs/*.

Rico Komenda
adesso SE
rico.komenda@adesso.de

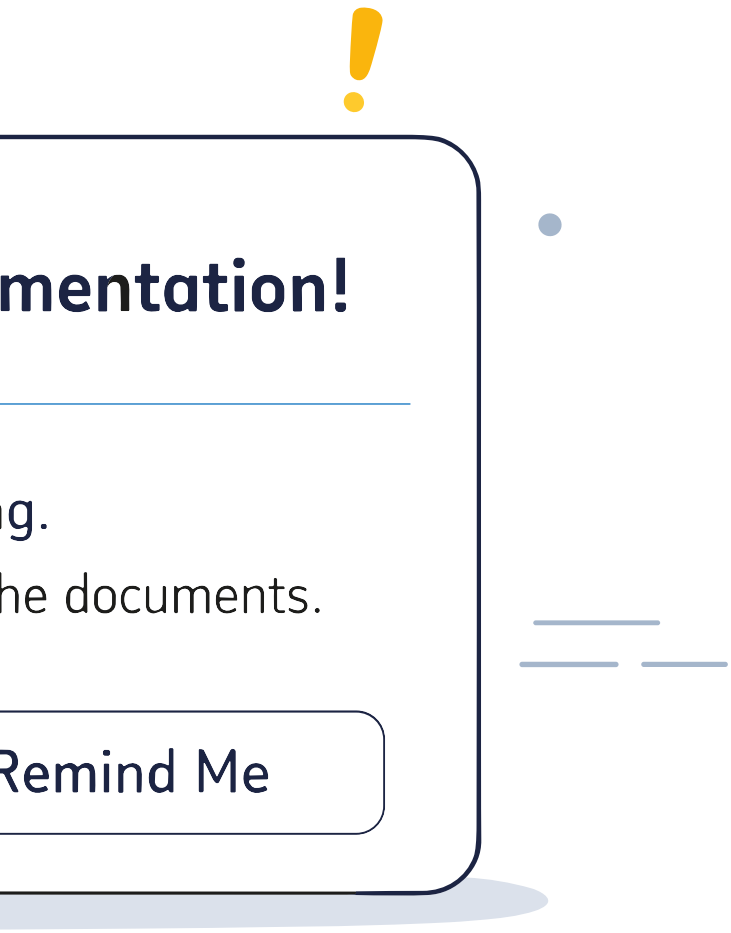
Rico Komenda beschäftigt sich mit der regulatorischen und technischen Sicherheit von KI bei der adesso SE. Als IT-Sicherheitsexperte arbeitet er eng mit Kunden zusammen, um maßgeschneiderte Sicherheitslösungen zu entwickeln und umzusetzen.

Von Pflichtübung zu Werkzeug: Software-Dokumentation richtig angehen

Liam Bergh, docuco GbR



Dieser Artikel befasst sich damit, warum Dokumentation in der Praxis häufig scheitert, welche grundlegenden Hürden dabei eine Rolle spielen und weshalb guter Wille allein nicht ausreicht. Es wird erläutert, welche Faktoren nachhaltige Dokumentation ausmachen und wie sie sinnvoll ineinandergreifen. Dabei wird deutlich, dass die richtige Herangehensweise darüber entscheidet, ob Dokumentation zum Selbstzweck wird oder ein wirkungsvolles Hilfsmittel darstellt.



Wieso versagt Dokumentation?

Software-Dokumentation hat bei vielen einen schlechten Ruf: aufwendig, starr, nicht wertschöpfend. Sie scheitert im Alltag, veraltet schnell und wird selten wirklich gelesen. Diese Probleme werden oftmals der Dokumentation selbst zugeschrieben, tatsächlich liegt die Ursache aber in der Regel woanders, nämlich in der Herangehensweise.

In vielen Teams wird dokumentiert, ohne zuvor systematisch zu klären, welchen Zweck die Dokumentation erfüllen soll. Es entsteht „irgendetwas Geschriebenes", Inhalte ohne klar definierte Fragestellung, ohne Struktur, ohne langfristige Pflegeperspektive. Dokumentation ist dann kein bewusst gestaltetes Artefakt, sondern ein zufälliges Nebenprodukt.

Hinzu kommen ungeeignete Tools und lückenhaftes Wissen über Erstellung und Pflege von Dokumentation. Häufig wird weder in der formalen Ausbildung noch in der Einarbeitung richtig vermittelt, wie man zielgerichtet dokumentiert, welche Arten von Dokumentation es gibt oder wie sie sinnvoll in Entwicklungsprozesse integriert wird. Dokumentation bleibt implizites Wissen und damit zufällig in Qualität und Nutzen.

Ein weiterer häufiger Fallstrick ist die externe Motivation. In manchen Fällen wird Dokumentation primär für Management, Kundschaft oder Auditierende erstellt. Sie dient der Absicherung und Kontrolle, nicht der täglichen Arbeit der Entwicklungsteams. Obwohl sie sich inhaltlich auf die entwickelte Software bezieht, bietet sie denjenigen, die daran arbeiten, kaum Mehrwert. Dokumentieren wird dadurch als Pflicht wahrgenommen, nicht als Unterstützung.

All diese Probleme liegen jedoch nicht in der Dokumentation selbst begründet, sondern sind das Ergebnis fehlender Strategie, unklarer Ziele und nicht geklärter Verantwortlichkeiten. Gute Dokumentation entsteht nicht zufällig, sie ist das Resultat einer bewussten Haltung und systematischen Herangehensweise.

Wozu dokumentieren wir überhaupt?

Bevor es um konkrete Lösungsansätze geht, lohnt sich ein kurzer Blick auf den eigentlichen Zweck der Dokumentation. Warum ist es überhaupt wichtig, sich diesen Schwierigkeiten zu stellen und Antworten darauf zu finden? Dokumentation ist kein Selbstzweck und kein formales Anhängsel der Softwareentwicklung. Ihr Sinn liegt darin, Wissen zugänglich, nutzbar und dauerhaft verfügbar zu machen. Eine wirkungsvolle Dokumentation beantwortet konkrete Fragen, reduziert Abhängigkeiten von Einzelpersonen und unterstützt tägliche Arbeit in allen Bereichen: Entwicklung, Betrieb, Wartung und Weiterentwicklung. Sie schafft Orientierung, erleichtert Entscheidungen und senkt langfristig Reibungsverluste.

Welche Schwierigkeiten sind zu bewältigen?

Wie erreicht man nun das Ideal einer nachhaltigen, effektiven Dokumentation? Aus den bereits genannten Problemen lassen sich zentrale Hürden ableiten, die erfolgreicher Dokumentation im Weg stehen können. Diese Hindernisse sind nicht unüberwindlich, müssen aber bewusst adressiert werden.

Ein großer Stolperstein ist das fehlende konzeptionelle Fundament. Dokumentation, die ohne klares Ziel entsteht, verliert schnell an Wert. Bevor Inhalte erstellt werden, muss geklärt sein, welche Fragen die Dokumentation beantworten soll, für wen sie gedacht ist und in welchem Kontext sie genutzt wird. Dazu gehört auch die bewusste Entscheidung für geeignetes Tooling. Eingesetzte Software sollte die angestrebte Nutzung unterstützen, nicht diktieren. Ein ungeeignetes Tool verstärkt strukturelle Probleme, statt sie zu lösen.

Eine weitere Herausforderung bringt der Faktor Mensch ins Spiel. Selbst das beste Konzept bleibt wirkungslos, wenn Mitarbeitende nicht befähigt sind, es umzusetzen. Gute Dokumentation erfordert fachliche Grundlagen, die häufig weder in Ausbildung noch Einarbeitung systematisch vermittelt werden. Ebenso wichtig ist die Haltung: Dokumentation muss als Unterstützung im Arbeitsalltag verstanden werden, nicht als zusätzliche Pflicht oder Kontrollinstrument. Das setzt Sensibilisierung, klar kommunizierte Erwartungen und eine entsprechende Kultur voraus.

Diese Gesichtspunkte zeigen, dass Dokumentation kein rein technisches oder redaktionelles Thema ist. Sie bewegt sich im Spannungsfeld zwischen Menschen, Organisation und Technik. Genau diese drei Dimensionen bilden die Grundlage für nachhaltige Software-Dokumentation.

Die menschliche Dimension: Zielgruppe, Relevanz, Nutzbarkeit

Die erste und oft unterschätzte Dimension guter Software-Dokumentation ist der Mensch. Dokumentation entfaltet ihren Wert nur dann, wenn sie verstanden, genutzt und als hilfreich wahrgenommen wird.

Am Anfang steht die Zielgruppe. Bevor dokumentiert wird, muss klar sein, für wen die Inhalte gedacht sind. Davon hängen zentrale Entscheidungen ab: Welche fachliche Sprache ist angemessen? Welches Vorwissen kann vorausgesetzt werden? Wie tief müssen technische Details gehen? Und in welcher Form ist die Dokumentation am besten nutzbar – Referenz oder Tutorial? PDF oder Webpage? Text oder Bilder? Ohne diese Antworten entsteht schnell Dokumentation, die entweder überladen oder aber lückenhaft ist und damit an den Bedürfnissen der Lesenden vorbeigeht.

Eng damit verknüpft ist die Relevanz der Inhalte. Gute Dokumentation enthält alle Informationen, die benötigt werden, und nur diese! Dieses Prinzip von „just enough documentation" hilft, den Fokus zu wahren und Überfrachtung zu vermeiden. Ausschweifende Dokumentation ist schwer zu lesen, noch schwerer zu pflegen und wird häufig ignoriert. Relevanz ist damit nicht nur eine Frage des Nutzens, sondern auch der Nachhaltigkeit.

Der dritte Aspekt ist die Nutzbarkeit: Lesbarkeit, Auffindbarkeit und praktische Verwendbarkeit der Informationen. Dokumentation, die nicht gefunden oder verstanden wird, ist faktisch wirkungslos. „Write-only-Dokumentation" kostet Zeit, ohne Mehrwert zu liefern. Klare Struktur, verständliche Formulierungen, konsistente Begriffe und ein für die Zielgruppe geeignetes Format sind daher entscheidend. Ebenso wichtig ist, dass das eingesetzte Tooling die Nutzung unterstützt, statt sie zu erschweren.

Diese drei Aspekte zeigen: Gute Dokumentation beginnt nicht beim Schreiben, sondern beim Verstehen der Menschen, die sie nutzen sollen. Zielgruppe, Relevanz und Nutzbarkeit bilden das Fundament, auf dem alle weiteren Dimensionen aufbauen.

Die organisatorische Dimension: Verantwortung, Struktur, Aktualität

Neben der menschlichen Perspektive ist gute Dokumentation in hohem Maß eine organisatorische Aufgabe. Viele der entscheidenden Voraussetzungen liegen nicht bei einzelnen Mitarbeitenden, sondern beim Unternehmen selbst.

Zentral ist die Frage der Verantwortung. Es muss klar definiert sein, wer für die Pflege der Dokumentation zuständig ist, wer Inhalte überprüft oder freigibt und wer Entscheidungen über Tools und Prozesse trifft. Ebenso wichtig sind benannte Ansprechpersonen für Rückfragen, Korrekturen oder Verbesserungsvorschläge zum Dokumentationsprozess. Ohne diese Klarheit bleibt Dokumentation unverbindlich und wird im Alltag leicht verdrängt.

Darauf aufbauend braucht es klare Strukturen. Dazu zählen nicht nur eindeutige Rollen, sondern auch einheitliche Vorlagen, Gliederungen und nachvollziehbare Prozesse. Diese sollten praxistauglich sein und sich sinnvoll in bestehende Arbeitsabläufe integrieren lassen. Dokumentation darf nicht nur als Ideal propagiert werden; sie muss organisatorisch so verankert sein, dass sie tatsächlich entsteht und gepflegt wird.

Ein wesentlicher Prüfstein dieser Verankerung ist die Aktualität. Dokumentation ist kein statisches Artefakt, sondern verändert sich zusammen mit der Software. Damit sie verlässlich bleibt, muss ihre Aktualisierung fester Bestandteil der täglichen Arbeit sein, zum Beispiel gekoppelt an Entwicklungs-, Review- oder Release-Prozesse. Gleichzeitig ist Anerkennung auf Planungsebene erforderlich; auch schlanke, effiziente Dokumentation kostet Zeit, die bewusst eingeplant werden muss.

Über den operativen Nutzen hinaus ist Dokumentations-Governance häufig auch Voraussetzung für Compliance. Regulatorische

Anforderungen, Audits oder vertragliche Verpflichtungen lassen sich nur erfüllen, wenn Zuständigkeiten, Versionen und Freigaben nachvollziehbar geregelt sind. Gute Governance verbindet damit den internen Mehrwert von Dokumentation mit externen Anforderungen, ohne sie auf reine Kontroll- oder Absicherungszwecke zu reduzieren.

Diese organisatorischen Rahmenbedingungen sind keine Nebensache. Sie entscheiden darüber, ob gute Dokumentation möglich ist oder ob sie dauerhaft an fehlender Verbindlichkeit scheitert. Unternehmen schaffen damit die organisatorischen Voraussetzungen, unter denen Dokumentation entstehen, wachsen und ihren Nutzen entfalten kann.

Die technische Dimension: Versionierung, Wiederverwendung, Automatisierung

Die technische Dimension beantwortet die Frage, wie Technik die Dokumentationspraxis unterstützt und in bestimmten Fällen überhaupt erst ermöglicht. Sie ist kein Ersatz für Konzept oder Kultur, aber ein entscheidender Hebel für Skalierbarkeit und Nachhaltigkeit.

Ein Aspekt ist die Versionierung. Sobald mehrere Versionsstände einer Software parallel im Einsatz sind, ist eine versionierte Dokumentation unverzichtbar. Sie macht Änderungen nachvollziehbar, ordnet Inhalte eindeutig einem Softwarestand zu und verhindert überladene Dokumente mit zahllosen Varianten. Statt einer einzigen, schwer lesbaren Dokumentation, entstehen klar getrennte Versionen mit direktem Bezug zu Software-Ständen. Versionierung ist damit Grundvoraussetzung für Verlässlichkeit.

Eng damit verbunden ist die Wiederverwendung von Dokumentation. Prinzipien wie „Don't repeat yourself" gelten nicht nur für Code. Funktional identische oder gemeinsam genutzte Komponenten benötigen keine mehrfach gepflegte Beschreibung. Durch modulare Inhalte lassen sich Dokumentationsbausteine wiederverwenden, was Pflegeaufwand reduziert, und Konsistenz erhöht. Besonders in der User-Dokumentation ist dieser Ansatz wirkungsvoll, etwa bei wiederkehrenden Bedienkonzepten oder gemeinsamen Features. Grundsätzlich lässt er sich aber überall dort anwenden, wo sich Strukturen oder Inhalte wiederholen.

Der größte Hebel liegt schließlich in der Automatisierung. Automatisierung kann Dokumentationsarbeit deutlich reduzieren, etwa durch die Ableitung von Inhalten aus bestehenden Artefakten oder durch automatisierte Prüfungen und Veröffentlichungen. Sie ist jedoch kein Einstiegspunkt. Ohne geklärte Grundlagen, wie beispielsweise Zweck, Struktur und technische Tiefe, erzeugt Automatisierung lediglich schneller schlechte Dokumentation. Erst auf Basis eines tragfähigen Konzepts entfaltet sie ihren Mehrwert.

Die technische Dimension zeigt damit klar ihre Rolle: Sie verstärkt gute Entscheidungen, kann sie aber nicht ersetzen. Technik macht Dokumentation effizienter, konsistenter und zugänglicher, aber nur, wenn die inhaltlichen und organisatorischen Voraussetzungen sind geschaffen.

Fazit

Gute Software-Dokumentation entsteht nicht allein durch umfangreiche Notizen oder moderne Tools. Sie ist das Ergebnis des Zusammenspiels menschlicher, organisatorischer und technischer Faktoren. Erst wenn diese drei Dimensionen aufeinander abgestimmt sind, kann Dokumentation ihren Zweck erfüllen: Wissen zugänglich machen, Arbeit erleichtern und langfristig tragfähig bleiben.

Dokumentation scheitert selten am tatsächlichen Aufwand. Sie scheitert an fehlendem Konzept, unklaren Erwartungen und einer Kultur, die sie als Pflicht oder Kontrollinstrument versteht. Wird Dokumentation hingegen als integraler Bestandteil der täglichen Arbeit gedacht und entsprechend gestaltet, verliert sie ihren Schrecken und wird zu dem, was sie sein sollte: eine praktische Unterstützung statt einer lästigen Pflicht.

Gleichzeitig ist Dokumentation kein monolithisches Großprojekt, das vollständig durchdacht und abgeschlossen sein muss, bevor sie Nutzen stiftet. Im Gegenteil: Sinnvolle Dokumentation lässt sich schrittweise aufbauen. Schon kleine, bewusst gesetzte Verbesserungen, zum Beispiel eine klar definierte Zielgruppe, eine gemeinsame Struktur, geklärte Zuständigkeiten, können spürbaren Mehrwert schaffen. Entscheidend ist nicht der Umfang, sondern die Richtung.

Liam Bergh
docuco GbR
Liam.Bergh@docuco.de

Zunächst im Bereich Software-Entwicklung tätig, wechselte Liam dann in den Bereich der Dokumentation und arbeitete als technischer Dokumentator für ein Software-Unternehmen. Mit docuco ging Liam den nächsten Schritt und bietet Software-Unternehmen als externer Dienstleister Unterstützung bei allen Fragen rund um das Thema Software-Dokumentation an. Liams Anliegen ist es, das Thema Software-Dokumentation populärer zu machen und zu zeigen, dass Dokumentation nicht aufwendig und lästig sein muss.

Faszinierend! BDD mit dem Spock Framework

Sebastian Lauth, Deichmann SE

Tests sind das Sicherheitsnetz jeder Softwareentwicklung. Oft fühlen sie sich jedoch leider wie eine lästige Pflicht an – schwer zu lesen und mit der Zeit noch schwerer zu warten oder erweitern. Hier tritt das Spock Framework auf den Plan. Durch die Symbiose aus Groovy-Eleganz und der Philosophie des Behavior-Driven Development (BDD) macht Spock unsere Tests nicht nur effizienter, sondern regelrecht faszinierend.

Einleitung: Warum Spock?

In der modernen Softwareentwicklung ist eine vernünftige Testabdeckung längst kein Diskussionspunkt mehr, sondern eine Grundvoraussetzung. Doch Hand aufs Herz: Wie oft fühlen sich unsere Test-Suiten wirklich wie ein verlässliches Sicherheitsnetz an – und wie oft eher wie eine bleierne Last, die jegliches Refactoring zum Albtraum macht? Klassische Unit-Tests in Java neigen dazu, mit der Zeit unübersichtlich zu werden. Wir kämpfen mit tief verschachtelten Mocks, unendlichen `assertEquals`-Ketten und Testmethodennamen, die so lang sind, dass sie kaum auf den Bildschirm passen.

