Ausgabe 01/2018

Javaaktuell

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

Javaaktuell

iJUG Verbund
www.ijug.eu

Die Programmiersprache Go
Einfaches Deployment, hohe Performance

Continous Delivery
Grundlagen des Jenkins-Pipeline-Plug-ins

Praktische Erfahrungen
Crypto 101 für eine sichere Verschlüsselung

Die Zukunft von Java EE

Inselbegabung gesucht.

Werden Sie Java-Entwickler bei Valtech

Digital Automotive

Die Valtech entwickelt mit und für namhafte Automobilmarken zukunftsweisende digitale Mobilitätsdienste. Dabei steht das Connected Car im Mittelpunkt. Wir entwickeln die Software der Backend-Server und die Online-Dienste.

Digital Business

Unsere Projekte dienen der digitalen Transformation. Sie reichen von komplexen eCommerce- und Customer-Engagement-Plattformen bis zu auf Kundenbedürfnisse zugeschnittene Softwarelösungen. Dabei verwenden wir agile Methoden und stecken auch unsere Kunden gerne mit dem agilen Mindset an.

Lust auf spannende Projekte?

Sie haben Interesse an einem der Bereiche mitzuarbeiten?
Come on board! jobs@valtech.de

valtech_
where **experiences** are **engineered**

www.valtech.de

Wir sehen uns auf der JavaLand 2018

Bald ist es wieder so weit: Am 13. März startet die vierte Ausgabe der JavaLand. Alle, die in irgendeiner Form mit Java zu tun haben, treffen sich im Phantasialand in Brühl. Weltweit anerkannte Experten geben ihr Wissen weiter, Java-Entwickler profitieren vom Erfahrungsaustausch und die Community diskutiert die neuesten Entwicklungen.

Gerade die Community hat viel zu bereden: Wie bewährt sich das neue Java 9 in der Praxis? Wie entwickelt sich Java EE unter der Schirmherrschaft der Eclipse Foundation weiter? Was sind die neuesten Trends? All das ist ständig Thema in der Community Hall.

Aufgrund der deutlich mehr Vortragseinreichungen als in den Jahren zuvor konnte das JavaLand-Programmkomitee bei seinem letzten Treffen in Berlin ein sehr interessantes und abwechslungsreiches Programm für die JavaLand 2018 zusammenstellen. Es ist online unter *„https://www.javaland.eu“* einsehbar.

Besonders gespannt bin ich auf die beiden Keynotes, in denen wieder alle Teilnehmer in einem riesigen Vortragsraum versammelt sind.

Bereits am Vortag treffen sich die Entwickler von morgen an gleicher Stelle auf der JavaLand4Kids. Dabei lernen Kinder im Alter von 8 bis 14 Jahren spielerisch das Programmieren kennen und erkunden in drei Gruppen und drei spannenden Workshops kreativ den Umgang mit Computern.

Nicht zuletzt sorgen die Attraktionen des Phantasialands, das Catering und die Band für einen schönen Community-Abend.

Ich freue mich auf ein Treffen in Rahmen der JavaLand 2018.

Ihr

Wolfgang Taschner
Chefredakteur Java aktuell

Inhalt

19

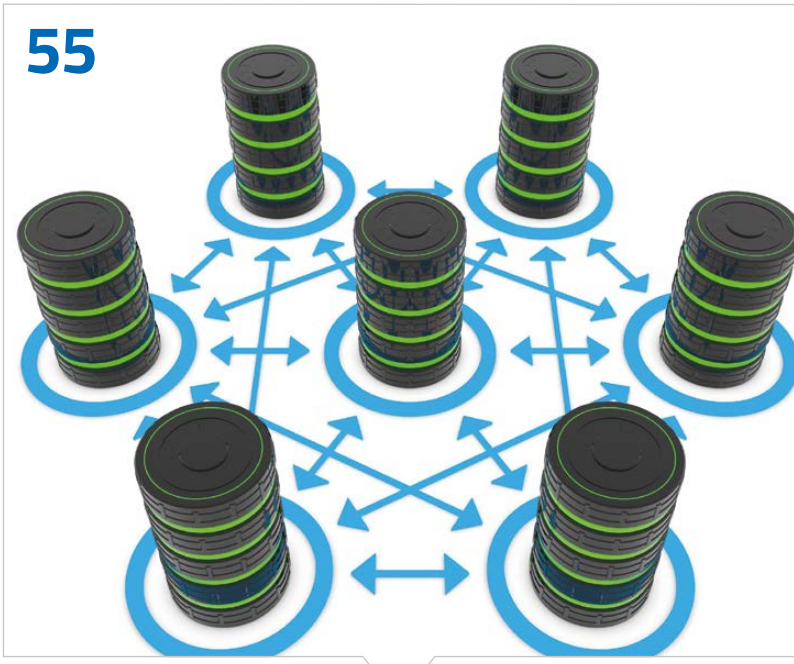
Jenkins

30

Go ist eine einfache, statisch getypte und kompilierte Programmiersprache mit hoher Lesbarkeit

So lässt sich eine Pipeline mithilfe des Jenkins-Pipeline-Plug-ins an zentraler Stelle als Code definieren

Ein gutes Software-Produkt benötigt neben seiner Funktionalität auch eine gute Schnittstelle zum Benutzer



Resilient Software Design Patterns sind ausgewählte Entwurfsmuster zur Erstellung einer Software

Das Java-Tagebuch

Andreas Badelt, stellv. Leiter der DOAG Java Community

Das Java-Tagebuch gibt einen Überblick über die wichtigsten Geschehnisse rund um Java – in komprimierter Form und chronologisch geordnet. Der vorliegende Teil widmet sich den Ereignissen im dritten Quartal 2017.

12. September 2017

And the winner is ... The Eclipse Foundation

Im August erst hat Oracle bekannt gegeben, dass sie Java EE gerne an eine Open Source Foundation übergeben würden. Wenn sie sich bei Oracle zu etwas entschlossen haben, verlieren sie keine Zeit: Nach Gesprächen mit IBM und Red Hat, den anderen großen Firmen hinter Java, sowie den Kandidaten verkündet David Delabassee, dass es die Eclipse Foundation sein soll. Ganz wichtiges Detail seines Posts: Alle von Oracle geführten Java-EE-Technologien sollen an Eclipse relizenziert werden – inklusive der Test Compatibility Kits (TCKs). Die TCKs werden endlich Open Source – ein Schritt, der auch die Apache Software Foundation freuen wird, selbst wenn sie nicht den Zuschlag erhalten hat. Namentlich erwähnt wird auch MicroProfile, die ja selbst Ende 2016 bei Eclipse untergeschlüpft sind („wir freuen uns auf die Zusammenarbeit"). Mal sehen, was auf der JavaOne zu dem Thema zu hören sein wird. Das Projekt soll jedenfalls so schnell wie möglich gestartet werden, wenn EE 8 final ist. Aber es dürfte auch eine ganze Weile dauern, bis die Übergabe abgeschlossen ist und das Projekt bei Eclipse Fahrt aufnimmt.

