Ausgabe 01/2023

Java aktuell

Praxis. Wissen. Networking. Das Magazin für Entwickler

D: 4,90 EUR | CH: 9,80 CHF | A: 5,60 EUR | Benelux: 5,80 EUR

4 191978 304903 01

Java aktuell

iJUG Verbund
www.ijug.eu

Enterprise
Jakarta Server Faces 4.0, Textblöcke

Moderne Frontends
mit Spring Boot, Thymeleaf und HTMX

Ethik
Ethisches Handeln in der Softwareentwicklung

ENTERPRISE

#CloudLand2023

Eventpartner: Heise Medien

Liebe Leserinnen und Leser,

los geht es mit dem Thema Enterprise mit Bernd Müller, der einen Blick auf das aktuelle Update 4.0 der ehemaligen JavaServer Faces (nun Jakarta Faces) wirft und Neuerungen sowie Änderungen aufzeigt. Im Anschluss präsentiert Frederik Hahne eine Möglichkeit, ein modernes Frontend mit Thymeleaf als Template Engine sowie HTMX und Spring Boot zu entwickeln. Venkrat Subraniam stellt Textblöcke vor, die in den letzten Java-Versionen hinzugefügt wurden. Die Verwendung dieser vereinfacht das Arbeiten mit mehrzeiligen Strings und ermöglicht es, prägnanten und eleganten Code zu erstellen. Früher war alles besser und Developer sind nicht blind jedem Hype hinterhergelaufen? Von wegen! Mark Struberg nimmt uns mit auf seine subjektive Reise zurück zu den verschiedenen Hypewellen der IT-Welt.

Alle in einem Unternehmen definierten Regeln und Aktivitäten stellen Prozesse dar. Häufig sind diese jedoch so kompliziert gestaltet, dass sie einen fließenden Ablauf behindern können. Wie diesem negativen Einfluss mithilfe von Prozessbeschreibung und -optimierung entgegengewirkt werden kann, verrät uns Marco Schulz. Auf ethisch und moralisch korrektes Handeln in der Softwareentwicklung möchte uns Thomas Matzner in seinem Artikel ab Seite 38 sensibilisieren. Ein besonderes Augenmerk legt der Autor dabei auf die unterschiedlichen Ethikaspekte und gibt konkrete Moralempfehlungen.

Alles Wissenswerte, Spannende und Neue rund um Java und die Eclipse Foundation findet ihr wie gewohnt im Java-Tagebuch und der Eclipse Corner ganz am Anfang der Ausgabe.

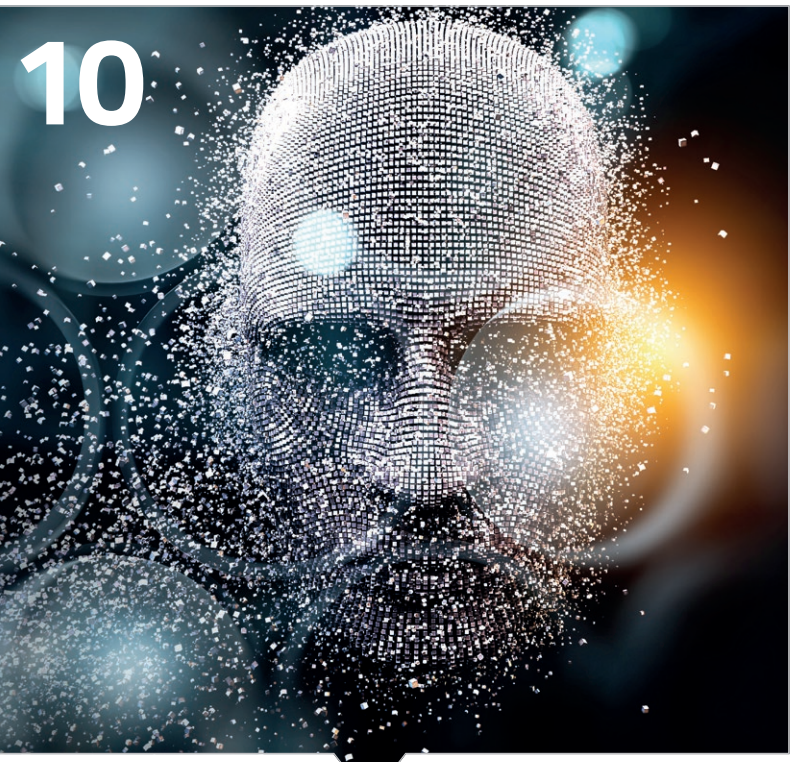
Wir wünschen euch und euren Liebsten eine entspannte Advents- und Weihnachtszeit und freuen uns, euch im kommenden Jahr wieder mit Informationen und Wissen rund um Java versorgen zu dürfen!

Wir wünschen euch viel Spaß beim Lesen!

Eure

Lisa Damerow
Redaktionsleitung Java aktuell

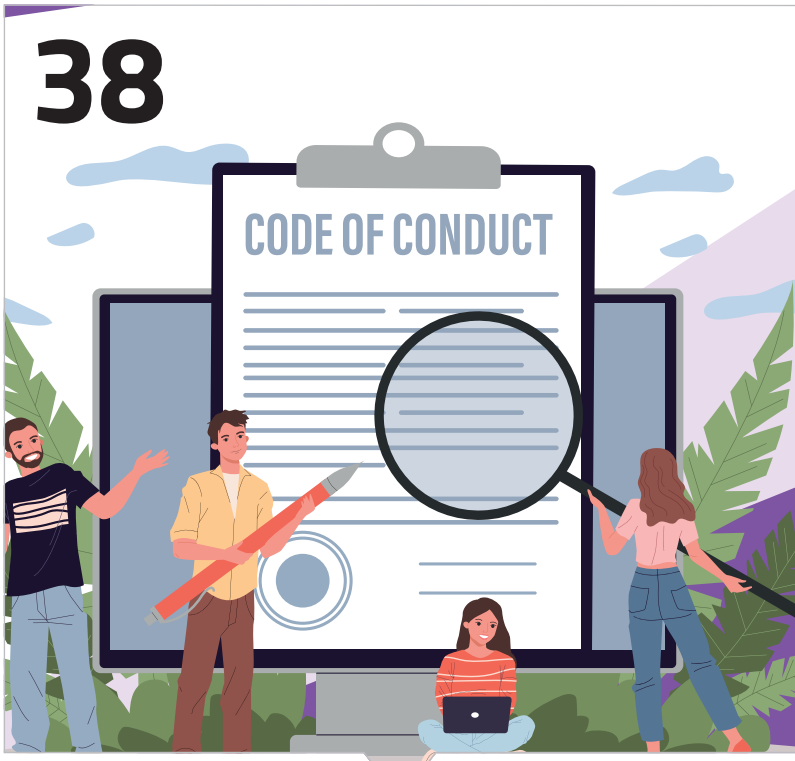
Inhalt



Jakarta Faces 4.0 im Überblick

Thymeleaf, Spring Boot und HTMX in der Frontendentwicklung

Früher war alles besser:
IT-Hypes revisited



Wie entwickle ich
möglichst moralisch und ethisch korrekte Software?

Das Java-Tagebuch

Andreas Badelt, stellv. Leiter der DOAG Java Community

10.09.2022

Jakarta EE Working Group bekommt asiatischen Schwerpunkt

Kurz nach der japanischen NEC Corporation gibt es ein weiteres asiatisches Mitglied in der Jakarta Working Group: die chinesische Ping An Technology Co. Ltd., den Technologiearm der Ping An Group mit (nach eigener Aussage) Schwerpunkten in den Bereichen AI und Cloud Services und dem Ziel, genau mit dieser Mischung global führend zu werden. Könnte hinhauen – aber wahrscheinlich kennt Ping An hierzulande kaum jemand, da es sich laut aktuellem „Forbes Global 2000"-Ranking nur um das siebzehntgrößte Unternehmen des Planeten handelt. Für Jakarta definitiv ein guter Zugewinn.

11.09.2022

Java 8: Breaking Change

Die verbliebenen Java-8-Nutzer kommen demnächst in den Genuss eines eher seltenen Phänomens: ein Breaking Change innerhalb der Major-Version (vermutlich sind aber nicht viele Programme betroffen). Die Änderungen sind aufgrund einer kritischen Sicherheitslücke nötig *[1]* und werden mit jdk8u352 ausgeliefert. Sie sind in Java 9+ sowieso bereits enthalten, müssen jetzt aber auch nachträglich in 8 eingebaut werden. Zum einen wird die Methode `Runtime.runFinalizersOnExit()` ersatzlos entfernt – diese ist eh seit Java 1.2 (!) „deprecated". Zum anderen wird `java.lang.ref.Reference.clone()` zukünftig eine „CloneNotSupportedException" werfen, um eine Race Condition mit dem Garbage Collector zu vermeiden. Der Workaround besteht darin, eine neue Instanz mit demselben „Referenten" zu erzeugen.

19.09.2022

JavaFX 19

Java 19 kommt morgen raus, das hatte ich mit dem letzten Eintrag des letzten Tagebuchs schon vorab angekündigt. Wovon ich nicht geschrieben hatte, ist, dass JavaFX, das Oracle schon 2018 aus dem OpenJDK entfernt und an die OpenJFX-Community übergeben hatte, weiterhin denselben Release-Takt einhält. JavaFX 19 ist einen Tag vor Java 19 freigegeben worden. Die Liste neuer Features und Verbesserungen ist überschaubar, im Wesentlichen sind es Fluent Bindings für das Interface ObservableValue und Support für den H.265/HEVC Codec für HTTP-Live-Streaming. Details dazu stehen hier *[2]* beziehungsweise hier *[3]*.

21.09.2022

Jakarta EE – Strategische Ausrichtung

Zum Sammeln von Ideen für „Jakarta EE next" (also erst mal 11) gibt es jetzt ein – wie üblich öffentlich verfügbares – Google Doc *[4]*. Einige Themen sind praktisch zwingend, etwa EJBs vollständig durch CDI zu ersetzen und weiteres Alignment mit MicroProfile (wie die Übernahme von MP JWT). Aber auch ein paar weniger präsente Ideen sind dabei, etwa zukünftiger Support für die virtuellen Threads in Java SE, ein Test-Framework oder Interceptors in der Expression Language.

Das Dokument soll Futter für eine breitere Initiative und einen Foliensatz liefern, in dem weniger detaillierte Ideen, sondern eher die großen Leitlinien beschrieben und priorisiert werden. Der Inhalt des Foliensatzes wird dann auch eine Grundlage für das Marketing des neuen Releases sein. Deadline für dieses zweite Dokument ist schon Mitte Oktober, aber die Inhalte bleiben ja interessant *[5]*. Ich gehe davon aus, dass mindestens das erste Dokument auch über das Datum und das nächste Release hinaus zum Sammeln von Ideen taugt. Aber im Zweifel kann jede(r) Ideen auch direkt in den Jakarta-E-Mail-Verteiler einkippen, oder über den iJUG.

22.09.2022

Jakarta EE, doch noch mal 9 Tage später

„Was soll jetzt noch passieren?", habe ich in der letzten Tagebuchausgabe gedacht und Jakarta EE 10 schon mal vorab angekündigt. Dann wurden es doch noch mal neun Tage mehr – das ist auf die Gesamtlaufzeit und vorhergehende lange Verzögerung bezogen unerheblich, aber symptomatisch. Zur Entstehungsgeschichte hat Markus Karg in der „Eclipse Corner" der letzten Java aktuell schon – deutliche – Worte gefunden. Wer etwas ausführlicher lesen möchte, was EE 10 so alles an Features bringt, dem sei die Eclipse Corner in dieser Ausgabe sowie der Artikel von Lars Röwekamp bei heise.de ans Herz gelegt *[6]*.

27.09.2022

Jakarta EE-Umfrage 2022

Interessant: Ohne neue Features in den letzten Jahren ist die Nutzung von Jakarta EE trotzdem um zirka sechs Prozentpunkte auf 53 % gestiegen, gegenüber einem leichten Rückgang bei MicroProfile und (dem weiterhin führenden) Spring. So zumindest steht es in der Auswertung des „2022 Jakarta EE Developer Survey" (Mehrfachnennungen waren möglich). Gut – dass unter den Beteiligten eines Jakarta-Surveys eh viele Jakarta nutzen, klingt logisch. Aber sowohl die Gesamtbeteiligung als auch die Jakarta-Nutzung haben gegenüber der gleichen Umfrage im Jahr 2021 zugelegt. Darunter sind natürlich ganz viele Jakarta-EE-8-Nutzer (quasi das alte Java EE), aber der Anteil derer, die auf EE 9 beziehungsweise 9.1 migriert haben, liegt inzwischen bei 14 %, und noch deutlich mehr wollen dies demnächst machen. Weitere 19 % planen, direkt auf Jakarta EE 10 zu springen.

Die Top-3-Wünsche der Teilnehmenden waren übrigens: native Integration mit Kubernetes, bessere Unterstützung für Microservices und schnellerer Support der Jakarta-EE-Hersteller/Cloud-Provider. Punkt 3 war letztes Jahr noch die Innovationsgeschwindigkeit, jetzt kommt ja mit EE 10 etwas Bewegung rein. Die Unterstützung für

Microservices wird unter anderem durch die weitere Zusammenarbeit mit MicroProfile sicherlich wachsen. Was die Integration mit Kubernetes angeht: Vermutlich stehe ich mit der Meinung ziemlich allein da, aber verglichen mit der kombinierten Lebenszeit von Java EE und Jakarta EE ist Kubernetes ja auch nur so etwas wie die neueste Sau, die durchs globale Dorf getrieben wird (zugegeben eine sehr fertile). Also Unterstützung ja – aber sich eng an Kubernetes zu binden, scheint mir langfristig riskant zu sein. Aber warten wir's ab und vertrauen der Working Group, dass sie den richtigen Mittelweg finden. Oder machen mit, höre ich Markus Karg jetzt sagen *[7]*.

29.09.2022

Liberty Server mit InstantOn

Ich habe die Termine der Blog-Einträge nicht manipuliert, falls jemand misstrauisch wird. Aber kaum schreibe ich vom Jakarta-Survey und vom Wunsch nach besserer Microservice-Unterstützung, kommt ein neuer Beitrag des IBM-Projekts OpenLiberty, in dem das neue InstantOn-Feature des Jakarta-EE- und MicroProfile-kompatiblen Frameworks beschrieben wird. Die Startup-Zeiten von JVM-Anwendungen werden ja immer noch oft thematisiert, auch wenn sich hier in den letzten Jahren nicht nur mit Quarkus viel getan hat. Mit „instant" ist bei OpenLiberty wohl eine Zahl im niedrigen dreistelligen Millisekunden-Bereich gemeint. Erreicht werden soll die Beschleunigung durch das Erstellen von „Checkpoints" beim Starten der Applikation, die dann als Basis für weitere Starts (aka „Restore") dienen *[8]*.

03.10.2022

Jakarta EE – Committee-Wahlen

Die diesjährigen Wahlen zu den verschiedenen Jakarta Committees sind abgeschlossen. Die vollständige Liste der Gewählten findet sich hier *[9]*. Nicht wundern, mit „London Jamocha Community" (ein Instant-Coffee-Milchshake!?) ist die London Java Community gemeint, eine Namensänderung ist mir nicht bekannt. Stichwort LJC: Sie wird im Steering Committee zukünftig nicht mehr von Martijn Verburg vertreten. Er hat aber Jakarta nicht den Rücken zugekehrt, sondern wollte als – inzwischen – Microsoft-Angestellter nur vermeintlichen oder tatsächlichen Interessenkonflikten vorbeugen, da das Unternehmen hier ja ebenfalls vertreten ist.

11.10.2022

MicroProfile 6.0 im Dezember

Das MicroProfile 6.0 Release wird noch etwas verschoben. Am 5. Dezember, dem Tag vor dem nächsten JakartaOne LiveStream soll es nun so weit sein. Es gab beziehungsweise gibt noch ein paar kleinere Probleme mit MP Telemetry und MP Metrics. Hauptgrund für die Verzögerungen ist aber, dass MicroProfile keine eigene Referenz-Implementierung hat, sondern der Prozess für ein Release mindestens eine kompatible Implementierung verlangt. Hier hängt MP nun am OpenLiberty Beta Release Train (da IBM als einzige überhaupt zu einer schnellen Umsetzung bereit waren). Ansonsten wurde im „Release Plan Review" noch über Namensinkonsistenzen diskutiert sowie über den richtigen Umgang mit Minor Releases der Komponenten-Spezifikationen, um Probleme mit der semantischen Versionierung („semver") zu vermeiden. Um diese (teilweise auch schon älteren) Details geradezuziehen, hat der iJUG, vertreten durch Jan Westerkamp, gegen den Plan gestimmt. Auswirkungen auf das Release hat das aber eh nicht, zumal die „deutsche Gründlichkeit" nicht geteilt wurde.

11.10.2022

Kotlin Multiplatform mit Beta-Release

JetBrains hat für Kotlin Multiplatform Mobile (KMM) jetzt die Beta-Phase angekündigt. Das SDK sei reif für den Einsatz in Projekten. Mit KMM lässt sich die Businesslogik mobiler Anwendungen einheitlich in Kotlin implementieren, für das Erstellen der UI werden dann plattformspezifische Projekte für iOS und Android generiert.

12.10.2022

Spring 6.0 Release-Kandidat

Auch Spring kommt mit einer Beta heraus: Für das Spring Framework 6.0 gibt es den ersten Release-Kandidaten (RC1), Spring Boot 3.0 soll mit einem eigenen folgen – mit Java 17 als Minimal-Anforderung und APIs aus Jakarta EE 9+ (anstelle der Java EE APIs mit alten Package-Namen). Beide sollen noch dieses Jahr freigegeben werden. Diejenigen, denen es mit Java 17 oder sonstigen Versionssprüngen zu schnell geht, können davon ausgehen, dass Spring Boot (2.7.x) noch bis Ende 2023 freien (OSS) Support hat und danach ein weiteres Jahr kommerziellen Support; für das Spring Framework (5.3.x) besteht OSS-Support bis Ende 2024 und kommerzieller Support bis Ende 2026. Allerdings werden sie wohl keine (offizielle) Unterstützung für das im nächsten Jahr herauskommende neue LTS-Release Java SE 21 mehr bieten.

Parallel zur Ankündigung gab es im Spring-Blog einen Beitrag zu ihrer Sicht auf die neuen virtuellen Threads. Kurz zusammengefasst: Sie experimentieren schon lange damit, und der Beitrag enthält Tipps, wie man sie jetzt schon minimalinvasiv in Spring nutzen kann. Spring sieht sie aber aktuell nur als Ergänzung zu den ReactiveX APIs. Zum einen wegen des bekannten Problems der virtuellen Threads mit blockierenden Calls in „synchronized"-Abschnitten („Platform Thread Pinning"); zum anderen halten sie für viele Szenarien ReactiveX für den geeigneteren Ansatz – aufgrund seiner deklarativen Nebenläufigkeit und Features jenseits eines reinen „non-blocking API" *[10]*.

18.10.2022

Log4Shell Reloaded

Wieder eine Apache Library mit einem „Feature", das eine kritische

Sicherheitslücke darstellt. Der StringSubstitutor in „commons-text" kann unter anderem URLs und DNS-Anfragen im Input String auflösen sowie Skripte ausführen. Dass das ein Sicherheitsproblem sein könnte, war beim Einbau im Jahr 2018 scheinbar noch nicht bekannt. Wer braucht so etwas? Und, wenn wirklich mehr als eine Person, muss ich das zusammen mit den allgemein nützlichen Sachen in „commons-text" packen? Oder wäre hier eine eigene Bibliothek geeigneter – „esoteric-text" oder „nuclear-powered-text-use-at-own-risk"?

Nun ja, in Version 1.10.0 sind die drei beschriebenen Lookups zumindest nicht mehr per Default aktiv, sie müssen explizit freigeschaltet werden.

Referenzen

[1] *https://blog.adoptium.net/2022/09/availability-of-jdk8u352-b05-ea/)*
[2] *https://openjfx.io/highlights/19/*
[3] *https://github.com/openjdk/jfx/pull/434*
[4] *https://docs.google.com/document/d/1m-dkvbL0iFFzitO4vt1S-Vq6GGSJyFdCDM2NU_FzGS10/*
[5] *https://docs.google.com/presentation/d/1rugEgECY-ghIlbYls-BtlDb0CTk4aeOFG/*
[6] *https://www.heise.de/news/Jakarta-EE-10-Das-erste-grosse-Release-seit-5-Jahren-zielt-auf-Cloud-Anwendungen-7272855.html*
[7] *https://eclipse-foundation.blog/2022/09/26/survey-says-confidence-continues-to-grow-in-the-jakarta-ee-ecosystem/*
[8] *https://openliberty.io/blog/2022/09/29/instant-on-beta.html*
[9] *https://www.eclipse.org/lists/jakarta.ee-wg/msg00912.html*
[10] *https://spring.io/blog/2022/10/11/embracing-virtual-threads*

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

Andreas Badelt ist stellvertretender Leiter der DOAG Cloud Native Community. Er ist seit dem Jahr 2001 ehrenamtlich im DOAG e.V. aktiv, zunächst als Co-Leiter der SIG Development und später der SIG Java. Seit 2015 ist er stellvertretender Leiter der Java Community, die ihren Bereich erweitert hat und nun Cloud Native Community heißt. Beruflich hat er seit dem Jahr 1999 als Entwickler und Architekt für deutsche und globale IT-Beratungsunternehmen gearbeitet und ist seit dem Jahr 2016 als freiberuflicher Software-Architekt unterwegs („www.badelt.it").

Markus' eclipse-Corner

Jakarta EE 10 ist da! Und allen Jubelns der Marketingmannschaft der Jakarta EE Working Group zum Trotz kann ich mich nicht wirklich richtig darüber freuen. Ich hätte nämlich sehr, sehr gerne über die tollen neuen Features berichtet. Doch nachdem ich euch jahrelang immer wieder mit Durchhalteparolen abspeisen musste, hatte ich mir vor einigen Monaten geschworen, dass ich euch an dieser Stelle keine Features schmackhaft machen werde, die ihr nicht selber auch sofort produktiv einsetzen könnt. Und so kam es, wie es kommen musste: Nicht nur, dass ich meiner Chefredakteurin schon wieder den Angstschweiß auf die Stirn getrieben habe, weil ich mal wieder bis zum letzten Drücker (ok, ich bin ehrlich, sogar zwei Tage über den internen Redaktionsschluss hinaus) mit meiner aktuellen Kolumne gewartet hatte, in der Hoffnung, dass doch noch das finale Release irgendeines Jakarta-EE-10-zertifizierten Produkts auf mei-

nem Schreibtisch landet. Mehr noch, da dies bis dato nicht eingetreten ist, musste ich ihr auch noch den großen Bericht über die neuen Features wegen dieses mir selbst gegebenen Schwurs absagen. Nun mag Lisa sicherlich ohne schwere Folgeschäden darüber hinwegkommen. Trotzdem schade, dass kein Anbieter es geschafft hat, ein zertifiziertes Produkt auf den Markt zu werfen. Ehrlich gesagt fragt man sich ja schon, wieso eigentlich. Denn was in Jakarta EE 10 drin sein würde und wie die neuen APIs aussehen würden, war ja der Öffentlichkeit bereits vor vielen Monaten zu 100 % bekannt. Letztendlich ist es ja nicht so, dass irgendein Hersteller von irgendetwas erst jetzt, ganz überraschend, urplötzlich und unerwartet, erfahren hätte. Ich überlasse es euch, den Hersteller eurer Wahl mal ganz konkret mit der Frage zu konfrontieren, was denn eigentlich sein spezifisches Problem ist und wann er gedenkt, sein Produkt auch wirklich mit Produktivsupport zu versehen. Geht man auf die Website *[1]* von Jakarta EE 10, findet sich ja schon eine ganze Menge an Herstellern, die mit diesem „Bembel" werben. Nun wird es Zeit, dass diese auch liefern.

Was hierbei auffällt, ist, dass die Seite leider nicht klar sagt, welche Version von Jakarta EE 10 diese Produkte überhaupt erfüllen und ob es ein Alpha-, ein Beta- oder ein Final-Release ist. Ich will niemandem unterstellen, dass diese Augenwischerei Absicht ist, aber zumindest ist der Zustand unglücklich und nervig. Dort wird beispielsweise Apache TomEE genannt und ein Klick darauf führt zu dessen Version 9-M7; M7 heißt: nicht fertig. Fertig ist hingegen TomEE 8, dieser ist aber nur für Jakarta EE 8 zertifiziert. Hm, also probieren wir mal Payara. Auch dort ist Payara 6 erst ein Alpha-Release. Supportet ist nur die 5, und die nur für Jakarta EE 9.1. Schade, denn Payara basiert auf GlassFish, für den es ja selbst keinen Produktivsupport gibt. Aber halt! War da nicht was? Arjan, Ondro & Co., also ehemalige Payara-Ingenieure und somit GlassFish-Experten, haben sich doch unter dem Label „OmniFish" selbstständig gemacht und bewerben seit geraumer Zeit Produktivsupport *[2]* für GlassFish. Also gleich mal googeln… und schon wieder enttäuscht: GlassFish 7 ist ebenfalls nur ein Milestone, also nicht fertig. Ich überlasse es euch, die anderen Produkte anzuklicken und ähnliche Enttäuschungen zu erfahren! Also wieder nix mit Features ausprobieren und nix mit Artikel schreiben. Schade.

Insofern verstehe ich nicht, weshalb die Freigabe von Jakarta EE 10 überhaupt irgendeine Rolle spielt und wieso das Jakarta Marketing ein solches Fass deswegen aufmacht. Ausprobieren und damit rumspielen konnte man doch auch schon vorher, es hat sich also nichts geändert. Insofern verspreche ich euch: Sobald es ein Produkt gibt, dass wir alle produktiv anwenden können, schreibe ich euch den Artikel zu Jakarta EE 10. Aber erst dann. Bis dahin gilt: Nach dem Jakarta Release ist vor dem Jakarta Release. Im wahrsten Sinne des Wortes!