Hier setzt das Spock Framework an. Es bietet einen strukturierten Ansatz, um die Kluft zwischen technischer Implementierung und fachlicher Anforderung zu überbrücken. Dabei bedient es sich einer Sprache, die für viele Java-Entwickler zunächst ungewohnt klingen mag: Groovy. Doch genau hier liegt der strategische Vorteil: Groovy erlaubt es Spock, eine „Domain Specific Language" (DSL) bereitzustellen, die technisches Testen mit der Klarheit einer Spezifikation verbindet.

Die Philosophie: Behavior-Driven Development (BDD)

Bevor wir uns der technischen Umsetzung widmen, lohnt ein Blick auf die methodische Basis. Behavior-Driven Development (BDD) verschiebt den Fokus weg von der rein technischen Verifikation einzelner Code-Einheiten hin zur Beschreibung des erwarteten Verhaltens eines Systems.

Der Grundgedanke ist so simpel wie effektiv: Softwarefehler entstehen oft nicht durch mangelhaft geschriebenen Code, sondern durch Missverständnisse über die Anforderungen. BDD adressiert dieses Kommunikationsproblem, indem es eine gemeinsame Sprache („Ubiquitous Language" im Sinne des Domain Driven Design) zwischen Fachbereich, Qualitätssicherung und Entwicklung etabliert und so alle Beteiligten an einen Tisch bringt. Statt kryptischer Testmethoden treten Szenarien in den Vordergrund, die einer festen logischen Struktur folgen:

- **Given:** In welchem Zustand befindet sich das System zu Beginn? Welche Vorbedingungen sind erfüllt?
- **When:** Welche spezifische Interaktion führt der Nutzer oder ein externes System aus?
- **Then:** Welche beobachtbare Veränderung oder Antwort wird vom System erwartet?

Dieser Dreisatz transformiert abstrakte Anforderungen in eine Form, die sowohl für Menschen lesbar als auch für Maschinen ausführbar ist. In der Java-Welt wurde dieser Ansatz lange Zeit primär durch Werkzeuge wie Cucumber realisiert, die eine Trennung zwischen textuellen Feature-Files und technischer Glue-Code-Implementierung erfordern.

BDD verfolgt das Ziel, die „Spezifikation" zur „Dokumentation" zu machen. Ein Testfall ist somit kein Abfallprodukt der Entwicklung, sondern dient als lebendiger Beleg dafür, dass die Geschäftsregeln korrekt verstanden und umgesetzt wurden. Das senkt nicht nur die Einstiegshürde für fachfremde Stakeholder, sondern schärft auch den Blick der Entwickler für das „Was" vor dem „Wie".

Die Anatomie einer Spock-Spezifikation

Während klassische Test-Frameworks oft eine strikte Trennung zwischen Testlogik und Dokumentation erzwingen, verschmilzt in Spock beides zu einer Einheit. Eine Testklasse wird hier konsequenterweise als „Specification" bezeichnet. Sie bildet den Rahmen, in dem das fachliche Verhalten in technisches Design übersetzt wird.
Der erste auffällige Unterschied zu JUnit ist die Benennung der Testmethoden. Spock (beziehungsweise Groovy) erlaubt die Verwendung von beliebigen Strings als Methodennamen. Das mag trivial klingen, ist aber ein entscheidender Faktor für die Ausdrucksstärke: Statt `testShouldCalculateDiscountForPremiumUsers()` schreiben wir `def "Ein Premium-Nutzer erhält 20% Rabatt auf den Warenkorb"()`.

Innerhalb dieser Methoden erzwingt Spock durch seine Labels die im BDD-Teil besprochene Struktur. Diese Labels sind kein bloßes Beiwerk, sondern steuern die Ausführungslogik des Frameworks:

- **given:** Hier bereiten wir die Testumgebung vor.
- **when:** Hier wird die zu testende Aktion ausgeführt.
- **then:** Hier finden die Verifikationen statt. Jede Zeile, die einen Wahrheitswert liefert, wird implizit als Assertion gewertet.
- **expect:** Eine kompakte Kombination aus when und then für einfache Abfragen.
- **cleanup:** Stellt sicher, dass Ressourcen (wie temporäre Dateien oder DB-Verbindungen) auch im Fehlerfall freigegeben werden.

Diese Struktur führt dazu, dass der Code „atmen" kann. Die klare Trennung verhindert das typische „Test-Knäuel", bei dem Setup-Logik und Assertions ununterscheidbar vermischt sind. Ein Beispiel für eine komplette Spock-Spezifikation ist in *Listing 1* abgebildet.

```
class MyFirstSpec extends Specification {
  def "Eine simple Spezifikation zur Demonstration"() {
        given: "ein initialer Zustand"
        def list = [1, 2, 3]

        when: "eine Aktion ausgeführt wird"
        list.add(4)

        then: "tritt das erwartete Ergebnis ein"
        list.size() == 4
  }
}
```

Listing 1: Beispiel für eine komplette Spock-Spezifikation

```
def "Analyse einer komplexen String-Operation"() {
     given:
           def name = "Java aktuell"
           def version = 2026

     expect:
           name.toUpperCase().startsWith("K") && version > 2030
}
```

Listing 2: Eine Spezifikation, die fehlschlagen wird

```
Condition not satisfied:

name.toUpperCase().startsWith("K") && version > 2030
|    |             |              |  |       |
|    JAVA AKTUELL  false          |  2026    false
Java aktuell                     false
```

Listing 3: Detailliertes Feedback zum fehlgeschlagenen Test

```
def "Berechnung der Versandkosten nach Bestellwert"() {
  expect:
  service.calculateCosts(bestellwert) == versandkosten

  where:
  bestellwert || versandkosten
  10.00       || 4.95
  25.50       || 3.95
  50.00       || 0.00
  150.00      || 0.00
}
```

Listing 4: Data-Driven Testing

Das Faszinierende daran ist die Barrierefreiheit: Eine gut geschriebene Spock-Spezifikation kann oft von einem Product Owner oder einem QA-Experten verstanden werden, ohne dass dieser tief in die Syntax eintauchen muss. Spock schlägt damit die Brücke zwischen der Business-Logik und dem ausführbaren Code, was die „Single Source of Truth" für das Systemverhalten schafft.

Das Ende der Raterei: aussagekräftiges Feedback bei Fehlschlägen

Ein frustrierender Moment im Entwickleralltag ist ein fehlschlagender Test in der CI-Pipeline, der lediglich eine Fehlermeldung wie `java.lang.AssertionError: expected <true> but was <false>` ausgibt. Ohne lokales Debugging ist die Ursache dann kaum zu ermitteln. Spock löst dieses Problem elegant mit den in Groovy sogenannten *Power Assertions*.

Konkret bedeutet das: in Spock benötigt man keine speziellen Assert-Methoden wie `assertThat` oder `assertEquals`. Jedes Statement innerhalb eines `then`- oder `expect`-Blocks, das einen Wahrheitswert liefert, wird *automatisch* als Assertion behandelt. Scheitert diese, erzeugt Spock eine aussagekräftige visuelle Aufschlüsselung des gesamten Ausdrucks. Nehmen wir als Beispiel eine einfache Spezifikation wie in *Listing 2*.

Wie der geneigte Entwickler schnell feststellen wird, schlägt dieser Test natürlich fehl. Spock präsentiert uns daraufhin im Terminal eine detaillierte ASCII-Grafik, die jeden Zwischenschritt der Auswertung zeigt *(siehe Listing 3)*.

Diese Darstellung macht den Zustand des Systems zum Zeitpunkt des Fehlers auf einen Blick transparent. Man sieht sofort, dass nicht nur der String-Vergleich fehlschlägt, sondern auch die Versionsprüfung. Besonders bei komplexen Objekthierarchien oder Collections spart diese Eigenschaft massiv Zeit beim Debugging. Da Spock die Bytecode-Instrumentierung nutzt, um diese Informationen zu extrahieren, muss der Entwickler keinerlei Zusatzaufwand betreiben – die Magie passiert komplett unter der Haube.

Data-Driven Testing: Die Magie der Tabellen

In der klassischen Testautomatisierung mit JUnit führt die Prüfung verschiedener Datenkonstellationen oft zu zwei unschönen Szenarien: Entweder man kopiert den Testfall mehrfach (Code-Duplizierung) oder man nutzt `@ParameterizedTest`, was häufig in unhandlichen Argument-Streams oder externen CSV-Dateien endet.
Spock bietet hierfür eine syntaktische Eleganz, die ihresgleichen sucht: den `where`-Block.

Hier werden Testdaten direkt in einer Markdown-ähnlichen Tabellenstruktur im Quellcode definiert. Dies ist nicht nur übersichtlich, sondern dient gleichzeitig als lebende Dokumentation der Business-Logik, wie das Beispiel in *Listing 4* verdeutlicht.

Ein potenzieller Nachteil parametrisierter Tests ist, dass sie in der IDE oft als ein einziger großer Testlauf erscheinen. Schlägt ein einzelner Datensatz fehl, ist die Ursache im Testreport schwer zu isolieren. Spock löst dies über das sogenannte „Method Unrolling", welches in neueren Versionen standardmäßig aktiviert ist. Durch die Verwendung von Platzhaltern im Methodennamen wird dabei jeder Tabelleneintrag als individueller Testfall in der IDE angezeigt *(siehe Listing 5)*.

```
def "Bestellwert #bestellwert ergibt Versandkosten von #versandkosten"() {
     // ... wie in Listing 4
}
```

Listing 5: Method Unrolling

In der Test-Execution-View der IDE oder im CI-Report erscheint nun jede Zeile der Tabelle als eigenständiger, aussagekräftig benannter Testschritt („Bestellwert 25,50 ergibt Versandkosten von 3,95“).

Die Tabellen im where-Block sind dabei nicht auf primitive Datentypen beschränkt. Da es sich um echten Groovy-Code handelt, können dort wie in *Listing 6* auch komplexe Objekte, Listen oder sogar Methodenaufrufe stehen.

Durch die Trennung mit „||“ (optional, aber empfohlen für die visuelle Trennung von Input und Output) wird auf den ersten Blick klar, welche Eingangsgrößen zu welchem Ergebnis führen sollen. Für Fachbereiche oder Product Owner ist dieser Teil des Codes oft ohne Programmierkenntnisse lesbar, was die Brücke zwischen Technik und Business schlägt.

Mocking und Stubbing: Native Integration ohne Ballast

In der Java-Welt ist Mockito der De-facto-Standard für das Mocking von Abhängigkeiten. Obwohl Mockito ein mächtiges Werkzeug ist, führt es oft zu einem „Arrange-Act-Assert“-Muster, das durch explizite `when(...).thenReturn(...)` Aufrufe aufgebläht wird. Spock integriert Mocking, Stubbing und Spy-Funktionalitäten direkt in seine Kernsyntax, was zu deutlich schlankeren Tests führt.

Ein Mock wird in Spock einfach über den Aufruf `Mock()` erzeugt. Das Besondere: Die Definition des Verhaltens (Stubbing) und die Überprüfung der Interaktion (Mocking) können wie in *Listing 7* elegant zusammengefasst werden.

Spock nutzt eine sehr intuitive Syntax für die Kardinalität von Aufrufen. Möchte man sicherstellen, dass eine Methode niemals aufgerufen wurde, schreibt man einfach `0 * service.method()`. Für eine beliebige Anzahl nutzt man den Unterstrich: `_ * service.method()`.

Besonders faszinierend ist zudem die Flexibilität bei den Argumenten:

- `service.process(_)`: Akzeptiert jedes Argument.
- `service.process(!null)`: Akzeptiert nur Argumente, die nicht null sind.
- `service.process({ it.amount > 100 })`: Nutzt einen Closure (Code-Block) für komplexe Validierungen.

Während Mocks zur Überprüfung von Interaktionen dienen (Hat der Service die E-Mail wirklich versendet?), nutzt man `Stub()`, wenn man lediglich Testdaten bereitstellen möchte, ohne die Interaktion selbst zu validieren. Ein Stub antwortet immer auf die gleiche Weise, egal wie oft er aufgerufen wird.

Durch die lockere Typisierung von Groovy im Testumfeld entfällt zudem das oft mühsame Hantieren mit `ArgumentCaptor`. Man kann Argumente einfach während der Interaktionsprüfung abgreifen und gegen sie validieren. Das Ergebnis ist Testcode, der sich auf das *Verhalten* konzentriert, anstatt sich in technischer Setup-Logik zu verlieren.

Integration in den Java-Alltag: Keine Angst vor Groovy

Ein häufiges Missverständnis ist die Annahme, dass der Einsatz von Spock ein „Groovy-Projekt“ voraussetzt. In der Realität ist das Gegenteil der Fall: Spock ist ein hervorragender Gast in reinen Java-Projekten. Da Groovy zu Standard-Bytecode kompiliert wird, bemerkt die JVM zur Laufzeit keinen Unterschied zwischen einer Java-Klasse und einer Spock-Spezifikation.

Die Integration in Build-Management-Tools ist zudem denkbar einfach. In **Gradle** ist die Unterstützung durch das groovy-Plugin eingebaut. In **Maven** hilft das `gmavenplus-plugin`, um die Test-Sourcen während der Test-Phase zu kompilieren. Der entscheidende Vorteil: Die Groovy-Abhängigkeit wird nur für den `test`-Scope benötigt. Das bedeutet, dass Ihre produktive Artefakt-Datei (JAR/WAR) keinerlei Groovy-Libraries enthält.

Für Java-Enterprise-Entwickler ist zudem die Integration mit Spring Boot essenziell. Dank des `spock-spring`-Moduls fühlt sich Spock

```
where:
user                    | action        || expectedStatus
new User(role: "GUEST") | Action.DELETE || Status.FORBIDDEN
adminUser               | Action.DELETE || Status.SUCCESS
```

Listing 6: Code im where-Block

```
def "Einen Rabatt nur gewähren, wenn der Gutschein gültig ist"() {
     given:
     def couponService = Mock(CouponService)
     def calculator = new PriceCalculator(couponService)

     when:
     def finalPrice = calculator.calculate(100.0, "SAVE20")

     then:
     // Stubbing und Mocking in einer Zeile:
     // Wir erwarten genau 1 Aufruf mit "SAVE20" und geben `true` zurück
     1 * couponService.isValid("SAVE20") >> true
     finalPrice == 80.0
}
```

Listing 7: Kombiniertes Mocking und Stubbing

hier wie zu Hause. Die bekannten Annotationen wie `@SpringBootTest`, `@MockBean` oder `@DataJpaTest` funktionieren nahtlos.

Man muss sich zudem nicht entscheiden: Spock und JUnit können friedlich nebeneinander im selben Projekt existieren. Da Spock 2.0 auf der JUnit 5 Platform (JUnit Platform Engine) basiert, werden Spock-Spezifikationen von allen modernen IDEs und Build-Tools wie ganz normale JUnit-Tests behandelt und ausgeführt.

Dies erlaubt Teams eine „sanfte Migration": Neue, komplexe Business-Features werden mit Spock spezifiziert, während bestehende JUnit-Tests unverändert bleiben. Die Lernkurve für Java-Entwickler ist dabei flach, da Groovy fast wie „Java ohne Semikolons" gelesen werden kann und die Mächtigkeit der Spock-DSL die syntaktischen Unterschiede schnell vergessen lässt.

Fazit: Ein Gewinn für die Code-Qualität und das Team

Spock ist mehr als nur ein weiteres Test-Framework; es ist ein Werkzeug, das die Art und Weise verändert, wie wir über Anforderungen und deren technische Umsetzung nachdenken. Durch die konsequente Ausrichtung auf Behavior-Driven Development (BDD) zwingt es uns förmlich dazu, Tests zu schreiben, die nicht nur den Code verifizieren, sondern ihn auch erklären.

Die Symbiose aus der Lesbarkeit von Groovy und der strukturellen Strenge der `given-when-then`-Blöcke sorgt dafür, dass Tests zu einer verlässlichen Dokumentation werden. Die Power Assertions nehmen dem Debugging den Schrecken, während die Data Tables eine Testabdeckung ermöglichen, die in purem Java oft zu unübersichtlichem Boilerplate-Code führt.

Für Java-Entwickler ist die Hürde dabei minimal. Dank der Integration in die JUnit 5 Plattform und der hervorragenden Unterstützung durch Build-Tools und IDEs lässt sich Spock ohne Risiko evaluieren. Man muss nicht über Nacht die gesamte Codebasis umstellen – es genügt, mit einer einzigen, komplexen Business-Anforderung zu beginnen, um die Vorzüge der „magischen" Tabellen und der nativen Mocks zu erleben.

Wer einmal die Klarheit einer scheiternden Power Assertion gesehen oder die Effizienz eines tabellenbasierten Tests gespürt hat, wird nur ungern zu klassischen Assertions zurückkehren. Spock macht das Testen nicht nur einfacher, sondern tatsächlich regelrecht faszinierend. Es ist eine Einladung an uns Entwickler, die Qualitätssicherung nicht als lästige Pflicht, sondern als integralen, kreativen Teil des Software-Designs zu begreifen.

Sebastian Lauth
sebastian_lauth@deichmann.com
sebastian@slauth.de

Sebastian verantwortet als Tech & Team Lead bei Deichmann Entwicklungsprojekte, die die nahtlose Verknüpfung zwischen dem stationären Filialgeschäft und dem Online-Handel sicherstellen. In seiner Rolle kombiniert er technische Führung mit der Leidenschaft für moderne Software-Methodiken, um komplexe Handelslogik in zukunftsfähige IT-Lösungen zu überführen.

Technical Debt ist eine Entscheidung, kein Zustand

Christian Seifert, gofore

„Das sind halt technische Schulden", ein Satz, der in Softwareprojekten erstaunlich oft das Ende einer Diskussion markiert. Doch was, wenn er in Wahrheit den Beginn eines Problems darstellt? Warum Technical Debt kein Freifahrtschein für schlechte Software ist, weshalb Schulden ohne Rückzahlplan keine Schulden sind – und warum Qualität keine Option, sondern eine Verpflichtung ist.

In nahezu jedem professionellen Softwareprojekt gibt es diesen Moment: Im Architektur-Review, in der Retrospektive oder bei der Planung der nächsten größeren Erweiterung sagt jemand sinngemäß: *„Ja, das wissen wir – das sind technische Schulden."* Häufig folgt ein zustimmendes Nicken, manchmal ein Schulterzucken, und dann geht man zum nächsten Punkt über.

Was hier auf den ersten Blick effizient wirkt, ist bei genauerem Hinsehen problematisch. Denn der Begriff wird in diesen Situationen oft nicht verwendet, um eine bewusste technische oder architektonische Entscheidung zu markieren, sondern um eine Diskussion abzukürzen. *Technical Debt* wird zu einer Art Containerbegriff: Er klingt sachlich, technisch und professionell – und verhindert gerade dadurch, dass genauer hingeschaut wird. Die eigentliche Frage, *worum es konkret geht* und *warum der Zustand so ist, wie er ist*, bleibt unbeantwortet.