https://blogs.oracle.com/theaquarium/opening-up-java-ee

21. September 2017

Kein eigenständiges Oracle Java SE Embedded mehr

Oracle Java SE Embedded wird mit dem JDK 9 eingestellt. Mit der neuen Plattform-Modularisierung (Stichworte: „Jigsaw" und „JLink") sowie dem noch experimentellen „Ahead of Time Compiling" (AOT) werden die Bemühungen um ein Extra-JDK für kleine Geräte, insbesondere die Compact Profiles, überflüssig.

https://blogs.oracle.com/java-platform-group/convergence-of-oracle-java-se-embedded-with-oracle-jdk

21. September 2017

Java EE 8 und SE 9 sind da

Java SE 9 ist freigegeben worden – zeitgleich mit der Java-EE-8-Spezifikation und ihrer Referenz-Implementierung GlassFish 5.0. Ein großer Tag für Java-Fans. Die Hersteller werden für weitere Applikations-Server hoffentlich auch zügig EE-8-Versionen herausbringen. GlassFish selber dürfte ja höchstens noch für Experimente genutzt werden – außer in der modifizierten Version von Payara.

https://www.theregister.co.uk/2017/09/22/java_se_9_java_ee_8_debut/

29. September 2017

EE4J

Nächster Schritt in der Übergabe von Java EE: Bei der Eclipse Foundation gibt es jetzt ein neues Top-Level-Projekt „Eclipse Enterprise for Java" (EE4J). Ein Entwurf der „Project Charter" ist gerade auf *„eclipse.org"* veröffentlicht worden. Die Community wird um Feedback und Mitarbeit in der E-Mail-Liste „ee4j-community" gebeten.

https://dev.eclipse.org/mhonarc/lists/ee4j-community/

1. Oktober 2017

JavaOne – Community Day

Am Sonntag vor der JavaOne ist traditionell Community Day, mit einer Mischung aus technischen Vorträgen und Informationen von und für JUGs. Mein Eindruck: Die JavaOne und die parallele Oracle OpenWorld scheinen im Vergleich zum Vorjahr insgesamt ein bisschen kleiner zu sein, die JavaOne ist größtenteils im Moscone West untergebracht und nicht mehr im zehn Fußminuten entfernten Hilton-Hotel. Der Community Day ist wieder in einzelne Tracks unterteilt, unter anderem die NetBeans Days, die meist für volle Säle sorgen. Erwartungsgemäß ist die Teilnehmerzahl beim JUG-Track nicht so groß, etwa bei der Podiums-Diskussion „Growth-Hack Your Community"-Session, die von der Israeli Java Community, der holländischen NLJUG und der DOAG Java Community veranstaltet wird. Aber auch gut zwanzig weitere Teilnehmer sind prima, da ein reger Austausch zwischen JUG-Vertretern aus aller Damen und Herren Länder stattfindet und am Ende alle neue Ideen mit nach Hause nehmen. Wie uns die BarCamp-Regel lehrt: „Wer immer da ist, es sind die richtigen Leute". Ein großes Thema auf der JavaOne ist natürlich die Zukunft von Java EE bei der Eclipse Foundation. Vor ein paar Tagen hat Eclipse den Entwurf der „Project Charter" und den gewählten Projektnamen „Eclipse Enterprise for Java" veröffentlicht. Allein die Namensgebung löst große Diskussionen aus. Aber einen Namen für ein Projekt von der Größenordnung von Java EE zu finden, inklusive aller (marken-)rechtlichen Fragestellungen, ist wohl nicht ganz einfach. Und der Name ist auch erst mal nur für das Projekt gedacht und nicht für das Marketing. Ein neuer Name, um das Zertifikat Java EE zu ersetzen, wird noch gesucht. Wichtiger als der Name sind aber wohl Fragen wie: Wird der Prozess weiterhin vom JCP beziehungsweise von dessen Executive Committee kontrolliert? Wie ist das Verhältnis zu MicroProfile? Diese sind noch völlig ungeklärt. Zur ersten Frage sagt Eclipse-Chef Mike Milinkovich immerhin, dass dies nach seinem Verständnis nicht der Fall sein soll – aber da gibt es auch andere Meinungen.

https://projects.eclipse.org/projects/ee4j/charter

2. Oktober 2017

JavaOne – Tag 1

Der große Saal im Moscone North Center wirkt bei der diesjährigen Java-Keynote etwas leerer als letztes Jahr. Dabei gibt es jede Menge zu erzählen und zu zeigen. Java SE 9 ist gerade erschienen, ebenso

Java EE 8, und es geht seitdem Schlag auf Schlag weiter mit großen Ankündigungen und Änderungen wie „Eclipse Enterprise for Java." Ein Trend dazu: Alles, was nicht als Alleinstellungsmerkmal gegenüber dem Wettbewerb taugt, wird zu Open Source. Bei Firmen wie Red Hat ist das schon lange so, aber auch Oracle, IBM und Intel werben auf der Keynote mit ihren „Contributions". Oracle gibt während der Keynote das „Fn Project" (Cloud Functions, Amazon nennt sie ja schon „Lambdas") frei – so übertrieben spontan im Dialog zwischen zwei Vice Presidents, dass es schon wieder witzig ist. Das ist aber auch der eher nervige Aspekt der Keynote: Die Beiträge werden inzwischen von den Unternehmen wohl so auf Punkt und Komma geprüft, dass gerade die Top-Manager lange Redebeiträge oft nur noch vom Prompter ablesen. Nur Larry Ellison kann noch frei reden (allerdings nicht hier, sondern auf der parallelen OpenWorld) und bringt damit sicher seine Rechtsabteilung ins Schwitzen. Am Rande notiert – ein wenig Munition für die nächste „Mein Haus, mein Auto, meine Programmiersprache"-Diskussion mit PHP-Entwicklern bringt Gastredner James Governor, Analyst und Mitgründer von RedMonk mit „When web companies grow up, they turn into Java Shops". Microservices, Docker, Kubernetes sind weitere Themen, unter anderem präsentiert vom Chef-Architekten von Spotify. Der letzte Teil der Keynote ist dann wieder Mark Reinhold vorbehalten. Genau wie im letzten Jahr fällt bei ihm mitten im Vortrag die Technik aus – diesmal ist es der Teleprompter (siehe oben unter „Spontanität"). Vielleicht ja doch ein Streich unter Kollegen? Anders als im letzten Jahr wird die Keynote aber – ganz unamerikanisch – einfach um zehn Minuten überzogen, statt mitten in den Demos abzubrechen. Das große Thema ist natürlich Jigsaw. Neun Jahre, so Reinhold, haben er und viele Kollegen daran gearbeitet, die „Classpath Hell" abzuschaffen und den massiven Monolithen JDK aufzubrechen. Der Aussage „Jigsaw breaks everything" tritt er gleich entgegen: „Jigsaw breaks some things" – aber nur bei Zugriffen auf interne APIs. Hier wurden die Restriktionen ja deutlich gelockert, die meisten dieser Zugriffe werden erst in zukünftigen Releases gar nicht mehr funktionieren, sodass die Projekte Zeit für eine Migration haben. Allerdings ist der Zeitrahmen vielleicht gar nicht so üppig: Die schon vor einiger Zeit angekündigte Änderung in den Release-Zyklen soll nämlich durchgezogen werden. Java erhält einen strikten „Release Train" mit einem Release alle sechs Monate. Java 10 soll daher im März 2018 kommen. Die ersten praktisch fertigen Features, etwa eine deutliche Verbesserung der Switch Statements mit implizitem Type Casting für weniger Boilerplate Code und mehr Lesbarkeit, stellt Chief Language Architect Brian Goetz vor.