Referenzen

[1] Angeblich Jakarta-EE-kompatible Produkte: *https://jakarta.ee/compatibility/*

[2] Omnifish bieten Produktivsupport für GlassFish: *https://omnifish.ee/solutions/#support*

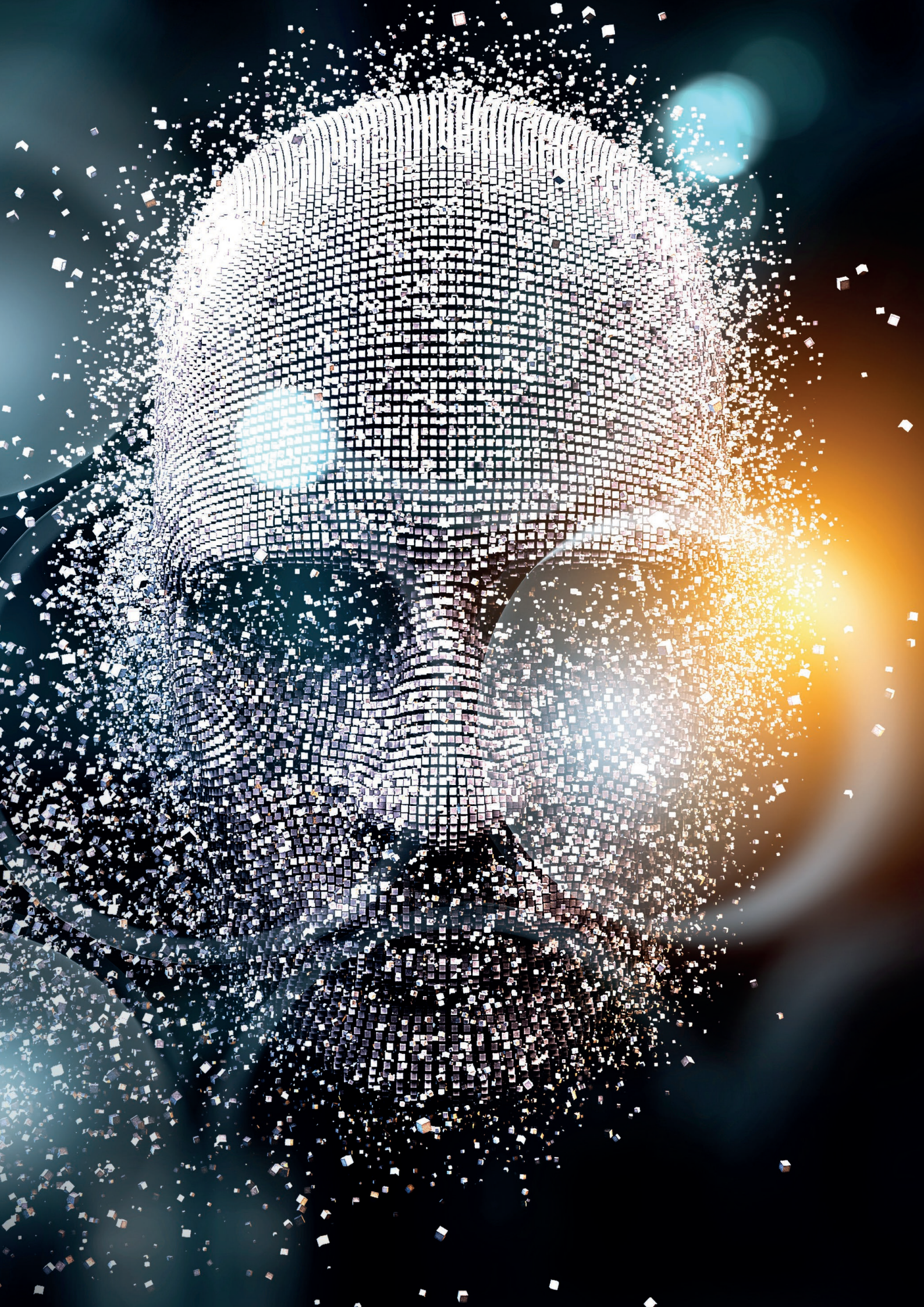
Markus Karg

markus@headcrashing.eu

Markus Karg ist Entwicklungsleiter eines mittelständischen Softwarehauses sowie Autor, Konferenzsprecher und Consultant. JAX-RS hat der Sprecher der Java User Group Goldstadt von Anfang an mitgestaltet, zunächst als freier Contributor, seit JAX-RS 2.0 als Mitglied der Expert Groups JSR 339 und JSR 370.

Totgesagte leben länger: Jakarta Faces 4.0 ist erschienen

Bernd Müller, Ostfalia



JavaServer Faces sind ganz bestimmt nicht mehr en vogue, wenn es um die Oberflächenerstellung geht, viele gehypte JavaScript-Frameworks dagegen schon. Die Halbwertszeit der JavaScript-Frameworks ist jedoch zum Teil recht niedrig, die von JavaServer Faces sehr hoch. Sie entwickeln sich immer noch weiter, sodass ein neues Release erschienen ist. Mit dem Umzug von Java EE zur Eclipse Foundation wurden die JavaServer Faces umbenannt und heißen nun Jakarta Faces. Im Rahmen von Jakarta EE 10 wurden Jakarta Faces 4.0 veröffentlicht, die wir in diesem Artikel kurz vorstellen.

Ein kleiner Rückblick

JavaServer Faces 1.0 erblickte im März 2004 das Licht der Welt. Mit der im Mai 2004 nachgeschobenen Version 1.1 waren JavaServer Faces dann auch tatsächlich in Produktivsystemen verwendbar. Im Bereich der Software-Entwicklung sind JavaServer Faces damit bereits in einem recht hohen Alter. Die letzte unter dem Dach von Java EE erschienene Version war 2.3, veröffentlicht im März 2017. Die Überführung der umfangreichen Code-Basis von Java EE zur Eclipse Foundation und vor allem die Klärung rechtlicher Fragen zog sich recht lange hin. Im Rahmen von Jakarta EE 8 wurde Jakarta Server Faces 3.0 veröffentlicht. Wie bei allen anderen EE-Spezifikationen auch, war das API identisch zur Vorgängerversion. Jakarta EE 9 vollzog die Änderung der Package-Namen, wobei der Präfix javax zu jakarta wurde. Jakarta EE 10 war das erste Release, das tatsächlich die Gelegenheit hatte, Neuerungen zu realisieren. JavaServer Faces wurden zu Jakarta Faces und die Version 4.0 deutet an, dass es wirklich Neuerungen gab. Im Folgenden werden wir meist von Faces sprechen, unabhängig von der Version.

Die Faces-Spezifikationen und viele andere Spezifikationen der EE-Familie beziehungsweise die Personen, die maßgeblich an der Erstellung der Spezifikation beteiligt waren, haben aus den Fehlern der frühen EJB-Spezifikationen gelernt und spezifizierten Framework-Eigenschaften nur, wenn die zugrunde liegenden Konzepte und Ideen bereits implementiert und auch erfolgreich eingesetzt wurden. Bei Faces sind hier vor allem MyFaces Codi, DeltaSpike *[1]* und OmniFaces *[2]* zu nennen, die die Entwicklung maßgeblich, auch personell, begleiteten. Durch das recht hohe Alter ist Faces ein sehr ausgegorenes und hochentwickeltes Framework, das wenig Wünsche offenlässt. Da ein grundlegendes Paradigma von Faces die Unterstützung von HTML ist, ist dieser Aspekt tatsächlich mehr oder weniger abgeschlossen. Für Features, die über Standard-HTML hinausgehen, sieht Faces die Verwendung von Komponentenbibliotheken vor, von denen PrimeFaces *[3]* aktuell sicher die bekannteste ist.

Die mit Faces 4.0 bezüglich HTML realisierten Neuerungen sind daher eher kosmetischer Natur. Auch die anderen neuen Features können hauptsächlich unter der Überschrift vereinfachte Entwicklung subsumiert werden. Faces 4.0 bricht aber mit einer sehr alten Tradition von Enterprise-Java, die der Rückwärtskompatibilität. Einige Framework-Eigenschaften, die schon länger nicht mehr wirklich genutzt werden, wurden tatsächlich entfernt.

Bei der Vorstellung der Änderungen von Faces 4.0 beginnen wir mit den Features, die in Zusammenhang mit HTML stehen. Danach gehen wir auf Neuerungen ein, die die Entwicklung mit Faces vereinfachen, um am Ende entfernte Features zu erwähnen.

Neuerungen mit HTML-Bezug

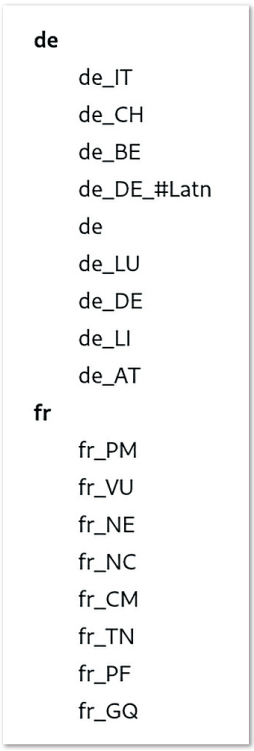
Da Faces ein Server-seitig HTML-generierendes Framework ist, sind die Neuerungen mit Bezug auf HTML von besonderem Interesse. Als Zielsprache wird schon seit mehreren Versionen HTML5 erzeugt. Zunächst konkurrierend arbeiten das W3C und die WHATWG (Web Hypertext Application Technology Working Group) mittlerweile gemeinsam an der Weiterentwicklung von HTML. Während jedoch das W3C in der Vergangenheit jeweils stabile Momentaufnahmen des Standards als HTML5, 5.1, und 5.2 *[4]* spezifizierte, arbeitet die WHATWG an einem sogenannten Living Standard *[5]* mit Namen HTML, das heißt ohne Versionsnummer. Mittlerweile verweisen die W3C-Seiten auf den Standard der WHATWG. Unabhängig von diesen beiden Varianten kann man HTML durchaus bescheinigen, dass die Weiterentwicklung im Fluss war und ist. Ein Framework wie Faces hat es damit recht schwer, up to date zu sein. Sehr vorausschauend wurden daher mit JSF 2.2 die sogenannten Pass-Through-Attribute und Pass-Through-Elemente definiert, die es ermöglichen, neue HTML-Tag-Attribute an die entsprechenden Faces-Tags unverändert durchzureichen (pass through). Die ersten der nun vorzustellenden Änderungen sind unter diesem Blickwinkel einzuordnen. Sie erleichtern dem geübten HTML-Nutzer die Arbeit, sind aber nicht zwingend notwendig. Um etwa im generierten HTML eine einfache Eingabe für eine E-Mail-Adresse zu erzeugen (`<input type="email">`), wurde bisher `<h:inputText pt:type="email" />` verwendet, wobei pt der entsprechende XML-Namensraum für Pass-Through-Attribute ist. Mit Faces 4.0 kann man nun `<h:inputText type="email" />` schreiben. Keine ganz grundlegende Verbesserung, aber einfacher zu lesen und zu schreiben. Werden Eingabetypen verwendet, für die dedizierte JSF-Tags existieren, wird im Development-Modus eine entsprechende Warnung mit Verweis auf das zu verwendende JSF-Tag ausgeben. Wird etwa der Eingabetyp password verwendet, so verweist die Warnung auf das JSF-Tag `<h:inputSecret>`, das an seiner Stelle zu nutzen ist.

```
<h:selectOneMenu>
  <f:selectItemGroups value="#{locales.localesFor('de', 'es', 'fr')}"
        var="languageLocales" itemLabel="#{languageLocales.key}">
     <f:selectItems value="#{languageLocales.value}"
        var="locale" itemLabel="#{locale}" />
  </f:selectItemGroups>
</h:selectOneMenu>
```

Listing 1

Ähnlich verhält es sich mit dem Faces-Tag `<h:inputFile>`, das gleich zwei neue Attribute bekommen hat: `multiple` und `accept`. Wird das boolesche Attribut `multiple` auf `true` gesetzt, können im Browser mehrere Dateien selektiert werden. Das Attribut `accept` erwartet eine durch Kommas getrennte Liste von Mime-Typen, also etwa `image/png` oder `application/pdf`. Vom Browser werden dann nur Dateien dieser Typen zur Auswahl angeboten.

Faces enthält eine Reihe von Tags, die zu HTML-Auswahlkomponenten gerendert werden. Die entsprechenden Auswahldaten werden mit `<f:selectItem>` und `<f:selectItems>` und der Klasse `SelectItem` definiert. Das daraus gerenderte HTML besteht aus `<option>`-Elementen. Um diese gruppieren zu können (`<optgroup>`), musste bisher die Klasse `SelectItemGroup` Java-seitig verwendet werden. Eine Gruppierung direkt in der View war nicht möglich. Mit Faces 4.0 wurden die Tags `<f:selectGroup>` und `<f:selectGroups>` eingeführt, die dies ermöglichen. Das *Listing 1* zeigt ein kleines Beispiel zur Verwendung, das die Lokalisierungen der JVM als Auswahldaten anzeigen soll. Die in `<f:selectItemGroups>` als Wert verwendete Methode `localesFor()` erwartet einen Varargs-Parameter von Strings, den Sprach-Codes von Lokalisierungen, und liefert eine Map zurück, die als Schlüssel die übergebenen Sprach-Codes und als Werte eine Liste der Lokalisierungen dieser Sprache hat. Die Iterationsvariable `languageLocales` wird nun verwendet, um im Beispiel über die drei Map-Einträge zu iterieren. Das geschachtelte `<f:selectItems>`-Tag iteriert über die Liste der Lokalisierungen der jeweiligen Sprache. Die *Abbildung 1* zeigt das resultierende Pop-up-Fenster nach Drücken des Auswahlmenüs. Das ebenfalls neue Tag `<f:selectItemGroup>` wird verwendet, wenn nur ein einzelnes `<optgroup>` gewünscht ist.



Die letzte Neuerung in Bezug auf HTML betrifft die Faces-Tags `<h:selectManyCheckbox>` und `<h:selectOneRadio>`. Das gerenderte HTML verwendete eine Tabelle für das Layout. Dies entspricht nicht mehr modernen Anforderungen für das Layout. Mit dem neuen Wert `list` für das Attribut `layout` wird eine `<ul><li>`-Struktur für die Checkboxen beziehungsweise Radio-Buttons verwendet. Die entsprechenden CSS-Definitionen sind zu ergänzen.

Nicht ganz korrekt unter der HTML-Überschrift einsortiert ist das neue Attribut `onerror` des `<f:websocket>`-Tags. Als Wert des Attributs ist eine JavaScript-Funktion anzugeben, die als Event-Handler für Verbindungsfehler des WebSocket dient.

Programmatische Erstellung von Views

Faces-Views werden mit einer Seitenbeschreibungssprache, Facelets genannt, deklarativ erstellt. Die mit JSF 1.0 ursprüngliche Definition mithilfe von JavaServer Pages wurden mit der Version 2.0 als deprecated erklärt. Mit der Version Faces 4.0 wurden nun alle mit JSP in Zusammenhang stehenden Artefakte entfernt. Dies ist eine nicht rückwärtskompatible Änderung, die allerdings zu verschmerzen sein wird, da JSPs nie richtig mit zentralen Faces-Konzepten harmoniert haben. Die programmatische Erstellung von Views war schon immer möglich, da die Elemente der Seitenbeschreibungssprache in Java-Objekte gewandelt werden, die alternativ auch direkt erzeugt werden können. Man benötigte aber eine minimale Faces-View, in der Regel mit `<f:view>` als Wurzelelement. Mit Faces 4.0 ist es nun möglich, eine View komplett mit Java, ganz ohne XHTML-Datei, zu erstellen. Da eine XHTML-Datei aber namensgebend für die View-Id ist, wird dies nun durch die neue `@View`-Annotation realisiert. *Listing 2* zeigt ein Beispiel für diese programmatische View-Definition. Wir verzichten auf eine Erläuterung, da *Listing 3* das Äquivalent in der gängigen View-Syntax zeigt und eine Gegenüberstellung der beiden Alternativen durch den Leser den Java-Code selbsterklärend macht. Abschließend lässt sich sagen, dass *Listing 3* deutlich kürzer, leichter verständlich und – nach Meinung des Autors – dem *Listing 2* vorzuziehen ist. Die programmatische Erstellung einer View ist aber nur ein Angebot. Der Entwickler hat wie immer die letzte Entscheidung.

Neuer CDI-Scope für Browser-Tabs

Mit der neuen CDI-Scope-Annotation @ClientWindowScoped kann die Lebensdauer einer Backing-Bean an ein Browser-Tab gebunden werden. Damit ist es sehr einfach möglich, dieselbe Faces-Seite, beispielsweise die der Kundenstammdaten, in mehreren Tabs für verschiedene Daten zu öffnen.

Neue Methode `getLifecycle()` der Klasse `FacesContext`

Die abstrakte Klasse FacesContext enthält alle für einen Faces-Request vorhandenen Informationen. Ein Zugriff auf den Lebenszyklus eines Faces-Request war bisher nur mit erheblichem Aufwand möglich. Die neue Methode `getLifecycle()` ändert dies und erlaubt damit zum Beispiel das programmatische Hinzufügen oder Entfernen eines Phase-Listener als Einzeiler.

Für alle bisher vorgestellten Neuerungen in Faces 4.0 stellen wir auf GitHub ein Projekt bereit *[6]*, das an kleinen Beispielen das bereits erläuterte noch einmal praktisch vorstellt.

Weitere Neuerungen, Änderungen und Löschungen

Es gibt noch weitere Neuerungen, die aber für die meisten Faces-Entwickler eher weniger relevant sein werden, sodass wir sie nicht explizit aufführen. Für eine komplette Übersicht empfehlen wir What's new in Faces 4.0? *[7]*, erstellt von Bauke Scholtz, besser bekannt unter dem Pseudonym BalusC, und unter diesem Pseudonym auf Stack Overflow der Nutzer mit den meisten Antworten zu Faces-Fragen.

Bei den Änderungen sind vor allem die XML-Namensräume wichtig. Diese waren bisher als URLs mit dem Präfix `http://xmlns.jcp.org/jsf/*` bekannt und wurden nun zu URNs mit dem Präfix `jakarta.faces.*`. Wir haben dies bereits in *Listing 3* verwendet. Weitere Änderungen sind eher kosmetischer Natur und ebenfalls unter *[7]* nachzulesen.

Als nicht rückwärtskompatible Löschungen ist neben der JSP-Löschung vor allem auf das Entfernen der Faces-eigenen Anno-

```
@View("/programmatic-facelet.xhtml")
@ApplicationScoped
public class ProgrammaticFacelet extends Facelet {

     @Override
     public void apply(FacesContext facesContext, UIComponent root) throws IOException {
            if (!facesContext.getAttributes().containsKey(IS_BUILDING_INITIAL_STATE)) {
                  return;
            }
            ComponentBuilder components = new ComponentBuilder(facesContext);
            List<UIComponent> rootChildren = root.getChildren();
            UIOutput output = new UIOutput();
            output.setValue("<html xmlns=\"http://www.w3.org/1999/xhtml\">");
            rootChildren.add(output);
            HtmlBody body = components.create(HtmlBody.COMPONENT_TYPE);
            rootChildren.add(body);
            HtmlForm form = components.create(HtmlForm.COMPONENT_TYPE);
            form.setId("form");
            body.getChildren().add(form);

            HtmlOutputLabel label = components.create(HtmlOutputLabel.COMPONENT_TYPE);
            label.setValue("Enter some text: ");
            form.getChildren().add(label);
            HtmlInputText input = components.create(HtmlInputText.COMPONENT_TYPE);
            input.setId("input");
            form.getChildren().add(input);
            String textEl = "#{backingBean.text}";
            FacesContext fc = FacesContext.getCurrentInstance();
            ValueExpression valueExpression = fc.getApplication().getExpressionFactory()
        .createValueExpression(fc.getELContext(), textEl, String.class);
            input.setValueExpression("value", valueExpression);

            HtmlCommandButton actionButton = components.create(HtmlCommandButton.COMPONENT_TYPE);
            String actionEl = "#{backingBean.action}";
            actionButton.setId("button");
            Class<?>[] parameterTypes = new Class[0];
            MethodExpression methodExpression = fc.getApplication().getExpressionFactory()
        .createMethodExpression(fc.getELContext(), actionEl, String.class, parameterTypes);
            actionButton.setActionExpression(methodExpression);
            actionButton.setValue("Do action");
            form.getChildren().add(actionButton);

            output = new UIOutput();
            output.setValue("</html>");
            rootChildren.add(output);
     }

     private static class ComponentBuilder {

            FacesContext facesContext;

            ComponentBuilder(FacesContext facesContext) {
                  this.facesContext = facesContext;
            }

            <T> T create(String componentType) {
                  return (T) facesContext.getApplication().createComponent(facesContext, componentType, null);
            }
     }

}
```

Listing 2

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
        xmlns:h="jakarta.faces.html">