Genau darin liegt die Gefahr. Ein Begriff, der ursprünglich helfen sollte, Verantwortung sichtbar zu machen, wird zum sprachlichen Abschluss. Anstatt Entscheidungen nachvollziehbar zu machen, schafft er Distanz. Anstatt Klarheit zu erzeugen, nivelliert er Unterschiede – zwischen bewusstem Kompromiss und stiller Vernachlässigung.

Das ist besonders problematisch, weil Softwareentwicklung längst nicht mehr nur aus Code besteht. Sie ist ein sozio-technisches System aus Architektur, Organisation, Kommunikation und Verantwortung. Begriffe wie *Technical Debt* wirken in diesem System nicht neutral. Sie strukturieren Diskussionen, beeinflussen Prioritäten und setzen implizite Standards dafür, was als akzeptabel gilt – und was nicht mehr hinterfragt wird.

Wenn Technical Debt zur bequemen Erklärung für alles wird, was schwierig oder unangenehm ist, verlieren wir genau diese Differenzierung. Und damit verlieren wir ein wichtiges Instrument, um Qualität bewusst zu steuern – technisch wie organisatorisch.

Was mit Technical Debt eigentlich gemeint ist

Der Begriff *Technical Debt* ist kein Synonym für einen allgemein „schlechten Zustand". Er beschreibt eine bewusste Entscheidung unter bekannten Zielkonflikten.

Technische Schulden entstehen dann, wenn ein Team – oder eine Organisation – sich entscheidet, eine Lösung zu implementieren, von der allen Beteiligten klar ist, dass sie nicht dem gewünschten langfristigen Qualitätsniveau entspricht, um dafür kurzfristig einen konkreten Nutzen zu erzielen.

Dieser Nutzen kann vielfältig sein:

- das Einhalten regulatorischer oder vertraglicher Deadlines
- das Nutzen eines engen Marktfensters
- die begrenzte Verfügbarkeit kritischer Ressourcen oder spezieller Expertise

Entscheidend ist: Diese Entscheidung wird **wissentlich** getroffen. Man weiß, dass man später einen Preis zahlen wird – und akzeptiert diesen Preis bewusst.

Ein typisches Beispiel aus dem Enterprise-Alltag:
Ein fachlich zentrales Modul soll in kurzer Zeit produktiv gehen, weil ein neuer Kunde angebunden werden muss oder ein regulatorischer Stichtag näher rückt. Die fachliche Logik wird sauber umgesetzt, die Architektur ist grundsätzlich tragfähig, – aber Details bleiben bewusst liegen. Persistenz-Zugriffe werden mit einfachen ORM-Defaults umgesetzt, obwohl klar ist, dass bei steigender Last Optimierungen notwendig sein werden. Fehlerfälle werden geloggt, aber fachlich noch nicht sauber behandelt. Tests konzentrieren sich auf Unit-Ebene, Integrations- oder Contract-Tests werden auf später verschoben.

Das System funktioniert. Es erfüllt seinen Zweck. Und gleichzeitig ist allen Beteiligten klar: So soll es langfristig nicht bleiben.

Typische, reale Ausprägungen solcher bewusst aufgenommenen Schulden sind:

- APIs, die funktional korrekt sind, aber schlecht geschnittene Ressourcen, inkonsistente Semantik oder unklare Verantwortlichkeiten aufweisen – zum Beispiel, weil ein sauberer Schnitt mehr Abstimmung mit anderen Teams erfordert hätte
- Persistenz-Schichten, die mit Default-Konfigurationen arbeiten, obwohl absehbar ist, dass bestimmte Abfragen bei wachsendem Datenvolumen nicht skalieren werden
- Implementierungen, die ausschließlich den „Happy Path" abdecken und Fehlerfälle nur rudimentär behandeln, um zunächst fachliche Kernprozesse zu ermöglichen
- reduzierte Testabdeckung, bei der bewusst auf Integrations- oder Contract-Tests verzichtet wird, weil deren Aufbau Zeit, Infrastruktur und Koordination erfordert
- Benutzeroberflächen, die intern als „ausreichend" gelten, aber nicht auf reale Nutzung, Barrierefreiheit oder zukünftige Erweiterbarkeit ausgelegt sind

All das kann sinnvoll sein – wenn klar ist, warum es so umgesetzt wurde, **welche Konsequenzen das hat** und **unter welchen Bedingungen investiert wird**, um diese Schulden wieder abzubauen.

Genau an dieser Stelle unterscheidet sich *Technical Debt* von schlechter Qualität: nicht durch das Ergebnis im Code, sondern durch die Bewusstheit der Entscheidung und den Umgang mit ihren Folgen.

Schulden bringen Zinsen mit – und zwar dauerhaft

Was in der Praxis häufig unterschätzt wird, sind die Folgekosten technischer Schulden. Technische Schulden sind kein statischer Zustand, den man einmal hinnimmt und dann ignorieren kann. Sie wirken dauerhaft – auf das System selbst und auf die Menschen, die täglich mit ihm arbeiten.

Mit jeder weiteren Änderung machen sich diese Schulden bemerkbar. Und zwar nicht abstrakt, sondern ganz konkret im Alltag von Entwicklung, Betrieb und Fachbereich.

Ein schlecht designtes API bedeutet zum Beispiel nicht nur „unschönen Code". Es bedeutet praktisch:

- längere Implementierungszeiten bei jeder Erweiterung, weil mehr Sonderfälle berücksichtigt werden müssen
- erhöhte Abstimmung zwischen Teams, weil Verantwortlichkeiten und Semantik nicht klar sind
- höhere kognitive Last für neue Teammitglieder, die sich mühsam in implizite Regeln einarbeiten müssen
- steigende Fehleranfälligkeit in Integration und Betrieb, weil Schnittstellen falsch oder uneinheitlich genutzt werden

Ähnlich verhält es sich mit fehlender oder unzureichender Fehlerbehandlung. Das bleibt selten ein rein technisches Problem. Stattdessen führt es zu:

- inkonsistenten Datenzuständen, die nachträglich analysiert und korrigiert werden müssen
- manuellen Eingriffen im Betrieb, oft unter Zeitdruck und mit unvollständigem Kontext
- wachsendem Supportaufwand, weil Anwender auf unerklärliche Zustände stoßen
- sinkendem Vertrauen der Anwender in das System und seine Verlässlichkeit

Unzureichende Testabdeckung wiederum hat häufig weniger unmittelbare Auswirkungen im Code, sondern vor allem auf die Organisation rund um das System:

- zusätzliche Freigabeprozesse, weil niemand mehr sicher ist, welche Nebenwirkungen Änderungen haben könnten
- manuelle Regressionstests, die Zeit kosten und fehleranfällig sind
- längere Release-Zyklen, weil jede Änderung aufwendiger abgesichert werden muss
- sinkender Mut, Änderungen vorzunehmen, insbesondere in zentralen oder kritischen Komponenten

Diese Effekte stehen nicht für sich. Sie **verstärken sich gegenseitig**. Je mehr technische Schulden bestehen, desto größer werden ihre Zinsen. Aus einzelnen Schwächen entsteht nach und nach ein System, das immer schwerer veränderbar wird.

Genau hier entstehen die **Zinseszinsen** technischer Schulden: Jede weitere Entscheidung wird teurer als die vorherige. Nicht, weil Entwickler schlechter arbeiten, sondern weil das System selbst Widerstand gegen Veränderung aufbaut. Und genau an diesem Punkt wird aus einem zunächst akzeptablen Kompromiss ein ernsthaftes Risiko für die Weiterentwicklung des Systems.

Technische Schulden ohne Rückzahlplan sind keine Schulden

Ein Aspekt ist entscheidend – und wird dennoch häufig ignoriert: **Technische Schulden existieren nur dann, wenn sie einen Rückzahlplan haben**.

Ohne eine Vorstellung davon, wie und unter welchen Bedingungen Schulden wieder abgebaut werden sollen, handelt es sich nicht um einen bewussten Trade-off, sondern um einen Zustand, mit dem man sich arrangiert hat.

Ein Rückzahlplan bedeutet dabei nicht zwangsläufig ein festes Datum oder eine detaillierte To-do-Liste. Er bedeutet vor allem Klarheit:

- Es ist bekannt, was konkret verbessert werden muss – und nicht nur, dass „irgendetwas unschön ist".
- Es ist klar, warum diese Verbesserung bisher nicht erfolgt ist – etwa wegen Zeitdruck, fehlender Ressourcen oder bewusster Priorisierung.
- Es gibt eine Vorstellung davon, unter welchen Bedingungen investiert wird - zum Beispiel bei der nächsten größeren Erweiterung oder ab einer bestimmten Systemlast.

Das kann unterschiedliche Formen annehmen. Es kann bedeuten, dass ein Team bewusst über mehrere Iterationen hinweg weniger neue Features liefert, um bestehende Schwächen systematisch abzubauen. Es kann bedeuten, dass bestimmte Erweiterungen nur noch in Kombination mit Refactorings umgesetzt werden. Es kann auch bedeuten, dass man sich bewusst gegen weitere Funktionalität entscheidet, bis ein Mindestmaß an Qualität und Stabilität wiederhergestellt ist.

Wichtig ist: All das sind **strategische Entscheidungen**. Sie betreffen nicht nur Entwickler, sondern Produktverantwortliche, Architekten und Management gleichermaßen. Technische Schulden sind damit kein rein technisches Thema, sondern eine Frage gemeinsamer Verantwortung.

Fehlt dieser Plan, handelt es sich nicht um technische Schulden. Dann handelt es sich um **strukturelle Vernachlässigung** – unabhängig davon, wie oft der Begriff *Technical Debt* verwendet wird.

Wenn alles *Technical Debt* ist, wird nichts mehr klar

Der inflationäre Gebrauch des Begriffs Technical Debt ist deshalb so problematisch, weil er **reale Probleme unsichtbar macht**. Was ursprünglich helfen sollte, bewusste technische Entscheidungen einzuordnen, wird zu einem Sammelbegriff für alles, was unbequem, teuer oder schwierig geworden ist.

Fehlendes Wissen wird dann nicht mehr als Lern- oder Qualifikationsproblem adressiert, sondern als technischer Zustand hingenommen. Anstatt in Weiterbildung, Pairing oder gezielten Wissenstransfer zu investieren, arrangiert man sich mit dem Status quo.

Schlechte Implementierungen werden nicht mehr als handwerkliche Mängel benannt. Unscharfe Architektur, unklare Verantwortlichkeiten oder fehlende Tests gelten plötzlich als „historisch gewachsen". Der Begriff *Technical Debt* übernimmt die Rolle einer stillschweigenden Entschuldigung.

Permanenter Lieferdruck wird ebenfalls gerne hinter dem Begriff verborgen. Wenn Teams ohne Spielraum für Qualität dauerhaft unter Zeitdruck arbeiten, handelt es sich nicht um technische Schulden, sondern um ein organisatorisches Versagen. Wird auch das als Technical Debt etikettiert, verschwindet das eigentliche Problem aus der Diskussion.

Alles verschwindet hinter einem technisch klingenden Begriff, der Verantwortung verwischt. Das ist bequem – weil niemand klar benennen muss, wo Entscheidungen getroffen oder eben nicht getroffen wurden. Aber es ist hochgradig gefährlich, weil es systematisch verhindert, dass Ursachen sichtbar werden.

Denn was wir nicht klar benennen, können wir auch nicht gezielt verbessern. Und genau deshalb schadet uns der unscharfe Umgang mit *Technical Debt* am meisten selbst – als Entwickler, als Teams und als Organisationen.

Technische Schulden, Qualität und Sprache – warum Begriffe keine Nebensache sind

Technische Schulden sind nicht dasselbe wie schlechte Qualität. Diese Unterscheidung ist keine akademische Feinheit, sondern hat ganz praktische Auswirkungen auf die Art und Weise, wie wir Software entwickeln und verbessern.

Technische Schulden entstehen aus einem bewussten Kompromiss. Sie sind das Ergebnis einer Entscheidung, bei der kurzfristige Vorteile gegen langfristige Kosten abgewogen werden. Schlechte Qualität hingegen entsteht nicht aus einem solchen Trade-off. Sie ist das Resultat von Nachlässigkeit, Überforderung, fehlendem Wissen oder dauerhaftem Druck – und sie liefert keinen kurzfristigen Nutzen, der spätere Kosten rechtfertigen würde.

Wenn wir schlechte Qualität als *Technical Debt* bezeichnen, verwischen wir genau diese Grenze. Wir machen aus einem Problem einen scheinbar legitimen Zustand. Fehlende Tests, fragile Architektur oder unklare Schnittstellen werden nicht mehr als Mängel wahrgenommen, sondern als etwas, das man „halt hat". Der Begriff verleiht ihnen eine technische Seriosität, die sie nicht verdienen.

Hier kommt die Rolle der Sprache ins Spiel. Begriffe wie *Technical Debt* sind nicht neutral. Sie steuern Wahrnehmung, Diskussionen und Prioritäten. Wer etwas als technische Schuld bezeichnet, signalisiert implizit: Das ist bekannt, das ist akzeptiert, das ist Teil des Systems. Genau dadurch verlieren Probleme ihre Dringlichkeit.

Susan Sontag brachte es prägnant auf den Punkt: *„Naming is a way of controlling reality."* In der Softwareentwicklung zeigt sich das sehr konkret. Wenn wir alles, was unangenehm ist, unter dem Begriff Technical Debt zusammenfassen, nehmen wir uns selbst die Möglichkeit, präzise zu unterscheiden – zwischen bewusster Entscheidung und strukturellem Versagen, zwischen temporärem Kompromiss und dauerhaft schlechter Qualität.

Damit schaden wir uns selbst. Wir senken den Qualitätsanspruch, ohne es bewusst zu entscheiden. Wir normalisieren Zustände, die eigentlich Anlass zur Verbesserung wären. Und wir machen es schwerer, Verantwortung zuzuordnen und gezielt zu handeln.

Gerade deshalb ist präzise Sprache kein Nebenthema. Sie ist ein zentrales Steuerungsinstrument. Nur wenn wir Probleme beim richtigen Namen nennen, können wir entscheiden, wie wir mit ihnen umgehen wollen – und ob wir sie überhaupt akzeptieren dürfen.

Technical Debt bewusst einsetzen – Qualität aktiv sichern

Am Ende geht es nicht um perfekte Definitionen oder theoretische Modelle. Es geht um **Verantwortung** und darum, wie wir sie im Alltag wahrnehmen.

Technische Schulden dürfen kein bequemer Sammelbegriff sein, hinter dem alles verschwindet, was unangenehm ist. Wenn wir den

Begriff verwenden, dann bewusst. Technische Schulden müssen benannt, begründet und eingeordnet werden. Und sie müssen Konsequenzen haben – für Planung, Priorisierung und Umsetzung.

Das ist nichts Abstraktes. Jede Entwicklerin und jeder Architekt kann und muss hier aktiv werden.

Konkret bedeutet das:

- **Technische Schulden sichtbar machen:** Technische Schulden gehören nicht in implizite Annahmen oder in die Köpfe Einzelner, sondern in Architekturentscheidungen, Backlogs und technische Roadmaps. Nicht als Schlagwort, sondern konkret: *Was ist die Schuld? Wo liegt sie? Welche Auswirkungen hat sie?*
- **Klar benennen, was die Schuld ist und was nicht:** Nicht alles, was schwierig oder teuer geworden ist, ist *Technical Debt*. Schlechte Qualität, fehlende Tests oder unsaubere Architektur müssen auch als solche benannt werden. Nur so bleiben Probleme adressierbar.
- **Erweiterungen an Rückzahlung koppeln:** Neue Features auf bestehenden Komponenten sollten nicht losgelöst von bekannten Schulden umgesetzt werden. Wer erweitert, muss auch verbessern. Das ist kein Selbstzweck, sondern eine Investition in zukünftige Entwicklungsfähigkeit.
- **Qualität nicht relativieren, auch wenn Druck entsteht:** Zeitdruck erklärt Entscheidungen, rechtfertigt aber keine dauerhafte Absenkung von Qualitätsstandards. Tests, saubere Schnittstellen und verständliche Architektur sind keine Extras, die man nur dann einbaut, wenn „noch Zeit übrig ist".
- **Sprache bewusst einsetzen:** Begriffe setzen Maßstäbe. Wer schlechte Qualität sprachlich als *Technical Debt* verharmlost, senkt implizit den Qualitätsanspruch. Präzise Sprache ist ein aktiver Beitrag zu besserer Software.

Das alles ist nicht immer bequem. Es erfordert Haltung, manchmal auch Widerspruch. Aber genau das ist Teil professioneller Softwareentwicklung.

Technische Schulden bewusst einzugehen ist professionell. Schlechte Qualität zu akzeptieren ist es nicht.

Qualität ist kein Luxus und kein Idealismus. Sie ist Voraussetzung dafür, dass Software langfristig wartbar, erweiterbar und vertrauenswürdig bleibt – für Anwender, für Organisationen und nicht zuletzt für uns selbst.

Wenn wir bessere Software bauen wollen, müssen wir aufhören, uns hinter dem Begriff Technical Debt zu verstecken. Stattdessen müssen wir ihn bewusst einsetzen – und gleichzeitig klar für Qualität einstehen. Genau das liegt in unserer Hand.

Christian Seifert
gofore
perdian@perdian.de

Christian Seifert ist Brückenbauer zwischen Code, Köpfen und Komplexität. Als Principal Software Architect bei gofore unterstützt er Unternehmen dabei, ihre Softwarelandschaften zukunftssicher zu gestalten – technisch fundiert, nachhaltig und mit Fokus auf starke Teams. Seit über 20 Jahren gestaltet und verantwortet er Softwareprojekte in unterschiedlichen Branchen und hat dabei gelernt, dass erfolgreiche Architektur nicht nur aus technischen Details besteht, sondern vor allem aus Menschen, die gemeinsam an guten Lösungen arbeiten.

CLOUD NATIVE FESTIVAL

im Heide Park Soltau

CloudLand
www.cloudland.org

19. – 22. MAI 2026

Aufregende Atmosphäre eines Freizeitparks!

Neueste Trends & Innovationen rund um Cloud Native!

Ein Treffen mit den Hyperscalern AWS, Microsoft Azure & Google Cloud!

Spannende Vorträge, interaktive Workshops & kreatives Networking!