https://www.youtube.com/watch?v=Tf5rlIS6tkg

3. Oktober 2017

JavaOne – Tag 2

Die vielleicht größten Hype-Themen auf der JavaOne und OpenWorld sind nicht die selbst-administrierende Datenbank oder die neuen Cloud-Angebote von Oracle, sondern Blockchain. Nicht verwunderlich, wenn man den Analysten glaubt, dass im Jahr 2027 zehn Prozent des Bruttoinlandsprodukts (global) über Blockchain-Transaktionen verwaltet werden und dass die Technologie nicht nur die Finanzbranche umkrempeln wird. Oracle hat dafür gesorgt, dass sie an diesem Wachstumsmarkt beteiligt sind, und den Blockchain Cloud Service angekündigt. Auf der Developer Keynote werden Oracles neue Aktivitäten zur Einbindung der Community beworben, „Developer Relations" heißt das heute. Bruno Borges und Stephen Chin reden über die „Oracle Code Events", die letztes Jahr ins Leben gerufen wurden und die sie binnen zwölf Monaten schon durch siebzehn Länder führten, mit insgesamt zwanzig Events und mehr als 10.000 Teilnehmern. „Be where developers are" lautet das Motto – statt darauf zu warten, dass diese zu den großen Konferenzen kommen. Oracle verstärkt seine Bemühungen deutlich, die Entwickler mit einer Offensive von Social Media, kostenlosen Online-Kursen etc. an das Unternehmen zu binden. Der Rest der Veranstaltung dreht sich darum, den Teilnehmern den neuen „Application Development Stack" – in erster Linie das Cloud-Angebot – vorzustellen. Als roter Faden dient eine Mobile App zum Kauf gebrauchter Autos, mit der sich zu einem beliebigen Autokennzeichen detaillierte Informationen inklusive eines von der App vorgeschlagenen Kaufpreises abrufen lassen (vorgezogenes Halloween für Datenschützer, aber wir sind hier ja nicht im Land der Bedenkenträger und es ist ja nur ein Beispiel). Unter anderem kommen das kürzlich zugekaufte API-Management von Apiary und die neue „Conversational AI Platform" zum Einsatz. Für Letztere ist auch eine Integration in Slack geplant.

https://youtu.be/Tit3PsK_oCg

4. Oktober 2017

JavaOne – Tag 3

Der vorletzte Tag der JavaOne bringt eine Podiums-Diskussion „Accelerating the Adoption of Java EE 8 with MicroProfile". Auf der Bühne sitzen Vertreter der Gründerfirmen von MicroProfile: Kevin Sutter (IBM), Mike Croft (Payara), Mark Little (Red Hat) und David Blevins (Tomitribe), außerdem ist Oracle durch David Delabassee vertreten. Durch die Diskussion führt John Clingan von Red Hat. Gut zu sehen, dass die Kommunikation zwischen MicroProfile auf der einen Seite und Oracle beziehungsweise Java EE 8 auf der anderen Seite gut zu funktionieren scheint – auch wenn nicht alle immer die gleiche Meinung teilen. David Blevins schätzt, dass MicroProfile, da demnächst beide Projekte bei der Eclipse Foundation beheimatet sind, schon bald unter den Schirm von Java EE schlüpfen wird – als „innovative Speerspitze". David Delabassee erwidert, dass es für beide Projekte erst mal genug zu tun gibt und man später darüber nachdenken kann. „Genug zu tun" ist ein gutes Stichwort: Das EE4J-Projekt wird mit Sicherheit für viele Monate damit beschäftigt sein, die technischen, insbesondere aber auch rechtlichen und organisatorischen Herausforderungen zu lösen, die mit dem Umzug verbunden sind. Die technischen Probleme haben auch mit dem Freigeben der bislang geheimen TCKs zu tun, die teilweise auf Test-Frameworks aus dem Zeitalter vor JUnit basieren und so wohl nicht in einem Open-Source-Projekt überleben werden. Das sollte aber nur eine Kleinigkeit gegenüber den nicht-technischen Herausforderungen sein, und in den JSR-Projekten wird sich erstmal nicht viel tun (aber die Situation kennen wir ja bereits aus dem Jahr 2016). Ob sich da ein Termin im Jahr 2018 halten lässt, wie ihn Oracle mal für Java EE 9 angekündigt hat, darf bezweifelt werden. Aber bis dahin wird auf jeden

Das Java-Tagebuch

Andreas Badelt, stellv. Leiter der DOAG Java Community

Fall einiges von MicroProfile zu erwarten sein, was dann später in Java EE 9 (oder wie immer es dann heißen wird) fließen kann: etwa das Configuration API oder vielleicht auch ein neues Transaction-Framework, das auch auf der JavaOne vorgestellt wurde und auf Basis des Narayana-Projekts „Long Running Actions" unterstützen soll (anstelle der gängigen ACID-Transaktionen, die für verteilte Services nicht gut funktionieren).

https://youtu.be/BhMLxwf0AMM

5. Oktober 2017

JavaOne – Tag 4

Das Moscone Center ist bereits leergeräumt. Vermutlich aus Kostengründen finden die – gegenüber den Vortagen reduzierten – Veranstaltungen des letzten Tages nur noch im benachbarten Marriott-Hotel statt. Was etwas schade ist, da bereits am Mittwochnachmittag die Ausstellung und die sonstigen Aktivitäten wie Hackergarten, Demo Labs etc. eingestellt werden. Die Amerikaner sind sehr effizient darin, von einer Sekunde auf die andere „klar Schiff" zu machen. Am letzten Tag der JavaOne findet traditionell die Community-Keynote statt, bestehend aus einem informativen Teil und einer Art Komödie, aufgeführt von Laiendarstellern aus den JUGs, in der die neueren Entwicklungen von Java aufs Korn genommen werden. Der riesige „Yerba Buena Ballroom" im Marriott neben dem Moscone Konferenzzentrum ist gut gefüllt.

Der informative Teil wird wie letztes Jahr von IBM bestritten. John Duimovich berichtet über das gerade erst als Open Source freigegebene J9 (bzw. jetzt OpenJ9 – die JVM heißt schon lange so, der Name hat also nichts mit der Release-Nummer zu tun). Der neue „Low Pause"-Garbage-Collector wird ebenso beworben wie der geringe Speicherbedarf, der insbesondere in der Cloud die Kosten drücken soll. Genau wie Oracle baut IBM sein Cloud-Angebot für die Entwickler massiv aus. Ein wichtiger Bestandteil davon ist weiterhin der Applikations-Server Liberty, dessen 3,5 Millionen Code-Zeilen auch kürzlich als Open Liberty freigegeben wurden: „The biggest open source contribution ever" (ob das nach EE4J auch noch stimmt, ist bislang nicht klar). Das Thema „Open Source" zieht sich durch die Konferenz.