     <h:body>
            <h:form id="form">
                  <h:outputLabel value="Enter some text: " />
                  <h:inputText id="input" value="#{backingBean.text}" />
                  <h:commandButton id="button" value="Do action" action="#{backingBean.action}" />
            </h:form>
     </h:body>
</html>
```

Listing 3

tationen für Dependency Injection hinzuweisen. Diese wurden mit JSF 2.0 zu einer Zeit eingeführt, als sich Dependency Injection insgesamt etablierte, aber CDI noch nicht verfügbar beziehungsweise innerhalb von Java EE noch nicht etabliert war. Die seit JSF 2.3 als deprecated gekennzeichneten Annotationen @ManagedBean sowie die Scope-Annotationen, die namensgleich zu den entsprechenden CDI-Annotationen sind, wurden entfernt. Es wurden noch einige wenige weitere Klassen entfernt, die schon seit geraumer Zeit deprecated waren. Aufgrund der geringen Relevanz zählen wir diese aber nicht auf. Faces 4.0 macht durch den Abwurf dieses alten Ballasts einen sehr aufgeräumten Eindruck. Es ist zu hoffen, eher zu erwarten, dass viele Altanwendungen die nun entfernten Klassen schon seit geraumer Zeit nicht mehr verwenden und mit Faces 4.0 problemlos lauffähig sind.

Zusammenfassung

JavaServer Faces, nun Jakarta Faces oder kurz Faces, existieren bereits seit vielen Jahren. Durch das für IT-Systeme hohe Lebensalter kann man Faces mit Fug und Recht als sehr ausgereift bezeichnen. Die neueste Version, Faces 4.0, ist die erste Version unter dem Jakarta-Dach, die mit wirklichen Änderungen aufwartet. Dazu zählen Neuerungen, tatsächliche Änderungen an Bestehendem, aber auch das Entfernen alter Zöpfe. Durch den hohen Reifegrad von Faces sind die Änderungen mehrheitlich als Features zu bewerten, die die Entwicklung vereinfachen. Es gibt ganz einfach keine grundlegenden Feature-Requests für wirklich Neues. Es bleibt abzuwarten, ob in einem der nächsten Releases derartig Neues Einzug in Faces hält und Faces damit konkurrenzfähig bleiben.

Referenzen

[1] DeltaSpike, *https://deltaspike.apache.org/*.
[2] OmniFaces, *https://omnifaces.org/*.
[3] PrimeFaces, *https://www.primefaces.org/*.
[4] W3C HTML 5.2, *https://www.w3.org/TR/2017/REC-html52-20171214/*.
[5] WHATWG HTML, *https://html.spec.whatwg.org/*.
[6] Faces4.0, *https://github.com/BerndMuller/faces4*.
[7] What's new in Faces 4.0?, *https://balusc.omnifaces.org/2021/11/whats-new-in-faces-40.html*.

Bernd Müller
Ostfalia
bernd.mueller@ostfalia.de

Nach seinem Studium der Informatik und der Promotion arbeitete Bernd Müller für die IBM und die HDI Informationssysteme. Er ist Professor, Geschäftsführer, Autor mehrerer Bücher zu den Themen JSF und JPA, sowie Speaker auf nationalen und internationalen Konferenzen. Er engagiert sich im iJUG und speziell in der JUG Ostfalen.

Moderne Frontends mit Thymeleaf, Spring Boot und HTMX

Frederik Hahne, adesso SE

Frontends für Webanwendungen werden heutzutage (fast) immer mit einem großen Framework wie Angular, React oder Vue umgesetzt. Das erhöht die Komplexität, da nicht nur eine robuste Serverseite (etwa mit Spring Boot oder Micronaut) entwickelt werden muss, sondern auch ein Frontend. Daher muss das Team ein weiteres Build-System und (Test)-Framework beherrschen, externe Abhängigkeiten verwalten sowie potenzielle Fehlerquellen kennen. Speziell im Enterprise-Umfeld ist die Schnelllebigkeit der JavaScript-Welt oft von Nachteil. In diesem Artikel wird eine Möglichkeit gezeigt, wie sich mit Thymeleaf als Template Engine und HTMX moderne Frontends entwickeln lassen, die einer Implementierung mit zum Beispiel Angular in nichts nachstehen.

Spätestens seit Thoughtworks in ihrem Technologieradar „SPA by default" auf Hold gesetzt haben, ist das Bewusstsein für die Probleme von Single-Page-Anwendungen gewachsen *[4]*. Frameworks wie Turbo/Stimulus oder auch Alpine.js ermöglichen es, auf dem Server erzeugtes HTML anzureichern, um dynamische Webanwendungen zu entwickeln. HTMX reichert HTML ebenfalls an, verfolgt aber einen anderen Ansatz. Über eine Reihe von Attributen kann die gewünschte Logik deklarativ in HTML beschrieben werden. Essenziell ist hierbei, dass dadurch jedes HTML-Element eine Serveranfrage auslösen kann und sich das Ergebnis an jeder Stelle des DOM einbinden lässt.

Wer sich hier an JSPs, AJAX und jQuery erinnert fühlt, der liegt nicht so falsch. HTMX lenkt das Ganze jedoch in geordnete Bahnen und kann noch einiges mehr, als mit jQuery möglich ist und war.

Abbildung 1: TODOMVC-Anwendung

Im Folgenden wird zunächst die bekannte TODOMVC-Anwendung *[5]* mit Thymeleaf und Spring Boot als klassische Multi-Page-Anwendung umgesetzt. Ausgehend von dieser Implementierung wird die Anwendung schrittweise angereichert, sodass am Ende eine dynamische, moderne Webanwendung steht. Danach wird die Anwendung mithilfe des Playwright-Frameworks Ende-zu-Ende getestet.

TODOMVC

Als Ausgangsbasis dient eine Umsetzung der TODOMVC-Anwendung *(siehe Abbildung 1)*. Diese wurde ursprünglich entwickelt, um unterschiedliche MVC-Frameworks (beispielsweise Angular und Backbone) mit derselben Anwendung zu vergleichen. Im Gegensatz zu einer Umsetzung als Single-Page-Anwendung verwendet die Spring-Boot-/Thymeleaf-Variante HTML-Formulare und das Post-Redirect-Get Pattern *[6]*, um folgende Funktionen umzusetzen:

- To-do hinzufügen und löschen
- To-do als erledigt markieren
- To-do filtern

Eine detaillierte Beschreibung würde den Rahmen dieses Artikels übersteigen, daher sei auf die Blogserie von Wim Deblauwe *[3]* verwiesen, die detailliert Schritt für Schritt beschreibt, wie sich eine solche Anwendung umsetzen lässt. Dem/der geneigten Leser/in sei auch das passende Buch „Taming Thymeleaf" ans Herz gelegt, um mehr über Thymeleaf zu erfahren *[2]*.

Die Anwendung basiert auf Spring Boot und folgt der typischen Package-Struktur *(siehe Abbildung 2)*. Das HTML-Template (index.html) basiert auf dem offiziellen Template der TODOMVC-Anwendung *[10]*. Die notwendigen Stylesheets werden per webjars eingebunden *(siehe Listing 1)* und entsprechend in der index.html referenziert *(siehe Listing 2)*. Der komplette Quellcode für diesen Artikel findet sich auf GitLab *[7]*.

Power Boost für HTML

HTMX ist eine 0-Dependency JavaScript-Bibliothek. Sie verspricht, moderne Browserfunktionen direkt in HTML verfügbar zu machen, statt JavaScript-Code schreiben zu müssen. HTMX versucht, die grundlegenden Konzepte von HTML als Hypertext zu erweitern, um so innerhalb von HTML neue Möglichkeiten zu eröffnen:

- Jedes Element (nicht nur a-Tags und Formulare) kann einen Request absetzen.
- Jedes Element kann einen Request auslösen.
- Jedes http-Verb ist erlaubt.
- Jedes Element (nicht nur der Body) kann als Ziel für die Antwort ausgewählt werden.

Wichtig und zunächst ungewohnt: Der Server antwortet mit (partial) HTML statt mit JSON oder XML.

Bevor die Anwendung „HTMX-ifiziert " werden kann, muss HTMX als neue Abhängigkeit hinzugefügt *(siehe Listing 3)* und dann in der index.html eingebunden werden *(siehe Listing 4)*.

Mit hx-boost ist es möglich, eine bestehende Anwendung mit HTMX zu erweitern. Wenn man zum Beispiel hx-boost in das oberste HTML-Element einfügt *(siehe Listing 5)*, werden alle Formulare auto-

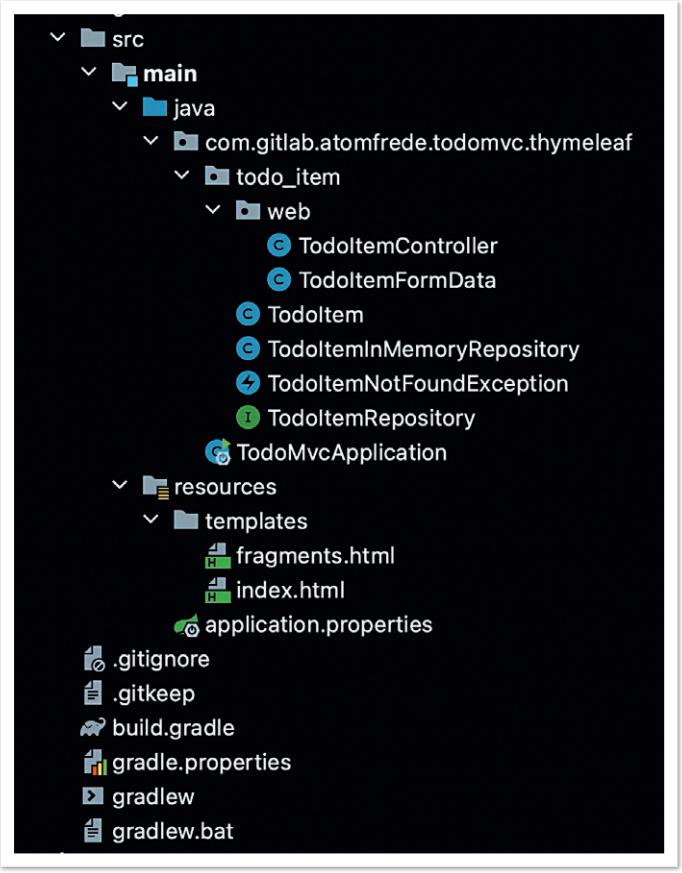


Abbildung 2: Struktur der TODOMVC-Anwendung

matisch in einen AJAX-Request umgewandelt und die Antwort der Anfrage in den Body der Seite gerendert. Der Vorteil von hx-boost ist, dass der Servercode nicht geändert werden muss und auch Anwender/innen ohne aktiviertes JavaScript die Anwendung ohne Einschränkungen verwenden können.

```
implementation "org.webjars:webjars-locator:0.45"
implementation "org.webjars.npm:todomvc-common:1.0.5"
implementation "org.webjars.npm:todomvc-app-css:2.4.1"
```

Listing 1

```
  <link rel="stylesheet" th:href="@{/webjars/todomvc-common/base.css}">
  <link rel="stylesheet" th:href="@{/webjars/todomvc-app-css/index.css}">
```

Listing 2

```
implementation "org.webjars.npm:htmx.org:1.8.0"
```

Listing 3

```
<script type="text/javascript" th:src="@{/webjars/
htmx.org/dist/htmx.min.js}"></script>
```

Listing 4

Damit HTMX beim Anklicken der „erledigt"-Checkbox den Request abfangen und durch einen AJAX-Request ersetzen kann, muss „form.submit()" durch „form.requestSubmit()" ersetzt werden *(Listing 6)*. Mit diesen Anpassungen wird die Seite nie neu geladen, als wäre die Anwendung eine Single-Page-Anwendung.

HTMX mit mehr Kontrolle

In der Regel möchte man nicht immer den kompletten Body austauschen, sondern angelehnt an die Funktionsweise von SPAs nur einen Teil der Seite aktualisieren oder austauschen. Beispielsweise ist es ausreichend, nach dem Hinzufügen eines neuen To-dos nur das Markup für dieses neue To-do, statt der kompletten Seite, zurückzuschicken.

Dies erfordert neben Anpassungen im HTML auch Änderungen am Servercode. Im Folgenden werden die notwendigen Änderungen anhand des Hinzufügens eines neuen To-dos erläutert. Eine komplette Schritt-für-Schritt-Anleitung mit allen Funktionen kann auf Wim Deblauwes Blog nachgelesen werden *[9]*.

Als Erstes muss das hx-boost-Attribut entfernt werden. Danach kann das Formular wie in *Listing 7* gezeigt angepasst werden.

Die hx-Attribute beschreiben folgendes Verhalten: Wenn „Enter" (hx-trigger) gedrückt wird, dann soll das Formular via POST an die Adresse „/" geschickt werden (hx-post) und die Antwort in das HTML-Element mit der ID „todo-list" am Ende (hx-swap) eingefügt werden (hx-target). Eine Liste aller HTMX-Attribute findet sich in der offiziellen Dokumentation *[11]*.

Die existierenden Handler im Controller können nicht verwendet werden, da diese einen Redirect nach dem POST (PRG-Pattern), statt (partial) HTML, senden. Der Controller muss eine Methode bereitstellen, die nur das Mark-up eines einzelnen To-dos zurückgibt.

Wenn eine Anfrage von HTMX gestellt wird, dann werden spezielle Header mitgeschickt, sodass der Server unterscheiden kann, ob etwa die komplette Seite gefragt ist oder nur ein bestimmtes Snippet. Daher kann der Controller so angepasst werden wie in *Listing 8* gezeigt. Diese Methode wird nur ausgewählt, wenn die Anfrage den Header „HX-Re-

```
<section class="todoapp" hx-boost="true">
```

Listing 5: hx-boost-Attribute

```
<input th:id="|toggle-checkbox-${item.id}|"
  class="toggle" type="checkbox"
  onchange="this.form.requestSubmit()"
  th:attrappend="checked=${item.completed?'true':null}">
```

Listing 6

```
<form id="new-todo-form" th:action="@{/}" method="post" th:object="${item}">
        <input id="new-todo-input" class="new-todo" placeholder="What needs to be done?" autofocus
        autocomplete="false"
        name="title"
        th:field="*{title}"
        hx-target="#todo-list"
        hx-swap="beforeend"
        hx-post="/"
        hx-trigger="keyup[key=='Enter']"
        >
    </form>
```

Listing 7

```
@PostMapping(headers = "HX-Request")
  public String htmxAddTodoItem(TodoItemFormData formData, Model model, HttpServletResponse response) {
            TodoItem item = repository.save(new TodoItem(repository.nextId(), formData.getTitle(), false));
            model.addAttribute("item", toDto(item));

            response.setHeader("HX-Trigger", "itemAdded");
            return "fragments :: todoItem";
    }
```

Listing 8

```
<span th:fragment="active-items-count"
        id="active-items-count"
        class="todo-count"
        hx-get="/active-items-count"
        hx-swap="outerHTML"
        hx-trigger="itemAdded from:body, itemCompletionToggled from:body, itemDeleted from:body">
           <th:block th:unless="${numberOfActiveItems == 1}">
                 <span class="todo-count"><strong th:text="${numberOfActiveItems}">0</strong> items left</span>
           </th:block>
           <th:block th:if="${numberOfActiveItems == 1}">
                 <span class="todo-count"><strong>1</strong> item left</span>
           </th:block>
</span>
```

Listing 9

quest" enthält. Es wird ein neues To-do angelegt sowie das entsprechende HTML-Fragment für ein To-do erzeugt und zurückgegeben.

Zusätzlich wird der Response Header „HX-Trigger" mit dem Wert „itemAdded" erzeugt. Wenn ein solcher Header zurückgegeben wird, dann erzeugt HTMX ein DOM-Event, das verwendet werden kann, um zum Beispiel wiederum Anfragen auszulösen. In diesem Fall, um die Anzahl der offenen To-dos zu aktualisieren. Hierzu muss das Thymeleaf-Fragment wie in *Listing 9* erweitert werden. Ein Blick in die Browser-Konsole zeigt, ob alles korrekt funktioniert *(siehe Abbildung 3)*.

Der komplette Quellcode ist auf GitLab verfügbar *[7]*.

Out of Band Swaps

Die zuvor gezeigte Lösung, um die Anzahl der unerledigten To-dos zu aktualisieren, funktioniert tadellos. Allerdings muss dafür ein zusätzlicher Request abgeschickt werden. Mit sogenannten Out of Band Swaps *[13]* kann zusätzlicher Inhalt neben dem Hauptinhalt an beliebigen Stellen im DOM ausgetauscht werden. Hierbei muss das Element entweder eindeutig durch eine ID referenziert oder mit hx-select ausgewählt werden.

Damit kann mit dem notwendigen Request/Response, um ein To-do anzulegen, direkt „Huckepack" das HTML mitgeschickt und ausgetauscht werden, um die Gesamtzahl der To-dos zu aktualisieren. Um Out of Band Swaps mit Thymeleaf zu verwenden, muss eine weitere Bibliothek eingebunden werden *(siehe Listing 10)*. Mithilfe dieser Bibliothek können nicht nur OOB-Swaps umgesetzt werden, die Controller-Methoden lassen sich mit einigen wenigen zusätzlichen Annotationen vereinfachen, um zum Beispiel hx-trigger Header zu setzen.

In *Listing 11* sind alle Anpassungen zu sehen. Die Annotation „HxRequest" sorgt dafür, dass diese Controller-Methode nur verwendet wird, wenn auch der HX-Request Header geschickt wird. Als Rückgabewert wird statt String ein HtmxReponse verwendet, der es ermöglicht, mehrere Templates zu kombinieren.

Nach dem Speichern des neuen To-dos wird das Model mit den be-

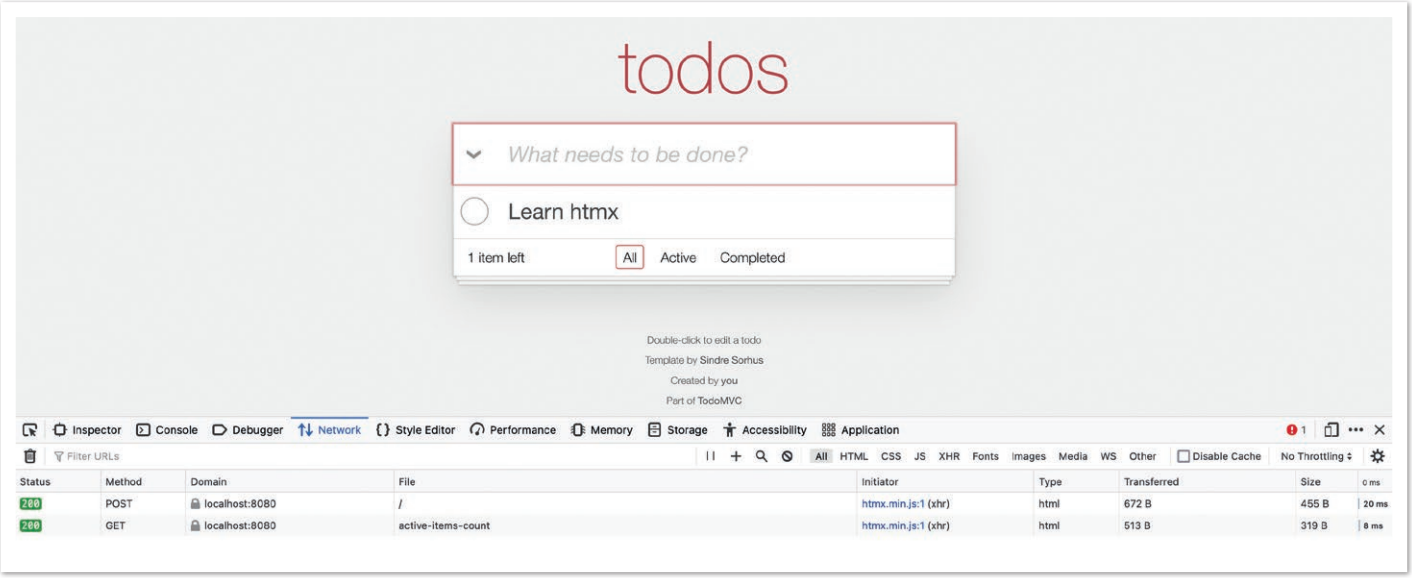


Abbildung 3: Nach dem Hinzufügen eines To-dos wird die Anzahl der offenen To-dos durch einen zweiten Request aktualisiert.

nötigten Daten (das To-do selbst und die Gesamtzahl der offenen To-dos) befüllt.

Der HtmxResponse wird mit dem Haupttemplate erzeugt und mit „.and" können beliebig weitere Snippets hinzugefügt werden, die dann via Out of Band Swaps austauschbar sind. Ein Blick in die Browser-Konsole zeigt, dass nun nur noch eine Anfrage gestellt wird und die Antwort sowohl das Mark-up für das neue To-do als auch das Mark-up für die Anzahl der offenen To-dos enthält *(siehe Abbildung 4)*. OOB-Swaps sind ein mächtiges Konstrukt, um innerhalb eines Request/Response Cycle mehrere Teile der Webanwendung dynamisch zu aktualisieren oder auszutauschen, womit sich die Dynamik einer Single-Page-Anwendung gut nachbilden lässt.

Testing mit Playwright

Da das Frontend fast vollständig in HTML beschrieben ist, braucht es kein besonderes Frontend-Testing-Framework. Mit Spring(-Boot)-Bordmitteln und MockMVC kann beispielsweise getestet werden, ob alle Templates korrekt gerendert werden würden und ob der Server Fehler korrekt behandelt. Nichtsdestotrotz wäre es wünschenswert zu testen, ob sich das Frontend im Browser wie gewünscht verhält.

Um nicht die Java-Umgebung zu verlassen, bietet es sich an, Playwright Java *[11]* und JUnit zu verwenden. Mit Playwright entfällt unter anderem das Set-up eines Selenium Grid, was End-to-End-Tests häufig schwierig macht.

Playwright Java kann wie jede andere Bibliothek als Abhängigkeit eingebunden werden *(siehe Listing 12)*. Da das Starten eines Browsers teuer ist, sollte dieser nicht für jeden Test neu erstellt werden. Damit alle Tests dennoch isoliert ablaufen, sollte ein neuer Browser-Context für jeden Test erzeugt werden. Die kann mit den JUnit Lifecycle Hooks @BeforeAll, @AfterAll, @BeforeEach und @AfterEach erreicht werden. Das Setup lässt sich in eine eigene Klasse *(siehe Listing 13)* auslagern, sodass diese dann mit einem @SpringBootTest kombinierbar ist *(Listing 14)*.

```
implementation "io.github.wimdeblauwe:htmx-spring-boot-thymeleaf:0.2.0"
```

Listing 10

```
    @PostMapping
    @HxRequest
    public HtmxResponse htmxAddTodoItem(TodoItemFormData formData,
                        Model model) {
           TodoItem item = repository.save(new TodoItem(repository.nextId(), formData.getTitle(), false));

           model.addAttribute("item", toDto(item));
           model.addAttribute("numberOfActiveItems", getNumberOfActiveItems());

           return new HtmxResponse()
                        .addTemplate("fragments :: todoItem")
                        .and(new HtmxResponse().addTemplate("fragments :: active-items-count"));
    }
```

Listing 11

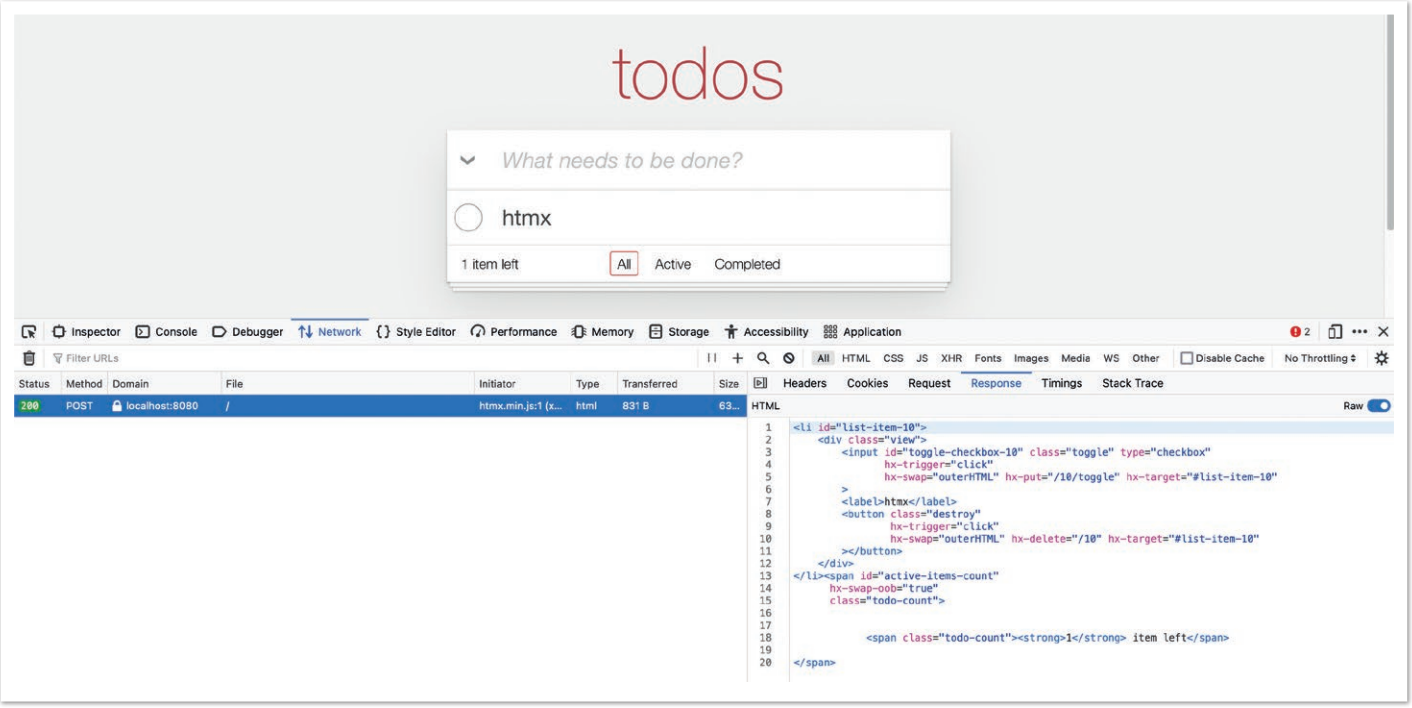


Abbildung 4: Mit OOB-Swaps wird nur noch eine Anfrage gestellt und die Antwort enthält mehrere Snippets.

Playwright unterstützt verschiedene Browser. Über spezielle LaunchOptions kann der Browser auch Headfull gestartet werden. Es werden Screenshots und Videoaufzeichnungen unterstützt.

Ein einfacher Test für die entwickelten Funktionen sollte das Eingabefeld mit einem Wert füllen und mit Enter ein neues To-do anlegen. Danach sollte das neue Element auf der Seite angezeigt werden. Da Playwright alle Prüfungen mit einem Time-out versieht, muss man als Entwickler/in keine speziellen Vorkehrungen treffen, um zum Beispiel auf das Vorhandensein eines HTML-Elementes zu warten,

```
testImplementation "com.microsoft.
playwright:playwright:1.25.0"
```

Listing 12

```
@TestInstance(TestInstance.Lifecycle.PER_CLASS)
public class PlaywrightSetup {

      protected Playwright playwright;
      protected Browser browser;
      protected BrowserContext context;
      protected Page page;

      @BeforeAll
      void launchBrowser() {
            playwright = Playwright.create();
            browser = playwright
                        .firefox()
                        .launch(new BrowserType.LaunchOptions().setHeadless(false));
      }

      @AfterAll
      void closeBrowser() {
            playwright.close();
      }

      @BeforeEach
      void createContextAndPage() {
            context = browser.newContext();
            page = context.newPage();
      }

      @AfterEach
      void closeContext() {
            context.close();
      }
}
```

Listing 13

```
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)
public class TodoMvcTest extends PlaywrightSetup{

	@LocalServerPort
	private int port;

	private String baseUrl() {
		return "localhost:%s".formatted(port);
	}

	@Test
	void shouldAddNewTodo() {
		page.navigate(baseUrl());
		page.locator("#new-todo-input").fill("htmx");
		page.locator("#new-todo-input").press("Enter");
		page.waitForSelector("#list-item-10");
	}
}
```

Listing 14

um danach eine Aktion ausführen zu können. Dadurch sind Tests gut les- und wartbar.

Der zuvor skizzierte Test ist in *Listing 14* dargestellt.

Fazit

Mit HTMX und Thymeleaf lassen sich moderne Frontends bauen, die einer typischen SPA in nichts nachstehen. Typische Anwendungsfälle wie Tabellen, modale Dialoge und Lazy Loading lassen sich mit sehr viel weniger Code umsetzen. Da die komplette Logik auf dem Server implementiert ist, muss diese nicht zusätzlich auf Clientseite implementiert werden. Sollte dennoch mehr Logik im Frontend benötigt werden, reicht es aus, clientseitiges Verhalten mit beispielsweise Alpine.js zu entwickeln.

Mit Playwright Java besteht eine Möglichkeit, zuverlässige End-to-End-Tests zu schreiben, sodass man die Anwendung mit ruhigem Gewissen ausliefern kann.

Quellen

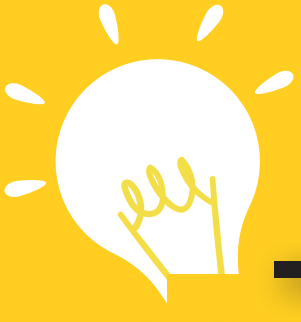
[1] *https://htmx.org/*
[2] *https://www.wimdeblauwe.com/books/taming-thymeleaf/*
[3] *https://www.wimdeblauwe.com/blog/2021/09/20/todomvc-with-spring-boot-and-thymeleaf-part-1*
[4] *https://www.thoughtworks.com/de-de/radar/techniques?blipid=202203006*
[5] *https://todomvc.com/*
[6] *https://de.wikipedia.org/wiki/Post/Redirect/Get*
[7] *https://gitlab.com/atomfrede/modern-frontends-with-thymeleaf-and-htmx*
[8] *https://www.webjars.org/*
[9] *https://www.wimdeblauwe.com/blog/2021/10/04/todomvc-with-thymeleaf-and-htmx/*
[10] *https://github.com/tastejs/todomvc-app-template*
[11] *https://htmx.org/reference/#attributes*
[12] *https://playwright.dev/java/*
[13] *https://htmx.org/docs/#oob_swaps*

Frederik Hahne
adesso SE
frederik.hahne@adesso.de

Frederik entwickelt seit 2007 vorwiegend Webanwendungen, er besitzt aber auch Erfahrung im Bau von Desktopanwendungen, angefangen von JSP-basierten Anwendungen über Wicket bis hin zu Angular und Vue.

Seit 2022 bringt Frederik sein Wissen als Senior Software Engineer für die adesso SE am wunderschönen Standort Paderborn in verschiedene, meist Java-basierte Projekte ein. In der knappen freien Zeit organisiert er die Java User Group Paderborn, um den Wissensaustauch in und um Paderborn zu fördern.

Er ist seit 2015 teil des JHipster Core Teams und kümmert sich dort insbesondere um den Gradle Support und in letzter Zeit um die Integration mit Spring Native und GraalVM.



Die Intelligenz hinter Textblöcken

Venkat Subramaniam, Agile Developer, Inc.
Aus dem Englischen übersetzt von Marcus Fihlon (@McPringle), Java User Group Switzerland.

Die Arbeit mit mehrzeiligen Strings war bisher umständlich und eher unangenehm. Dank der neuen Textblöcke können wir prägnanten und eleganten Code erstellen, wenn wir mit mehreren Textzeilen arbeiten. In diesem Artikel gehen wir speziell auf die intelligente Einrückung ein, die ein wichtiger Teil von Textblöcken ist.

Prägnante mehrzeilige Texte

Mit der Weiterentwicklung von Java können wir mehrzeilige Zeichenketten einfach und prägnant als Textblock erstellen. Angenommen, wir möchten eine Begrüßungsnachricht aus einer Funktion wie `main()` ausgeben, so kann das wie in *Listing 1* aussehen.

Ein Textblock beginnt und endet mit drei doppelten Anführungszeichen (""""). Er benötigt keine Escape-Zeichen, um Sonderzeichen wie doppelte Anführungszeichen selbst enthalten zu können. Der Code aus *Listing 1* enthält beispielsweise einen Textblock mit dem Wort „Gruß" in Anführungszeichen. Die Zeilenumbrüche bleiben ebenfalls erhalten. *Abbildung 1* zeigt die Ausgabe des kleinen Programms und