#CLOUDLAND2026

Der undichte Karriereweg: Status quo und Perspektiven für Frauen in der IT

Ildikó Tárkányi, QAware GmbH

Softwareentwicklung war ursprünglich weiblich geprägt – heute ist sie eine Branche, die an kritischen Schnittstellen massiv Talente verliert. Dieser Artikel analysiert das Phänomen des „undichten Karrierewegs": Warum verlässt fast die Hälfte der weiblichen IT-Talente das System zwischen Studium und Führungsebene? Von den historischen Ursachen über kulturelle Barrieren bis hin zu neuen gesetzlichen Transparenzpflichten beleuchtet der Beitrag die strukturellen Bruchstellen. Er liefert konkrete Lösungsansätze, um den Karriereweg nachhaltig abzudichten und Diversität als operativen Erfolgsfaktor zu etablieren.

Heute wird oft so getan, als sei Informatik von Natur aus ein männliches Feld. Wer jedoch in der Geschichte zurückblättert, stellt fest: Softwareentwicklung war ursprünglich oft „Frauenarbeit". Die Geschichte unserer Branche ist untrennbar mit den Leistungen von Frauen verbunden, auch wenn die öffentliche Wahrnehmung über Jahrzehnte hinweg ein völlig anderes Bild zeichnete.

Ein Blick zurück: Als Code noch weiblich war

In den 1940er Jahren galt das Bauen der Hardware als prestigeträchtige Männerarbeit. Die Ingenieure verstanden dies als klassische, harte Technik. Das Programmieren der Maschinen hingegen sahen sie als rein ausführende, fast mechanische Tätigkeit an. Zeitgenossen verglichen es mit dem Stricken oder dem Befolgen eines Kochrezepts – Tätigkeiten, die die Gesellschaft damals primär Frauen zuschrieb.

Ein Team aus sechs Frauen programmierte beispielsweise den ENIAC, den ersten programmierbaren Universalrechner der Welt. Bereits im 19. Jahrhundert legte Ada Lovelace das theoretische Fundament, indem sie den ersten Algorithmus für eine mechanische Rechenmaschine entwarf. In den 1950er Jahren trieb Grace Hopper die Entwicklung voran: Sie erfand den ersten Compiler und ebnete damit den Weg für moderne Programmiersprachen wie COBOL. Dass wir heute sicher auf dem Mond landen könnten, verdanken wir ebenfalls einer Frau: Margaret Hamilton leitete die Entwicklung der Bordsoftware für die Apollo-Missionen. *Abbildung 1* zeigt eine Darstellung dieser und weiterer Pionierinnen der Softwareentwicklung.

Der kulturelle „Hard-Reset" der 80er Jahre

Wie konnte sich dieses Bild so radikal drehen? In den frühen 80ern realisierte die Wirtschaft, dass Software ganze Gesellschaften revolutionieren würde. In dem Moment, als Code „wertvoll" wurde, stieg das Ansehen des Berufsfeldes massiv an.

Parallel dazu hielten Heimcomputer Einzug in die Kinderzimmer. Das Marketing dieser Geräte zielte jedoch einseitig auf Jungen ab. Die Hersteller positionierten Computer als Spielzeug für Technikbegeisterte Jungs, während Mädchen Spielzeugpuppen erhielten. Dieser kulturelle Bruch führte dazu, dass eine ganze Generation von Jungen mit einem Vorsprung an technischem Selbstvertrauen aufwuchs. Mädchen hingegen erhielten systematisch die Botschaft: „Das ist nichts für dich."

Status quo: Eine Bewegung zwischen Hype und Notwendigkeit

Wir erleben heute eine Zeit, in der das Thema „Women in Tech" allgegenwärtig scheint. Ob auf großen Fachkonferenzen, in Business-Netzwerken oder in den Geschäftsberichten der globalen Player: Die Forderung nach mehr Diversität in der IT ist präsenter denn je.

Dabei ist die Bewegung längst kein kurzfristiger Trend mehr. Heute treiben wirtschaftliche Notwendigkeit, gesellschaftlicher Wandel und neue gesetzliche Vorgaben das Thema voran. Viele Unternehmen begreifen Vielfalt inzwischen als echten Erfolgsfaktor für ihre Innovation. Doch wir müssen uns kritisch fragen: Wie tief geht dieser Wandel wirklich? Erleben wir gerade eine dauerhafte Veränderung unserer Strukturen?

Der undichte Karriereweg in Zahlen

Um die Nachhaltigkeit der „Women in Tech"-Bewegung zu bewerten, müssen wir die statistischen Trends genau analysieren. Die Datenlage zeichnet ein Bild von vorsichtigem Optimismus, offenbart aber gleichzeitig auch, wie fest alte Strukturen noch sitzen. Es wird deutlich: Wir verlieren auf dem Weg von der Ausbildung bis in die Chefetage an jeder Stufe wertvolle Talente.

Bildungsweg: Rekorde mit Beigeschmack

Das Statistische Bundesamt meldete für das Studienjahr 2024 einen historischen Höchststand *[1]*: In Deutschland waren 35,5 % der Studienanfänger in den MINT-Fächern weiblich. Die Informatik ist dabei besonders beliebt und zieht mit über 45.800 Anfängerinnen die meisten Frauen an.

Ein Blick auf die langfristige Entwicklung zeigt jedoch die mühsame Natur dieses Aufstiegs *(siehe Abbildung 2)*: Im Jahr 2014 lag dieser

Abbildung 1: Pionierinnen der Softwareentwicklung
(© Copyright-Hinweis / KI-generiert mit Google Gemini)

Anteil noch bei 31,3 %. In zehn Jahren haben wir also lediglich einen Zuwachs von etwa vier Prozentpunkten erreicht – ein vergleichsweise langsamer Fortschritt. Noch deutlicher stagniert die Entwicklung in der betrieblichen IT-Ausbildung, wo der Frauenanteil seit über einem Jahrzehnt bei 12 % stagniert.

Jahr	Anzahl Männer	Anzahl Frauen	Anteil Frauen
2014	231 080	105 449	31,3 %
2015	237 262	109 746	31,6 %
2016	237 648	113 220	32,3 %
2017	236 233	115 134	32,8 %
2018	232 639	116 840	33,4 %
2019	229 629	119 134	34,2 %
2020	215 979	112 588	34,3 %
2021	201 159	105 900	34,5 %
2022	198 177	106 976	35,1 %
2023	199 938	108 741	35,2 %
2024	205 494	113 265	35,5 %

Abbildung 2: Entwicklung der Anzahl weiblicher Studienanfängerinnen in MINT-Fächern (2014 - 2024) (© Copyright-Hinweis / https://www.destatis.de, redaktionell bearbeitet durch Ildikó Tárkányi)

Der Arbeitsmarkt: Die technische Unterrepräsentanz

Während der Frauenanteil in IT-Berufen in Deutschland 2013 noch bei etwa 14 % lag, stieg er bis 2023 auf knapp 18 % an *[2]*. In absoluten Zahlen bedeutet dies einen Zuwachs auf rund 64.000 berufstätige Frauen *(siehe Abbildung 3)*. Dieser Aufwärtstrend täuscht jedoch: Während an den Universitäten bereits über 35 % Frauen starten, finden sich im Berufsleben lediglich 18 % wieder. Wir verlieren also fast die Hälfte der ausgebildeten Talente direkt beim Übergang vom Studium in die Praxis.

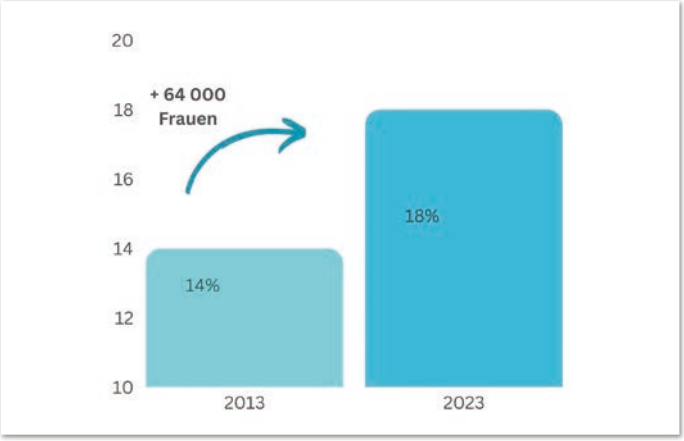


Abbildung 3: Entwicklung der Anzahl weiblicher Studienanfängerinnen in MINT-Fächern (2014 - 2024) (© Copyright-Hinweis / https://www.destatis.de, redaktionell bearbeitet durch Ildikó Tárkányi)

Darüber hinaus bleibt Deutschland im europäischen Vergleich auch im hinteren Mittelfeld zurück *(siehe Abbildung 4)*. Spitzenreiter wie Bulgarien oder Rumänien zeigen uns, dass deutlich höhere Quoten möglich sind *[3]*. Interessanterweise wählen Frauen in weniger prosperierenden Ländern häufiger MINT-Fächer als sicheren Weg zu finanzieller Stabilität. In Deutschland hingegen scheint der Wohlstand paradoxerweise dazu beizutragen, dass wir alte Rollenbilder langsamer ablegen.

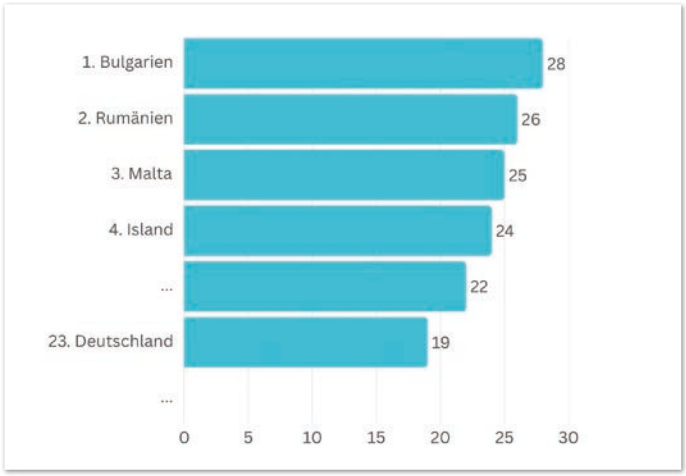


Abbildung 4: Frauenanteil in IT-Berufen im europäischen Vergleich (© Copyright-Hinweis / ec.europa.eu, redaktionell bearbeitet durch Ildikó Tárkányi)

Big Tech in Zahlen: Wo die Diversität im Engineering aufhört

Beim Blick auf die globalen Branchenführer der „Big Tech"-Gruppe setzt sich dieses Bild auf einer noch spezifischeren Ebene fort. Analysiert man die Belegschaften von Giganten wie Meta, Apple, Google und Microsoft, stellen Frauen insgesamt zwar etwa ein Drittel der Beschäftigten *(siehe Abbildung 5)*, doch wir müssen genauer hinschauen: Sobald wir uns die technischen Schlüsselpositionen wie Software Engineering oder Architektur ansehen, sinkt dieser Anteil auf etwa 23 bis 25 % *[4]*. Damit zeigt sich, dass echte Technik-Jobs bei den großen Playern weiterhin primär männlich besetzt sind.

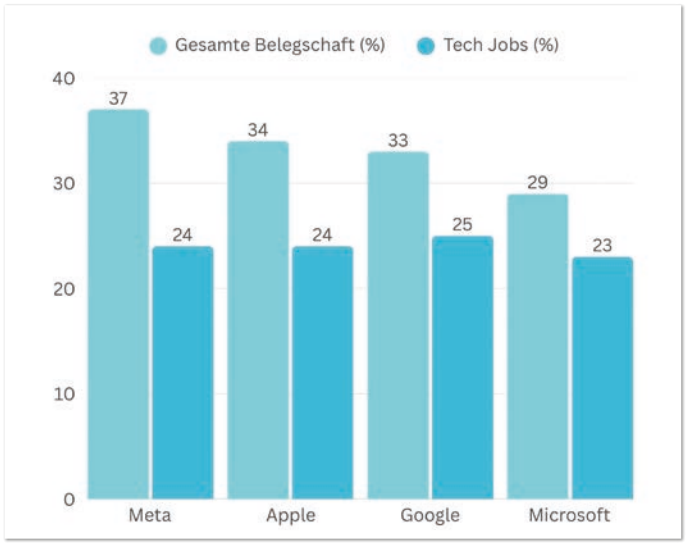


Abbildung 5: Frauenanteil bei Big-Tech-Unternehmen (© Copyright-Hinweis / https://www.statista.com, redaktionell bearbeitet durch Ildikó Tárkányi)

Die gläserne Decke: Stillstand an der Spitze

Besonders deutlich wird die strukturelle Schieflage in den obersten Hierarchieebenen. Die „gläserne Decke" fungiert hier als fast unüberwindbare Barriere *(siehe Abbildung 6)*. Weltweit halten Frauen lediglich etwa 16 % der Management-Positionen in der IT-Branche *[4]*. Schaut man direkt auf die Spitze, wird es noch einsamer: Nur 3 % der CEO-Positionen in der IT sind weiblich besetzt *[4]*. Damit bildet die Tech-Branche das Schlusslicht unter fast allen Industrien.

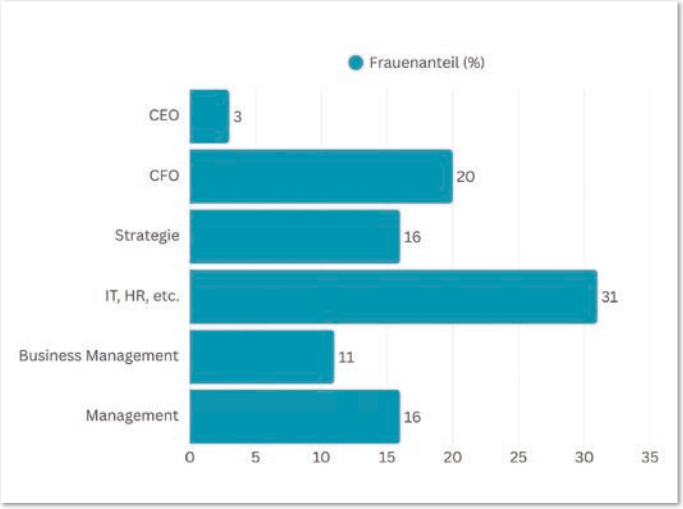


Abbildung 6: Frauenanteil in den obersten Hierarchieebenen (© Copyright-Hinweis / https://www.statista.com, redaktionell bearbeitet durch Ildikó Tárkányi)

Ursachenforschung: Warum der Karriereweg undicht ist

Hinter den ernüchternden Zahlen steht kein Naturgesetz, sie sind eine Kombination aus kulturellen und strukturellen Barrieren. Wir müssen die Bruchstellen identifizieren, an denen wir Talente verlieren: Warum werden Mädchen schon frühzeitig aus der technologischen Zukunft herausgefiltert?

Frühe Prägung: Das Bildungssystem als Filter

Die Weichenstellung beginnt lange vor dem ersten Arbeitstag. Unser Bildungssystem marginalisiert Mädchen oft durch unbewusste Barrieren und veraltete Lehrpläne. Während die moderne Informatik heute eine hochgradig kollaborative Disziplin ist, betonen viele Schulen weiterhin einseitig die abstrakte Theorie und isolierte Einzelarbeit. Dieser Fokus steht den Interessen vieler Schülerinnen häufig diametral entgegen.

Veraltete Rollenbilder bei den Lehrkräften verschärfen das Problem: Jungen wird oft ein natürliches Talent für Logik zugesprochen, während der Erfolg von Mädchen lediglich als Ergebnis von Fleiß gilt. Diese subtile Exklusion führt dazu, dass das Interesse von Mädchen an der Informatik bereits in der Mittelstufe massiv sinkt.

Mangel an Vorbildern: Die unsichtbare Barriere

Neben strukturellen Hürden bremst der Mangel an weiblichen Identifikationsfiguren den Nachwuchs psychologisch aus. Vorbilder entscheiden maßgeblich darüber, ob junge Frauen eine Karriere in der IT für realistisch und erreichbar halten. In der Realität gestaltet sich die Suche nach diesen Pionierinnen jedoch mühsam: Studien zeigen, dass nur 22 % der Befragten eine berühmte Frau aus der Tech-Branche benennen können *[3]*. Im Gegensatz dazu assoziieren zwei Drittel sofort einen prominenten männlichen Akteur.

Wenn Vorbilder fehlen, fehlt jungen Talenten die Inspiration. Ohne ein passendes Netzwerk fühlen sich viele Frauen isoliert, und der eigene Weg nach oben bleibt unsichtbar. Ohne eine bewusste Steigerung der weiblichen Sichtbarkeit – sei es auf Konferenzbühnen oder in Führungsetagen – bleibt die Bewegung für mehr Diversität ein abstraktes Versprechen.

Die „Nerd-Kultur" und exklusive Netzwerke

In der Praxis stoßen Frauen oft auf eine Barriere, die tief in der Identität der Tech-Community verwurzelt ist: die „Nerd-Kultur". Diese Kultur suggeriert, dass echte IT-Expertise eine fast schon obsessive, isolierte Beschäftigung mit Hardware und Code voraussetzt, die idealerweise bereits im Kinderzimmer begonnen hat. Dieser Stereotyp schreckt nicht nur Talente ab, die einen anderen Zugang zur Informatik haben, sondern definiert Fachwissen häufig einseitig über eine rein technische Biografie.

Ein weiteres Problem sind die gewachsenen, informellen Strukturen. Während Karrieren oft in geschlossenen Zirkeln gefördert werden, bleibt Frauen der Zugang zu diesen Netzwerken häufig verwehrt. In diesen Kreisen entstehen jedoch meist die strategischen Entscheidungen. Ohne den Zugang zu diesen informellen Informationsflüssen bleiben Frauen ausgeschlossen. Das Ergebnis ist eine homogene Kultur, die sich mangels neuer Impulse immer wieder selbst reproduziert.

Beförderungsdynamik: Die kritische Phase der Karriereplanung

Die typische Beförderungsdynamik verschärft die Situation zusätzlich. Fast 50 % aller Karrieresprünge finden in der Altersgruppe zwischen 31 und 40 Jahren statt *[5]*. Dies ist exakt der Zeitraum, in dem viele Frauen eine Familie gründen oder familiäre Verantwortung übernehmen. Ohne flexible Arbeitsmodelle und eine Kultur, die den Wiedereinstieg aktiv fördert, entsteht an dieser Stelle ein massiver Verlust an Fachkräften – der Karriereweg erweist sich hier als besonders undicht. Hochqualifizierte Talente verlassen das System, weil die Branche keine Kompatibilität zum Privatleben bietet.

Die Kosten der Ungleichheit: Warum wir mehr Frauen in der IT brauchen

Mangelnde Diversität ist kein rein gesellschaftliches Problem, sondern eine handfeste Bremse für den wirtschaftlichen und technologischen Fortschritt. Wenn wir Frauen systematisch ausbremsen, zahlen wir als Branche einen hohen Preis – in Form von fehlenden Talenten, entgangenen Gewinnen und fehlerhaften Produkten.