Zum Schluss betritt MicroProfile-Co-Projektleiter John Clingan die Bühne, um die bereits bei der Panel-Diskussion am Vortag diskutierte Zusammenarbeit auch dem breiten Publikum vorzustellen: Die Zusammenarbeit zwischen MicroProfile und EE4J ist fest eingeplant, aber ob sie rein über JSRs durchgeführt wird (wie z.B. mit dem Configuration API bereits gestartet) oder ob MicroProfile sich unter das EE4J-Top-Level-Projekt begibt, ist noch nicht entschieden.

Es folgt das Stück „Community Keynote Reloaded", angelehnt an die Matrix-Filme, mit „Director" Stephen Chin (eigentlich „Director Oracle Technology Network"): „They screwed up part two and three of the movie, so we have to fix that." Die Aufnahme wird es vermutlich nicht auf Netflix oder Amazon schaffen, aber wie viele andere JavaOne-Inhalte auf YouTube zu finden sein.

https://www.youtube.com/embed/L1BNeRGhl34

21. Oktober 2017

Java Community Process: Wahlen

Die Kandidaten für die diesjährigen Wahlen zum Executive Committee stehen fest. Für die „ratified Seats" (zum Abnicken durch die JCP-Mitglieder) sind es ARM, Credit Suisse, Fujitsu, HP, IBM, Intel, Red Hat und SouJava. Um die drei frei wählbaren Sitze bewerben sich Alibaba, Hazelcast, Tomitribe, Twitter und UEDB Limited. Für die „assoziierten Sitze" (Vertreter der „Mitglieder light" – die Kandidaten selbst müssen aber Vollmitglieder sein) sind es Andres Almiray, Andrew Gumbrecht, Magesh Kasthuri, Werner Keil, Geir Magnusson Jr. und Andres Cespedes Morales.

Es bleibt natürlich abzuwarten, wie viel Einfluss das EC in Zukunft noch haben wird. Mit Java EE geht ein signifikanter Teil an Eclipse, der dann vermutlich nicht mehr im Rahmen des JCP standardisiert wird – auch wenn das noch nicht feststeht.

https://jcp.org/en/whatsnew/elections

24. Oktober 2017

EclipseCon Europe: EE4J continued

Am ersten Tag der EclipseCon Europe in Ludwigsburg gibt es einiges zum EE4J-Projekt zu hören. Eclipse-Foundation-Chef Mike Milinkovich berichtet in seiner Keynote von dem Projekt, das auch für die Foundation ein enormes Unterfangen ist, aber auch eine große Chance: „Eclipse Foundation is becoming the source for innovation and collaboration in the Java ecosystem", heißt es auf einer Folie. David Delabassee, Oracles Java-EE-Evangelist (heißen die nicht woanders inzwischen alle „Developer Advocate"?), kann in seinem Vortrag „Java EE 8 is final … Now what?" neben dem Überblick über EE 8 nur kurz auf die Zukunft eingehen – aber es gibt ja noch eine Panel-Diskussion und abends eine „Birds of a Feather"-Session. Die Panel-Diskussion wird von Mike Milinkovich geleitet. Beteiligt sind die „Project Management Committee"-Vertreter der großen an EE4J beteiligten Firmen: Dimitry Kornilov von Oracle, Kevin Sutter (IBM), Heiko Rupp (Red Hat), außerdem David Delabassee. „Was wäre ein Erfolgskriterium für EE4J innerhalb eines Jahres?", fragt Mike die Runde unter anderem. Für Delabassee wäre es ein Erfolg, wenn alle Oracle-Projekte bei der Eclipse Foundation liegen und die dann quellcodeoffenen TCKs erfüllen – quasi ein Release-Kandidat für EE4J 1.0, der im Grunde nur Java EE 8 repliziert. Angesichts der schieren Größe des Migrations-Projekts wäre das aber wohl tatsächlich eine anzuerkennende Leistung. Einige der weitergehenden Fragen werden nicht vollständig beantwortet, weil die Panel-Teilnehmer noch nichts dazu sagen können – oder dürfen, etwa die Frage, wie sich das Verhältnis zwischen dem OpenJDK und EE4J gestalten wird. Immerhin so viel ist klar: OpenJDK-Code, der nur für EE relevant ist (dieser wurde mit Java 9 unter „java.se.ee" zusammengefasst), soll an Eclipse gehen. Eine Standardisierung von EE4J soll es auch weiterhin geben – das klang schon mal anders. Die Devise heißt aber: „Code first". Welchen Fahrplan der Release-Train hat – alle zwei Jahre, zwei Mal pro Jahr … – dazu traut sich niemand eine verlässliche Aussage zu. „Schneller als bei Java EE soll es ge-

hen, aber das trifft ja auf so ziemlich alle Projekte zu", heißt es augenzwinkernd. An der BoF-Session nimmt bis auf Mike Milinkovich die gesamte Panel-Besetzung teil. Hier wird insbesondere das Thema „Community-Beteiligung" etwas ausführlicher diskutiert. Was kann die Community tun? Zunächst mal etwas Geduld haben, weil die Projekte organisatorisch und technisch laufen müssen. Auch die Übertragung der „IP" von Oracle wird noch eine Weile dauern. Trotzdem hilft es natürlich, bereits für das Projekt zu werben. Und auch wenn erst mal nur die Mitglieder der Java EE Expert Groups als „Committer"-Kandidaten für die Eclipse-Projekte gesehen werden, erhofft man sich doch eine größere Beteiligung aus der Community. Eine Beteiligung an den Projekten, etwa in Form von Pull Requests, ist auch sehr einfach. Die Hürden für Committer, mit Schreibrechten in den Repositories, sind aber logischerweise etwas höher. Neben einem gewissen „track record" aus Open-Source-Projekten gehört dazu ähnlich wie beim JCP das Unterzeichnen eines „Individual Contributor Agreement" und für abhängig Beschäftigte auch eine Bescheinigung des Arbeitgebers. Letztere könnte in manchen Firmen für Schwierigkeiten sorgen, aber scheinbar ist es gar nicht so schlimm, wie es sich anhört. Es klingt jedenfalls nach einer guten Aufgabe für den iJUG, hier rechtliche Aufklärungsarbeit zu leisten.
https://www.eclipsecon.org/europe2017

9. November 2017

Oracle macht beim MicroProfile mit

Oracle hat sich ganz offiziell dem MicroProfile-Projekt angeschlossen, das ja inzwischen bei der Eclipse Foundation liegt. Noch mehr Unterstützung für das bislang schon recht erfolgreiche Projekt, das sich aus der Enterprise-(Edition)-Sicht speziell den Microservices widmet wird. Damit sind nun die großen drei des Java Ökosystems – IBM, Oracle und Red Hat – alle bei MicroProfile aktiv. Hoffentlich führt das auch zu mehr Zusammenarbeit in anderen Bereichen.
https://www.infoq.com/news/2017/11/oraclejoinsmicroprofile