```
package com.agiledeveloper.example;

public class Greet {
  public static void main(String[] args) {
     String message = """
     Hello there,
     Hope you are doing well. It is "great" to see you.
     Happy Coding.
     """;

     System.out.print(message);
  }
}
```

Listing 1: Ein einfaches Beispiel

```
Hello there,
Hope you are doing well. It is "great" to see you.
Happy Coding.
```

Abbildung 1: Ausgabe des Programms von Listing 1

wie du siehst, wurde unser Begrüßungstext inklusive Anführungszeichen und Zeilenumbrüchen exakt so ausgegeben, wie wir ihn eingegeben haben.

Sprachen sind wie Kunst

Java ist nicht die erste Sprache mit Unterstützung von Textblöcken. Etliche andere Sprachen haben diese oder eine vergleichbare Funktion unter anderen Namen wie beispielsweise Heredocs *[1]*. In gewisser Weise können wir Programmiersprachen als Kunst betrachten und Sprachentwickler als Künstler. Die größten Künstler der Geschichte sind oft dafür bekannt, dass sie die Arbeit anderer Künstler nachahmen, dabei aber auch ihren eigenen Stil oder ihre eigenen Merkmale einbringen. Wenn es um Funktionen geht, sind Programmiersprachen in dieser Hinsicht ähnlich – sie ahmen die Funktionen anderer Sprachen nach und bringen doch ihre eigenen Nuancen in der Art und Weise, wie die Funktionen bereitgestellt oder implementiert werden, mit ein.

Die Funktion der Heredocs ist in anderen Sprachen genauso mächtig wie in Java. Allerdings haben einige von ihnen einige Ecken und Kanten, was die Behandlung von Einrückungen betrifft. Um den Textrand zu ermitteln, erfordern einige Sprachen eine spezielle Syntax oder einen höheren Programmieraufwand. Dies kann, gelinde gesagt, aus Sicht eines Programmierers von einem kleinen Ärgernis bis hin zu einer echten Zeitersparnis reichen.
Die Entwickler von Java waren nicht nur sehr daran interessiert, die Leistungsfähigkeit von Heredocs in anderen Sprachen kennenzulernen, sondern gleichzeitig auch daran, von den Nachteilen und Eigenheiten in diesen Sprachen zu lernen. Das Ergebnis zeigt sich in der Art und Weise, wie Textblöcke in Java implementiert wurden. Lasst uns hier einen Aspekt davon näher betrachten.

Intelligente Einrückungen

Der Java-Compiler ist sich der Tatsache bewusst, dass Java-Programmierer beim Schreiben von Methoden häufig Einrückungen verwenden. Es ist sehr wahrscheinlich, dass ein Textblock innerhalb mehrerer Einrückungsebenen platziert wird, zum Beispiel innerhalb einer `if`-Anweisung. In solchen Fällen würden die Einrückungen vor dem Text die Einrückung des Codes darstellen und sollten nicht in die Einrückung des zu definierenden Textes übertragen werden. Der Compiler entfernt standardmäßig diese Einrückungen *(siehe Listing 2)*.

Der Textblock erscheint innerhalb der verschachtelten if-Anweisung und innerhalb von drei Einrückungsebenen. Der Compiler erkennt jedoch, dass die Einrückung **auf den Code bezogen** und *nicht Teil des Textblocks* ist und entfernt sie, wie wir in der in *Abbildung 2* gezeigten Ausgabe sehen.

Einführung von Einrückungen

In jeder Zeile des mehrzeiligen Textes kannst du dem Compiler mitteilen, ob eine Einrückung notwendig ist. Führen wir zum Beispiel

```
package com.agiledeveloper.example;

public class TwoLevelIndentation {
  public static void main(String[] args) {
     String message = "Hi";

     if (Math.random() > 0.1) {
        message = """
        Hello there,
        Hope you're doing well.
        Happy Coding.""";
     }

     System.out.print(message);
  }
}
```

Listing 2: Ein eingerückter Textblock

```
Hello there,
Hope you're doing well.
Happy Coding.
```

Abbildung 2: Ausgabe des Programms von Listing 2

eine Einrückung in der zweiten Textzeile ein *(siehe Listing 3)*.

Da der Textblock innerhalb der dritten Einrückungsebene liegt, entfernt der Compiler die überflüssigen Leerzeichen vor jeder Zeile. Nach der Umwandlung stehen vor der ersten und der dritten Zeile keine Leerzeichen mehr. Der Compiler entfernt auch die überflüssigen Leerzeichen, sechs an der Zahl, aus Zeile 2, lässt aber die drei beabsichtigten Leerzeichen stehen. Wir können dies in der in *Abbildung 3* gezeigten Ausgabe sehen.

```
     String message = "Hi";

     if (Math.random() > 0.1) {
        message = """
        Hello there,
           Hope you're doing well.
        Happy Coding.""";
     }
```

Listing 3: Ein Textblock mit Einrückungen

```
Hello there,
   Hope you're doing well.
Happy Coding.
```

Abbildung 3: Ausgabe des Programms von Listing 3

Einrückungen beibehalten

Gelegentlich möchte man vielleicht die Einrückungen tatsächlich beibehalten oder zumindest den Compiler anweisen, seine Entscheidung zwischen überflüssigen und gewollten Einrückungen zu ändern. Lasst uns mit dem Textblock fortfahren, den wir im vorigen

Beispiel betrachtet haben. Nehmen wir an, wir wollen, dass jede Zeile um zwei Leerzeichen eingerückt wird und die zweite Zeile um weitere drei Leerzeichen. Wir können dies realisieren, indem wir die drei Anführungszeichen am Ende des Textblocks in eine separate Zeile setzen. *Listing 4* zeigt das entsprechende Beispiel.

Die einzige Änderung, die wir vorgenommen haben, besteht darin, das abschließende Begrenzungszeichen in eine neue Zeile zu verschieben und die Einrückung des Begrenzungszeichens um zwei Leerzeichen zu verringern. Jetzt berechnet der Compiler die zu entfernende Einrückung auf der Grundlage des kleinsten Einrückungswerts aller Zeilen im Textblock einschließlich des abschließenden Begrenzungszeichens. Da die Zeile mit dem abschließenden Begrenzungszeichen den kleinsten Einzug aller Zeilen des Textblocks (vier) aufweist, wird dieser Wert von vier zum zu entfernenden Einzug. Alle Einrückungen, die größer als vier sind, werden als gewollte Einrückung betrachtet. Das Ergebnis ist, dass jede Zeile, mit Ausnahme von Zeile 2, zwei Leerzeichen Einrückung hat. Die Zeile 2 hat nun insgesamt fünf Leerzeichen. Dies ist in der in *Abbildung 4* gezeigten Ausgabe zu sehen.

```
String message = "Hi";

if(Math.random() > 0.1) {
   message = """
   Hello there,
       Hope you're doing well.
   Happy Coding.
""";
```

Listing 4: Einrückungen ganz unter unserer Kontrolle