Fachkräftemangel: Eine Überlebensstrategie für den Wirtschaft

Der ökonomische Druck in Deutschland steigt durch den akuten Fachkräftemangel massiv an. IT-Leiter weltweit berichten, dass der Mangel an qualifizierten Talenten ihre Organisationen massiv zurückhält. In Deutschland teilen knapp zwei Drittel der Unternehmen diese Sorge *[6]*: Ohne Frauen in Digital- und IT-Berufen verspielt die deutsche Wirtschaft ihre Zukunftsfähigkeit. Wir können es uns schlicht nicht mehr leisten, die Hälfte des potenziellen Talentpools zu ignorieren.

Die Mobilisierung von Frauen ist daher keine Frage der Fairness, sondern eine notwendige Überlebensstrategie für unseren Wirtschaftsstandort. Analysen zeigen das enorme Potenzial: Eine Verdopplung des Frauenanteils in der europäischen Tech-Branche könnte das Bruttoinlandsprodukt der EU um bis zu 600 Milliarden Euro steigern [7].

Wirtschaftliche Hebelwirkung: Höhere Margen durch Vielfalt

In einer globalisierten Wirtschaft ist die Heterogenität von Entwicklungsteams ein Wettbewerbsvorteil. Studien belegen, dass diverse Teams eine höhere Wahrscheinlichkeit haben, ihre Mitbewerber in puncto Profitabilität zu übertreffen [8].

Wer Diversität ignoriert, akzeptiert freiwillig einen Wettbewerbsnachteil.

Produktdesign-Fehler: Wenn die Zielgruppe ignoriert wird

Besonders schmerzhaft wird mangelnde Diversität bei der Produktentwicklung. Wenn Frauen in den Designteams fehlen, entstehen Technologien, die für die Hälfte der Weltbevölkerung schlicht schlechter funktionieren. Ein klassisches Beispiel aus der Software-Historie ist Microsofts Assistent „Clippy“: Das überwiegend männliche Team ignorierte Berichten zufolge das negative Feedback von 90 % der weiblichen Testnutzerinnen. Die Frauen empfanden die Optik als unangenehm, was letztlich zum Scheitern und schnellen Ende des Produkts beitrug [4].

Lösungsansätze: Den Karriereweg konsequent verbessern

Wir müssen die „Women in Tech“-Bewegung von einer rein moralischen Diskussion zu einer operativen Strategie weiterentwickeln. Echte Veränderung erfordert einen tiefgreifenden Umbau unserer Strukturen.

Frühe Intervention: Bildung neu denken

Der entscheidende Hebel liegt darin, Mädchen frühzeitig und kontinuierlich mit der Informatik in Kontakt zu bringen. Es ist essenziell, den Stereotyp des isolierten Programmierers aufzubrechen, bevor es sich als festes Rollenbild manifestiert. Stattdessen müssen wir die Informatik als kreative und kollaborative Disziplin präsentieren. In Deutschland existieren bereits zahlreiche Initiativen, die durch gezielte Projekte eine beachtliche Anzahl an Schülerinnen erreichen und das Interesse an Technologie nachhaltig fördern. *Abbildung 7* zeigt eine Auswahl solcher Organisationen.

Auch auf universitärer Ebene erzielen gezielte Strategien messbare Erfolge. Die Universität Bamberg zeigt mit ihrer „CS30-Strategie“, wie man den undichten Karriereweg erfolgreich abdichtet. Durch Vernetzungsangebote wie „makeIT“ und Coaching-Programme wie „CoachNet“ steigerte die Universität den Anteil der Informatik-Studentinnen im ersten Semester von 7,2 % im Jahr 2005 auf beachtliche 37,1 % im Jahr 2017 [4]. Diese Beispiele belegen: Wenn wir die Bildungslandschaft aktiv umgestalten, erreichen wir auch die angestrebte Parität.

Sichtbarkeit und Mentoring: Wege ebnen

Um das Klischee des männlichen Programmierers zu brechen, müssen wir Expertinnen sichtbar machen – als Keynote-Speakerinnen auf Fachkonferenzen, in Entscheidungspositionen und in den Fachmedien. Viele bedeutende Tech-Konferenzen in Deutschland haben bereits diesen Bedarf erkannt: Etablierte Formate bieten heute Stipendien, Networking-Events oder Mentoring-Programme an. Plattformen – wie Women in Tech e. V., Informatica Feminale, femtec oder Global Digital Women – fördern nicht nur die Sichtbarkeit, sondern ermöglichen den notwendigen fachlichen Austausch.

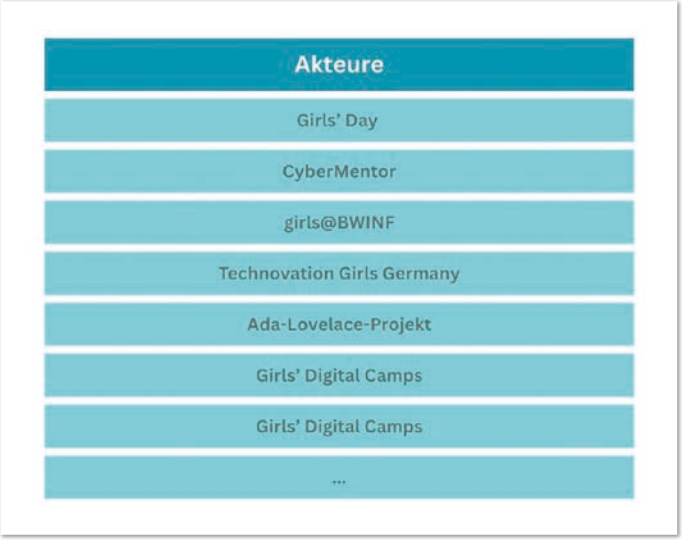


Abbildung 7: Akteure des Wandels: Organisationen für die Förderung weiblicher Nachwuchstalente. (© Copyright-Hinweis / Ildikó Tárkányi)

Flexibilität als kultureller Standard

Die Vereinbarkeit von Beruf und Privatleben bleibt der kritische Faktor, um Frauen langfristig in der IT zu halten. Unternehmen müssen Flexibilität als Wettbewerbsvorteil begreifen. Remote-Arbeit, Teilzeitmodelle – explizit auch für Führungskräfte – und eine konsequente Ergebnisorientierung bilden hier die Grundvoraussetzungen. Wir müssen die starre Präsenzkultur durch moderne Arbeitsmodelle ersetzen, die individuelle Lebensphasen berücksichtigen, ohne die Karriereentwicklung zu blockieren.

Governance und Transparenz: Diversität als messbarer Standard

Während einige Unternehmen Inklusion lediglich für Marketingzwecke nutzen, zwingt die zunehmende rechtliche Verpflichtung zur Transparenz nun zum Handeln. Die Corporate Sustainability Reporting Directive (CSRD) und die European Sustainability Reporting Standards (ESRS) verpflichten Unternehmen seit 2024 dazu, ihre Diversitätsstrategien und Kennzahlen im Lagebericht offenzulegen. Diversität und Frauenförderung rücken damit aus der Peripherie der HR-Abteilungen direkt in das strategische Zentrum der Unternehmensführung. Diese Transparenz trägt maßgeblich dazu bei, den bisher undichten Karriereweg langfristig abzudichten.

Fazit

Die Geschichte der Informatik zeigt deutlich: Frauen waren von Beginn an keine Randerscheinung, sondern die Architektinnen der digitalen Welt. Wenn wir heute über „Women in Tech“ sprechen, fordern wir kein neues Privileg.

Die Analyse der aktuellen Zahlen macht jedoch deutlich, dass wir uns nicht auf historischen Erfolgen ausruhen können. Ein undichter Karriereweg ist kein unveränderliches Schicksal, sondern das Ergebnis kultureller und struktureller Probleme. Um diese zu behe-

ben, brauchen wir ein grundlegendes „Update“ unserer gesamten Industrie.

Das bedeutet: Die Bewegung „Women in Tech“ darf kein kurzfristiger Trend bleiben. Wir müssen das Bildungssystem für neue Rollenbilder öffnen, Expertinnen die Bühne geben, die sie verdienen, und Flexibilität nicht als Ausnahme, sondern als Standard definieren.

Quellen

[1] o.V., KORREKTUR: Immer mehr Frauen entscheiden sich für ein MINT-Studium, in: *https://www.destatis.de, https://www.destatis.de/DE/Presse/Pressemitteilungen/2026/01/PD26_N006_213.html*, letzter Zugriff: 13. Februar 2026

[2] o.V., Frauenanteil in der technischen Forschung und Entwicklung binnen zehn Jahren von 11 % auf 18 % gestiegen, in: *https://www.destatis.de*, *https://www.destatis.de/DE/Presse/Pressemitteilungen/Zahl-der-Woche/2024/PD24_17_p002.html*, letzter Zugriff: 13. Februar 2026

[3] Jenny Tiesler, Wo sind die Frauen in der IT?, in: *https://www.get-in-it.de/*, *https://www.get-in-it.de/magazin/arbeitswelt/it-arbeitsmarkt/wo-sind-die-frauen-in-der-it*, letzter Zugriff: 13. Februar 2026

[4] Shanhong Liu, Women in tech, in: *https://www.statista.com*, *https://www.statista.com/study/74785/women-in-tech*, letzter Zugriff: 13. Februar 2026

[5] o.V., Diversity Benchmarking Studie 2024, in: *https://www.swissmem.ch/*, *https://www.swissmem.ch/de/themen/fachkraefte/diversity-benchmarking-studie-2024.html*, letzter Zugriff: 13. Februar 2026

[6] Merle Wiez, Leah Schrimpf und Pauline Meimberg, IT- und Digitalberufe: Frauen weiter unterrepräsentiert, in: *https://www.bitkom.org/*, *https://www.bitkom.org/Presse/Presseinformation/IT-Digitalberufe-Frauen-weiter-unterrepraesentiert*, letzter Zugriff: 13. Februar 2026

[7] Sven Blumberg, Melanie Krawina, Elina Mäkelä, und Henning Soller, Women in tech: The best bet to solve Europe’s talent shortage, in: *https://www.mckinsey.com*, *https://www.mckinsey.com/capabilities/tech-and-ai/our-insights/women-in-tech-the-best-bet-to-solve-europes-talent-shortage*, letzter Zugriff: 13. Februar 2026

[8] o.V., Deutschlands IT-Unternehmen wollen Frauenanteil erhöhen, in: *https://bitkom-akademie.de*, *https://bitkom-akademie.de/news/deutschlands-it-unternehmen-wollen-frauenanteil-erhoehen*, letzter Zugriff: 13. Februar 2026

Ildikó Tárkányi
QAware GmbH
ildiko.tarkanyi@qaware.de

Ildikó Tárkányi ist IT-Projektmanagerin und Agiler Lerncoach bei der QAware GmbH. Sie unterstützt die Entwicklung komplexer Softwaresysteme und begleitet Teams dabei, Zusammenarbeit wirksam zu gestalten. Sie setzt sich leidenschaftlich für eine IT-Kultur ein, in der Diversität und Inklusion keine Schlagworte bleiben, sondern zu operativen Erfolgsfaktoren werden.

Vom Wunscherfüller zum Kundenflüsterer – Das Geheimnis gesunder Grenzen

Julia Pedak, wlb-coaching.de

„Der Kunde ist König!", lautet so das Credo einer professionellen IT-Beratung? Nicht, wenn dadurch wiederholt Grenzen überschritten werden! Denn das schadet sowohl der mentalen Gesundheit der Beratenden als auch der Zufriedenheit der Kundinnen und Kunden. In diesem Artikel erfahrt ihr, welche psychologischen Muster bei fehlender Abgrenzung aktiv werden, wie ihr sie ab jetzt bei euch selbst durchschaut und schließlich Kundenbeziehungen auf Augenhöhe führen könnt.

Es ist 19:20 Uhr. Chris, seines Zeichens Junior IT-Consultant, will gerade den Rechner herunterfahren. Er freut sich auf den Feierabend mit seinen Kumpels im Gym und ist schon gedanklich dort. Da ertönt der bekannte Teams-Jingle. Der Name des Product Owners (PO) des Kunden blinkt hartnäckig. Was kann der jetzt noch wollen? Chris nimmt den Call an – und wird mit mehreren kurzfristigen Anforderungen konfrontiert. Es gehe um „kritische Punkte", die natürlich noch vor dem Release morgen umgesetzt werden müssen. Chris weiß, was das heißt: Tschüss Feierabend, hallo Überstunden …

Wer kennt solche Situationen nicht? Insbesondere im IT-Consulting wird kundenfreundliches und serviceorientiertes Verhalten von vielen Mitarbeitenden erwartet. Die Ansprechpartner auf Kundenseite vertrösten oder ihnen gar einen Wunsch abschlagen? Das ist für viele Beratende unvorstellbar. Das eigene Privatleben und die Gesundheit hintenanstellen: Das geht dagegen immer wieder erstaunlich leicht. Dabei schlagen häufig zwei Herzen in einer Brust: Auf der einen Seite haben wir den klassischen Wunscherfüller, der die Extrameile geht, zur Not auf dem Zahnfleisch. Auf der anderen Seite ist da diese Stimme, die Einwände erhebt und auf den Feierabend pocht. Doch wem sollte und darf man in solch einer Zwickmühle nachgeben?

Das Work-Life-Balance-Dilemma

Natürlich ist den meisten von uns klar, dass Arbeiten ohne Pause auf Dauer der Gesundheit schadet – Stichwort Burnout. Wir brauchen Pausen, um uns zu erholen, einen klaren Kopf zu bekommen und frische Ideen zu finden. Es ist auch kein Geheimnis, dass es neben der Arbeit noch ein Privatleben gibt, das je nach Lebenssituation Aufmerksamkeit und Energie fordert und im besten Fall auch Spaß verspricht. Warum fällt es vielen Menschen dennoch so schwer, Prioritäten zu setzen und sich abzugrenzen? Schauen wir uns das Beispiel von Chris an:

Chris hat zum einen klassische Ängste, die viele Angestellte kennen: Wenn er nicht performt, könnte er Ärger bekommen. Vom Kunden oder von seinen Vorgesetzten. Im schlimmsten Fall könnte er sogar seinen Job verlieren, fürchtet er. Bei der aktuellen wirtschaftlichen Lage kein angenehmer Gedanke. Neben solchen extrinsischen Bedrohungen existieren auch intrinsische: Chris hat Sorge, dass er nicht ausreichend Anerkennung bekommt und als Versager dasteht.

Gegenüber all dem, was Chris vermeiden möchte, stehen aber auch große Ziele und Hoffnungen: Er will noch Karriere machen, befördert werden und eine Gehaltserhöhung bekommen. Außerdem treibt ihn innerlich der Wunsch nach Perfektion an. Er behält dabei auch gerne die Kontrolle. Also macht er alles zu 150 Prozent und am liebsten allein, was die Arbeitslast entsprechend erhöht (siehe Abbildung 1).

Abbildung 1: Gründe für fehlende Abgrenzung (© Julia Pedak)

Feuer und Flamme oder schon Burnout?

Das Tückische an Beispielen wie dem von Chris ist der Faktor des Unbewussten. Wir haben meist nicht klar vor Augen, was uns innerlich so sehr antreibt oder ängstigt, dass wir darüber Dinge wie Sport, Freunde und Familie vernachlässigen. Es fühlt sich auch gut an, wenn wir helfen können und gebraucht werden. Das Projekt macht Spaß und fordert uns heraus – bis es kippt und zu einer Überforderung wird, weil die Energiereserven nicht mehr ausreichend aufgetankt werden. Dass dieser schleichende Prozess voranschreitet, wird bei den meisten Menschen an ganz konkreten Symptomen sichtbar:

- Die Gedanken kreisen um die Arbeit
- Das Ein- und Durchschlafen fällt schwer
- Kopf und Nacken schmerzen
- Der Appetit verändert sich
- Freizeit und soziale Kontakte leiden
- Ein Gefühl von Überforderung stellt sich ein
- Es scheint keine Handlungsalternativen zu geben

In meiner Beratungspraxis sehe ich viele Menschen mit diesen Beschwerden, die lernen wollen, wie sie ihre Work-Life-Balance wieder herstellen. Auch im Trainingskontext berichten Teams vermehrt von Überlastung, die durch ständige Erreichbarkeit, kurzfristige Umpriorisierungen und ungeplante Ad-hoc-Anfragen zusätzlich verstärkt wird. Kurzfristig erlernbare Lösungen wie besseres Zeitmanagement und Entspannungsmethoden helfen den Betroffenen akut. Um das Grundproblem zu lösen, bedarf es aber ehrlicher (und daher manchmal ungemütlicher) Selbstreflexion.

Die Geburtsstunde des Ja-Sagers

Was Menschen aktuell dazu verleitet, Ja zu sagen und über ihre Grenzen zu gehen, haben wir schon gesehen. Warum entwickeln manche Personen aber solche Tendenzen – und andere nicht? Hier hilft ein Blick in die Biografie und ins aktuelle Umfeld.

Von Bezugspersonen lernen wir von Kindesbeinen an, welches soziale Verhalten von uns erwartet wird. Je nach Erziehungsstil wird Gehorsam mehr oder weniger belohnt beziehungsweise Ungehorsam bestraft. Das kann auch sehr subtil durch Nudging erfolgen und lässt sich tatsächlich gar nicht vollständig vermeiden. Denn als Kind

machen wir natürlich das, was uns die Aufmerksamkeit und Zuwendung der Erziehenden sichert. Das ist pure Überlebensstrategie. Kritisch wird es spätestens, wenn wir uns als Erwachsene noch immer nach Zuneigung sehnen und unsere eigenen Bedürfnisse leugnen, um anderen zu gefallen. Dieses People Pleasing hat zu einem gewissen Grad Vorteile, wenn es zur Beliebtheit beiträgt. Es kann aber auch dazu führen, dass andere Personen immer mehr fordern, weil sie bemerken, dass hier viel zu holen ist. So auch bei Chris:

Dass der PO Chris anruft, ist nicht das erste Mal. Er weiß, wen er abends noch erreichen und mit Sonderwünschen beauftragen kann. Leider variieren diese Wünsche zu manchen Zeiten recht häufig. Das frustriert Chris. Wo er anfangs noch voll Enthusiasmus das Unmögliche möglich gemacht hat, fühlt er sich nun nur noch ausgelaugt. Es ist ein Fass ohne Boden, das er aufgemacht hat. Wenn er nur wüsste, wie er aus der Nummer wieder rauskommt!

Falsche Vorbilder

In früher Jugend konditioniertes Ja-Sagen ist nur ein Teil der Faktoren, die Abgrenzung erschweren. Auch im aktuellen Umfeld finden sich bei den meisten Fällen Gegebenheiten, die das Problem aufrechterhalten. Oft in den eigenen Team-Reihen oder mit Blick auf die Vorgesetzten. Denn diese leben vor, wie es (nicht) geht. Ist der Team Lead auch am Wochenende und spät abends erreichbar, scheint dieses Verhalten auch bei den Teammitgliedern erwünscht und wird unbewusst abgeschaut.