16. November 2017

Wahlen zum EC gehen in die Verlängerung

An sich sind die Wahlen zum Executive Committee des Java Community Process ja immer relativ unspektakulär: Ein sehr kurzer Wahlkampf ohne große Kontroversen, kein Fernsehduell, keine Hochrechnungen nach dem Ende der Stimmabgabe – ein paar Tage nach der 14-tägigen Wahlphase wird ganz nüchtern das Ergebnis präsentiert, und das war's. Aber jetzt, wo der JCP eher an Bedeutung verliert, kommt mal ein bisschen Spannung auf. Beim einzigen zu vergebenden Associate Seat gibt es ein Unentschieden zwischen den Kandidaten Andres Almiray und Werner Keil. Der JCP hat aber auch hierfür einen klaren – und irgendwie auch lustigen – Prozess, der sich an RFC 2777 der IETF („Publicly Verifiable Nomcom Random Selection") orientiert. Zunächst sind den beiden Kandidaten anhand des Endstands des NASDAQ-Index die Zahlen 1 und 2 zugeordnet worden (der Index schloss am 15.11. mit einer geraden Zahl, daher werden die Kandidaten in alphabetischer Reihenfolge aufgelistet, also die 1 für Andres). Als nächstes werden nun für den 16.11. drei weitere „Zufallszahlen" ermittelt und in ein Online-Formular vom W3C eingetragen, das auf RFC 2777 basiert *(siehe „https://www.w3.org/2001/05/rfc2777")*. Die Schlussstände des S&P-500-Index und des Dow Jones Industrial Average (ohne Nachkommastellen) sowie der „Winning Score" des Basketballspiels zwischen den Golden State Warriors und den Boston Celtics. Man sieht, das Ganze ist fest in amerikanischer Hand – aber verglichen mit einer Präsidentenwahl wohl nur sehr schwer manipulierbar: Die drei Quellen wurden einen Tag vorher festgelegt. Wer alle drei manipulieren kann, sich aber immer noch für den JCP interessiert, dürfte außerdem als harmlos gelten.
https://www.jcp.org/en/whatsnew/elections

22. November 2017

Wahlen zum Executive Committee: Endergebnis

Mit ein paar Tagen Verzögerung stehen die Ergebnisse jetzt fest: Bei den Associate Seats (nur einer war zu vergeben) hat sich Andres Almiray bei Stimmengleichheit per Losverfahren gegen den Sitzverteidiger und JCP-Urgestein Werner Keil durchgesetzt. Von den restlichen Kandidaten konnte nur Andy Gumbrecht einigermaßen mithalten, die anderen waren weit abgeschlagen. Bei den „Ratified Seats" sind alle Kandidaten abgenickt worden. Bei den frei gewählten Sitzen haben sich Twitter, Hazelcast und Tomitribe durchgesetzt. Alibaba folgt mit deutlichem Abstand, UEDB weit abgeschlagen. Herzlichen Glückwunsch an alle alten und neuen EC-Mitglieder, und viel Erfolg bei den aktuellen Herausforderungen, insbesondere dem Auseinanderdriften von SE und EE.

Herzlichen Dank an Werner Keil für den langjährigen Einsatz im EC.
https://www.jcp.org/aboutJava/communityprocess/elections/2017.html

Andreas Badelt
stellv. Leiter der DOAG Java Community

Andreas Badelt ist stellvertretender Leiter der DOAG Java Community. Er ist seit dem Jahr 2001 ehrenamtlich in der DOAG Deutsche ORACLE-Anwendergruppe e.V. aktiv, zunächst als Co-Leiter der SIG Development und später der SIG Java. Seit dem Jahr 2015 ist er stellvertretender Leiter der neugegründeten Java Community innerhalb der DOAG. 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").

Java EE – die nächste Generation

Peter Doschkinow, ORACLE Deutschland B.V. & Co. KG

Die Fertigstellung von Java EE 8 und die Ankündigung von Oracle, die Java-EE-Spezifikationen, Test-Compatibility-Kits (TCK) und Referenz-Implementierungen vollständig zu öffnen und an die Eclipse Foundation zu übergeben, wurden in der Java Community als ein Befreiungsschlag empfunden.

Diese Schritte sowie die positive Stimmung auf der JavaOne 2017 haben nach monatelangen Diskussionen und Verunsicherungen über den Zustand, das Management und die Zukunft der Java-EE-Plattform App-Server-Hersteller sowie Entwickler neu mobilisiert und eine neue, erfolgreiche Zukunft für Enterprise Java mit mehr Community-Beteiligung und effizienteren Prozessen in Aussicht gestellt.

Java EE 8

Java EE 8 und die Referenz-Implementierung GlassFish 5 wurden am 21. September 2017 zeitgleich mit der Java SE 9 freigegeben. Im Vergleich zur vorherigen Version Java EE 7 ist der Umfang der neuen Features geringer ausgefallen; einige der vorgesehenen Teilspezifikationen wie MVC, JMS 2.1 und Management API 2.0 wurden ausgelassen. Ein Grund dafür ist, wie Java-EE-8-Spec-Lead Linda DeMitchiel in ihrer JavaOne-Session erklärte, dass führende Mitarbeiter zeitweise mit anderen, höher priorisierten Aufgaben beschäftigt waren. Auch der Umzug der Java-EE-/GlassFish-Infrastruktur von *„java.net"* in die moderne GitHub-Umgebung unter *„https://github.com/javaee"* während der Entwicklung hat viel Zeit gekostet. Er war jedoch notwendig, um die Plattform für interessierte Entwickler zugänglicher zu machen und ihre Zusammenarbeit besser zu unterstützen.

Die Java Community freut sich nun auf die neue Version. Java EE wurde weiter modernisiert und ihre Nutzung in automatisierten CI/CD-Pipelines („continuous integration and delivery") ist einfacher geworden. Neue APIs standardisieren und erleichtern den Umgang mit modernen Web-Technologien. Zu den wichtigsten Neuerungen gehören:

- Java Servlet 4.0 API mit HTTP/2-Support: Die Unterstützung für das HTTP/2-Protokoll mit Request/Response Multiplexing, Server Push und Protokoll-Upgrade von HTTP/1.1 ermöglicht dem Web-Client (unter Beibehaltung der HTTP/1.1-Semantik) Nebenläufigkeit, ohne mehrere TCP-Verbindungen zum Server aufzubauen, und verbessert somit seine Latenz-Zeiten.
- Erweiterter JSON-Support einschließlich eines neuen JSON-Binding-API vereinfacht den Umgang sowie die Modifikation von JSON-Strukturen und kann über Java-Annotations ein anpassbares Mapping zwischen diesen und Java-Objekten definieren.
- Ein neues REST-Reactive-Client-API, motiviert durch Java-Bibliotheken wie RxJava und das Reactive-API von Jersey, die Referenz-Implementierung von JAX-RS sowie ein Reactive-Client-API zeigen ihre Stärken am besten im Use-Case Microservice-Orchestrierungen. Wenn die Ergebnisse eines Aggregations-Service von mehreren Services abhängen, die aus Performance-Gründen asynchron und parallel abgefragt werden müssen, so werden der Code und die Fehlerbehandlung schnell komplex und unübersichtlich. Das Reactive-Client-API ist ein Fluent-API für asynchrone Data-Streams, das das Observer-Muster implementiert und für diesen Fall eine einfachere, besser lesbare und zuverlässigere Implementierung ermöglicht.
- Asynchrone CDI-Events verbessern die Skalierbarkeit und die Zuverlässigkeit bei der Zustellung von Events zwischen Produzenten und Konsumenten, weil sie in verschiedenen Threads stattfindet.
- Ein neues, portierbares Security-API vereinfacht die Konfiguration der Anwendungs-Authentifizierung (BASIC, DIGEST, FORM, oder CERT). Es standardisiert den Umgang mit dem benötigten Repository für User-Credentials, Gruppen-Zugehörigkeit und Autorisierung, egal ob eingebettet oder auf einer Datenbank beziehungsweise LDAP-Server basierend. Das API definiert auch einen Plattform-übergreifenden Security Context, der den Zugriff auf Ressourcen deklarativ und programmatisch einheitlich regeln kann (bisher zersplittert in Servlet, JAX-RS, CDI, JAX-WS, EJB, JSF und WebSocket API).
- Server-Sent Events Support (Client- und Server-seitig) standardisiert ein übliches Kommunikations-Muster, bei dem der Server dem Web-Client mehrere Updates über dieselbe HTTP-Verbindung schickt.
- Unterstützung für die neuen Java-SE-8-Features (wie Data & Time API, Streams API, Annotations).