```
Hello there,
    Hope you're doing well.
Happy Coding.
```

Abbildung 4: Ausgabe des Programms von Listing 4

Diese Beispiele zeigen, dass der Java-Compiler einen recht vernünftigen Algorithmus zur Bestimmung der Einrückung hat und eine sehr bequeme Möglichkeit bietet, von seinem Standardverhalten abzuweichen.

Zusammenfassung

Textblöcke sind eine nette Erweiterung von Java, die in den letzten Versionen dazukam. Obwohl das Konzept in anderen Sprachen schon länger bekannt ist, hat Java seine eigene Art, die Funktion zu implementieren. Die Entwickler der Sprache haben nicht nur eine schöne und nützliche Funktion in die Sprache eingebracht, sondern sich auch die Mühe gemacht, sie sehr intuitiv, einfach und bequem nutzbar zu gestalten. Sie wurde sorgfältig mit Blick auf die Entwickler entworfen, die diese Sprache verwenden.

Das neue Buch von Venkat Subramaniam „Cruising Along with Java" ist ab sofort erhältlich und behandelt viele weitere neue Features von Java. Erschienen bei „The Pragmatic Programmers", ISBN 978-1-68050-981-6.

Referenzen

[1] *https://de.wikipedia.org/wiki/Heredoc*

Venkat Subramaniam
Agile Developer, Inc.
venkats@agiledeveloper.com

Dr. Venkat Subramaniam ist preisgekrönter Autor, Gründer von Agile Developer, Inc., und Lehrbeauftragter an der Universität von Houston. Er hat Tausende von Softwareentwicklern in den USA, Kanada, Europa und Asien geschult und als Mentor betreut und ist ein regelmäßig eingeladener Redner auf mehreren internationalen Konferenzen.

Venkat ist (Mit-)Autor mehrerer Fachbücher, darunter das 2007 mit dem „Jolt Productivity Award" ausgezeichnete Buch „Practices of an Agile Developer". Eine Liste seiner Bücher ist unter agiledeveloper.com zu finden.

Kontakt Übersetzung:

Marcus Fihlon
Java User Group Switzerland
marcus.fihlon@jug.ch

Eine kurze Geschichte der IT-Hypes

Mark Struberg, RISE GmbH

Früher war alles besser, sagt man. Damals haben Entwickler noch richtige Programme geschrieben, und nicht immer nur planlos von Stack Overflow Sachen zusammenkopiert. Und wir sind damals nicht jedem Hype blind nachgelaufen, hört man dann oft. Aber stimmt das auch? Oder war es früher nicht auch schon so?

Ich tendiere in der Zwischenzeit dazu, in „Wellen" zu denken. Ideen entstehen auch in der Software-Industrie nicht aus dem Nichts. Sie bauen auf vorhergehenden Ideen auf und erweitern diese oder versuchen, Schwachstellen der jeweiligen Ansätze zu beseitigen. Allerdings gibt es bei großen Änderungen seit jeher immer auch einen gewissen Trade-off, vor allem wenn es um sehr grundsätzliche Mechanismen geht. Entscheidet man sich für eine gewisse neue Architektur, dann aufgrund dessen, dass ihr ein gewisser Vorteil innewohnt, der ein gerade brennendes Problem löst. Oft wird dabei jedoch vergessen, dass jede Entwurfsentscheidung auch inhärente, systembedingte Nachteile aufweist!

Meine erste Hype-Welle

Die ersten Computer, die in den 1950er Jahren aufkamen, waren große Schränke und natürlich nur zentral verwendbar. Später gab es dazu passende Terminals und Betriebssysteme, die im Time-Sharing-Modus quasi zeitgleich mehrere Benutzer bedienen konnten. Mit der Zeit wuchs die Distanz zwischen Zentralrechner und Terminals immer weiter. Beispielsweise hatten viele Versicherungen Leitungen in ihre Landesstellen, um von dort Kunden mit berechneten Tarifen ausstatten zu können. Dies hatte allerdings ein paar Nachteile: Einwahlleitungen waren nicht sehr zuverlässig und Standleitungen fast unbezahlbar. Stand das zentrale System fehlerbedingt nicht zur Verfügung, so betraf dies immer alle Sachbearbeiter/innen – der zentrale Computer war also sehr betriebskritisch. Und auch die Performance des zentralen Systems ließ – aufgrund der hohen parallelen Nutzung – oft zu wünschen übrig.

Als Ende der 1980er PCs zu einem vernünftigen Preis breit verfügbar waren, sind viele Firmen deshalb auf eine dezentrale Architektur umgestiegen. Versicherungsunternehmen waren so nicht mehr davon abhängig, ob die Leitung und der zentrale Host funktionierten, sondern konnten Variantenberechnungen direkt im Beisein der Kunden von ihrem Arbeitsplatz aus durchrechnen. Dies brachte, wie oben beschrieben, viele Vorteile.

Allerdings übersah man dabei auch einige Nachteile, die erst mit der Zeit richtig bewusst wurden. Hat man ein dezentrales Versicherungsprogramm, so muss dieses (oder zumindest die Berechnungsformeln) zu den Sachbearbeiter/innen geschickt und dort auf den dezentralen PCs installiert werden. Schnell waren also einige Dutzend Administratoren damit beschäftigt, 3.000 Disketten pro Monat zu kopieren und zu verschicken (Online-Updates wären in Zeiten von 64-kBit-Leitungen pro Landesstelle noch viel zu zeitaufwendig gewesen). Zudem gingen von diesen 3.000 Disketten beim Transport gut ein Drittel kaputt und mussten erneut gesendet werden. Man hat also ein Problem gelöst (Availability) und dabei ungewollt ein systembedingtes anderes eingeführt (Distribution/Consistency).

Als ich mit einem Kollegen 1997 in einer großen Leasingfirma eine Web-Applikation zur Leasingberechnung vorgestellt habe, ernteten wir dafür vom Abteilungsleiter ein Lob, das mir bis heute in Erinnerung geblieben ist: „Das gefällt mir, das ist wie mein Host, nur viel hübscher und mit Bildern!" Und tatsächlich, vom Architekturansatz her ist eine Web-Applikation einer traditionellen Host-Anwendung sehr ähnlich, nämlich zentral. Diese Wellenbewegung zwischen zentraler und verteilter Applikationsarchitektur sollte sich im Laufe der Zeit noch einige Male wiederholen. Und zwar nicht nur am Frontend (Benutzeroberfläche), sondern auch in den Backend-Services und anderen Bereichen.

Es sollte nicht der letzte Hype bleiben

Auch in anderen Bereichen gab es immer wieder Hypes, die dann auf einmal jeder haben musste! Auf ein paar besonders prägende will ich im Folgenden eingehen.

Service-Oriented Architecture

Um das Jahr 2005 herum musste jede Applikation in XML und SOAP sein – alle Technologien, die man vorher gemacht hat, möge man am besten gleich vergessen! Das Internet war endlich auch bei großen Konzernen nicht mehr wegzudenken und jeder wollte seine Dienste „as a Service" anbieten – und zwar per SOAP. Bald schon begannen Consulting-Firmen diese Art der Kommunikation nicht nur für große, firmenübergreifende Dienste zu verwenden, sondern auch für interne Mechanismen. Schnell wurde die Granularität der abgebildeten Funktionalitäten immer kleiner und kleiner – bis man letztlich bei „Entity Services" angelangt war. Während eigentlich initial damit nur „Business Entities", also große fachliche Bereiche gemeint waren, wurden diese zunehmend kleiner und kleiner.

Eine Absurdität, die ich damals in einer Applikation gesehen habe, war ein „CustomerService", dem ein separater „AddressService" zur Seite gestellt war. Die Frage, wie man mit diesem Design gedenke, trotz der fehlenden Datenbanktransaktionsgarantien die konsistente Anlage eines Kunden mitsamt seiner Adressen zu garantieren, hat zuerst beim Consultingunternehmen, danach beim Kunden selbst Panik ausgelöst. Sobald man den sicheren Hafen eines gut aufgesetzten Frameworks innerhalb eines Prozesses verlässt und remote mit einem anderen Computer über das Netzwerk interagiert, ist man gnadenlos den „Fallacies of distributed computing" [Peter L. Deutsch, Sun Microsystems, 1994] ausgesetzt. Legt man einen Kunden mit fünf Adressen an und stolpert nach der zweiten Adresse zufällig jemand über das Netzwerkkabel, so hat man im Endeffekt nur die halbe Information in der Datenbank.

Somit sind wir bei der Fehlerbehandlung: Wie verfährt man, wenn man einen Fehler bemerkt, obwohl man schon etliche schreibende Operationen in anderen Systemen durchgeführt hat? Der von SOA populär gemachte Ansatz nennt sich „Compensation Logic" oder „Compensational Operation". Das klassische SOA-Beispiel ist das Buchen eines Fluges und das Bestellen eines Hotels per Web-Service. Ich benötige jedoch den Flug nicht, wenn ich kein Hotel bekomme, und umgekehrt. Beides gemeinsam atomar (wie etwa bei einer Transaktion innerhalb einer Datenbank) wird man nicht machen können, also muss man die Aufträge parallel oder hintereinander, in jedem Fall unabhängig voneinander, ausführen. Kann eine der beiden Operationen nicht erfolgreich abgeschlossen werden (zum Beispiel kein Flug verfügbar), so kommt die „Compensation" ins Spiel. In unserem Fall auf die Art, dass man die andere Operation mit einem zweiten Aufruf versucht, fachlich rückgängig zu machen, in unserem Fall das Hotel storniert.

Das klingt wunderbar und ist auch machbar, der Teufel liegt jedoch im Detail.

1. Sind es, wie im Beispiel, nur zwei Operationen, dann ist die Compensation Logic einfach – bei mehreren Operationen wird es aber schnell kompliziert.
2. Tritt innerhalb der Compensation ein Fehler auf, etwa wenn durch einen lokalen Netzwerkfehler auch das Hotel nicht mehr

storniert werden kann, so muss man sich diesen Zwischenstatus irgendwo wegschreiben und später nochmal wiederholen. In den 1960er Jahren hat man dies beispielsweise mit oftmals Hunderten von Batches gemacht.

3. Jede Compensation muss explizit programmiert werden. Wird eine Zustandskombination übersehen, so können Dateninkonsistenzen unbemerkt bestehen bleiben.

Ich habe von Telekom-Unternehmen gehört, die mehrere 100 Millionen Euro in ihre SOA-Plattformen gesteckt haben und dennoch grandios gescheitert sind. Heutzutage kenne ich keine Firma, die stolz von sich sagt, ihre gesamte interne Software basiere auf SOA. Dennoch haben mehr im Zuge des SOA-Hypes etablierte Mechanismen überlebt als vielen bewusst ist! Compensations sind auch in Microservices-Projekten (zu denen komme ich später noch) nicht wegzudenken. Viele große Firmen, gerade im Government-Bereich, verwenden noch immer SOAP zum Datenaustausch und zur Kommunikation. Hier hat SOAP gerade aufgrund seiner starren Handhabe und strengen Validierungsmechanismen (WSDL) einen großen Vorteil gegenüber vielen moderneren Datenaustauschverfahren. Allerdings werden die Applikationen nicht mehr als SOA bezeichnet.

Big Data

Obwohl der Begriff „Big Data" schon in die 1990er Jahre zurückgeht, wurde die Idee erst um 2008 herum, vor allem durch den Erfolg von Apache Hadoop, so richtig populär. Mit Big Data bezeichnet man kurz gesagt die Verarbeitung von Datensätzen, die so komplex oder groß sind, dass sie nicht mehr auf einer Box Platz haben oder mit einer Box verarbeitet werden können. Das war cool. Das war hip. Noch dazu war es auf einmal auch im Gartner Report! Spätestens ab diesem Zeitpunkt war der Terminus „Big Data" in den Management-Abteilungen der großen Firmen angekommen – mit den üblichen Konsequenzen. Plötzlich wollte jeder Big Data machen. Vom Großkonzern bis zum kleinen Klempner. Von einigen findigen Hardwareherstellern gab es sogar „Big Data Appliances" zu kaufen. Das war nichts anderes als ein 3HE-Rack-Server mit mittelmäßig vielen Festplatten und ein paar starken CPUs. Dass man damit die Definition des Begriffs ad absurdum führte, da die Definition ja geradezu sagt, dass die Daten für eine einzelne Kiste/Rack viel zu groß sind, verstanden die wenigsten Manager. In den allermeisten Fällen wären diese Firmen mit einer herkömmlichen Serverapplikation mit einer klassischen Datenbank besser bedient gewesen. 300.000 Kunden sind zwar wirklich viele – aber von Big Data ist man damit noch meilenweit entfernt!

Macht Big Data also gar keinen Sinn und war alles nur ein nutzloser Hype?

Das sicher nicht, es gab und gibt auch etliche sinnvolle Anwendungen! Sämtliche Dienste der großen Cloud-Firmen zum Beispiel. Oder eine Anwendung, die mir persönlich besonders gefallen hat: die Speicherung von Magnetresonanz- und Computertomographie-Scans, Langzeit-EKGs sowie digitale Röntgenbilder etc. in Krankenhäusern. Hier kommen schnell mehrere 100 TB an Daten pro Jahr zusammen, und diese müssen bis zu 30 Jahre lang aufbewahrt werden. Mit einer einzelnen Storage Box ist das nicht mehr zu schaffen. Nimmt man aber eine Apache-Hadoop-Installation mit 50 günstigen 2HE-Servern mit je 12 HDDs (damals 8 TB HDDs, heute würde man vermutlich 32 TB SSDs nehmen), so kommt man schon auf eine gewaltige Speichermenge. Mit dem Hadoop Distributed File System

(HDFS) werden die Festplatten im Netzwerk quasi zu einem großen Software-RAID verbunden. Die je 8 CPU-Kerne (16 Threads) pro Box, also zusammen 800 Threads, konnten nebenher mittels MapReduce wissenschaftliche Forschung rechnen. Sollte irgendwann der Platz zu klein werden, kann man einfach weitere Server in den Hadoop-Cluster hinzufügen. Doch Hand aufs Herz: Brauchen Sie in Ihrer Firma tatsächlich Dutzende Petabyte an Festplattenspeicher und haben Berechnungen für mehrere Tausend CPU-Kerne?

NoSQL

Um das Jahr 2011 herum schoss dann der nächste große Hype gen Himmel: NoSQL. Wiederum waren die Technologien, die dahintersteckten, schon alt und nur der Name und die Verbreitung stieg Hype-getrieben sprunghaft an. NoSQL ist lediglich ein Sammelbegriff für viele nicht-relationale Datenbanken. Die meisten von ihnen skalieren sehr gut und sind so für sehr große Datenmengen ausgelegt. Wobei die prinzipielle Funktionsweise oft sehr unterschiedlich ist.

Die bekanntesten Spielarten sind:

- Key/Value Stores: Quasi eine persistierende Hash-Map mit einem Schlüssel und einem Wert, wobei es im Normalfall keine Indexierung auf den Wert gibt
- Dokumentenspeicher: Hinter einem Schlüssel liegt ein komplettes Datendokument, beispielsweise im JSON-Format
- Graphdatenbanken: Diese sind sehr gut darin, Beziehungen zwischen einzelnen Einträgen aufzubauen. Ein bekannter Vertreter davon ist Neo4J
- Spaltenorientierte Datenbanken, wie zum Beispiel Apache Cassandra

Und dann gibt es noch eine Vielzahl von Systemen dazwischen, wie zum Beispiel Apache Lucene, das schon seit den späten 1990er Jahren sehr erfolgreich ist. Allerdings hat damals noch niemand den Begriff NoSQL dafür verwendet...

Gemeinsam haben die meisten NoSQL-Datenbanken, dass ohne Relationen auch die Normalisierung flachfällt. Auch unterstützen NoSQL-Datenbanken im Normalfall keine Transaktionen – die Atomarität ist lediglich für einzelne Zeilen-Operationen gewährleistet.

Haben NoSQL-Datenbanken nun tatsächlich klassische relationale Datenbanken komplett ersetzt, wie damals vollmundig prophezeit wurde? Das kann man klar verneinen. Die meisten Anwendungsfälle erreichen bei Weitem nicht die Datengröße, ab der klassische relationale Datenbanken wie PostgreSQL, Oracle, MySQL und MariaDB Probleme bekommen würden. Obwohl bei manchen Anwendungen gewisse NoSQL-Datenbanken Vorteile hätten, überwiegen oft die non-funktionalen Vorteile von im Hause bereits eingesetzter Technologie. Beispielsweise kann man die Flexibilität einer schemalosen Datenbank mit einem klassischen RDBMS (Relational DataBase Management Sytem) einfach mit dem Speichern von JSON in einer CLOB-Spalte nachbasteln, solange die Datenmenge und Komplexität überschaubar sind. Die meisten relationalen Datenbanken bieten darüber hinaus sogar schon eigene Datentypen und Operationen für JSON- und XML-Spalten an.

Wiederum gibt es Anwendungen, bei denen wir NoSQL so selbstverständlich verwenden, dass es uns gar nicht mehr bewusst ist. Einem Telefon-Anbieter wird es schwerfallen, mehrere 100 Milliarden Verbindungsinformationen pro Monat in einer relationalen Datenbank zu speichern. Fast jede große Applikation verwendet heutzutage Apache Lucene für Freitext-Suche. Noch weiter verbreitet: LDAP (OpenLDAP, ActiveDirectory) und DNS. Ja, auch das Domain-Name-System entspricht der Definition von NoSQL. NoSQL ist heutzutage also tatsächlich überall.

Cloud

Cloud is here to stay! Aber müssen jede Applikation und jede Installation automatisch bei Cloud-Providern gehostet sein? Und wie dynamisch müssen Applikationen skalieren können? Meiner Meinung nach wird hier bei vielen Kundenprojekten zu wenig hochtrabend gedacht. Aus Platzgründen will ich jedoch nur einige wenige Punkte ausführen, die man aus meiner Sicht im Hinterkopf behalten sollte:

- Wird meine Applikation einen Skalierungsbedarf haben? Netflix hat in der Weihnachtszeit vermutlich den 100-fachen Traffic im Vergleich zu einem Dienstagvormittag mitten im Jahr. Hier macht eine Cloud-basierte Lösung sehr viel Sinn, denn man kann die Anzahl der Server vollautomatisch rauf- und wieder runterskalieren. Nehmen wir allerdings interne Applikationen einer Versicherung her, so werden diese kaum einen Skalierungsbedarf aufweisen. Das Unternehmen hat eine fixe Anzahl an Mitarbeitern, die gut vorhersehbar von Montag, 7 Uhr bis Freitag, 17 Uhr eine recht gut abschätzbare und über das ganze Jahr konstante Last produzieren. Ein normales Load-Balancing zur Ausfallsicherheit würde in diesem Fall möglicherweise reichen. Schafft man es für diese Applikation, eine Cloud-Readiness mit wenig Mehraufwand zu gewährleisten, so wird man dies natürlich tun. Aber massive Verschlechterungen bei Codekomplexität und somit Wartung, Betrieb und Datenhoheit wird man möglicherweise nicht in Kauf nehmen wollen. Bei einer auch von einer unbekannten Zahl externer Kunden benutzbaren Applikation mag man zu einer gänzlich anderen Einschätzung kommen.

- Darf (rechtlich) beziehungsweise will (Datenhoheit, Betriebsspionage) ich meine Daten überhaupt bei einem Cloud-Provider hosten? Klar kann man auch bei AWS, Azure etc. ein gewisses Grundmaß an Datensicherheit sicherstellen. Doch dies bedeutet, dass sämtliche Server verschlüsselt sein müssen, so gut wie kein „Service" des Cloud-Providers benutzt werden darf etc. Eine einhundertprozentige Sicherheit erhält man dann noch immer nicht, wie man schmerzhaft bei Bekanntwerden der Meltdown- und Spectre-Vulnerabilities feststellen musste.

- Muss man, aus welchen Gründen auch immer, die Server selbst (oder bei einem Hosting-Provider) betreiben und ist die Dynamik gering, so ist es oft wesentlich einfacher, eine Virtualisierung auf KVM-Basis zu verwenden als ein komplettes Kubernetes-Cluster. Obwohl die Werkzeuge zur Verwaltung von Kubernetes immer besser werden, darf man die Komplexität eines produktiven Echtbetriebes nicht unterschätzen. Diese Erfahrung hat sogar Tesla machen müssen – dort wurde die Kubernetes Admin Console gehackt, um auf den Tesla-Servern Cryptomining zu betreiben (Tesla cryptojacking attack). Auch die Verwaltung von Netzwerken (beziehungsweise deren Isolation) und Plattenspeichern ist im Produktivbetrieb nicht gerade trivial.

Microservices

Wie beim Thema Cloud geht es auch bei Microservices zumindest zum Teil um Skalierbarkeit, vor allem des Backends. Hier konnte man die gleiche Welle betrachten, die ich schon anfangs erwähnt habe: Von CICS am Host (zentral), über EJBs (verteilt), zu clustered Monoliths mit Spring (zentral), dann SOA (Service-Oriented Architecture, verteilt), noch einmal zurück zu Spring oder CDI-basierten Monolithen, um dann mit Microservices wieder verteilt zu agieren.

Bei einer genaueren Betrachtung sind Microservices tatsächlich ein direkter Abkömmling des SOA-Ansatzes. Die Technologien SOAP und XML sind allerdings JSON und HTTP-REST gewichen. Wobei hier zwar oftmals der Begriff REST verwendet wird, die Umsetzung in den allermeisten Fällen aber nur selten den REST-Prinzipien der PhD-Thesis meines geschätzten Apache-Software-Foundation-Kollegen Roy Fielding folgt. Denn während REST eigentlich Ressourcen-orientiert arbeitet, sind die meisten Microservices klassisch imperativ umgesetzt (mach dies, mach das, ...).

Viele Mechanismen wurden direkt von SOA übernommen, etliche andere auch aus älter zurückliegenden Erkenntnissen. Zum Beispiel kennt man State-Tracking und Eventual-Consistency von alten COBOL-Applikationen aus einer Zeit vor SQL und Transaktionen (also vor 1973). Das daraus abgeleiteten SAGA-Pattern stammt von 1987 *[1]*, der allgemeine Ansatz von „Event-Driven Architecture" aus dem Anfang der 1980er Jahre.

Aber warum brauchen wir all diese alten Tricks überhaupt? Um es kurz zu machen: Seit Ted Codd 1970 SQL erfunden hat und kurze Zeit später Transaktionen dazukamen, sind wir verwöhnt geworden. Heute sind wir gewohnt, eine ACID-Transaktion mit BEGIN zu starten, danach Dutzende Datenänderungen in den angebundenen Datenbanken zu machen und am Ende ein atomares COMMIT abzusetzen. Oder ein atomares ROLLBACK, falls irgendwo ein unerwarteter technischer oder fachlicher Fehler auftritt. Im Normalfall brauchen wir uns nicht einmal selbst darum kümmern, sondern es wird durch Java Interceptors direkt in den verwendeten Frameworks abgehandelt. Dies funktioniert in Spring, EJB, Apache DeltaSpike etc. auf sehr ähnlichem Weg. Wobei die Frameworks im Endeffekt den `javax.transaction.TransactionManager` bedienen, womit große Portabilität sichergestellt wird. Auf jeder Serviceebene wird im Interceptor überprüft, ob eine Transaktion im aktuellen Thread bereits geöffnet ist. Falls nicht, wird eine neue Transaktion begonnen. Verschachtelte Serviceaufrufe verhalten sich transparent. Kommt man im Call-Stack aus den aufgerufenen Servicefunktionen wieder zu jenem Interceptor zurück, der die Transaktion begonnen hat, wird entweder `TransactionManager#commit()` oder – im Falle einer aufgetretenen Exception – ein `TransationManager#rollback()` aufgerufen. Haben wir in unserem Programm eine Fehlerbehandlung vergessen oder fliegt uns aus sonst einem Grund eine Exception um die Ohren, bleibt somit immerhin die Konsistenz unserer Daten gewährleistet! Hier liegt der entscheidende Vorteil gegenüber verteilten Applikationen, die eine Datenkonsistenz (wie vor SQL) mühsam durch manuelle Schritte sicherstellen müssen.