Zusätzlich kann noch ein falsch verstandenes Wettbewerbsdenken hinzukommen. Wenn Überstunden und Extrameilen eine Art Status-Symbol sind und unter Kolleginnen und Kollegen bewundert werden, facht dies das Feuer noch mehr an.

Wege aus der People-Pleasing-Falle

Ob frühere oder heutige Konditionierung: Sich über die Ursprünge und die dahinterliegenden Bedürfnisse klar zu werden, kann ein echter Augenöffner sein. Fangen wir an, uns zu beobachten und zu reflektieren, erkennen wir:

- In welchen Situationen wir uns schlecht abgrenzen können.
- Bei welchen Personen wir häufiger Ja sagen und Nein denken.
- Welche Sorgen oder Hoffnungen dahinterstecken.

Hilfreich ist die Frage, was wir uns davon versprechen, wenn wir anderen ständig entgegenkommen und über unsere Grenzen gehen. Ein Gefühl der Zugehörigkeit? Lob und Anerkennung? Meist sind es solche Sehnsüchte nach Bestätigung. Als Kinder können wir uns diese Bestätigung nicht selbst geben und sind auf andere (Autoritäts-) Personen angewiesen. Als Erwachsene wäre es allerdings fatal, sich ständig von der Anerkennung anderer abhängig zu machen und sich dafür zu verbiegen und zurückzustellen. Also lassen wir das People Pleasing doch einfach sein!

Alte Muster umlernen

Ganz so einfach ist es leider nicht. Wer schon einmal versucht hat, ein unerwünschtes Verhalten zu ändern, kennt das sicher. Bevor wir etwas loslassen können, muss erst ein neues attraktiveres Ziel am Horizont erscheinen. Typische Beispiele: die Trauer um einen verlorenen Job schwindet mit der neuen Stelle, Doomscrolling ist nicht mehr relevant, wenn ein neuer Freundeskreis lockt, der Rauchstopp gelingt leichter, wenn die Motivation wirklich klar vor Augen steht.

Beim Setzen von Grenzen funktioniert es genauso: Soll das alte Verhalten (Grenzen überschreiten, um Anerkennung zu erhalten) abgelegt werden, muss erst ein neues Verhalten erlernt werden. Wenn der Wunsch nach Anerkennung hinter dem Ja-Sagen steckt, muss die Anerkennung also zunächst aus einer anderen Quelle sprudeln *(siehe Abbildung 2)*.

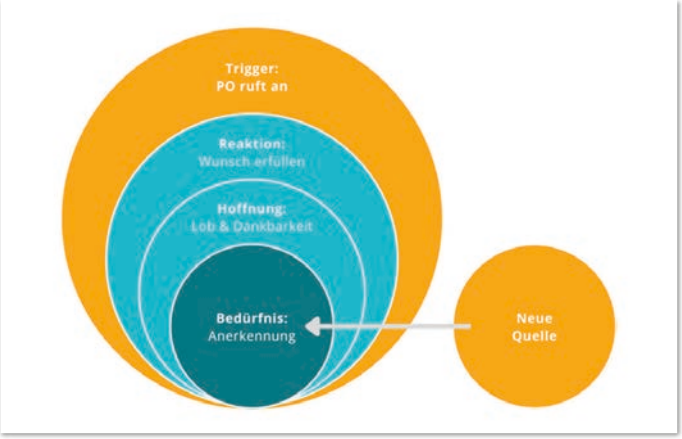


Abbildung 2: Versteckte Bedürfnisse erkennen (© Julia Pedak)

Drei Hacks aus der Coaching-Kiste

Wo bekommt jemand wie Chris Anerkennung her, wenn er sie sich nicht mehr durch andere Personen holen soll? Und wie verhindert er, dass er in alte Muster zurückfällt? Dafür empfehle ich euch drei konkrete Tools, die jeweils für sich und in Kombination noch stärker wirken:

1. Proudness-Journaling
2. Erlaubnis-Training
3. Grenzen-Tracking

1. Proudness-Journaling: Worauf bin ich stolz?

Im Resilienz-Training ist das Dankbarkeitstagebuch eine gängige Methode, um Optimismus und Wertschätzung zu trainieren. In Anlehnung an diese Methode empfehle ich ein „Darauf-bin-ich-stolz-Tagebuch". Jeden Abend notiert ihr darin mindestens drei Dinge, auf die ihr stolz seid. Das können größere Erfolge und erreichte Ziele sein, aber auch alltägliche Dinge und Verhaltensweisen. Wichtig ist, dass ihr sie selbst „erzeugt" habt. Das ist auch der entscheidende Unterschied zu Dankbarkeitstagebüchern, in denen schnell Dankbarkeit für schönes Wetter steht, was außerhalb unseres Einflusses liegt.

In der Stolz-Variante geht es dagegen darum, sich selbst vor Augen zu führen, was wir jeden Tag leisten und uns selbst dafür Wertschätzung entgegenzubringen. Wer dies zwei Wochen lang täglich durchführt, wird bald einen Unterschied merken und sich selbstbewusster und zufriedener fühlen – unabhängig vom Lob anderer. Und: An schlechten Tagen hilft das Blättern in den vorherigen Einträgen, um uns wieder kraftvoll und zuversichtlich zu fühlen.

2. Erlaubnis-Training fürs Grenzen-Setzen

Gerade in hektischen Phasen geraten wir schnell in Stress und verfallen dann automatisch in alte Muster. Das Beispiel von Chris zeigt

es gut: Kurz vor Feierabend ist er nicht mehr so konzentriert, dass er die Situation mit kühlem Kopf analysieren und sich bewusst abgrenzen kann. Im Stress-Modus sagt er ganz automatisch Dinge zu, die er eigentlich auf einen späteren Zeitpunkt verschieben oder sogar ganz ablehnen möchte. Damit das nicht passiert, hilft das Einüben von Erlaubersätzen, auch als Affirmationen bekannt, die täglich geübt werden sollten. Damit sind keine „Ich schaffe alles, was ich will"-Phrasen gemeint. Es geht vielmehr darum, den eigenen Handlungsspielraum zu erweitern, indem wir alte, ungünstige Glaubenssätze durch hilfreichere neue Gedanken ersetzen.

Im Beispiel von Chris kann das so aussehen: Bisher war er der Überzeugung, dass „Wer A sagt, muss auch B sagen" gilt. Somit muss er in seiner eigenen Logik alle Kundenwünsche erfüllen, wenn er einmal ein Projekt begonnen hat. Hilfreicher wäre für ihn folgender Satz: „Ich darf mir Zeit einräumen." Damit verschafft er sich eine Pause, in der er prüfen kann, ob er dafür realistisch Kapazitäten hat oder ob eine Neupriorisierung erforderlich wäre. Andere hilfreiche Sätze können sein:

- „Ich darf auch für mich sorgen."
- „Ich setze professionelle Grenzen."
- „Ein Nein zu anderen ist ein Ja zu mir."

Das sind Serviervorschläge. Ich empfehle jeder Person, ihre eigenen Sätze zu entwickeln und durchaus auch täglich mit leichten Variationen zu spielen. Auch ob diese Sätze auf Notizzetteln am Bildschirm kleben oder morgens vor dem Spiegel gedacht, gesagt oder gesungen werden, bleibt eine individuelle Entscheidung. Der Effekt stellt sich nach zwei bis drei Monaten ein. Es braucht also etwas Geduld.

3. Tracking: Fortschritte sehen und schätzen

Wer analytisch unterwegs ist, wird diesen Tipp lieben: Man kann den eigenen Fortschritt sehr schön tracken, indem man Situationen sammelt, in denen die Abgrenzung gut oder zumindest besser gelungen ist als früher. Auch eine Skala, auf der man die eigene Fähigkeit zum Grenzen setzen einschätzt, kann das schön veranschaulichen. Je nach Vorliebe lässt sich das auf Papier festhalten oder auch in Apps oder selbstprogrammierten Dashboards.

Gleichzeitig lenkt das Tracken der Erfolgserlebnisse den Fokus auf die Situationen, die funktionieren. Das ist für viele Menschen eine neue Perspektive und kann die Motivation deutlich steigern.

Challenge your Customer

Wer bis hier gelesen hat, fragt sich vielleicht schon eine Weile, ob man Kunden ein „Nein" überhaupt zumuten darf. Tatsächlich ist die Antwort „Ja!" Gerade im Consulting werden Beratende wegen ihrer Expertise gebucht. Der Kunde zahlt also für die ehrliche Einschätzung der Lage. Wenn Extra-Wünsche außerhalb des vereinbarten Scopes liegen oder nur über einen Change Request realistisch umsetzbar wären, gehört es zur professionellen Rolle, dies offen zu spiegeln. Das ist kein Zeichen von Schwäche, sondern von Professionalität. Und es erfordert Mut, keine Frage. Denn die Sorge, dass der Kunde bei einem „Nein" verärgert ist, schwingt bei vielen mit, die vor dieser Herausforderung stehen.

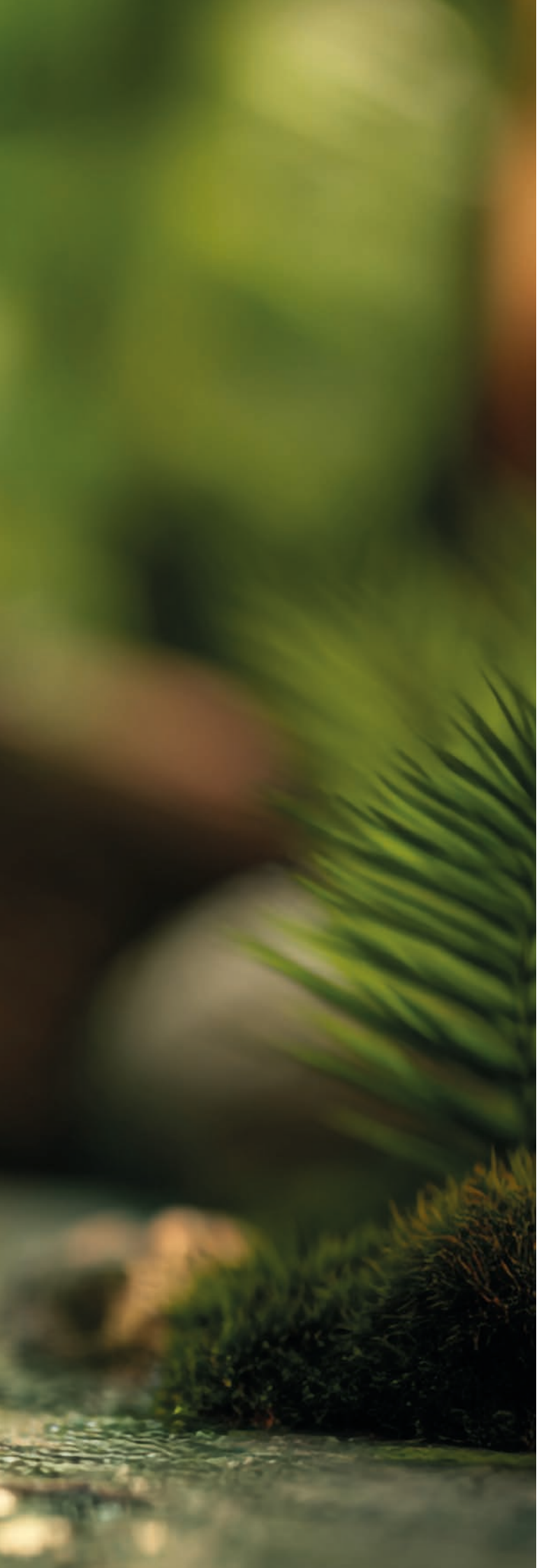
Paradoxerweise tritt viel häufiger das Gegenteil ein: Kunden sind dankbar für die ehrliche Antwort. Denn wiederkehrende Extrawünsche oder Meinungswechsel sind ein Zeichen von Unsicherheit. Ein klares, wertschätzend formuliertes „Nein" gibt dagegen Sicherheit und schafft eine Partner-Beziehung auf Augenhöhe, die nachhaltig wachsen kann.

Julia Pedak
wlb-coaching.de
beratung@wlb-coaching.de

Julia Pedak arbeitet als Systemische und Psychologische Beraterin in Bonn sowie online. Sie unterstützt Einzelpersonen rund um die Themen Work-Life-Balance, Karriereentwicklung und Führung sowie Teams beim Umgang mit Veränderungen und beim Klären von Konflikten. Als Instruktorin bildet sie zudem MHFA-Ersthelfende aus. Zuvor hat Julia Pedak in der Unternehmenskommunikation und im Marketing in einer IT-Beratung sowie bei einem Software-Hersteller gearbeitet. Sie verfügt insgesamt über mehr als acht Jahre Erfahrung in der IT-Branche.

Die Avocado trägt keine Schuld

Berend Semke, Bredex GmbH



Bei der Diskussion, weshalb Kollegen ein bestimmtes Verhalten zeigen oder unterlassen, wird meist zwischen extrinsischer und intrinsischer Motivation unterschieden. Diese vereinfachte Unterscheidung geht wissenschaftlich auf die Selbstbestimmungstheorie (SDT) nach Deci und Ryan zurück, die ein analoges Spektrum von externer Regulation, über internalisierter Regulation, zu intrinsischer Motivation beschreibt.

In diesem Artikel möchte ich erklären, weshalb ich das Bild der extrinsischen Motivation für einen maskierten Bug halte. Stattdessen ist es hilfreich, den Grad der (Un-) Reflektiertheit zu diskutieren und davon auszugehen, dass jeder Antrieb in den Lebewesen selbst liegt.

Lebewesen interagieren mit ihrer Umwelt. Eine Avocado-Pflanze trägt keine Schuld, wenn sie übergossen wird und eingeht. Warum sollten wir dann Menschen Schuld zuschreiben, wenn sie in ihrer Umgebung nicht performen?

> *„Sie können andere nicht motivieren. Als Führungskraft ist es vielmehr Ihre Aufgabe, Demotivation von Ihren Mitarbeitern fernzuhalten!"*

Dieser Merksatz, gepaart mit einem unwohlen Gefühl in meiner Magengrube, war mein Mitbringsel aus dem Seminar „Motivation", als ich mich vor knapp 20 Jahren in meiner ersten Führungskräfteausbildung befand. Vorausgegangen war die Vermittlung eines mir sympathischen Menschenbildes: Jeder Mitarbeitende nähme höchstmotiviert eine neue Tätigkeit auf. Die Interaktion mit der Organisation und den individuellen Gegebenheiten vernichte anschließend einen Teil der vorhandenen Motivation. Führungskräfte hätten demnach die Aufgabe, Störfaktoren zu identifizieren und zu eliminieren, damit Mitarbeitende in ihrer Motivation möglichst wenig gestört würden *(siehe Abbildung 1)*.

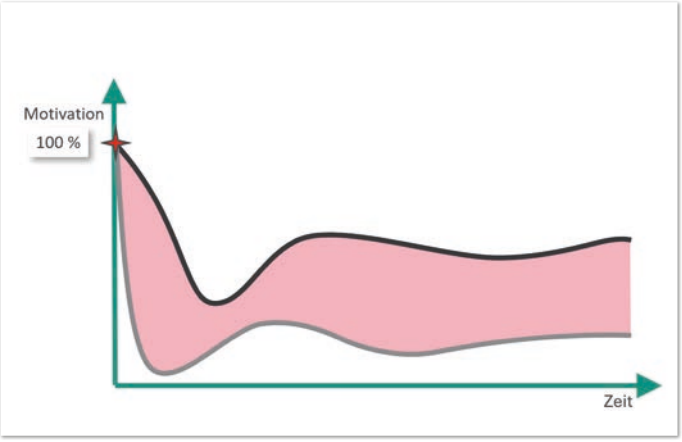


Abbildung 1: Führungskräfte, die Demotivation identifizieren und eliminieren wollen, wirken in der roten Fläche. (© Berend Semke)

Als ich nach diesem Seminartag zu Hause ankam, sah ich rosa Elefanten *[1]* und konnte einige Nächte schlecht schlafen. Seit meiner Zeit als Tischtennistrainer habe ich mir angewöhnt, möglichst im Lösungsraum zu kommunizieren. Der Datenzugriff im menschlichen Gehirn ist assoziativ. Wenn wir Dinge klar benennen und Zielzustände klar beschreiben, regen wir automatisch Denkprozesse an, die Wege zur Lösung aufzeigen.

Es hat mich gestresst, mit einem Merksatz umgehen zu müssen, der gleich zwei negative Formulierungen beinhaltete: Das Substantiv *„Demotivation"* beschreibt die Abwesenheit von Motivation. Das Verb *„fernhalten"* beschreibt die Tätigkeit, dafür zu sorgen, dass etwas abwesend ist.

> *Ich soll dafür sorgen, dass die Abwesenheit von Motivation abwesend ist?*

Die doppelte Verneinung lässt sich nach den Regeln der Boolescher Algebra auflösen: Wenn ich dafür sorgen soll, dass die Abwesenheit von Motivation vermieden wird, dann kann ich mich auch gleich um die Anwesenheit von Motivation bemühen.

Dieses Bemühen wird von mächtigen Menschen gerne als „andere motivieren" interpretiert („ich sende dir Motivation"). Eine Interpretation, die für Mächtige hilfreich ist, da sie vor Ohnmachtsgefühlen schützt. Leider funktioniert sie nicht. Dies hat meinen Ausbilder zur Erkenntnis gebracht, dass andere Menschen nicht motiviert werden könnten.

Wenn ich davon ausgehe, dass Menschen grundsätzlich motiviert sind, ich sie aber nicht direkt motivieren kann und mich gleichzeitig um die Motivation der Menschen bemühen möchte, können Entwurfsmuster der Softwareentwicklung helfen, um Wechselwirkungen zu modellieren und besser zu verstehen. Scheinbar sind Mitarbeitende und Führungskräfte lose gekoppelt und stehen dennoch in Resonanz zueinander. Durch die Manipulation von Umwelteinflüssen beeinflussen sie sich indirekt. Dank Polymorphismus kann eine Führungskraft zur selben Zeit Umwelteinfluss und Mensch sein.

Basierend auf dieser Überlegung entwarf ich in erster Iteration ein E/R-Modell *(siehe Abbildung 2)*:

- Menschen interagieren mit Umwelteinflüssen.
- Mitarbeitende sind Menschen.
- Führungskräfte sind Menschen.
- Menschen werden von anderen Menschen als Umwelteinfluss wahrgenommen.
- Führungskräfte können Umwelteinflüsse manipulieren und gestalten.
- Menschen sind motiviert.