Wie in der Vergangenheit üblich, steht neben GlassFish 5 auch ein Java-EE-8-Software-Developer-Kit zum Download bereit, in dem zusätzlich Code-Beispiele, die API-Dokumentation sowie ein Tutorial enthalten sind. *Abbildung 1* zeigt die neuen und modifizierten Java EE 8 Specs.

Eclipse Enterprise for Java (EE4J) – die neue Heimat von Java EE

Im August 2017 hat Oracle damit begonnen, verschiedene Alternativen zu untersuchen, um Java-EE-Technologien an eine Open-Source-Organisation zu übergeben. Ziel war, die Weiterführung der Java-EE-Standards agiler, flexibler und offener zu gestalten. Unter Einbeziehung der Sichten von IBM, Red Hat und der Community hat sich Oracle schließlich für die Eclipse Foundation entschieden, die sich als effizientes Zuhause für viele Open-Source-Projekte wie die IBM-J9-Java-Implementierung und EclipseLink erwiesen hat. Die Java-EE-Plattform, bestehend aus Spezifikationen, Referenz-Implementierungen, TCK und Dokumentationen, wird nun unter dem Dach des neuen Eclipse-Top-Level-Projekts EE4J weiterentwickelt. Im ersten Schritt sollen der Java-EE-8-Kern, GlassFish 5 und die von Oracle angeführten Java-EE-8-Technologien, inklusive Code und Build-Scripts, dorthin gebracht werden.

Der Nachfolger von Java EE 8 bekommt demnächst einen neuen Namen (Java EE ist ein Markenname von Oracle). Die Lizenzierung wird flexibilisiert und erfolgt zunächst nach Eclipse-Public-Lizenz (EPL) 2.0 mit der sekundären Lizenz GPL 2 mit Classpath Exception. Das bedeutet, dass EE4-Ergebnisse GPL-2-kompatibel sind und, falls mit GP-2-Artefakten kombiniert, zusammen unter der GPL-2-Lizenz bereitgestellt werden können.

Das EE4J-Open-Source-Projekt hat sich folgende Ziele gesetzt:

- Erstellung von Standard-APIs, Referenz-Implementierungen und TCK für Java-Runtimes für Entwicklung und Betrieb von Server-Side- und Cloud-Native-Anwendungen auf der Grundlage von Java EE 8
- Definition eines agilen und offenen Prozesses, flexible Lizenzierung und offenere Verwaltung für die Evolution der Plattform
- Kompatibilität zwischen Java EE 8 und EE4J-Folgeversionen
- Förderung der gemeinsamen Nutzung und leichte Integration der einzelnen Technologien sowie ihre Stand-alone-Nutzung auf Java SE, um leichtgewichtige und Microservices-Architekturen besser zu unterstützen

EE4J ist offen für die Integration von Innovationen, die in anderen Open-Source-Projekten stattfinden. Ein anderes spannendes und Java-EE-verwandtes Eclipse-Projekt ist zum Beispiel MicroProfile, das sich mit der Optimierung von Java-Enterprise für Microservice-Architekturen beschäftigt und sich als Innovations-Lab für Java-EE-Technologien versteht. EE4J wird untersuchen, wie eine effiziente Zusammenarbeit und Integration von MicroProfile-Ergebnissen gestaltet werden kann.

Oracle wird nach wie vor Support für bestehende Java-EE-Lizenznehmer und solche, die nach Java EE 8 migrieren, anbieten. Oracle beabsichtigt, auch existierende WebLogic-Versionen und Java EE 8 in einer künftigen WebLogic-Version zu unterstützen. Somit möchte Oracle einen fließenden Übergang von existierenden Java-EE-Standards in eine offenere Umgebung, in der sie erfolgreicher weitergeführt werden können.

Fazit

Java EE bildet seit vielen Jahren erfolgreich das Fundament für viele unternehmenskritische Anwendungen, weil es komplexe und wichtige Dienste für Entwickler über standardisierte und leicht zu nutzende Schnittstellen zur Verfügung stellt. Java EE 8 ist der letzte Meilenstein in Richtung Vereinfachung, Modernisierung und Anpassung der Plattform an aktuelle Web-Technologien und Trends. Die Übergabe von Java EE an die Eclipse Foundation hat Begeisterung und breite Zustimmung

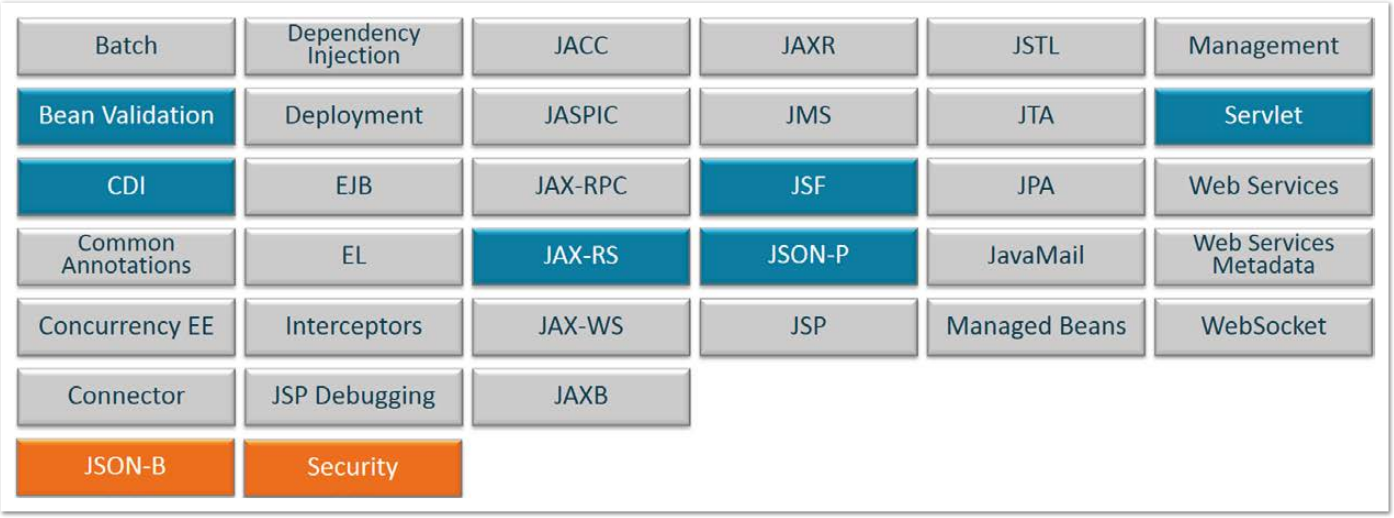


Abbildung 1: Die neuen Java EE 8 Specs in orange, die stark modifizierten in blau

in der Java-EE-Community ausgelöst sowie die berechtigte Hoffnung, dass effizientere Prozesse, Management und stärkeres Community-Engagement im Rahmen von EE4J-Enterprise Java schnell nach vorne bringen. Wie James Governor auf der JavaOne angemerkt hat, „haben sich in der Java-Welt die Dinge in den letzten drei Wochen wahrscheinlich mehr geändert als in den vergangenen dreizehn Jahren".