Bedeutet dies, dass wir in Monolithen überhaupt nicht auf Datenkonsistenz aufpassen müssen? Nein, denn sobald wir unseren lokalen Server verlassen und Daten in einem fremden System speichern (SAP, Dokumentenarchiv etc.), haben wir exakt die gleichen Probleme wie bei Microservices. Die gute Nachricht ist: An den paar wenige Stellen einer großen Business-Applikation, an denen wir diese Situation haben, kennen wir nun genügend Lösungsansätze. Denn dank Microservices-Hype sind diese nun einer großen Entwickler/innenschaft geläufig.

„If you only have a hammer, every problem appears to be a nail"

Die in all den Jahrzehnten durchlebten technischen Iterationen passierten nicht aus Jux und Tollerei, sondern weil man damit jeweils ein brennendes Problem zu lösen versuchte. In den allermeisten Fällen gelang es auch, *dieses* Problem zu lösen – wobei man sich jedoch oft ein systembedingtes *anderes* Problem wieder einhandelte. Jede Medaille hat eben zwei Seiten und jeder technische Mechanismus hat Vor-, aber auch gewisse systemimmanente Nachteile. Die wahre Frage ist, ob für den jeweils aktuell konkreten Anwendungsfall die Vorteile überwiegen. Zudem erhöht natürlich jeder zusätzlich zur Verfügung stehende Mechanismus unsere Handlungsoptionen und somit die Chance für uns beziehungsweise unsere Kunden, eine möglichst gute Lösung zu liefern.

Mein persönlicher Ansatz ist, nicht jeden neuen Hype blind auf das nächstbeste Kundenprojekt anzuwenden, sondern zuerst zu verstehen, wie die grundsätzlichen technischen Mechanismen funktionieren. Denn nicht alle Probleme lassen sich mit einem Hammer lösen...

Referenzen

[1] *https://www.cs.cornell.edu/andru/cs711/2002fa/reading/sagas.pdf*

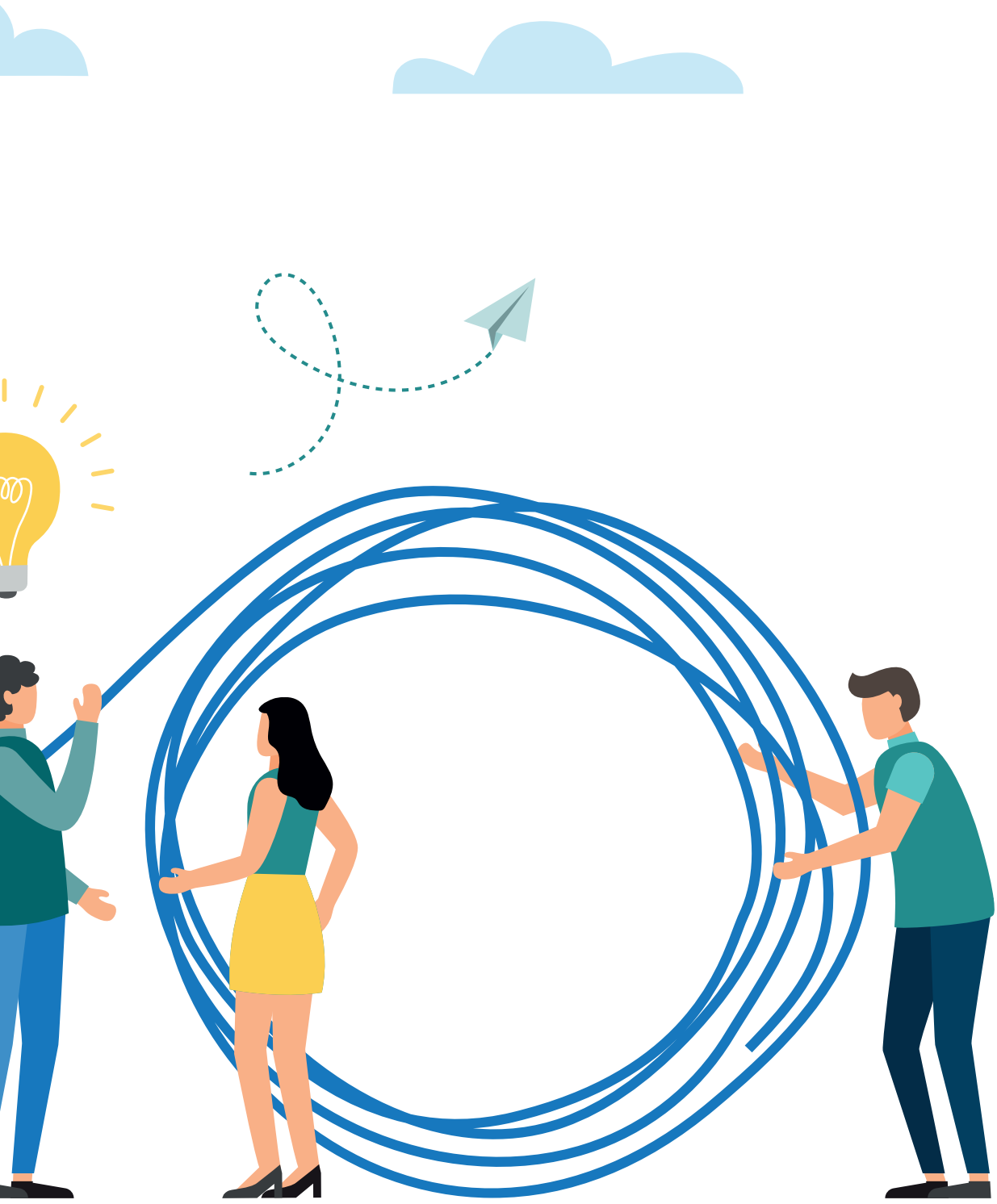
Mark Struberg
RISE GmbH

Mark ist seit 30 Jahren als Hard- und Softwareentwickler in der Industrie tätig, davon über zehn Jahre bei der Firma RISE GmbH in Wien. Seit nunmehr 20 Jahren arbeitet er aktiv an diversen Open-Source-Projekten bei der Apache Software Foundation und Spezifikationen im Java-Enterprise-Umfeld mit.

Prozesslandschaften

Marco Schulz

Sämtliche in einem Unternehmen aufgestellten Regeln und durchgeführten Aktivitäten stellen Prozesse dar. Deswegen kann auch pauschal gesagt werden, dass die Summe der Prozesse eine Organisation beschreibt. Leider sind manchmal die Prozesse so kompliziert gestaltet, dass sie sich negativ auf das Unternehmen auswirken. Was kann also getan werden, um die Situation zu verbessern?



Laut ISO-900-Definition ist ein Prozess ein Satz von in Wechselbeziehung stehenden Tätigkeiten, der Eingaben in Ergebnisse umwandelt. Dabei spielt es keine Rolle, ob der Prozess atomar ist, also nicht weiter zerlegt werden kann oder aus mehreren Prozessen zusammengesetzt wurde. An dieser Stelle ist es wichtig, auch kurz auf einige Begriffe einzugehen.

- **Choreografie**: beschreibt einzelne Operationen, aber nicht die Nachrichtenreihenfolge (Ablauf). Es behandelt die etablierte Kommunikation zwischen zwei Teilnehmern.
- **Orchestration**: beschreibt die Reihenfolge und Bedingungen der aufrufenden Teilprozesse.
- **Konversation**: beschreibt die Abfolgen zwischen Prozessen. Es wird die gesamte zulässige Kommunikation (Vollständigkeit) zwischen zwei Teilnehmern beschrieben.

Die aufgeführten Begrifflichkeiten spielen für die Beschreibung von Prozessen eine wichtige Rolle. Wenn Sie beispielsweise die Idee haben, die für Ihr Unternehmen wichtigen Geschäftsprozesse in einem Prozessbrowser visualisiert darzustellen, müssen Sie sich bereits im Vorfeld über die Detailtiefe der bereitgestellten Informationen im Klaren sein. Sollten Sie die Absicht hegen, möglichst alle Informationen in so einem Schaubild einzubringen, werden Sie schnell feststellen, wie sehr die Übersichtlichkeit darunter leidet. Wählen Sie daher immer die für die benötigte Anwendung geeignete Darstellung aus.

Ansichtssachen

Hier kommen wir auch schon zur nächsten Fragestellung: Was sind geeignete Mittel, um Prozesse verständlich darzustellen? Aus persönlicher Erfahrung hat sich in meinen Projekten eine Darstellung über den Informationsfluss gut bewährt. Dazu wiederum nutze ich die Business Process Model Notation, kurz BPMN, die für solche Zwecke geschaffen wurde. Ein frei verfügbares Werkzeug, um BPMN-Prozesse aufzuzeichnen, ist der BigAzi Modeler *[1]*. Die Möglichkeit, aus BPMN-Diagrammen wiederum softwaregestützte Programme mittels serviceorientierter Architekturen (SOA) zu erzeugen, ist für einen Großteil der Unternehmen weniger nutzbringend und nicht so einfach umzusetzen, wie es auf den ersten Blick scheint. Viel wichtiger bei einer Umsetzung zur grafischen Darstellung interner Unternehmensprozesse sind die so zutage geförderten versteckten Erkenntnisse über mögliche Verbesserungen.

Besonders Unternehmen, die eine eigenständige Softwareentwicklung betreiben und die dort angewendeten Vorgehensweisen möglichst in einem hohen Grad automatisieren wollen, können den Schritt zur Visualisierung interner Strukturen selten auslassen. Die hier viel zitierten Stichwörter Continuous Integration, Continuous Delivery und DevOps haben eine sehr hohe Automatisierungsstufe zum Ziel. Um in diesem Bereich erfolgreiche Ergebnisse erzielen zu können, ist es unumgänglich, möglichst einfache und standardisierte Prozesse etabliert zu haben. Das beschreibt auch das Paradoxon der Automatisierung:

„Prozessautomation reduziert das Risiko, dass Fehler gemacht werden. Aber hochkomplexe Prozesse sind naturgemäß nur sehr schwer zu automatisieren!"

Wenn Sie den Entschluss gefasst haben, die hauseigenen Geschäftsprozesse zu optimieren, benötigen Sie selbstredend zuerst eine realistische Analyse des aktuellen IST-Zustands, um daraus den gewünschten SOLL- Zustand zu beschreiben. Sobald diese beiden wichtigen Punkte feststehen, können Sie geeignete Maßnahmen ergreifen, mit der sie die Transformation vollziehen können.

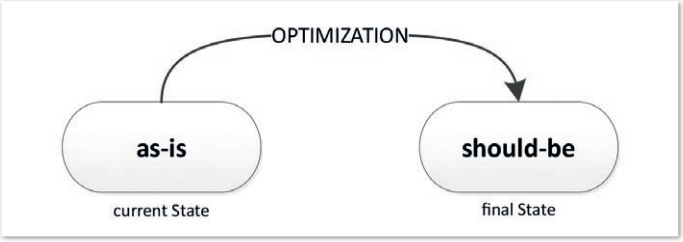


Abbildung 1: Die Transformation von der Ausgangssituation hin zur Zielstellung

Es wäre an dieser Stelle nicht sehr hilfreich, verschiedene Vorgehensmodelle dafür zu beschreiben, wie eine solche Transformation vonstattengehen kann. Solche Vorhaben sind stets sehr individuell und den tatsächlichen Gegebenheiten im Unternehmen geschuldet. Hier sei Ihnen nur ein wichtiger Ratschlag mit auf den Weg gegeben: Gehen Sie kleine, einfache Schritte und vermeiden Sie es möglichst, alles auf einmal umsetzen zu wollen. Manchmal entdecken Sie während einer Umstellung wichtige Details, die angepasst werden müssen. Das gelingt Ihnen gefahrlos, wenn Sie genügend Reserven eingeplant haben. Sie sehen, auch hier spiegeln sich agile Gedanken wider, die Ihnen die Möglichkeit geben, direkt auf Veränderungen einzugehen.

Richten Sie Ihr Augenmerk vor allem auf den zu erreichenden Sollzustand. Im Großen und Ganzen wird zwischen zwei Prozesstypen unterschieden. Autonome Prozesse laufen im Idealfall vollständig automatisiert ab und erfordern keinerlei manuelles Eingreifen. Demgegenüber stehen die interaktiven Prozesse, die an einer oder mehreren Stellen auf eine manuelle Eingabe warten, um weiter ausgeführt werden zu können. Ein sehr oft angestrebtes Ziel für den SOLL-Zustand der Prozesslandschaften sind möglichst kompakte und robuste autonome Prozesse, um den Automatisierungsgrad zu verbessern.

Folgende Punkte helfen Ihnen dabei, das gesteckte Ziel zu erreichen:

- Definieren Sie möglichst atomare Prozesse, die ausschließlich einen einzigen Vorgang oder einen Teilaspekt eines Vorgangs beschreiben.
- Halten Sie die Prozessbeschreibung möglichst einfach, orientieren Sie sich dabei an vorhanden Standards und suchen Sie nicht nach eigenen, individuellen Lösungen.
- Vermeiden Sie so gut wie möglich jegliche manuelle Interaktion.
- Wägen Sie bei Ausnahmen kritisch ab, wie oft diese tatsächlich auftreten, und suchen Sie nach möglichen Lösungen, diese Ausnahmen mit dem Standardvorgehen abarbeiten zu können.
- Setzen Sie komplexe Prozessmodelle ausschließlich aus bereits vollständig beschriebenen atomaren Teilprozessen zusammen.

Sicher stellen Sie sich die Frage, was es mit meinem Hinweis auf die Verwendung von etablierten Standards auf sich hat. Viele der in

einem Unternehmen auftretenden Probleme wurden meist schon umfangreich und bewährt gelöst. Nicht nur aus Zeit- und Kostengründen sollte bei der Verfügbarkeit bereits etablierter Vorgehensmodelle kein eigenes Süppchen gekocht werden. So erschweren Sie zum einen den Wissenstransfer zwischen Ihren Mitarbeitern und zum anderen die Verwendung von standardisierter Branchensoftware. Hierzu möchte ich Ihnen ein kleines Beispiel aus meinem Alltag vorstellen, in dem es darum geht, in Unternehmen möglichst automatisierte DevOps-Prozesse für die Softwareentwicklung und den Anwendungsbetrieb zu etablieren.

Die Kunst des Loslassens

Die größte Hürde, die ein Unternehmen hier nehmen muss, ist eine Neuorientierung an dem Begriff Release und dem dahinterliegenden Prozess, der meist eigenwillig interpretiert wurde. Die Abweichung von bekannten Standards hat wiederum mehrere spürbare Folgen. Neben erhöhtem Personalaufwand für die administrativen Eingriffe im Release-Prozess besteht auch stets die Gefahr, durch unglückliche äußere Umstände in zeitlichen Verzug zum aktuellen Plan zu geraten. Ohne auf die vielen ermüdenden technischen Details einzugehen, liegt das gravierendste Missverständnis in dem Glauben, es gäbe nach dem Erstellen eines Releases noch die Möglichkeit, die in der Testphase erkannten Fehler im selben Release zu beheben. Das sieht dann folgendermaßen aus: Nach einem Sprint wird beispielsweise das Release 2.3.0 erstellt, das dann ausgiebig in der Testphase auf Herz und Nieren überprüft wird. Stellt man nun ein Fehler fest, ist es nicht möglich, eine korrigierte Version 2.3.0 zu erzeugen. Die Korrektur hat ein neues Release zur Folge, das dann die Versionsnummer 2.3.1 trägt. Ein wichtiger Standard, der hier zum Tragen kommt, ist die Verwendung des Semantic Versioning, das jedem einzelnen Segment der Versionsnummer eine Bedeutung zuordnet. In dem hier verwendeten Beispiel zeigt die letzte Stelle die für ein Release durchgeführten Korrekturen an. Falls Sie sich etwas intensiver mit dem Thema Semantic Versioning beschäftigen mögen, empfehle ich dazu die zugehörige Internetseite *[2]*.

Was aber spricht nun dagegen, ein bereits geplantes und auf den Weg gebrachtes Release bei der Detektion von Fehlern zu stoppen, zu korrigieren und „repariert" erneut unter der bereits vergebenen Versionsnummer auf den Weg zu schicken? Die Antwort ist recht einfach: der erhebliche Arbeitsaufwand, der ausschließlich manuell durchgeführt werden muss, um den Fehler wieder auszubügeln. Abgesehen davon wird Ihre gesamte Entwicklungsarbeit für das Folge-Release erheblich ausgebremst. Ressourcen können nicht freigegeben werden und der Fortschritt beginnt zu stagnieren.

Deshalb ist es wichtig, sich so zu disziplinieren, dass ein bereits auf den Weg gebrachtes Release sämtliche Prozeduren durchläuft und erst im letzten Schritt dann die manuell ausgeführte Entscheidung getroffen wird, ob das Release für den produktiven Einsatz auch geeignet ist. Deswegen rate ich grundsätzlich dazu, den Begriff „Release-Kandidat" aus dem Sprachgebrauch zu streichen und besser von einem „Production-Kandidat" zu sprechen. Diese Bezeichnung spiegelt den Release-Prozess viel deutlicher wider.

Sollten sich während der Testphase Mängel aufzeigen, gilt es, zuerst zu entscheiden, wie schwerwiegend diese sind und deren Behebung zu priorisieren. Das kann so weit gehen, dass direkt ein Korrektur-Release auf den Weg gebracht werden muss, während parallel der nächste Sprint abgearbeitet wird. Weniger gravierende Fehler können dann auf die nächsten Folgesprints verteilt werden. Wie das alles in der täglichen Praxis umgesetzt werden kann, habe ich letztes Jahr in meinem Vortag „Rolling Stones: Vom Release überrollt" präsentiert. Den Videomitschnitt finden Sie frei zugänglich im Internet *[3]*.

Unter dem Gesichtspunkt der Prozessoptimierung bedeutet das für das aufgeführte Beispiel des Release-Prozesses, dass der Prozess beendet wurde, wenn aus dem Sourcecode erfolgreich ein binäres Artefakt mit einer noch nicht belegten Versionsnummer erstellt werden konnte. Das so entstandene Release wird umgehend an einer zentralen Stelle veröffentlicht (delivered), wo es in den Testprozess übergeben werden kann. Erst wenn der Testprozess mit dem Ergebnis abgeschlossen wurde, dass das erzeugte Release auch in Produktion verwendet werden darf, erfolgt die Übergabe in den Deployment-Prozess. Sie sehen, das, was vielerorts als ein gesamter Prozess angesehen wird, ist genau betrachtet eine Orchestration aus mindestens drei eigenständigen Prozessen.

Ein wichtiger Punkt, den Sie in *Abbildung 2* zum Thema DevOps ebenfalls herauslesen können, ist, dass der Schritt zwischen Continuous Delivery und Continuous Deployment besser nicht vollautomatisiert werden sollte, denn Deployment meint in diesem Kontext nicht das automatisierte bereitstellen der Anwendung auf allen verfügbaren Testinstanzen. Continuous Deployment meint in erster Linie ein automatisiertes Einsetzen der Anwendung in Produktion. Ob das immer eine gute Idee ist, sollte sehr sorgfältig abgewogen werden.

Ein wertvoller Aspekt der Prozessbeschreibung in Organisationen ist die Ausarbeitung wichtiger Kriterien, die erfüllt sein müssen, damit ein Prozess autonom ablaufen kann. Mit diesem Wissen können

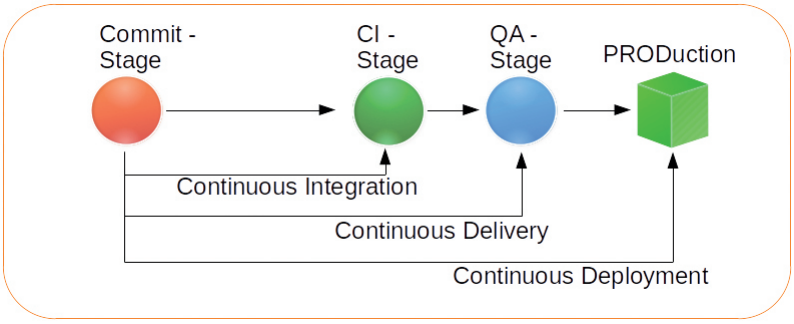


Abbildung 2: Continuous Delivery und Continuous Deployment

Sie bei der Evaluierung benötigter Werkzeuge sehr leicht einen Anforderungskatalog mit priorisierten Punkten erstellen, der einfach abgearbeitet wird. Kann das ins Auge gefasste Tool die aufgelisteten Punkte zufriedenstellend lösen und passt der aufgerufene Preis auch ins Budget, ist Ihre Suche erfolgreich beendet.

Fazit

Sehr oft wird mir entgegengebracht, dass durch moderne DevOps-Strategien der klassische Release-Prozess obsolet geworden ist. Dem kann ich nicht zustimmen. Es mag wenige Ausnahmen geben, in denen Unternehmen tatsächlich jede Codeänderung sofort in Produktion bringen. Aus Gründen der Gewährleistung und Haftung kommt für viele Firmen ein so vollständig automatisiertes Vorgehen aber nicht infrage. Auch der Datenschutz sorgt dafür, dass die Bereiche Entwicklung und Betrieb voneinander getrennt werden. Zudem benötigen umfangreiche Softwareprojekte auch eine strategische Planungsinstanz über die umzusetzenden Funktionalitäten. Diese Entscheidbarkeit wird auch künftig nicht beim Entwickler liegen, ganz gleich wie hervorgehoben der Punkt DevOps in der Stellenbeschreibung auch sein mag.

Wie Sie sehen, ist das Thema der Prozessbeschreibung und Prozessoptimierung nicht ausschließlich ein Thema für produzierende Branchen. Auch der vielerorts detailreich beschriebene Softwareentwicklungsprozess hält einiges an Verbesserungspotenzial bereit. Ich hoffe, ich konnte Sie mit meinen Zeilen ein wenig für das Thema sensibilisieren, ohne zu sehr ins Technische verfallen zu sein.

Referenzen

[1] *https://www.bizagi.com/en/platform/modeler*
[2] *https://semver.org*
[3] *https://youtu.be/m1m8Snjnjdc/*

Marco Schulz
Twitter: @ElmarDott

Marco Schulz studierte an der HS Merseburg Diplominformatik und twittert regelmäßig als @ElmarDott über alle möglichen technischen Themen. Seine Schwerpunkte sind hauptsächlich Build- und Konfigurationsmanagement, Software-Architekturen und Release-Management. Seit über fünfzehn Jahren realisiert er in internationalen Projekten für namhafte Unternehmen umfangreiche Webapplikationen. Er ist freier Consultant/Trainer. Sein Wissen teilt er mit anderen Technikbegeisterten auf Konferenzen, wenn er nicht gerade wieder einmal an einem neuen Fachbeitrag schreibt. Seine Homepage finden Sie unter: *https://elmar-dott.com*

DAS CLOUD NATIVE FESTIVAL

DAS EVENT DER DEUTSCHSPRACHIGEN CLOUD NATIVE COMMUNITY

on demand

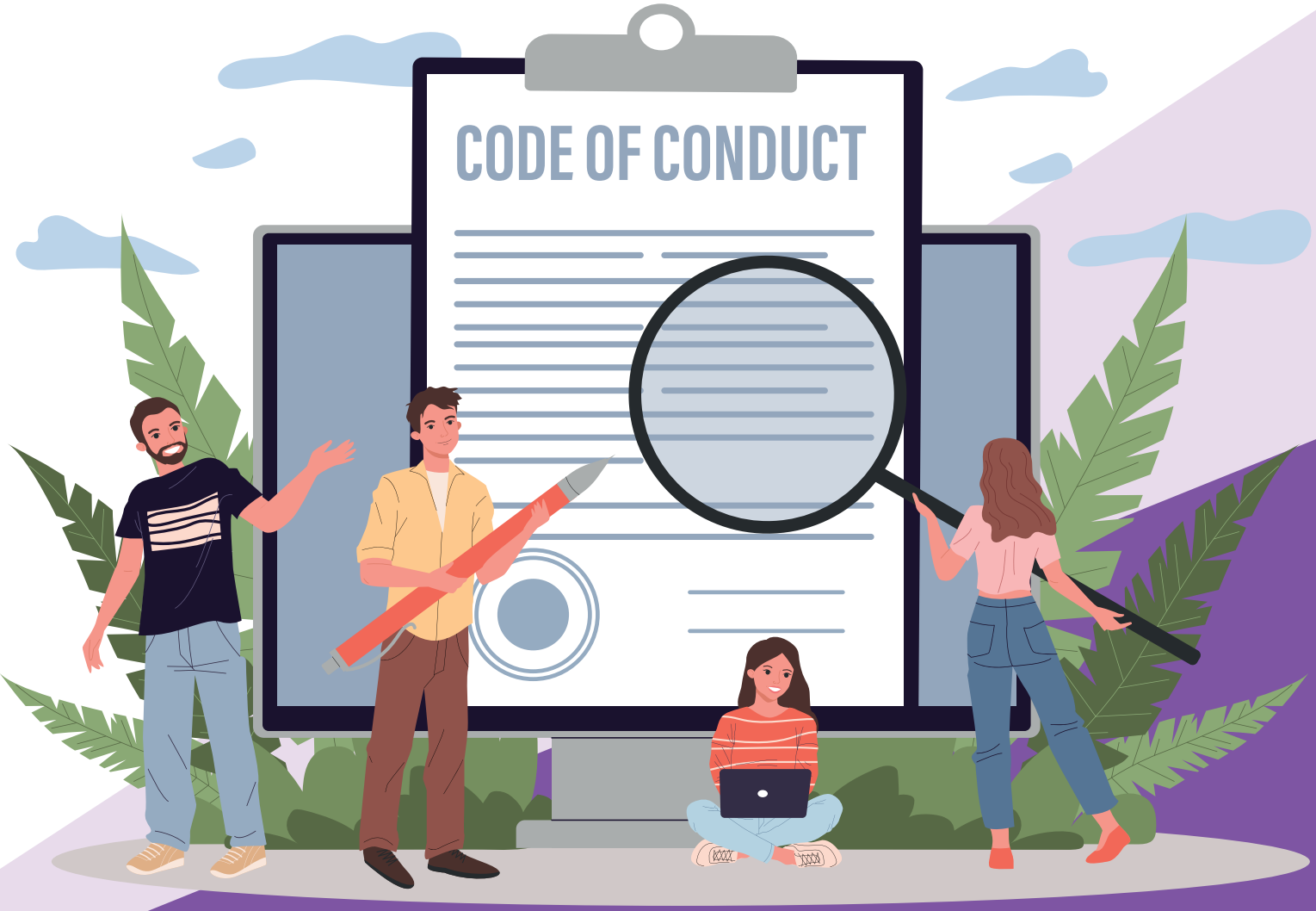
DAS CLOUD NATIVE FESTIVAL VERPASST?

JETZT ON-DEMAND-TICKET BUCHEN UND VORTRAGSAUFZEICHNUNGEN ANSCHAUEN!

Alle Angebote im On-demand-Ticket-Shop

Ethisches Handeln durch Software Engineering

Thomas Matzner



Ethische Fragen im Zusammenhang mit Informatiksystemen finden in den letzten Jahren zunehmendes Interesse. Ein Kernpunkt der von mir entworfenen Informatikethik (Matzner 2020) sind konkrete Moralempfehlungen, die denen, die an Entwicklung und Betrieb von Informatiksystemen beteiligt sind, zeigen, bei welchen Tätigkeiten welches Verhalten dazu dienen kann, moralisch erfreuliche Ergebnisse zu erzielen.

An drei Fallstudien möchte ich einen Ausschnitt dieser moralischen Erwägungen illustrieren:

- (Nezik 2019) berichtet von einer Software, die von der österreichischen Arbeitsmarktverwaltung eingesetzt wird, um die Vermittlungschancen von Langzeitarbeitslosen zu berechnen. Um die Gelder für Ausbildungsmaßnahmen nutzbringend einzusetzen, sollen Arbeitslose nur dann solche Maßnahmen erhalten, wenn sie voraussichtlich dazu führen, wieder eine Arbeitsstelle zu erhalten. Kritiker der Software bemängeln, dass sie nicht den gesamten Quellcode einsehen durften und deshalb befürchten, die Software könne Frauen und ältere Menschen diskriminieren, indem sie ihnen geringere Chancen bescheinigt.

- Bei Strafverfahren in den USA wird eine Software namens Compas eingesetzt, um die Rückfallgefahr und Gewaltbereitschaft verurteilter Straftäter zu ermitteln. Davon abhängig legt der Richter Maßnahmen während und nach der Haftdauer fest, die das Rückfallrisiko vermindern sollen. Etliche Medienberichte, etwa (Ziegler 2017), deuten an, diese Software habe Einfluss auf die Haftdauer. Das ist falsch, wie in (Wisconsin 2016) ausführlich dargestellt. Dennoch verbleibt Kritik an der Software, weil sie etwa dunkelhäutigen Menschen schlechtere Prognosen ausstellt als anderen, und das, obwohl die Hautfarbe nicht zu den Eingabedaten gehört.

- Ein klassisches Fallbeispiel ist die maschinelle Prüfung der Bonität von Antragstellern für einen Kredit. Die vorherrschende Befürchtung lautet, diese Systeme würden etwa die Kommunikationsplattformen im Internet durchsuchen und Leuten mit auffälligem Verhalten den Kredit verweigern. (O'Neil 2016, 198..199) äußert eine originellere Kritik. Sie lehnt es ab, etwa die Wohnanschrift der Antragsteller als Indikator für die Bonität zu verwenden, und das, obwohl sie erfahrungsgemäß ein guter Indikator ist. Sie bezeichnet es als ungerecht, einen Menschen nach dem Verhalten anderer zu beurteilen, nur weil diese ähnliche Eigenschaften haben wie er selbst. Gerecht sei ausschließlich, einen Menschen nach seinem eigenen Verhalten zu beurteilen.

Ethik, Moral und Software-Engineering

Kant hat die Philosophie in vier Teilgebiete eingeteilt und jedes mit einer Leitfrage versehen. Die Ethik ist eines davon, ihre Leitfrage lautet:

Was soll ich tun?

Sie enthält zwei entscheidende Wörtchen:

- *tun:* Ethik ist eine Handlungswissenschaft. Es gibt eine Vielzahl von Formaten, die zu unserem Thema erklären, was wir befürchten oder uns wünschen. Der Job des Ethikers ist aber erst dann erledigt, wenn klar ist, was getan werden muss, um einen erwünschten Zustand zu erreichen.

- *ich:* Gemeint ist hier natürlich nicht eine bestimmte Person, vielmehr eine Rolle, die Gesamtheit aller Personen mit einer bestimmten Aufgabe, Verantwortung oder in einer bestimmten Situation.

Hier ist schon sprachlich die Nähe zum Software-Engineering zu erkennen, denn auch dieses gibt Antworten zu Fragen dieser Form. Wie kommt ein Requirements-Engineer zu einer guten Vorgabe? Wie verständigen sich Tester, Entwickler und Management über die Folgen eines fehlgeschlagenen Tests?

Eine **Moral** ist eine Sammlung von Handlungsurteilen, zustimmenden oder ablehnenden. **Ethik** ist die Wissenschaft über Moral; sie verhält sich zur Moral also ähnlich wie die Physik zum Ingenieurwesen.