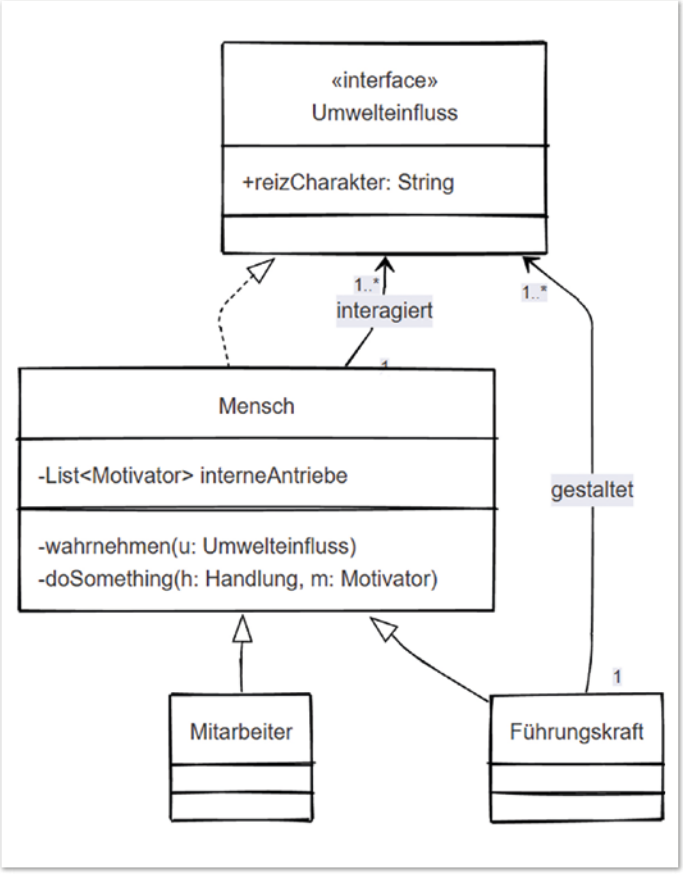


Abbildung 2: Führungskräfte sind Umwelteinflüsse für Mitarbeitende (© Berend Semke)

Daraus habe ich einen Merksatz abgeleitet, der mich seitdem zuverlässig durch meine Arbeit als Führungskraft begleitet:

> ***Motivation ist intrinsisch.***

Die Avocado

In dem Zimmer meines 10-jährigen Sohns steht eine Pflanze, die es eigentlich gar nicht geben dürfte. Auf 50 cm Höhe kämpft sich ein

kahler Haupttrieb empor, begleitet von zwei mickrigen Nebentrieben. Die kahlen Narben des Haupttriebes erinnern an Blätter vergangener Zeiten. An seiner Spitze drängen sich sieben unverhältnismäßig große, teilweise angetrocknete Blätter in zwei Ebenen. Gemeinsam mit einer Hand voll Sprösslingen in ihrer Mitte belegen die Blätter trotzig, dass noch Leben in dem kleinen Avocadobaum steckt.

Den beiden Nebentrieben ist ein Schatten-Trauma und unbändiger Lebenswille anzusehen: Beide Spitzen sind vor Monaten abgestorben. Der trockene Tod ist den Wurzeln 10 cm entgegengewandert, bis die Pflanze endlich in ein Fenster gestellt wurde. An dem Tag, als Sonnenlicht auf die Seitentriebe fiel, hat die Pflanze ihren Verfall gestoppt und ihre Kraft in das verbliebene, gesunde Holz gesteckt. Sie hat rechtwinklig neu ausgetrieben und ist in einem Viertelkreis der Schwerkraft entgegengewachsen, um jeweils vier Blätter zu produzieren, die von dem instabilen Provisorium geradeso gehalten werden können.

15 Blätter hat die Pflanze heute. In den vergangen zwei Monaten, in denen sie endlich direkter Sonneneinstrahlung ausgesetzt wurde, konnte sie einen unglaublichen Blätterzuwachs von 500 % realisieren *(siehe Abbildung 3)*.

Abbildung 3: Die Avocado „Addi", nachdem sie vor Demotivation geschützt wurde? (© Berend Semke)

Die Geschichte – manche nennen es Tortour – der kleinen Pflanze ist voller unfreiwilliger Anekdoten der Reihe „Was man beim Ziehen einer Avocado vermeiden sollte": Sie war dem Risiko von Staunässe ausgesetzt, ist mehrfach fast verdurstet und zu Beginn ihres Lebens fast verschimmelt. Sie hatte mit Parasiten zu kämpfen und stand die meiste Zeit ihres Lebens an einem Platz, der ihre Grundbedürfnisse ignorierte.

Heute – nachdem ich sie lange Zeit beobachtet habe, um ihr Leben bangte, ihr einen Namen gegeben habe („Addi" – ich weiß, nicht sehr originell) und deshalb von meiner Frau manchmal liebevoll-schräg angeschaut werde – erkenne ich an der Stellung ihrer Blätter im Vorbeigehen, ob sie durstig ist und versorge sie dann mit Wasser.
Ich unterstelle Addi, dass sie „überleben" positiv bewertet. Sie ist motiviert. Ihr Verhalten ist eine „aktivierende Ausrichtung des momentanen Lebensvollzugs auf einen positiv bewerteten Zielzustand" *[2]*. Das glaube ich nicht.

Bin ich damit zu einem Avocado-Motivator geworden?
Habe ich Addi vor Demotivation geschützt?

Vielmehr denke ich, dass Addi ihre Motivation immer in sich trug. Wie jedes Lebewesen interagiert sie mit der Umwelt entsprechend dem Reiz-Reaktions-Schema. Avocado-Pflanzen und Menschen unterscheiden sich hinsichtlich der Implementierung des Schemas, die Interaktion als solche bleibt ein Zeichen von Leben *(siehe Abbildung 4)*.

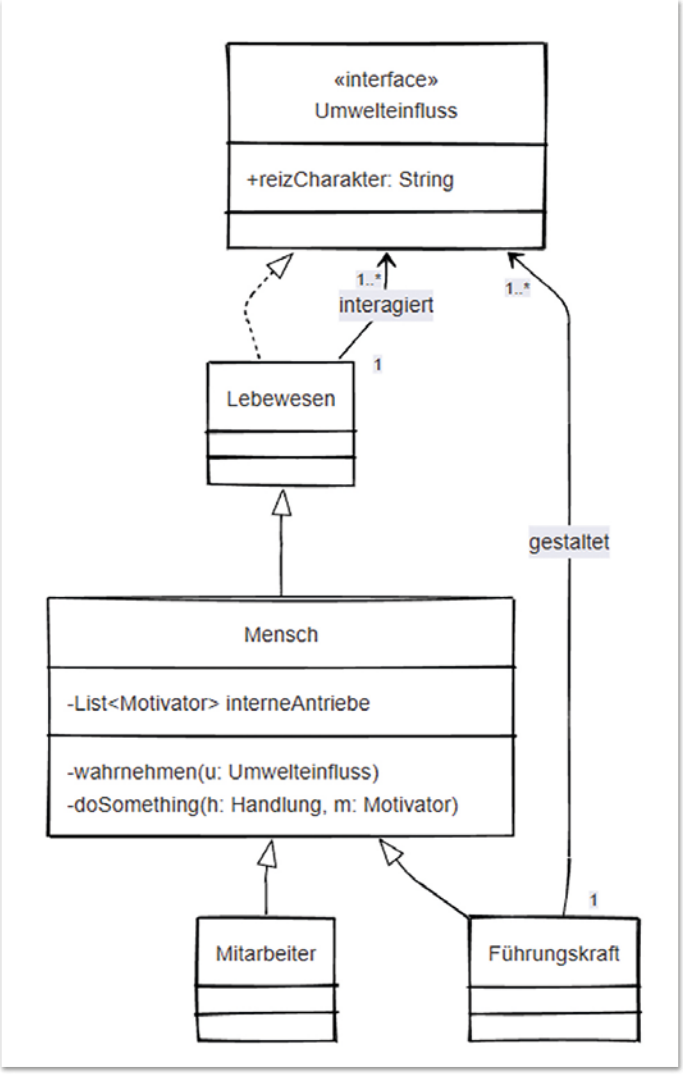


Abbildung 4: Menschen sind Lebewesen (© Berend Semke)

Die Selbstbestimmungstheorie

Spätestens seit der Industrialisierung hat sich die Wissenschaft zunehmend intensiv mit der Frage auseinandergesetzt, wie genau das menschliche Reiz-Reaktions-Schema aufgebaut ist: Wie können mächtige Menschen weniger mächtige Menschen dazu bringen, be-

stimmte Handlung durchzuführen (z. B. damit Waren am Fließband zuverlässig hergestellt werden)?

Im Bereich der Motivationsforschung genießt die Selbstbestimmungstheorie (SDT), in den 1960er und 70er Jahren von Deci und Ryan entwickelt, breite Anerkennung *[3]*. Sie stellt die These auf, dass jeder Mensch drei Grundbedürfnisse in sich trägt, die sein Handeln antreiben *(siehe Abbildung 5)*:

- **Das Bedürfnis nach Autonomie**
  Menschen bringen viel Energie auf, um Autonomie zu erleben. Kinder wollen sich zunehmend aus der Abhängigkeit zu ihren Eltern lösen, Kollegen für eine bekannte Fragestellung einen eigenen Lösungsweg gehen. Diese Energie ist die Triebfeder von Innovation. Nur wenn ausgetretene Pfade verlassen werden, können wir Wege finden, die der bekannten Lösung überlegen sind.
- **Das Bedürfnis nach Kompetenz**
  Sich einer herausfordernden Aufgabe zu stellen, setzt Endorphine frei, vor allem bei ihrer Bewältigung. Wiederholung macht zunächst Spaß, wir erlangen Übung. Irgendwann sinkt der Grenznutzen und uns wird langweilig. Schließlich wollen wir uns einer neuen Aufgabe stellen (und erneut Kompetenz erleben). Diese Energie ist Triebfeder von persönlicher Entwicklung.
- **Das Bedürfnis nach sozialer Eingebundenheit**
  Menschen sind soziale Wesen. Kinder lernen von ihren Eltern. Junior-Entwickler von Mentorinnen. Es gibt Herausforderungen im Leben, die wir nur gemeinsam bewältigen können. Diese Energie sichert soziale Ordnung, hilft bei der Synchronisation der verschiedenen Systeme und bei dem gemeinschaftlichen Aufbau von Wissen und Kompetenzen.

Aufgrund der individuellen Genetik, der persönlichen Prägung, der individuellen Lebenssituation und Lebenserfahrung sind diese Bedürfnisse bei jedem Menschen unterschiedlich priorisiert.

- Das Überleben von Neugeborenen ist von deren sozialer Eingebundenheit abhängig. Autonomie kann erst nach einigen Monaten/Jahren ausgelebt, artikuliert und realisiert werden.
- Fachexpertinnen mit Jahrzehnten professioneller Erfahrung haben oft ein stärkeres Bedürfnis nach Autonomie. Ein Bedürfnis nach sozialer Eingebundenheit kann in einer Mentorenrolle befriedigt werden.
- 60-jährige Techniker, die immer neugierig geblieben sind, verspüren ein ständiges Bedürfnis nach Kompetenz. Sie befriedigen dies, indem sie ihr Wissen kontinuierlich mit den neuesten Entwicklungen abgleichen.

Die Grundannahmen der Selbstbestimmungstheorie lassen sich problemlos modellieren, ohne eine Differenzierung zwischen „externer Regulation", „internalisierter Regulation" und „intrinsischer Motivation" vorzunehmen.

Das System „Mensch" ist in sich geschlossen, seine Attribute sind sauber gekapselt. Es besteht eine lose Kopplung zu anderen Systemen, insbesondere zu anderen Menschen. Zwischenmenschliche Beziehungen ergeben sich, weil jeder Mensch immer auch Umwelteinfluss für andere Menschen ist. Aufgrund des Bedürfnisses nach sozialer Eingebundenheit, das einen evolutionären Vorteil darstellt *[4]*, interagieren Menschen tendenziell eher mit anderen Menschen als mit leblosen Umwelteinflüssen.

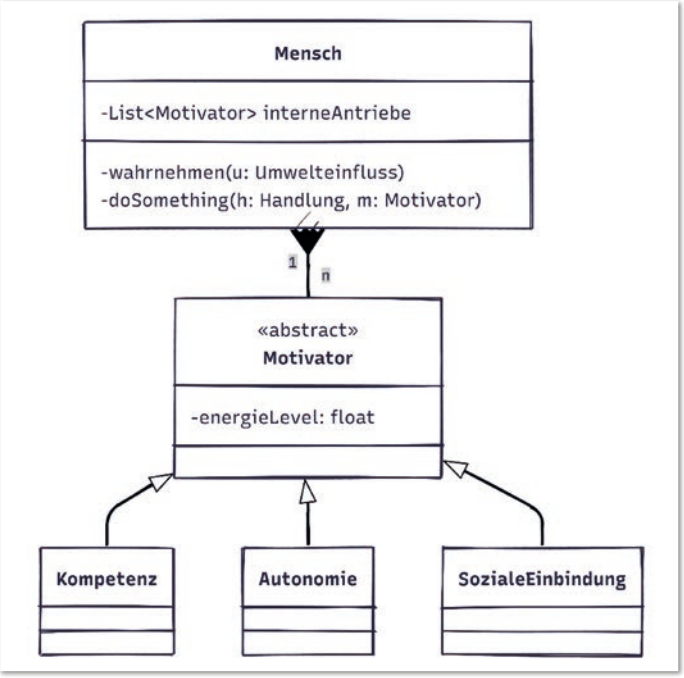


Abbildung 5: Menschliche Motivatoren lassen sich nach SDT in 3 Kategorien einordnen. (© Berend Semke)

Rutger Bregman beschreibt in seinem großartigen Buch „Im Grunde gut" *[5]*, wie Deci in den 1960er Jahren zu extrinsischer Motivation forschte. Es war anerkannte Lehrmeinung, dass menschliches Handeln gleichzeitig von externen als auch von internen Faktoren getrieben sei, dass zwischen Handlung und Motivation eine n-zu-m-Beziehung bestand.

Deci wollte den Einflussfaktor von extrinsischer Motivation isoliert messen. Dazu suchte er Menschen, die Handlungen ohne sichtbare Belohnung durchführen. Diese Menschen handelten nach vorherrschender Lehrmeinung ausschließlich intrinsisch. Würde diesen Menschen eine zusätzliche Belohnung angeboten, müsste sich eine Motivationssteigerung messen lassen: Der isolierte Einfluss des externen Motivators.

Überraschend hat er beobachtet, dass Studenten, die sich freiwillig zum Lösen von Kreuzworträtseln trafen, diese Aktivität früher beendeten und mit geringerer Güte durchführten, sobald man ihnen für die Tätigkeit ein paar Dollar zahlte.

Daraus folgerte Deci: „Wenn Geld als Belohnung für eine bestimmte Aktivität winkt, verliert das Subjekt seine eigentliche Motivation." *[6]* Basierend auf dieser Annahme entwarf er ein Konstrukt, das strikt zwischen extrinsischer und intrinsischer Motivation trennte, verschiedene komplexe Zwischenformen enthält und bei dem Versuch scheitert, mir plausibel zu erklären, wie genau dieser externe Eingriff in das Subsystem „Mensch" eigentlich funktioniert.

Priming von Motivatoren

Der Testmanager in mir erkennt hier eine klassische Verwirrung zwischen Verifikation und Validierung. Während bei der Validierung überprüft wird, ob sich ein System entsprechend seinen funktionalen Anforderungen verhält („Does the right thing"), testen wir bei der Verifikation, ob die Qualitätsanforderungen an den internen Aufbau des Systems eingehalten wurden („Does things right").

In der Kreuzworträtsel-Studie geht Deci zunächst davon aus, dass eine

Handlung von verschiedenen Motivatoren gleichzeitig angetrieben werden kann. Dass unterschiedliche Motivatoren kompatibel sind, lässt sich empirisch belegen: Manche Arbeitnehmer berichten, sie würden einer Tätigkeit nur wegen des Geldes nachkommen, gleichzeitig wird unsere Gesellschaft von ehrenamtlich tätigen Menschen getragen, die zum Beispiel Java User Groups in ihrer Freizeit organisieren.

Deci konfrontiert vermeintlich intrinsisch motivierte Menschen mit einem weiteren (externen) Motivator und geht davon aus, dass beide Motivatoren gleichzeitig Einfluss auf die Handlung hätten. Stattdessen beobachtet er, dass der intrinsische Antrieb „verloren geht" und ausschließlich die externe Regulation wirkt.

Diese Beschreibung ist ein typisches Fehlerbild bei einem fehlerhaft implementierten Datenmodell. Die n-zu-m-Beziehung zwischen Antrieb und Motivation konnte nicht verifiziert werden, stattdessen scheint hier eine 1-zu-n- oder eine 1-zu-1-Beziehung vorzuliegen.

Für die folgende Betrachtung gehe ich exemplarisch von einer 1-zu-1-Beziehung aus: Die Durchführung einer spezifischen Handlung benötigt genau einen spezifischen Motivator. Sobald dieser Motivator ausgewählt wurde, ist der Transmitter (das Register) belegt. Unabhängig davon, ob es andere kompatible Instanzen mit höherem Energieniveau gegeben hätte.

Der Datenzugriff im menschlichen Gehirn ist assoziativ *[7]*. Wenn die Wahrnehmung eines Menschen zuletzt durch die Verarbeitung von „Geld" geflutet wurde, liegen hauptsächlich Assoziationen zu „Geld" im Cache. Wird nun eine Handlung aufgenommen und eine kompatible Instanz von Motivator ist unmittelbar verfügbar, wird genau diese verwendet, um den Transmitter zu belegen *[8]*. Psychologen sprechen in diesem Zusammenhang von „Priming".

Der maskierte Bug

Ein Bug ist als eine Abweichung zwischen spezifiziertem und beobachtetem Verhalten definiert *[9]*. Meine These: Deci hat in der Studie einen Bug beobachtet und ihn auf der Validierungsebene interpretiert, obwohl der Testfall besser geeignet war, die Datenbeziehung zwischen Motivation und Handlung zu verifizieren.

In späteren Studien wurde der Versuchsaufbau umgedreht. Menschen, die scheinbar keine intrinsische Motivation zu einem spezifischen Verhalten zeigten, wurden mit Belohnung oder Bestrafung konfrontiert und so zu dem gewünschten Verhalten getrieben. Diese Beobachtung wurde als Nachweis externer Regulation verwendet, was ich für einen maskierten Bug halte.

Bei einem maskierten Bug kann ein Systemverhalten validiert werden, obwohl das System fehlerhaft ist, weil mindestens ein weiterer Systemfehler das Fehlverhalten (teilweise) überdeckt.