Links

Java EE: *http://www.oracle.com/technetwork/java/javaee/overview*
GlassFish: https://github.com/javaee/glassfish
Java EE Freigabe: *https://blogs.oracle.com/theaquarium/opening-up-ee-update*
EE4J: *https://projects.eclipse.org/projects/ee4j/charter*
The Future of Java EE (Mark Little, Red Hat): *http://middlewareblog.redhat.com/2017/09/11/the-future-of-java-ee*

Peter Doschkinow
peter.doschkinow@oracle.com

Peter Doschkinow arbeitet als Senior Java Architekt bei Oracle Deutschland. Er beschäftigt sich mit serverseitigen Java-Technologien und Frameworks, Web Services und Business Integration, die er in verschiedenen Kundenprojekten erfolgreich eingesetzt hat. Vor seiner Tätigkeit bei Oracle hat er wertvolle Erfahrungen als Java Architect and Consultant bei Sun Microsystems gesammelt.

Java EE 8 und danach – mit Cloud und Community

Werner Keil, JCP Executive-Committee-Mitglied, Java EE 8 EG Mitglied

Nachdem die Fertigstellung von Java EE 8 bereits maßgeblich durch Mitglieder der Java Community erfolgte, wirkt die geplante Übergabe an die Eclipse Foundation als Open-Source-Projekt wie ein logischer nächster Schritt. Insbesondere, da Oracle mit dem Fn Project seine eigenen Visionen und Angebote für die Cloud in einem „Serverless"-Umfeld eben erst auf der JavaOne 2017 und Oracle Code präsentiert hatte.

Man wendet sich damit bei Oracle doch etwas von Java ab, da zwar eine gute Unterstützung für Java-„Functions" in der Fn Cloud versprochen wurde, deren Infrastruktur aber komplett auf Google Go, Docker oder Amazon AWS und Lambda basiert. Dies erlaubt eine Vielzahl von Sprachen (von JavaScript über Python und Go bis Java), wodurch Java nur eine von vielen ist. Es bleibt abzuwarten, wie sehr sich Java hier gegen die Konkurrenz bewährt.

Java EE 8

Am 21. September 2017 wurden Java EE 8 *[1]* und die Referenz-Implementation GlassFish 5 *[2]* veröffentlicht. Anders als bei den JSRs, die in Java EE 8 einflossen, sind die Committer des GlassFish-Projekts fast ausschließlich Oracle-Mitarbeiter. Nach dem Umzug von *„java.net"* in die GitHub-Organisation *„https://github.com/javaee"* ist die Transparenz ein großer Vorteil von GitHub. Hier sind alle, die in ein Java-EE-Repository gepusht haben, klar ersichtlich. Dies kommt nicht zuletzt auch dem Transparency-Auftrag von „JCP.next" entgegen. Nachdem die wichtigsten Neuerungen in Java EE 8 von Oracle bereits hinlänglich beleuchtet wurden, möchte ich den Blick auf den Anteil der Community in den jeweiligen JSRs richten, die darin neu oder aktualisiert eingeflossen sind:

Java Servlet 4.0 API mit HTTP/2-Support: Dies wurde fast ausschließlich von Oracle als Spec Lead getragen, wenngleich die Mitglieder der Expertengruppe und die erweiterte Community durchaus gefragt wurden.

Java Server Faces: Dieser JSR unter Leitung von Ed Burns (ebenso wie Servlet 4.0) gehört traditionell zu den offensten und transparentesten. Selbst wenn Details der Implementierung oder das TCK bisher nicht offen entwickelt werden, ist hier ein starker Anteil der Java-Community und verschiedener JSF-Implementationen eingeflossen.

JSON-Binding 1.0 API: Wurde zwar zu erheblichen Teilen durch Spec Lead Dmitry Kornilov und seine Kollegen in Prag getragen, aber auch in dieser Expertengruppe sind Open-Source-Projekte und Einzelpersonen vertreten, die ihre Wünsche und Ideen einbrachten. So wurden auch das Logo und die Website durch Mitglieder der Community gestaltet. Da die Anfänge bei EclipseLink (JAX-B) lagen, die RI später allerdings in ein unabhängiges Eclipse-Projekt „Yasson" übergeführt wurde, kann man JSON-B als Paradebeispiel und Vorreiter dessen betrachten, was die gesamte Plattform unter EE4J noch vor sich hat. Dmitry wurde dafür auch zu Recht mit dem JCP-Award 2016 als bester Spec Lead ausgezeichnet.

JSON-P 1.1: Diese Verbesserung und Anpassung des JSON-JSR an neue JSON-Standards wurde von der Community, also einigen engagierten Mitgliedern der Expertengruppe, gerettet, als Oracle zwischen 2015 und 2016 begann, die Prioritäten zugunsten seiner Cloud-Produkte oder Projekte wie Fn zu verschieben. Dadurch waren viele Spec Leads gezwungen, die Arbeit an JSRs praktisch ganz einzustellen, und andere dazu, das Unternehmen früher oder später zu verlassen. Der ursprüngliche Spec Lead von JSR-374 war so ein Fall: Ich war als einziges EG-Mitglied bereits im Vorläufer JSON-P 1.0 auch in der Expertengruppe und damit bei allen Änderungen und Verbesserungen besonders an einer Kontinuität und Rückwärtskompatibilität interessiert. Gemeinsam mit anderen Mitgliedern der EG und Vertretern der Community, die als Contributor gewürdigt wurden, gelang es, diesen Spagat aus Erneuerung und Kontinuität relativ gut zu meistern. Weil JSON-P 1.1 auch als Grundlage für den neuen Standard JSON-B 1.0 dient, erklärte sich Dmitry Kornilov um die JavaOne 2016 bereit, diesen auch zu übernehmen. Beide gehören zu guten Beispielen dafür, wie der Spec-Lead-Vertreter bei Oracle sich der Community öffnete und sie einbezog, nicht zuletzt auch, da er neben seinen Aufgaben in der Cloud gleich 2 JSRs zu leiten und fertigzustellen hatte; wie übrigens auch Ed Burns – ein weiterer der „Super Spec Leads" bei Oracle, die wohl zu den ganz wenigen Vertretern im EE4J-Projekt gehören und dort die Zukunft von Enterprise Java weiter mitbestimmen dürfen. Dmitry ist wohl auch durch seine Rolle als Eclipse-Projektleiter für Yasson inzwischen dem EE4J PMC beigetreten.