Tätigkeiten und Rollen bei der Softwareentwicklung

Meine Überlegungen sind unabhängig von der Aufbau- und Ablauforganisation des Prozesses. Fast überall wird zwar behauptet, agil vorzugehen; man verwendet Begriffe wie Sprint, User Story und Product Owner. Die konkrete Ausformung variiert jedoch beträchtlich. Die folgende Aufstellung ist deshalb keine neue Entwicklungsmethode, sondern eine Abstraktion über alle solche Methoden *(siehe Abbildung 1)*. Die genannten Tätigkeiten müssen in jedem Fall ausgeführt werden, mal mehr oder weniger häufig oder intensiv. Wer immer gerade eine solche Tätigkeit ausführt, ist Träger der bei ihr genannten Rolle.

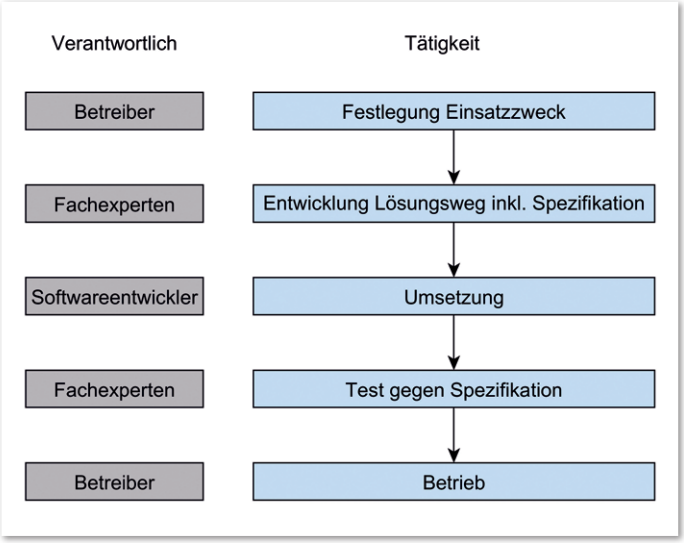


Abbildung 1: Rollen und Tätigkeiten bei der Softwareentwicklung (© Thomas Matzner)

Die erste entscheidende Frage beim Einsatz von Informatiksystemen ist ihr **Einsatzzweck**. Wer immer entscheidet, für einen bestimmten Zweck ein bestimmtes Verfahren einzusetzen, mit oder ohne Hilfe von Maschinen, ist **Betreiber** dieses Verfahrens. Oft ist dies die Leitung ei-

nes Unternehmens, einer Behörde oder einer sonstigen Organisation, aber nicht immer. Richter genießen eine hohe Unabhängigkeit. Wenn also etwa ein Richter entscheidet, zur Klärung eines bestimmten Sachverhalts einen Sachverständigen, eine Glaskugel oder eben eine Software einzusetzen, ist er in der Betreiberrolle.

Der Einsatzzweck soll möglichst konkret und nachprüfbar angegeben werden, also nicht nur „verbesserte Zuteilung von Fördermaßnahmen", sondern etwa in die Richtung: „Erhöhung der Vermittlungsquote geförderter Arbeitsloser von derzeit 60 auf 80 Prozent." Doch auch ein präziser Einsatzzweck lässt sich oft nicht direkt in Software umsetzen. Ein Softwareentwickler, der diese Zielvorgabe bekommt, würde zu Recht einwenden, keinen Sachverstand zu besitzen, um die Vermittlungschancen von Arbeitslosen zu beurteilen. Er würde eine **Spezifikation** fordern, die von **Fachexperten** auf der Fachdomäne erarbeitet wird, der der Einsatzzweck angehört. Genau betrachtet, braucht man häufig nicht nur diese Spezifikation, die das Verhalten der Software bestimmt. Das **Verfahren**, um den Einsatzzweck zu erreichen, kann neben maschinellen auch menschliche Tätigkeiten enthalten; die Beschreibung des Verfahrens, die auch die Spezifikation enthält, nenne ich den **Lösungsweg**.

Über die Umsetzung durch Softwareentwickler muss ich nicht viele Worte verlieren. Wichtig ist, dass der **Test gegen die Spezifikation**, hinreichend gründlich durchgeführt, Mängel in der Umsetzung erkennen lässt. Da der Betreiber, wie oben gefordert, die Verantwortung für den Einsatz des Verfahrens hat, hat er das Recht, aber auch die Verpflichtung, auf Basis des Testergebnisses zu entscheiden, ob er den Einsatz verantworten kann.

Einsatzzweck vs. Verfahren

Ein grundlegendes Prinzip lässt sich jetzt schon erkennen. Die moralische Bewertung des Einsatzzwecks lässt sich von der für das eingesetzte Verfahren trennen. Fragen nach den Chancen von Arbeitslosen, der Gefährlichkeit von Straftätern und der Kreditwürdigkeit wurden auch schon gestellt, bevor es Computer gab. Viele Formate zu unserem Thema vermischen das. Sie äußern Befürchtungen über – absichtlich oder nicht – schlecht gemachte Software, um dann Kritik etwa am kapitalistischen Wirtschaften, der Strafjustiz oder dem Sozialstaat zu üben. Eine konstruktive Lösung kommt umgekehrt zustande: Solange wir keine Einigkeit über die moralische Zulässigkeit des Einsatzzwecks haben, brauchen wir über unterschiedliche Lösungswege gar nicht zu sprechen.

Die Verantwortung für den Einsatzzweck trägt der Betreiber, womöglich langfristig unterstützt durch staatliche und gesellschaftliche Organe, die gesetzliche oder sonstige Moralvorschriften setzen.

Die Verantwortung der anderen Beteiligten beschränkt sich darauf, den Einsatzzweck durch ein Verfahren möglichst unverfälscht zu unterstützen.

Konkret: Warum rationiert man überhaupt die Fördermittel für Arbeitslose und gibt nicht allen einfach das, was sie sich wünschen? Das ist das Grunddilemma jedes Sozialstaates, denn das, was er an die aktuell Bedürftigen ausschüttet, muss er den aktuell Erfolgreichen abnehmen. Derselbe Arbeitslose, dem heute eine Fortbildung verweigert wird, hat fünf Jahre zuvor über die hohe Beitragslast gestöhnt. Für dieses Problem gibt es keine Patentlösung; es zu lösen, ist jedenfalls nicht unsere Kompetenz als Softwareentwickler. Selbst, wenn das ganze Informatikpersonal rund um das Arbeitsministerium sich zusammentäte und eine ihm gerecht erscheinende Umverteilungsmethode implementieren würde, wäre das undemokratisch. Nicht die Softwareentwickler haben den Auftrag, politische Fragen zu entscheiden, sondern über freie Wahlen letztlich der Arbeitsminister. Wer sich bei uns über ungerechte Verteilung der Mittel beschwert, dem ist zu sagen, er soll nächstes Mal andere Politiker wählen.

Überall hört und liest man über die Allgegenwart von Informatiksystemen. Tatsächlich sind wir in vielen, wenn auch nicht allen, Lebensbereichen mit unseren Systemen *unterstützend* tätig. Gerade deshalb ist es wichtig, nicht den Anspruch zu erheben, in allen Fachdomänen *mitzubestimmen*. Ein Student erzählte mir von einem Professor, der in einem Vortrag den Spruch brachte: „Die Informatik frisst die Welt." Genau das darf nicht passieren, denn „die Informatik" ist im Sinn der Ethik kein Akteur, dem man Verantwortung zuschreiben, Regeln auflegen und Verstöße sanktionieren kann. Es ist wichtig, die Betreiber der Systeme immer wieder darauf hinzuweisen, dass sie zwar nervtötende Rechenschritte an die Maschine abgeben können, nicht jedoch ihre Verantwortung und das Verständnis für ihre Domäne, das sie brauchen, um sie wahrzunehmen.

Voraussage vs. Entscheidung

Die eingangs vorgestellten Fälle decken natürlich nicht das gesamte Feld von Informatiksystemen ab, aber einen Ausschnitt, der offensichtlich besondere Besorgnis erregt. Sie haben ihre Grundstruktur gemeinsam: Sie bestehen aus einem Voraussage- und einem darauffolgenden Entscheidungsschritt. Auf die Voraussage (Wird ein geförderter Arbeitsloser wieder einen Job finden? Ein Straftäter rückfällig werden? Ein Kreditnehmer den Kredit vollständig bedienen?) folgt eine Entscheidung. Wir kennen das Entwurfsprinzip „divide and conquer", um schwer überschaubare Probleme auf handhabbare Größe herunterzubrechen. Trennung von Einsatzzweck und Verfahren war ein solcher Schritt, Trennung von Voraussage und Entscheidung ist der nächste. Welche Erleichterung bringt er uns?

In vielen Fällen ist der Voraussageschritt moralisch unkritisch, jedoch fachlich schwierig.

In vielen Fällen ist der Entscheidungsschritt moralisch kritisch, jedoch fachlich einfach.

Sehen wir uns die Kreditentscheidung an: Eine möglichst zutreffende Voraussage nützt bei vernunftorientierter Betrachtung beiden Parteien. Dem Kreditgeber sowieso, der gute von schlechten Risiken trennen kann. Aber auch für den Kreditnehmer ist eine negative Voraussage, vorausgesetzt sie ist zuverlässig, zwar momentan unangenehm, aber in Summe nützlich, bewahrt sie ihn doch womöglich von den unangenehmen Folgen der Überschuldung. Dass die Voraussage jedoch fachlich und in der Folge auch in der technischen Umsetzung schwierig ist, dürfte auf der Hand liegen. Die rein finanziellen Voraussetzungen, etwa Vermögen, Einkünfte, wirtschaftliche Lage des Berufsstandes, des Unternehmens und der Branche des Kreditnehmers, mögen noch fassbar sein. Aber wie sieht es mit menschlichen Risiken aus, mit Krankheiten, Auseinanderbrechen von Familien und sonstigen Schicksalsereignissen? Der Umgang mit solchen Unsicherheiten wird uns noch beschäftigen müssen.

Nun zum Entscheidungsschritt. Würde der Voraussageschritt in jedem Einzelfall ein klares Ja-Nein-Ergebnis liefern, wäre er trivial. Das können wir aber nicht erwarten, allenfalls eine Wahrscheinlichkeit für das jeweils erwünschte Ereignis. Der Banker könnte es sich nun einfach machen und eine betriebswirtschaftliche Rechnung darüber anstellen, wie hoch die Wahrscheinlichkeit auf Rückzahlung sein muss, um in Summe noch ein gutes Geschäft zu machen. Soll er bei 80 Prozent den Kredit vergeben? In vier von fünf Fällen macht er satten Gewinn, auch in dem fünften Fall verliert er nicht alles, sondern nur einen Teil der Summe. Das mag sich für ihn noch lohnen, lässt allerdings die Risiken für den Kreditnehmer außer Acht. Dieser ist im Fall der Zahlungsunfähigkeit wahrscheinlich überschuldet und muss das Verfahren der Privatinsolvenz über sich ergehen lassen. Dies bringt mit sich, über Jahre hinweg von allen erwirtschafteten Einkünften nur das gesetzlich festgelegte Existenzminimum behalten zu dürfen und alles darüber hinaus an die Gläubiger zu geben. Dieses Existenzminimum ist so festgelegt, dass es gerade ausreicht, ein menschenwürdiges Leben zu führen. Ein leichtfertig vergebener Kredit bringt den Kreditnehmer also zu weitgehendem Verlust oder der Einschränkung seiner Menschenrechte. Diese Folge überschattet alle anderen, die keinen Menschenrechtscharakter haben; also ist geboten, ihn davor bestmöglich zu schützen und dafür auch das Gewinnstreben der Bank zu begrenzen.

Wie hoch soll nun die Wahrscheinlichkeit für korrekte Rückzahlung sein, um den Kredit verantworten zu können? 100 Prozent werden wir angesichts der aufgezählten unkalkulierbaren Risiken nicht erreichen. Genügen 80 oder sollten wir 90 fordern? Auf **Grenzziehungen** wie diese gibt auch die Ethikliteratur keine praktisch verwertbare Antwort.

Hier hatten wir es wenigstens nur mit *einem* Menschenrecht zu tun, also einem klaren Fokus der Interessen. Beim Straftäter ist es noch komplizierter. Nehmen wir einen gefährlichen Fall an, etwa Misshandlung von Kindern. Bei solchen Tätern tun sich nach Abbüßung der Strafe und Therapie auch Fachleute schwer, eine Prognose für die Rückfallgefahr abzugeben, die als Voraussetzung dafür dient, dem Täter wieder die volle Freiheit zu gewähren. 80 Prozent erscheinen hier zu niedrig, bedeutet das doch, in einem von fünf Fällen wieder ein Kind zu schädigen. Aber wieder müssen wir irgendeine Grenze ansetzen. Nehmen wir an, sie liege bei 99 Prozent. Dann bedeutet das, dass in der Täterkohorte mit nur 98 Prozent Erfolgswahrscheinlichkeit 98 von 100 Tätern ihrer Freiheit beraubt werden, obwohl sie inzwischen ungefährlich sind. Auch dies ist ein schwerwiegender Eingriff in die Menschenrechte und läuft der Idee des Rechtsstaates, dass eine Tat schlussendlich verbüßt sein kann, zuwider.

Diese Beispiele sollten auch illustriert haben, dass Erwägungen wie diese nicht Sache des Informatikpersonals sein können. Sie zeigen auch einen Unterschied zwischen Ethik und Software-Engineering, nachdem ich anfangs eine Gemeinsamkeit gezeigt hatte. Wenn man etwa Modellierung lehrt, kann man am Ende jeder Unterrichtseinheit die Teilnehmer mit einer mustergültigen Lösung nach Hause schicken, über die sich alle freuen können. Für viele alltägliche Aufgaben der Ethik gibt es keine solche Lösung, über die wir in Jubel ausbrechen können. In manchen Fällen lässt sich der Konflikt abmildern, indem man statt einer Ja-Nein-Entscheidung differenziertere Maßnahmen einsetzt. So gibt es für Straftäter mit unklarer Prognose nicht nur die Möglichkeit der lebenslangen Sicherungsverwahrung, sondern abgestufte Maßnahmen der Überwachung nach der Haftentlassung. Die Kreditentscheidung ist hier unerbittlich: Ist der Kredit einmal vergeben, gibt es für alle Beteiligten nur mehr geringe Spielräume, um von den dadurch gesetzten Regeln abzuweichen.

Ist diese moralisch schwierige Entscheidung, bei welchem Voraussageergebnis welche Folge eintreten soll, einmal getroffen, ist ihre Umsetzung denkbar einfach. Ob man den Vergleich mit einem Schwellwert durch einen Menschen oder eine Maschine ausführen lässt, ist unerheblich.

Gerechtigkeit

Dies ist ein sperriges Konzept, von so vielen teils logisch widersprüchlichen Forderungen durchdrungen, dass es kaum möglich ist, für eine Aufgabe eine nachweisbar gerechte Lösung zu finden. Sehen wir uns wieder das Kreditwesen an. Basis dafür ist unser Schuldrecht, das unter dem Stichwort der Privatinsolvenz die Risiken zwischen Kreditgeber und Kreditnehmer verteilt: Der Kreditnehmer erleidet während deren Dauer erhebliche Nachteile, der Kreditgeber verzichtet danach auf die noch bestehende Restforderung. Dies führt dazu, dass beide bestrebt sind, Ausfälle zu vermeiden.

Das ist ungerecht, denn es führt dazu, dass Reiche immer reicher, Arme zumindest schwieriger reich werden. Wer schon zwei Mietshäuser besitzt, kann der Bank diese als Sicherheit anbieten und hat satte Mieteinnahmen, erhält also leicht einen Kredit für ein drittes Mietshaus. Alle drei vererbt er seinen Kindern, die ein zumindest finanziell sorgloses Leben führen können. Wer kein Kapital und nur ein durchschnittliches Arbeitseinkommen hat, bekommt womöglich keinen Kredit, vererbt seinen Kindern nichts, und alle müssen für ihren Lebensunterhalt arbeiten. Dies zu ändern, würde gravierende Eingriffe in das Schuldrecht erfordern, für die kaum ein praktikabler Ansatz erkennbar ist. Jede Entlastung einer der beiden Parteien würde zu einer Belastung der anderen führen und deren Bereitschaft mindern, das Geschäft einzugehen. Ein Teufelskreis.

Doch der anfangs zitierte Einwand gegen die Ermittlung von Indikatoren für die Kreditwürdigkeit zielte nicht auf diese grundsätzliche Ungerechtigkeit, sondern auf das Verfahren im konkreten Einzelfall. Was kennzeichnet eine gerechte Bonitätsprüfung?

Bei aller Sperrigkeit gibt es eine einfache Regel für gerechtes Handeln, nämlich Gleiches gleich zu behandeln. Wird einem rückzahlungsfähigen Kunden der Kredit verwehrt, ist das ungerecht, denn ihm wird im Gegensatz zu den anderen eine Chance verwehrt. Wird er einem nicht rückzahlungsfähigen gegeben, ist das auch ungerecht, denn er wird im Gegensatz zu den anderen nicht vor der Überschuldung geschützt.

Wir sehen also, dass zumindest in den Fällen, in denen kein Interessenkonflikt besteht, das gerechteste Verfahren dasjenige ist, das möglichst wenig Fehler macht. Das zweckmäßigste Verfahren ist gleichzeitig das gerechteste. Verfahren, die keine Trefferrate von 100 Prozent erzielen können, sind deshalb niemals vollständig gerecht, aber je besser die Quote, desto gerechter ist auch das Verfahren. Man beachte, dass für diesen Schluss kein Einblick in die inneren Abläufe des Verfahrens nötig war, sondern lediglich die Betrachtung seiner Außenwirkung, was uns zum nächsten Thema bringt.

Innen- vs. Außenwirkung, Bedeutung des Testens

Wenn ich Texte lese, in denen vor den Gefahren eines Informatiksystems gewarnt wird, weil man nicht weiß, welche schädlichen Wirkungen es entfaltet, frage ich mich immer: Warum hat niemand dem Autor erklärt, dass man diese Systeme testen kann und muss? Oder, wenn er es weiß, warum erklärt er es nicht seinen Lesern als effektives Mittel, die Befürchtungen zu entkräften?

Eine Einschränkung vorweg: Natürlich kann der Bankkunde, der Arbeitslose oder Straffällige ein Misstrauen gegenüber der Großorganisation empfinden, der er gegenübersteht. Er hat nicht die Möglichkeit, die dort eingesetzte Software systematischen Tests zu unterziehen; bestenfalls kennt er einen Testfall, seinen eigenen, den er nicht mit anderen vergleichen und etwa auf Gerechtigkeit prüfen kann. Dies ist jedoch keine Spezialität von Informatiksystemen, sondern der Moralregeln für die Wahrung der Grenzen zwischen Personen, seien es natürliche oder juristische. Auch bei rein menschlicher Abwicklung hätten die Genannten keinen Einblick in die Abläufe bei Bank, Arbeitsamt oder Gericht gehabt. Am transparentesten ist noch das Gericht, das öffentlich verhandelt und schriftlich begründet. Aber wie ist der Richter zu der Überzeugung gekommen, der Angeklagte sei weiterhin gefährlich und nach Entlassung mit einer elektronischen Fußfessel zu überwachen? Vielleicht hat er selbst gründlich nachgedacht, vielleicht seine Frau oder seinen alten Hochschullehrer angerufen, vielleicht auch eine Software rechnen lassen – wir werden es nicht erfahren. Wenn wir also von Tests sprechen, setzt das voraus, dass der Betreiber selbst ein Interesse an einem guten Ergebnis hat oder dass, wie bei Jahresabschlüssen oder bestimmten technischen Anlagen schon üblich, eine Prüfinstanz stellvertretend für den Rest der Bevölkerung die Abläufe bewertet.

Bei der Aufstellung einer Informatikethik benutze ich vorwiegend die Werkzeuge der **konsequentialistischen Ethik**, die Handlungs*folgen* bewertet, etwa im Gegensatz zu Tugend- oder Gesinnungsethik. Die Konzentration auf Folgen macht deutlich, dass es nicht auf die inneren Abläufe der Handlung ankommt, sofern sie die erwünschten Folgen erzeugt. Also kommt es innerhalb des Verfahrens auch nicht auf den inneren Aufbau und Abläufe der Software an, sondern auf die Folgen von deren Ausführung, also auf die Außensicht. Genau diese können wir durch Testen ermitteln. Damit erreichen wir zwar nicht die Sicherheit eines mathematischen Beweises, können jedoch den Test so intensiv gestalten, wie es der Kritikalität der Anwendung angemessen ist.

> *Die Kritikalität eines Einsatzzwecks misst sich an dem maximal möglichen immateriellen und materiellen Schaden, der bei seiner Verfehlung eintreten kann.*
>
> *Die Intensität des Tests einer Software muss der Kritikalität des Einsatzzwecks angemessen sein. Verantwortlich hierfür ist der Betreiber, der das Testergebnis für die Einsatzentscheidung braucht.*

Die Kritiker der Arbeitsamts-Software wollten sich erst mit Einblick in den Quellcode beruhigen lassen. Es gibt mancherlei Gründe, Quellcode offenzulegen, allerdings für manche Softwarehersteller auch Gründe dagegen. Für die Überprüfung der moralischen Integrität der Software ist der Test zu bevorzugen:

- Die Überprüfung des Quellcodes müsste durch eine von den ursprünglichen Entwicklern unabhängige Instanz geschehen, da die Entwickler Urheber des moralischen Problems sein könnten. Da in diesem Fall das Fehlverhalten gut versteckt sein könnte, muss die Prüfinstanz eine ihr unbekannte Software in ihrer Gänze überprüfen, was hohe Aufwände verursacht.
- Beim Lesen von Programmcode unterliegen wir kognitiven Verzerrungen, die uns hinsichtlich der Funktionsweise in die Irre führen. Gerade deshalb müssen wir ja testen.
- Die Ergebnisse eines Tests sprechen die Sprache des Betreibers, der die Einsatzentscheidung treffen muss.

Illustrieren wir das an der Frage, ob die Arbeitsamts-Software Frauen benachteiligt. Will man das am Code überprüfen, muss man den gesamten Code inspizieren. Man darf sich nicht darauf verlassen, dass eine eventuelle Ungleichbehandlung von Frauen in dem zentralen Modul zur Überprüfung der Chancen lokalisiert ist, auch nicht darauf, dass die entsprechenden Datenfelder an der Stelle semantisch korrekt mit „Geschlecht" oder Ähnlichem benannt sind. Also ist zeitraubende Detektivarbeit am gesamten Code angesagt. Ist nichts gefunden, könnte man es immer noch übersehen haben. Ist etwas gefunden, kann immer noch unklar sein, wie häufig sich der Unterschied auswirkt und ob er tatsächlich unerwünschte Ergebnisse liefert.

Ein Test hingegen kann folgendermaßen ablaufen: Man lässt zuerst die Eingabedaten für die Arbeitslosen der letzten zwei Jahre durchlaufen und hält die Ergebnisse fest. Nun ändert man wahllos in den Eingabedaten das Geschlecht und zur Sicherheit den Vornamen, falls da eine Plausibilitätsprüfung zuschlägt. Man lässt die Daten wieder durchlaufen. Ist das Ergebnis unverändert, hat man zwar keine absolute Sicherheit, kann aber sagen, dass immerhin in den letzten zwei Jahren keine Diskriminierung vorgekommen wäre. Gibt es Abweichungen, kann der Betreiber sich die Fälle zeigen lassen und entscheiden, ob sie sinnvoll sind. Wenn etwa Frauen seltener zu Straßenbauarbeiterinnen umgeschult werden sollen als Männer, kann der Arbeitsminister damit vermutlich leben. Wenn aber Frauen systematisch zu schlechter bezahlten Jobs umgeschult werden, ist das ein Alarmsignal.

Ist der Lösungsweg korrekt?

Ein Test gegen die Spezifikation kann Fehler in ihr nicht aufdecken. Wie sicher können wir uns des Lösungswegs und der in ihm enthaltenen Spezifikation sein? Das hängt von der Fachdomäne ab, der der Einsatzzweck angehört *(siehe Abbildung 2)*.

Es gibt Fachdomänen mit nachweisbar gültigen Regeln, also solchen, die nicht nur von einigen Personen für richtig gehalten werden, sondern die intersubjektiv beweisbar sind. Dazu gehören die exakten Wissenschaften und darauf aufbauend technische Produkte und Vorgänge. Es gibt auch Domänen, in denen die Menschen die Macht haben, solche Regeln als gültig festzulegen, etwa wirtschaftliche und administrative Transaktionen. Jemand bietet eine Ware an, jemand bestellt sie, sie wird geliefert, eine Rechnung geschrieben, Geld- und Warenfluss verbucht, Steuern darauf erhoben – das alles läuft nach vorbestimmten, zweifelsfrei gültigen Regeln.