Das Ergebnis der späteren Studien lässt sich auch durch die Datenbeziehung zwischen Handlung und Motivator in Verbindung mit dem assoziativen Caching erklären: Wenn ein Mensch mit der Aussicht auf Belohnung konfrontiert wird, ist dies zunächst eine Veränderung der Umwelteinflüsse. Diese werden wahrgenommen und verarbeitet. In der Folge ist der Cache mit Assoziationen zu der Belohnung gefüllt, darunter auch die Instanz von Motivator, die die Handlung antreibt. Diese Instanz wurde durch den Menschen selbst erstellt, basierend auf seinem individuellen Assoziationsmodellen.

Nach der Durchführung der Handlung entscheidet der Mensch/sein Unterbewusstsein, ob dieser Motivator es wert ist, langfristig abgespeichert zu werden. Sollte dies nicht der Fall sein, wird der Speicher freigegeben und die Handlung nicht wiederholt. Wird der erlebte Zustand positiv bewertet *[2]*, so wird der Motivator abgespeichert und steht später wieder zur Verfügung – sofern er jemals wieder in den Cache geladen werden sollte.

Dass der Motivator aus einem alltäglichen Kontext heraus nachgeladen wird, ist nämlich eher unwahrscheinlich, da er zunächst mit der Belohnung verknüpft ist, aus dessen Assoziation er entstand. Erst wenn weitere Assoziationen zu alltäglichen Bedürfnissen aufgebaut wurden, erhöht sich die Wahrscheinlichkeit, dass die Handlung weiterhin durchgeführt wird. Bis dahin braucht es einen neuen Belohnungs-Reiz, dessen Grenznutzen mit jeder Wiederholung sinkt, weil Menschen die Verarbeitung unbekannter Reize priorisieren.

Die Erkenntnis des persönlichen Nutzens einer Handlung definiere ich als Reflexion. Durch den Reflexionsprozess werden neue Assoziationen erstellt. Dieselbe Handlung kann mehr Freude bringen, sogar bei monetär entlohnten Tätigkeiten.

SDT Refactored

In meiner Wirklichkeit gibt es keine externe Regulation und keine Internalisierung von Regulatoren. Es gibt Umwelteinflüsse, Menschen und Motivatoren, die immer intrinsisch sind, da sie als Objekte einer Private Collection gehalten werden:

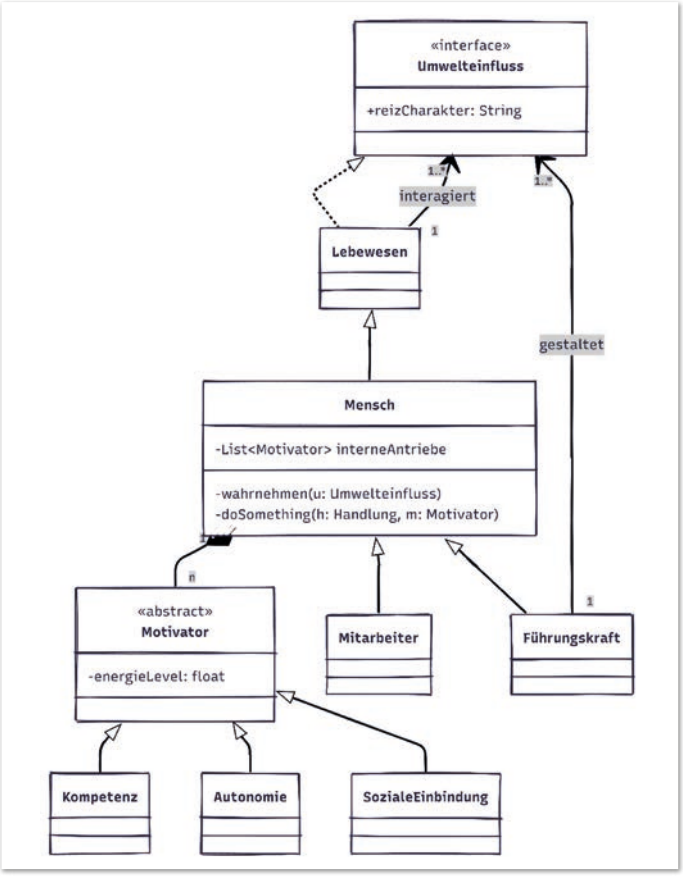


Abbildung 6: Das UML-Modell des SDT nach dem Refactoring (© Berend Semke)

Menschen nehmen die Umwelteinflüsse wahr und handeln anschließend. Dabei wird für jede Handlung die private Methode *doSomething()* aufgerufen, die mit der Handlung und ihrem Motivator genau zwei Attribute benötigt.

Führungskräfte wirken auf die Umwelteinflüsse ihrer Mitarbeiter ein. Dabei sind sie selbst ein Umwelteinfluss von höchster Relevanz. Zudem haben sie die Möglichkeit, auch andere Umwelteinflüsse zu gestalten.

Menschen haben keinen direkten Einfluss auf den Inhalt der Collection *interneAntriebe*, sie können aber die Auswahl in zirka 5 % der Fälle beeinflussen, in denen *doSomething()* aus der langsamen Handlungsplanung heraus aufgerufen wird *[8]*. Zudem können Reflexionstechniken beeinflussen, welche Motivatoren bei bestimmten Reizen nachgeladen werden.

So wird Reflexionsfähigkeit zu einer Schlüsselqualifikation. Erst wenn ich weiß, welche Objekte in der Collection liegen, kann ich genau den Motivator auswählen, der für meine aktuellen Bedürfnisse das beste Energielevel hat.

Die Collection *interneAntriebe* ist private. Nur der handelnde Mensch hat einen Zugriff darauf. Alle anderen können lediglich Beobachtungen kommunizieren, um den handelnden Menschen bei der Reflexionsarbeit zu unterstützen.

Schuld

Wir leben in einer Gesellschaft, in der die meisten Menschen darauf trainiert wurden, Ergebnisse (Output) mit Kompetenz gleichzusetzen.

- Schülern wird Kompetenz unterstellt, wenn sie in Klassenarbeiten geschickt betrügen. Dasselbe trifft zu, wenn sie die Ergebnisse während der Bearbeitungszeit geschaffen haben, unabhängig von der Energie, die sie in die Vorbereitung investierten.
- Vertriebsmitarbeiterinnen werden an „order intake" gemessen, meist unabhängig davon, ob und wie viel Zufall oder Vorarbeit von Kollegen beteiligt war.
- Agile Entwicklungsteams werden an ihrer „velocity" gemessen, meist unabhängig davon, ob und welche Impediments das Team gestört haben oder wie spezifisch die Akzeptanzkriterien beschrieben waren.

In einer Welt, in der Ergebnisse zum Messen von Kompetenz verwendet werden, entsteht bei enttäuschenden Ergebnissen ein Dilemma. Das Ergebnis zeigt Inkompetenz an, gleichzeitig haben Menschen das Bedürfnis, Kompetenz zu erleben. Daher sind Menschen in einem solchen Umfeld tendenziell motiviert, Inkompetenz (das Ergebnis) von sich abzuweisen.

Das Konzept der Schuld ist dabei enorm hilfreich für die Schuldzuweisenden:

- Indem ein spezifischer Umwelteinfluss als Ursache für das enttäuschende Ergebnis identifiziert wird,

  „Johann(a) ist schuld."

- wird Kompetenz erlebt

  „Ich habe die komplexe Situation verstanden."

- und das eigene Handeln als Kompetent bewertet.

  „Ich habe getan, was ich tun konnte.
  Es lag an Johann(a)."

- Findet diese Schuldzuweisung zudem die Zustimmung einer Gruppe von Mitmenschen, befriedigt sie das Bedürfnis nach sozialer Einbindung.

  „Die Kollegen sind auch der Meinung, dass Johann(a) das verbockt hat."

Arme(r) Johann(a).

Wenn wir die Gesundheit unserer Mitmenschen schützen und die Effizienz unserer Prozesse verbessern wollen, ist es hilfreich, das Konzept der Schuld aufzugeben. Lasst uns lieber die Umwelteinflüsse analysieren, die zu dem unerwünschten Ergebnis geführt haben *[10]*!

Das unerwünschte Ergebnis ist ein Bug. Eine Abweichung zwischen dem geplanten (spezifizierten) und dem beobachteten Verhalten in einem komplexen System, dass sich aus etlichen Subsystemen zusammensetzt. Wenn sich eine Schnittstelle ändert, müssen viele Anpassungen vorgenommen werden.

Addi

> *„Papa, wir müssen Addi umtopfen. Wir kaufen mir doch auch neue Schuhe, wenn ich aus den alten herausgewachsen bin."*

Er hatte recht. Addi musste sich mehr als ein Jahr lang mit einem kleinen Plastikbehälter begnügen, in dem vorher eine kleine Sonnenblume wuchs, die im Hochsommer auf der Terrasse leider verdorrte.

Es war Spätherbst und in unserem Gartenhaus fanden wir einen roten Übertopf in angemessener Größe. Mittlerweile wusste ich, dass Avocados keine Staunässe mögen, also bohrten wir Löcher in dem Boden des Topfes. Weil ich Angst hatte, den schönen Topf zu zerstören, nahm ich einen 5er-Bohrer und ordnete fünf Löcher symmetrisch an. Anschließend wurde Addi umgetopft und hatte fast das doppelte Volumen, um sich zu entfalten.

Wegen der Durstigkeit von Avocados, der Fußbodenheizung im Zimmer und der vielen Erde goss ich sie großzügig. Als die ersten Blätter abfielen und verwelkten, noch ein wenig großzügiger.

Irgendwann merkte ich, dass die Erde extrem feucht war und das Wasser über die fünf kleinen Löcher kaum abfließen konnte. Es stellte sich heraus, dass mein Sohn Addi auch helfen wollte und sie daher mit zusätzlichem Wasser versorgte, ohne dass wir das miteinander abgesprochen hatten *(siehe Abbildung 7)*.

Als mir klar wurde, dass wir im Begriff waren, die Pflanze zu ertränken, haben wir uns darauf geeinigt, auf das Gießen so lange zu ver-

Abbildung 7: Eine traurige Addi nach dem Umtopfen. (© Berend Semke)

zichten, bis sie vollkommen trockengefallen ist. Nach vier Wochen war dieser Zustand erreicht, in der Zwischenzeit hat sie allerdings sämtliche Blätter verloren.

Ich hätte meinem Sohn die Schuld geben können:
Schließlich hat er vorgeschlagen, die Pflanze umzutopfen.

Mein Sohn hätte mir die Schuld geben können:
Schließlich habe ich viel zu wenige,
viel zu kleine Löcher in den Topf gebohrt.

Ich hätte meinem Sohn die Schuld geben können:
Schließlich hat er die Pflanze gegossen,
ohne das mit mir abzusprechen.

Mein Sohn hätte mir die Schuld geben können:
Schließlich habe ich die Feuchtigkeit der Erde nicht gemessen.

Wir hätten gemeinsam der Avocado die Schuld geben können:
Schließlich war sie nicht ausreichend motiviert,
um das Beste aus der Situation zu machen
und die vorhandenen Ressourcen zum Weiterwachsen zu nutzen!

Die Avocado trägt keine Schuld.

Stattdessen haben wir an sie geglaubt. Wir haben sie wieder vorsichtig gegossen, ihr so viel Sonnenlicht gegeben, wie wir finden konnten. Und wir gaben ihr Zeit, damit sie sich erholen konnte *(siehe Abbildung 8)*.

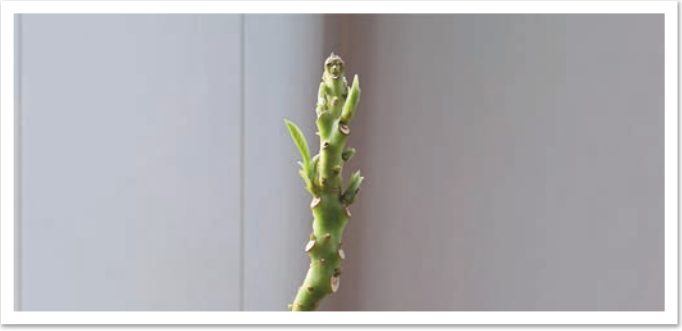


Abbildung 8: Hoffnung. (© Berend Semke)

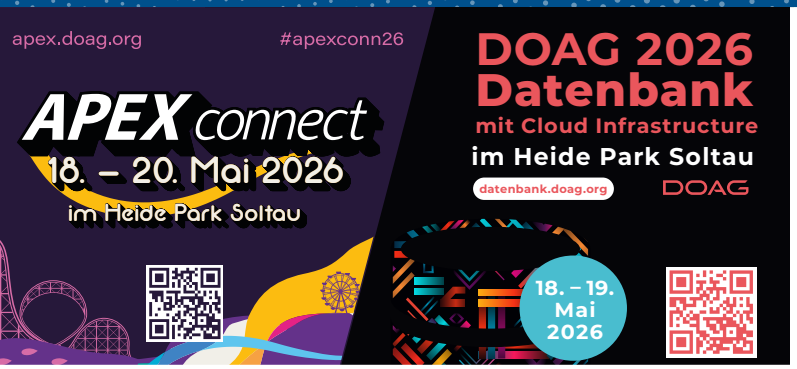
Quellen

[1] Semke, Berend (2024): Psychologie für Softwareentwickler:innen. In: Java Aktuell, Ausgabe 02/2024.
[2] Rheinberg, Falko (2004): Motivation. 5. Auflage. Stuttgart: Kohlhammer.
[3] Ryan, R. M., & Deci, E. L. (2017). Self-Determination Theory: Basic Psychological Needs in Motivation, Development, and Wellness. Guilford Press.
[4] Hare, Brian; Woods, Vanessa (2020): Survival of the Friendliest: Understanding Our Origins and Rediscovering Our Common Humanity. New York: Random House.
[5] Bregman, Rutger (2020): Im Grunde gut: Eine neue Geschichte der Menschheit. Hamburg: Rowohlt.
[6] Deci, Edward L. (1971): Effects of Externally Mediated Rewards on Intrinsic Motivation. In: Journal of Personality and Social Psychology, Vol. 18.
[7] Meyer, David E.; Schvaneveldt, Roger W. (1971): Facilitation in recognizing pairs of words: Evidence of a dependence between retrieval operations. In: Journal of Experimental Psychology, Vol. 90.
[8] Kahneman, D. (2011). Thinking, Fast and Slow (dt.: Schnelles Denken, langsames Denken).
[9] DIN EN ISO 9000:2015-11: Qualitätsmanagementsysteme – Grundlagen und Begriffe. Berlin: Beuth Verlag.
[10] Dekker, S. (2014). The Field Guide to Understanding 'Human Error'.

Berend Semke
Bredex GmbH
Berend.Semke@bredex.de

Berend Semke, Jahrgang 1983, ist seit Frühjahr 2022 Head of Project Management bei der Bredex GmbH. In seiner Arbeit verbindet Berend technische Paradigmen mit Aspekten der lösungsorientierten und tiefenpsychologischen Therapie, der systemischen Beratung und der integrativen Lerntherapie. Sein Psychologieverständnis speist sich aus 20 Jahren Austausch mit seiner Frau Lena Semke, dipl. Sozialpädagogin und integrative Lerntherapeutin, die in Braunschweig in eigener Praxis Hilfe für Erwachsene mit Lese- und Rechtschreibschwierigkeiten anbietet.

EVENT KALENDER



Impressum

Java aktuell wird von der Interessensgemeinschaft aus DOAG e.V., JavaLand GmbH und DevLand GmbH (Tempelhofer Weg 64, 12347 Berlin, java-aktuell.eu) herausgegeben.
Es ist das User-Magazin rund um die Programmiersprache Java im Raum Deutschland, Österreich und Schweiz. Es ist unabhängig von Oracle und vertritt weder direkt noch indirekt deren wirtschaftliche Interessen. Als herstellerunabhängiger Verein bietet die DOAG Software-Anwenderinnen und -Anwendern sowie Entwicklerinnen und Entwicklern eine Plattform für den Wissens- und Erfahrungsaustausch.

Java aktuell wird verlegt von der DOAG Dienstleistungen GmbH, Tempelhofer Weg 64, 12347 Berlin, Deutschland, gesetzlich vertreten durch den Geschäftsführer Fried Saacke, deren Unternehmensgegenstand Vereinsmanagement, Veranstaltungsorganisation und Publishing ist.

DOAG e.V. hält 100 Prozent der Stammeinlage der DOAG Dienstleistungen GmbH. DOAG e.V. wird gesetzlich durch den Vorstand vertreten; Vorsitzender: Jonas Janz. DOAG e.V. informiert kompetent über alle Oracle-Themen, setzt sich für die Interessen der Mitglieder ein und führen einen konstruktiv-kritischen Dialog mit Oracle.

Redaktion:
Sitz: DOAG Dienstleistungen GmbH
ViSdP: Fried Saacke
Redaktionsleitung: Lisa Damerow
Kontakt: redaktion@java-aktuell.eu

Redaktionsbeirat:
Andreas Badelt, Marcus Fihlon, Markus Karg, Manuel Mauky, Bernd Müller, Benjamin Nothdurft, Daniel van Ross, Bennet Schulz

Titel, Gestaltung und Satz:
Alexander Kermas,
DOAG Dienstleistungen GmbH

Bildnachweis:
Titel: Bild © fullvector
https://freepik.com
S. 10 + 11: pch.vector
https://freepik.com
S. 14 + 15: Bild © vetre
https://123RF.com
S. 22: Bild © Designed by freepil
https://freepik.com
S. 26 + 27: Bild - Imke Senst & Mike Kuketz
https://en.wikipedia.org/wiki/Fediverse
S. 31: Logo Fediverse
https://wikipedia.com
S. 40 + 41: Bild © macrovector_official
https://freepik.com
S. 50 + 51: Bild © Andrii
https://stock.adobe.com
S. 56 + 57: Bild © Coosh448
https://stock.adobe.com
S. 60 + 61: Bild © Willrow Hood
https://stock.adobe.com
S. 66 + 67: Bild © pikisuperstar / Freepik
https://freepik.com
S. 72: Bild © Viktor
https://stock.adobe.com
S. 78: Bild © robsonphoto
https://stock.adobe.com
S. 82 + 83: Bild © Sadia
https://stock.adobe.com

Anzeigen:
DOAG Dienstleistungen GmbH
Kontakt: sponsoring@doag.org
Mediadaten und Preise:
www.doag.org/go/mediadaten

Druck:
WIRmachenDRUCK GmbH
www.wir-machen-druck.de

Inserentenverzeichnis

APEX connect

18. - 20. Mai 2026

+

DOAG 2026 Datenbank

mit Cloud Infrastructure

18. - 19. Mai 2026

apex.doag.org

Heide Park Soltau

JavaLand

2.–4.
MÄRZ 2027

www.javaland.eu

Europa-Park
Rust

Präsentiert von: iJUG Verbund · heise medien · DOAG

javaland.eu