In anderen Fachdomänen gibt es solche Regeln nicht. Dazu gehört menschliches Verhalten. Kein Wirtschafts-Nobelpreisträger, der ein mathematisches Wirtschaftsmodell entwickelt hat, kann mit Sicherheit den morgigen Kurs einer Aktie voraussagen, geschweige

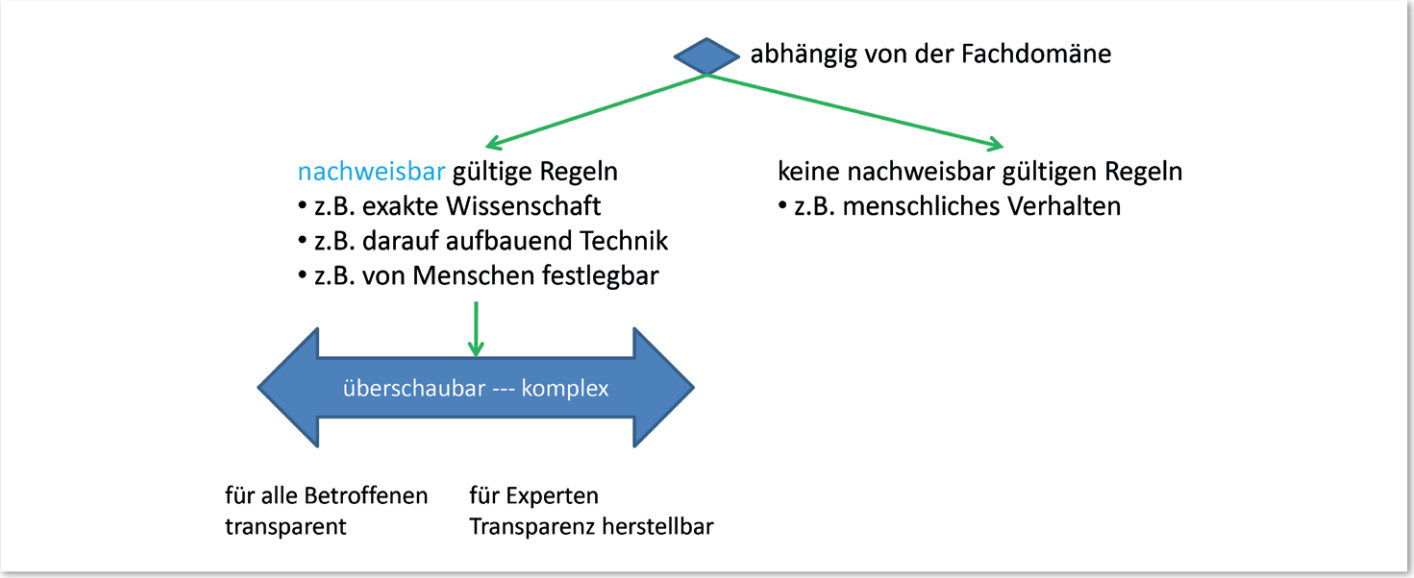


Abbildung 2: Klassifikation des Einsatzzwecks (© Thomas Matzner)

denn langfristige menschliche Entwicklungen wie in unseren Fallbeispielen. Auch die psychologische Literatur entwickelt zwar Handlungsmuster, aber nur für kurzfristig ablaufende Vorgänge wie etwa die Nahrungs- oder Partnersuche. Wir müssen uns damit abfinden, dass es für viele, auch alltägliche, Aufgaben keinen offensichtlich gültigen Lösungsweg gibt *(siehe Abbildung 3)*.

Hat man nachweisbar gültige Regeln, setzt man diese im Lösungsweg einfach um. Je nach Komplexität kann auch dies schwierig sein, so ist etwa die Rechnungsschreibung einfacher als die Ermittlung von Steuern.

Gibt es solche Regeln nicht, muss man sich anders behelfen. Man kann annähernde Regeln verwenden, bei der Kreditprüfung etwa Grenzen für Einkünfte und Startkapital festlegen. Will man aber etwa die gesundheitliche und familiäre Entwicklung des Kreditnehmers beurteilen, wird es schon schwieriger. Datengetriebene Verfahren, zu denen die meisten Techniken der künstlichen Intelligenz zählen, gehen einen anderen Weg: Sie ermitteln Korrelationen zwischen Ausgangsdaten und erwünschtem Ergebnis, etwa der tatsächlichen Rückzahlung des Kredits. Während man diese beiden Lösungswege ganz oder teilweise maschinell abwickeln kann, ist bei rein menschlicher Abwicklung noch ein dritter denkbar: das Vertrauen auf menschliche Intuition, bevorzugt durch Experten mit langjähriger einschlägiger Erfahrung.

Einsatzzwecke ohne nachweisbar gültige Regeln oder mit so komplexen Regeln, dass deren Umsetzung in einen Lösungsweg schwierig und fehlerträchtig ist, nenne ich **schwierig algorithmisierbar**.

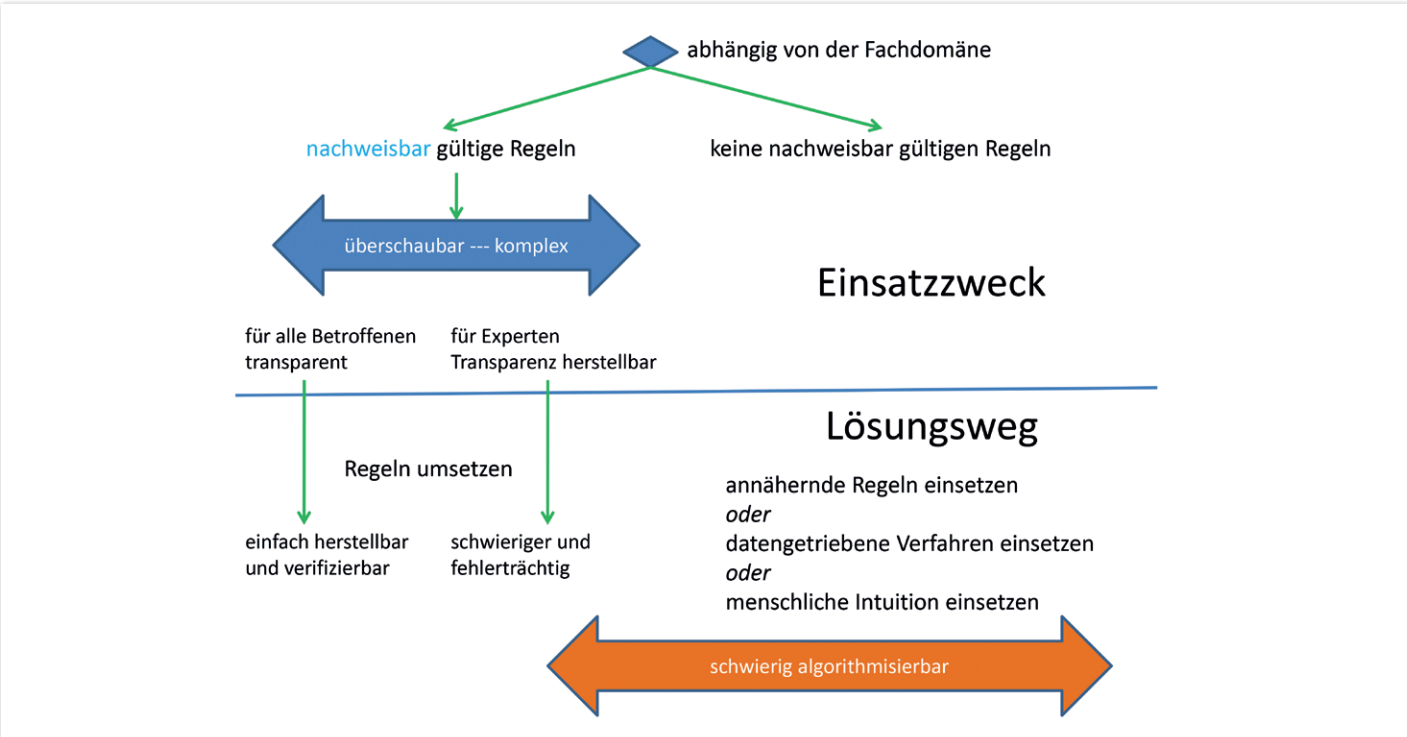


Abbildung 3: Mögliche Lösungswege abhängig vom Einsatzzweck (© Thomas Matzner)

Strenggenommen geht es hier nicht um die *Formulierung* eines Algorithmus, die man sich beliebig einfach machen kann, sondern um den Nachweis, dass Lösungsweg und Spezifikation tatsächlich den Einsatzzweck treffen.

Wie schon bei der Diskussion von Voraussagen erwähnt, ist in diesen Fällen ein *korrekter* Lösungsweg nicht zu erwarten, also einer, der in jedem Einzelfall das richtige Ergebnis liefert. Das Beste, was wir verlangen können, ist ein möglichst *verlässlicher* Lösungsweg.

> *Bei einem schwierig algorithmisierbaren Einsatzzweck ist von allen Lösungswegen der verlässlichste zu wählen, der die niedrigste Fehlerrate liefert.*

Das bringt uns zu der Frage, wie man diesen findet, wenn man seine Verlässlichkeit nicht direkt aus seiner Formulierung ableiten kann.

Test gegen die Wirklichkeit

Es gibt schon seit Längerem Informatiksysteme, die nicht nur gegen ihre Spezifikation getestet werden. Denken wir an vollautomatisch fahrende Kraftfahrzeuge. Es gibt sie schon seit Jahren auf unseren Straßen, aber nicht im Einsatz für normale Konsumenten. In ihnen sitzt ein Begleitfahrer, der das Verhalten des Fahrzeugs überwacht und im Notfall eingreifen kann. Wenn dieser Vorgang genügend lange durchgeführt wurde, um Vertrauen zu dem System haben zu können, kann es als praxisreif gelten. Dieses Vorgehen nenne ich **Test gegen die Wirklichkeit**, denn hier wird nicht gegen eine vorbestimmte Spezifikation auf Papier getestet, sondern gegen eine nicht immer vorbestimmbare Wirklichkeit *(siehe Abbildung 4)*.

Während also der Test gegen die Spezifikation Fehler bei der Umsetzung aufdeckt, leistet der Test gegen die Wirklichkeit dies für Lösungsweg und Spezifikation.

Der vorhin beschriebene Test auf Gender-Ungerechtigkeit war kein Test gegen eine Spezifikation, sondern gegen die Wirklichkeit: Wir hatten bei ihm kein Ergebnis fest vorgegeben, sondern erforscht, welche Wirkung eine angenommene Vertauschung der Geschlechter hätte.

Damit können wir die Forderung aus dem letzten Abschnitt konkretisieren:

> *Bei einem schwierig algorithmisierbaren Einsatzzweck ist durch Test gegen die Wirklichkeit zu ermitteln, welcher Lösungsweg die niedrigste Fehlerrate liefert.*

Diese Forderung ist zugegebenermaßen zu streng: Sie würde bedeuten, *alle* möglichen Lösungswege zu implementieren und gründlich gegen die Wirklichkeit zu testen. Das kann etwa eine einzelne Bank, wie reich sie auch sein möge, kaum leisten. Es müssten alle annähernden Regelsysteme, alle bisher entwickelten datengetriebenen Algorithmen und eine hinreichend große Anzahl von Menschen gegeneinander getestet werden. Was man aber in der Praxis verlangen kann, ist mindestens, für das derzeit bevorzugte Verfahren einen solchen Test durchzuführen sowie Verfahren, für die das mit wenig Aufwand möglich ist, etwa auf dem Markt angebotene fertige Software, damit zu vergleichen.

Mensch vs. Maschine

In manchen ethischen Leitlinien wird gefordert, dass bestimmte kritische Aufgaben, wie sie in unseren Fallstudien vorkommen, grundsätzlich oder auf Verlangen des Betroffenen von Menschen gelöst werden. Daraus könnte man schließen, dass Menschen diese Aufgaben zuverlässig besser lösen als Maschinen. Tatsächlich haben wir uns über Jahrtausende daran gewöhnt, dass Lehrer ihre Schüler beurteilen, Richter den Charakter der Prozessbeteiligten und Vorgesetzte die Leistung ihrer Mitarbeiter. Ist also die maschinelle Ausführung oder Unterstützung bei dieser Aufgabe nur eine Notlösung, um Kosten oder Zeit zu sparen?

(Kahneman 2011, 222..233) lässt uns daran zweifeln, dass menschliche Voraussagen von Ereignissen unter Unsicherheit der Goldstandard sind. Schon Mitte des vorigen Jahrhunderts, als noch kaum jemand an Computereinsatz dachte, war durch kontrollierte Experimente erwiesen, dass selbst eine einfache Scoring-Formel mindestens ebenso verlässliche Ergebnisse liefert wie intuitive Entscheidungen durch Experten. Diese Erkenntnisse laufen unserem Selbstbild, vor allem wenn wir berufs- und lebenserfahrene Experten sind, zuwider und werden selten in der Praxis thematisiert,

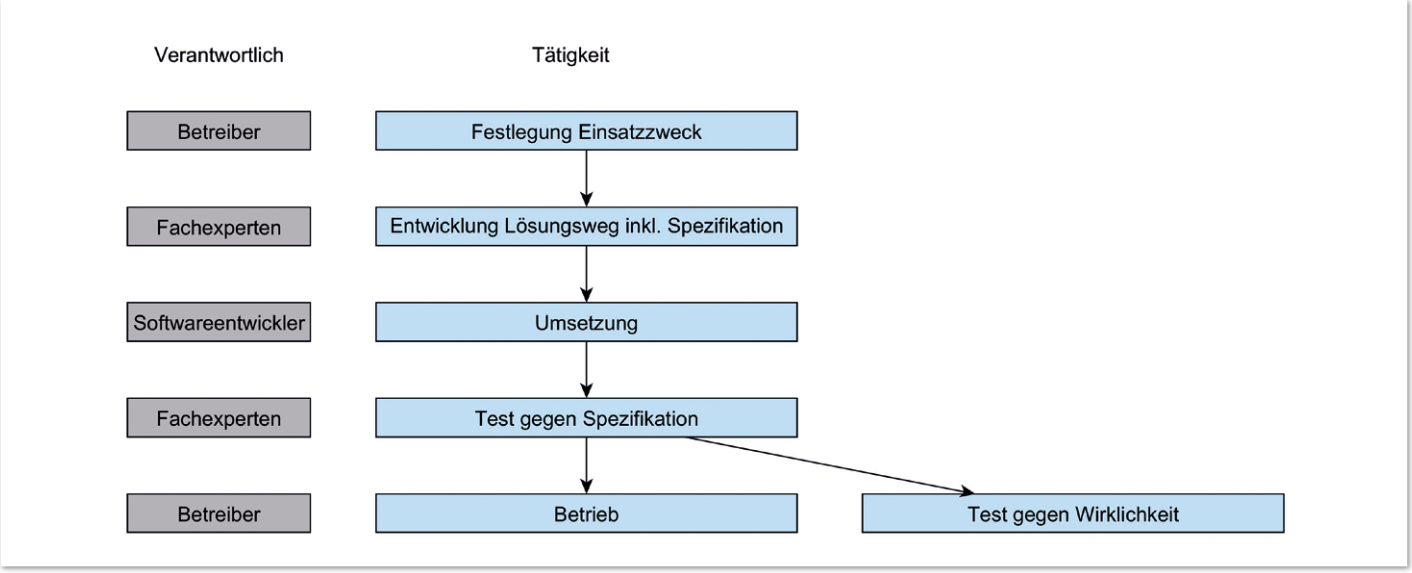


Abbildung 4: Softwareentwicklung mit Test gegen die Wirklichkeit (© Thomas Matzner)

obwohl sich das zitierte Buch gut verkauft hat. Als Ethiker habe ich hierzu nichts Eigenes beizutragen, muss aber die relevante Forschung zur Kenntnis nehmen und daraus schließen:

> *Bei einem Einsatzzweck, der Voraussagen unter Unsicherheit umfasst, darf nicht vorausgesetzt werden, dass ein rein durch Menschen ausgeführter Lösungsweg der verlässlichste ist.*

Das heißt nicht, dass der Mensch den Wettbewerb mit anderen Lösungswegen stets verliert, wohl aber, dass er sich ihm stellen muss, zumal es „den Menschen" gar nicht gibt, sondern viele unterschiedliche, deren Einschätzungen oft genug widersprüchlich sind. Die wichtigsten Schwächen der drei Lösungsweg-Typen für schwierig algorithmisierbare Aufgaben sind:

- Annähernde Regeln
  - Da sie per Voraussetzung nicht nachweisbar sind, entspringen sie menschlicher Intuition, siehe dort.
- Datengetriebene Verfahren
  - Korrelation bedeutet nicht Kausalität. Also liefert ein solches Verfahren auch bei häufiger Ausführung keine zusätzlichen *Einsichten* in die Mechanismen der Fachdomäne. Demnach ist es anfällig gegenüber Veränderungen in der Wirklichkeit, mangelhaften Trainingsdaten und anderen Versäumnissen.
- Menschliche Intuition
  - Die zugrunde liegende Datenbasis ist naturgemäß kleiner als bei den anderen Lösungswegen, bei denen potenziell alle überhaupt verfügbaren Daten genutzt werden können.
  - Sowohl unsere Erinnerungen wie auch die Schlüsse, die wir aus ihnen ziehen, unterliegen vielfältigen kognitiven Verzerrungen. Davon handelt das soeben zitierte Buch von Kahneman.
  - Also sind die vielfach befürchteten Vorurteile vorwiegend bei intuitiven Voraussagen oder aus Intuition gewonnenen annähernden Regeln zu erwarten.

Testbarkeit gegen die Wirklichkeit

Der Test gegen die Wirklichkeit ist je nach Einsatzzweck unterschiedlich schwierig. Vergleichen wir dazu das vollautomatisch fahrende Auto mit der Kreditentscheidung. Das Autofahren ist immens schwierig algorithmisierbar, umfasst es doch die korrekte Interpretation des Verkehrsgeschehens in Echtzeit inklusive Voraussagen über die Absichten der Verkehrsteilnehmer. Es ist aber relativ leicht gegen die Wirklichkeit testbar: Der Begleitfahrer kann, Kenntnis der Verkehrsregeln vorausgesetzt, in jeder Situation sofort sagen, ob sich das Auto korrekt verhalten hat.

Umgekehrt bei der Kreditentscheidung. Eine einfache Heuristik auf Basis der finanziellen Situation lässt sich in wenigen Tagen spezifizieren; damit hat man womöglich schon eine Lösung, die auch heute noch mancherorts produktiv eingesetzt wird. Aber wie gestaltet sich der Test gegen die Wirklichkeit? Wenn heute eine Entscheidung getroffen wird, ist erst in Jahren oder Jahrzehnten objektiv ermittelbar, ob sie richtig war.

Zu den Merkmalen Kritikalität und Algorithmisierbarkeit des Einsatzzwecks tritt also als drittes die **Testbarkeit gegen die Wirklichkeit**. Sie ist bei Voraussagen umso schwieriger, je länger der Voraussagezeitraum ist.

Aus diesem Grund ist in der vorangegangenen *Abbildung 4* der Test gegen die Wirklichkeit nicht einfach vor dem Betrieb, sondern parallel zu ihm eingezeichnet. Anders als der Test gegen die Spezifikation kann er nicht einfach vor Inbetriebnahme durchgeführt und dann abgehakt werden, sondern erfordert in schwierigen Fällen ständiges Nachprüfen. Hier einige zusätzliche Maßnahmen für solche schwierigen Fälle:

- In erster Linie denkt man natürlich bei langfristigen Voraussagen an die Verwendung historischer Daten. Man kann bei der Kreditentscheidung nicht abwarten, bis die Antwort für heute gewährte Kredite da ist, also geht man Jahre bis Jahrzehnte zurück. Man muss sich jedoch je nach Fachdomäne überlegen, wie gut historische Verhältnisse auf gegenwärtige und künftige übertragbar sind. So kann etwa eine Branche, die in den letzten Jahrzehnten glänzend florierte, zusammen mit den Regionen, in denen sie Arbeitsplätze bietet, zu guten Kreditverläufen führen. Wenn es der Branche künftig schlechter ergeht – man denke etwa an mögliche Verwerfungen im Zusammenhang mit der Energie- und Verkehrswende –, können historische Gewissheiten sich verflüchtigen.

- Deshalb sollte man während des Betriebs laufend überwachen, ob sich die Datenkonstellationen verändern, ob etwa bestimmte Muster mehr oder weniger häufig auftreten als bei den anfangs zugrunde gelegten Tests.

- Wenn mehrere Verfahren zur Verfügung stehen, etwa auch nur durch unterschiedliche Konfiguration desselben Grundverfahrens, kann es lohnen, diese parallel einzusetzen, um festzustellen, ob ein anderes als das aktuell entscheidende plötzlich Hinweise auf Veränderungen zeigt.

- Um all dies zu ermöglichen, müssen in den unterstützten Prozessen laufend Erfolgsdaten zuvor getroffener Entscheidungen eingespielt werden. Das ist derzeit nicht in allen Fällen selbstverständlich. Natürlich hat die Bank aus ihrem operativen Geschäft jederzeit vollständige Daten über die Bedienung von Krediten. Aber hat jeder Richter solche Daten über die Rückfallquote von Straftätern mit bestimmten Merkmalen zur Verfügung? Bei der Erfassung und Verarbeitung solcher Daten könnten Einwände hinsichtlich des Datenschutzes gemacht werden. Die Darstellung des moralischen Dilemmas beim Umgang mit therapierten Missbrauchstätern hat jedoch den Blick auf Menschenrechte geweitet. Es gibt nicht das eine Recht auf Schutz privater Daten, dem alle anderen Interessen unterzuordnen sind. Zu den Grundaufgaben der Angewandten Ethik gehört das ständige Austarieren solcher Werte ohne vorgefertigte Präferenzen.

Ein moralisches Dilemma bei schwierig algorithmisier- und testbaren Einsatzzwecken entsteht durch selbsterfüllende Prophezeiungen. Ist ein Kredit verweigert oder ein Straftäter unter dauerhafte Überwachung gestellt worden, kann sich der Akteur immer in der wohligen Überzeugung wiegen, großen Schaden verhindert zu haben. Ein objektives Feedback aus der Wirklichkeit kann nicht stattfinden, da ja sowohl das befürchtete Ereignis als auch sein Gegenteil unmöglich gemacht wurden. Also müsste man, um das Verfahren objektiv zu beurteilen, auch in einem gewissen Anteil der Fälle bewusst falsch entscheiden. Das ist aber

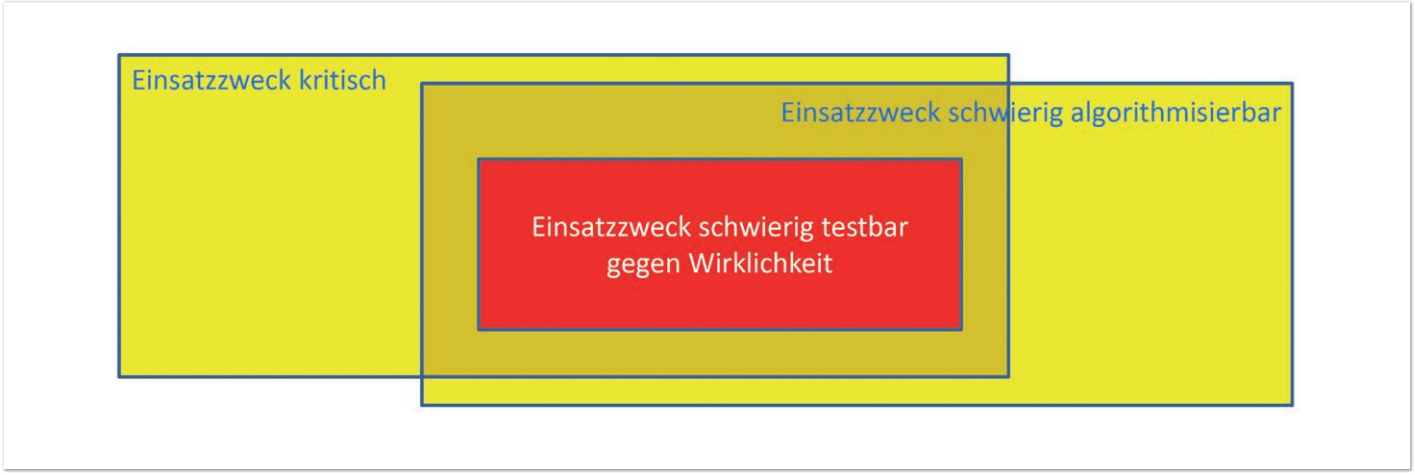


Abbildung 5: Stufen der Gefährlichkeit des Einsatzzwecks (© Thomas Matzner)

moralisch hochproblematisch, wie in dem Abschnitt über Konsequenzen fehlerhafter Kreditvergabe oder Freilassung gefährlicher Personen gezeigt.

Angst vor KI ist fehlgeleitet

Eine Vielzahl der Formate, die moralische Fragen im Zusammenhang mit IT behandeln, behandeln als Gipfel der Problematik die künstliche Intelligenz (KI). Ganze Institute gibt es für Erforschung der ethischen Konsequenzen aus KI. Wie komme ich dazu, die Konzentration auf KI in diesem Zusammenhang als fehlgeleitet zu betrachten?

Betrachten wir den Einsatzzweck eines Verfahrens, in dem KI eingesetzt wird. Ist er unkritisch, haben wir kein Problem. Das trifft etwa für KI-gestütztes Schachspiel, aber auch für außerspielerische Einsätze, etwa Predictive Maintenance zu, mit der vorausbestimmt wird, wann bestimmte technische Einrichtungen instandgesetzt werden sollen, um ihrem Ausfall zuvorzukommen. Diese ist zwar nicht vollkommen unkritisch, aber immerhin geht es bei ihr nicht um menschliche Schicksale, wodurch unsere Angst sich in Grenzen hält.

Ist der Einsatzzweck leicht algorithmisierbar, haben wir es mit Regeln zu tun, die zuvor klar bekannt sind und über eine Spezifikation in das Informatiksystem überführt werden. Hier gibt es keinen Grund für den Einsatz von KI, weshalb sie auch etwa bei naturwissenschaftlich-technischen Berechnungen oder Bestellabwicklungssystemen keine Rolle spielt.

Ist der Einsatzzweck schwierig algorithmisierbar und leicht gegen die Wirklichkeit zu testen, können wir jeden Lösungsweg, auch solche auf Basis von KI, durch hinreichende Tests gegen die Wirklichkeit absichern. Hierzu gehört etwa die Gesichtserkennung. Jeder Betreiber, sei es eine Privatperson mit ihrer Fotosammlung oder der Geheimdienst mit der Fahndungsdatei, kann sich vergewissern, dass das System keine oder hinreichend wenig Fehler macht, ohne dabei überhaupt wissen zu müssen, ob KI darin steckt oder nicht.

Erst bei kritischem, schwierig algorithmisierbarem und schwierig gegen die Wirklichkeit testbarem Einsatzzweck wird es mulmig. Hier können wir bei keinem Lösungsweg sicher sein, ob er hinreichend verlässlich ist, ob er schon heute oder, wie oben dargestellt, in der Zukunft anfängt, zu viele falsche Resultate zu produzieren. Gäbe es einen Lösungsweg, etwa außerhalb der KI, zu dem man hinreichendes Vertrauen hat, könnte man ihn als Grundlage für einen Test gegen die Wirklichkeit jedes anderen Verfahrens verwenden, wodurch der ganze Einsatzzweck nicht mehr schwierig

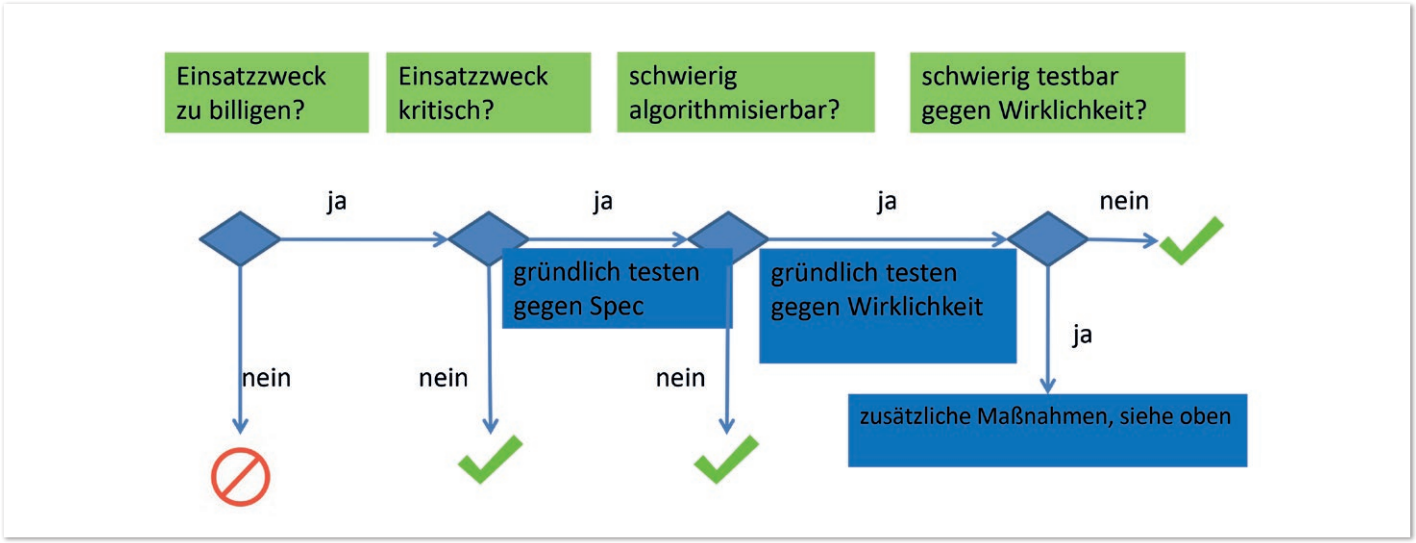


Abbildung 6: Fahrplan für die Gestaltung moralisch integrer Verfahren (© Thomas Matzner)

gegen die Wirklichkeit zu testen wäre. Vielmehr sind jedoch alle drei Kriterien gerade nicht abhängig vom gewählten Lösungsweg, sondern vom Einsatzzweck selbst.

So beschreibt etwa (O'Neil 2016, 12..22) den Fall eines tatsächlich wohl fehlgeleiteten Softwareeinsatzes zur Beurteilung der Berufsleistung von Lehrern. Genaues Studium des Textes legt jedoch den Verdacht nahe, dass dort keineswegs, wie vom Hersteller der Software behauptet, KI und Big Data am Werk waren, sondern ein recht einfacher Vergleich der Schülerleistungen am Ende des vorigen und des jetzigen Schuljahrs, den jeder Laie mit dem Taschenrechner ausführen könnte. Weshalb sollten wir vor einem solchen zu einfachen Verfahren, das Schaden anrichtet, weniger Angst haben als vor einem komplizierten?

Die Angst vor KI habe ich als fehlgeleitet, nicht als unbegründet, bezeichnet. Natürlich kann man vor Fehlfunktion eines Systems berechtigte Angst haben. Allerdings entstammen alle drei Kriterien, die eine solche Angst rechtfertigen, einem Zusammentreffen dreier ungünstiger Merkmale des Einsatzzwecks, nicht den zu seiner Verfolgung eingesetzten Techniken. KI einzusetzen, ist erst bei schwierig algorithmisierbaren Einsatzzwecken sinnvoll – die Gefahr entstammt jedoch der Aufgabe und würde nicht verschwinden, wenn man etwa den KI-Einsatz verbieten würde *(siehe Abbildung 5)*.

Dazu kommt, dass bei unseren Fallstudien, wie in vielen vergleichbaren Fällen, eine Instanz über den Betroffenen entscheidet, die größer ist, als mächtiger wahrgenommen wird und aus den beschriebenen Gründen für ihn nicht transparent ist. Auch diese Umstände sind unabhängig von der eingesetzten Technik.

Zusammenfassung: Fahrplan für die Gestaltung moralisch integrer Verfahren

Der folgende Fahrplan *(siehe Abbildung 6)* dürfte nach dem Lesen dieses Textes ohne weitere Erläuterung verständlich sein.

Literaturverzeichnis

- Kahneman, Daniel. *Thinking, Fast and Slow*. London: Penguin, 2011.
- Matzner, Thomas. *Informatikethik*. 2. Auflage. Norderstedt: BoD, 2020.
- Nezik, Ann-Kathrin. „Wenn Maschinen kalt entscheiden." ZEIT, 10 2019: 21.
- O'Neil, Cathy. *Angriff der Algorithmen. Wie sie Wahlen manipulieren, Berufschancen zerstören und unsere Gesundheit gefährden*. Hanser, 2016.
- Wisconsin, Supreme Court of. „State vs Loomis." 13. 7 2016. *https://bit.ly/2CgErVV* (Zugriff am 9. 7 2022).
- Ziegler, Peter-Michael. „Im Namen des Algorithmus. Wenn Software Haftstrafen verhängt." c't, 12 2017: 68.

Thomas Matzner

tam@tamatzner.de

Thomas Matzner ist selbstständiger Diplom-Informatiker. Sein Hauptarbeitsgebiet ist die Konzeption von Informationssystemen. Er arbeitet in der Rolle des Requirements Engineers, Product Owners, Business Process Managers und Business Analysts.

Er unterrichtet Informatikethik im Rahmen eines von ihm aufgebauten Wahlpflichtfachs an der Technischen Hochschule Nürnberg Georg Simon Ohm.

JavaLand

21. – 23. MÄRZ 2023

im Phantasialand bei Köln

Die Konferenz der Java-Community

2.000+ JAVA-FANS

160+ SESSIONS

50+ AUSSTELLER

100 % AUSTAUSCH

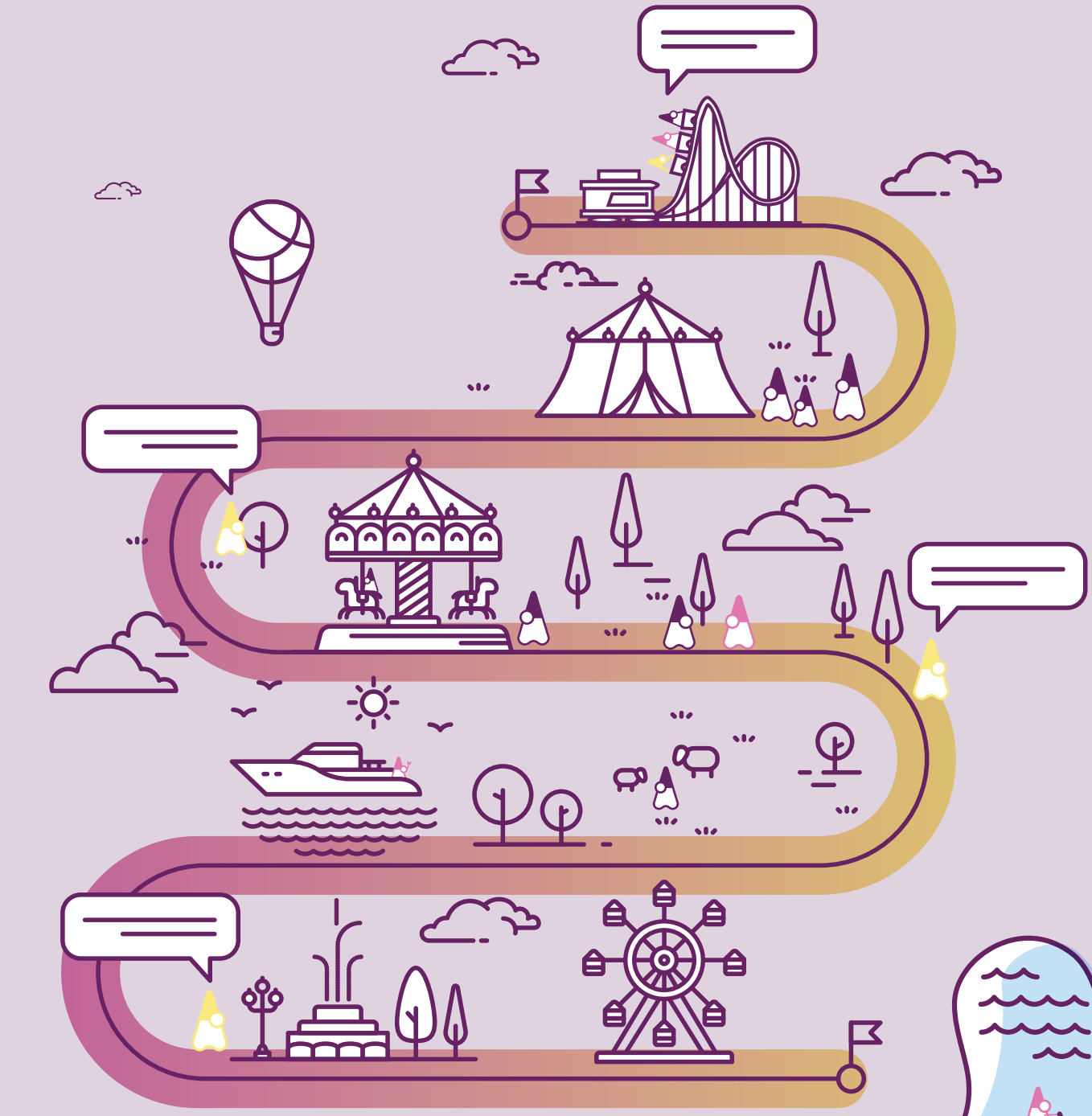
54 JAVA USERGROUPS

100 % SPASS

Präsentiert von: iJUG Verbund · Heise Medien · DOAG

www.javaland.eu

Programm jetzt online!



Veranstalter:

Mitglieder des iJUG

01 BED-Con e.V.
02 Clojure User Group Düsseldorf
03 DOAG e.V.
04 EuregJUG Maas-Rhine
05 JUG Augsburg
06 JUG Berlin-Brandenburg
07 JUG Bremen
08 JUG Bielefeld
09 JUG Bonn
10 JUG Darmstadt
11 JUG Deutschland e.V.
12 JUG Dortmund
13 JUG Düsseldorf rheinjug
14 JUG Erlangen-Nürnberg
15 JUG Freiburg
16 JUG Goldstadt
17 JUG Görlitz
18 JUG Hannover
19 JUG Hessen
20 JUG HH
21 JUG Ingolstadt e.V.
22 JUG Kaiserslautern
23 JUG Karlsruhe
24 JUG Köln
25 Kotlin User Group Düsseldorf
26 JUG Mainz
27 JUG Mannheim
28 JUG München
29 JUG Münster
30 JUG Oberland
31 JUG Ostfalen
32 JUG Paderborn
33 JUG Saxony
34 JUG Stuttgart e.V.
35 JUG Switzerland
36 JSUG
37 Lightweight JUG München
38 SOUG e.V.
39 SUG Deutschland e.V.
40 JUG Thüringen
41 JUG Saarland

iJUG Verbund
www.ijug.eu

Impressum

Java aktuell wird vom Interessenverbund der Java User Groups e.V. (iJUG) (Tempelhofer Weg 64, 12347 Berlin, *www.ijug.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. Vielmehr vertritt es die Interessen der Anwender an den Themen rund um die Java-Produkte, fördert den Wissensaustausch zwischen den Lesern und informiert über neue Produkte und Technologien.

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.

Die DOAG Deutsche ORACLE-Anwendergruppe e.V. hält 100 Prozent der Stammeinlage der DOAG Dienstleistungen GmbH. Die DOAG Deutsche ORACLE-Anwendergruppe e.V. wird gesetzlich durch den Vorstand vertreten; Vorsitzender: Björn Bröhl. Die DOAG Deutsche ORACLE-Anwendergruppe 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@ijug.eu

Redaktionsbeirat:
Andreas Badelt, Melanie Andrisek, Marcus Fihlon, Markus Karg, Manuel Mauky, Bernd Müller, Benjamin Nothdurft, Daniel van Ross, André Sept

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

Bildnachweis:
Titel: Bild © vectorpouch
https://freepik.com
S. 10 + 11: Bild © nexusplexus
https://123rf.com
S. 16 + 17: Bild © vectorjuice
https://freepik.com
S. 24: Bild © Harryarts
https://freepik.com
S. 27: Bild © lemonos
https://123rf.com
S. 32 + 33: Bild © StockVector
https://stock.adobe.com
S. 38: Bild © Bro Vector
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