JAX-RS 2.1: Dieser JSR wurde relativ stark von Oracle getragen und umgesetzt. Bereits im Vorfeld war ein Red-Hat-Vertreter aus der Expertengruppe ausgestiegen, offenbar nach Konflikten oder Meinungsverschiedenheiten mit dem Spec Lead, die anders nicht beilegbar schienen. Einige andere EG-Mitglieder repräsentieren aber sehr wohl die Java Community. Ein paar davon sind auch in den von Oracle aus Java EE 8 abgegebenen MVC-Standard involviert, da dieser ja JAX-RS massiv als Basis nutzt. Es bleibt abzuwarten, wie die Zukunft von JAX-RS im Rahmen eines neuen EE4J-Umfelds aussehen wird. Andererseits wurden viele Neuerungen wie REST-Reactive-Client-API bereits umgesetzt; also könnte sich der Änderungsbedarf hier auch vorerst in Grenzen halten.

CDI 2.0: Hier standen asynchrone Events im Vordergrund, ebenso die bessere Unterstützung von CDI in einem Java-SE-Umfeld, Stichwort „Serverless" oder „Microservices". Obwohl mit wenigen Ausnahmen fast alle Commits auch hier durch Vertreter der Spec-Lead-Firma Red Hat erfolgten, ist die Expertengruppe von einer starken Diskussionskultur getragen, bei der auch selbstständige, unabhängige EG-Mitglieder wie ich und andere ihre Meinung einbringen konnten und praktisch alle Entscheidungen sehr demokratisch nach dem Mehrheitsprinzip erfolgten, statt vom Spec Lead auf eigene Faust durchgeboxt zu werden. Auch hier wurde Antoine Sabot-Durand dafür sowohl mit einer Star-Spec-Lead-Auszeichnung gewürdigt, als auch Gewinner eines JCP-Awards 2017 in der Spec-Lead-Kategorie. CDI war für JSRs – beinahe exotisch – immer schon einer der offensten Standards. Mit API, RI, TCK und sogar der Spezifikation unter einer (Apache-2.0)-Open-Source-Lizenz. Beim Spec-Dokument bilden CDI und Bean Validation die seltene Ausnahme. Alle anderen JSRs halten sich hier ausschließlich an die Oracle-Spec-Lizenz, während Red Hat deren Gültigkeit nur auf von „jcp.org" heruntergeladene PDF-Dokumente beschränkt. Dass Red Hat zu den treibenden Kräften sowohl hinter Eclipse MicroProfile als auch hinter EE4J gehört, verwundert daher wenig. Wie JSRs inklusive CDI 2.1 oder 3.0 in Zukunft aussehen sollen und ob die Package-Struktur diese dann noch rückwärtskompatibel macht, ist dagegen noch Inhalt langwieriger Verhandlungen, die jenen über die „Jamaika"-Koalition in Deutschland nach der Bundestagswahl 2017 in nichts nachstehen.

Bean Validation 2.0: Die Expertengruppe, wie CDI von Red Hat geleitet, umfasst zahlreiche Mitglieder, viele von ihnen Einzelpersonen. Bean Validation 2.0 bezog nicht zuletzt Contributions aus der Community ein, etwa die JSR-354-Unterstützung (JavaMoney) *[3]*, die Zalando zuvor auf GitHub entwickelt hatte. Hier wurden also gerade bei der Referenz-Implementation Hibernate Validator viele Einflüsse aus der Gemeinschaft genutzt, während API oder TCK – offen und transparent – dennoch primär von Red Hat entwickelt wurden.

Java EE Security-API 1.0: Dieser neue Standard wurde noch stärker als JSON-B 1.0 massiv durch die Community getragen und gemeinsam aufgebaut. Es fing mit einer sehr demokratischen Abstimmung und Bewertung der User Stories und Requirements an, beinahe wie ein verteilter Agile-Planning-Poker. Auch hier musste der Oracle

Spec Lead leider sehr bald anderen Aufgaben in der Cloud gehorchen, und die EG war weitgehend auf sich allein gestellt. Dank der besonders aufopfernden Leistung von Arjan Tijms und einer Handvoll anderer (zumeist selbstständiger beziehungsweise unabhängiger) EG-Mitglieder wurde dieser JSR dennoch ein Erfolg und rechtzeitig für Java EE 8 fertiggestellt.

Unterstützung für neue Java-SE-8-Features: Diese wurden großteils als Maintenance Release (MR) und nicht als eigene JSRs für Java-EE-8-Komponenten wie JPA umgesetzt; naturgemäß durch den Spec Lead, da formell eine Expertengruppe für ein MR gar nicht mehr existiert und allein der Maintenance Lead dafür verantwortlich ist. Dennoch kann ich speziell aus jenen Expertengruppen wie JPA, in denen ich schon seit Java EE 6 oder 7 dabei bin, sagen, dass hier die Mitglieder über Entscheidungen befragt wurden und es eine teils auch recht rege Diskussion darüber gab.

Java EE Guardians

Der erwähnte Arjan Tijms gehört wie fast alle Einzelpersonen beziehungsweise Mitarbeiter von Software-Herstellern, die halfen, JSR-375 und viele andere Java EE 8 JSRs erfolgreich fertigzustellen, zu den „Java EE Guardians" *[4]*, einer Grassroot-Bewegung, die der frühere Java-EE-Evangelist bei Oracle, Reza Rahman, ins Leben rief, kurz nachdem er selbst Oracle verließ, als Antwort auf die Kürzungen bei der Standardisierung zugunsten meist kommerzieller Cloud-Angebote. Die Liste wird von so illustren Persönlichkeiten angeführt wie Java-Begründer James Gosling, der übrigens inzwischen auch mit Amazon AWS bei einem führenden Cloud-Anbieter arbeitet, also einerseits mit Oracle konkurriert, andererseits aber (siehe Fn Project) die Basis eigener Cloud-Angebote darstellt.

Die Java EE Guardians wurden anfänglich belächelt (auch ob des an Marvel angelehnten Namens), erwiesen sich jedoch durchaus wie eben jene sehr unterschiedlichen und manchmal chaotisch anmutenden Marvel-Helden als Retter, wenn nicht der Galaxis, so doch zumindest von Java EE 8. Ohne Vertreter der Java EE Guardians wären etliche vor allem der neueren JSRs in den Anfängen steckengeblieben oder hätten wie JCache (JSR-107) inzwischen gleich zum zweiten Mal den Anschluss an den Java-EE-8-Release-Train schlicht verpasst.

Die für Java SE und speziell OpenJDK sehr erfolgreiche Adopt-a-JSR-Bewegung fand davor an Java EE leider nie wirklich Gefallen. Erst die Java EE Guardians brachten selbst jene, die vielleicht nicht auf dieser Liste zu finden sind, zum Umdenken und zur Unterstützung von Java EE oder damit verwandten Projekten wie MicroProfile beziehungsweise EE4J. Heute findet man auch die „üblichen Verdächtigen" unter großen Java-User-Groups wie LJC oder SouJava, die nun Adopt-ähnliche Aktionen auch für Java EE anbieten oder MicroProfile und in naher Zukunft EE4J unterstützen. Ohne die Java EE Guardians als Zündfunke und Fackelträger dieser Bewegung wäre das vielleicht nie geschehen oder hätte deutlich länger gedauert.

Eclipse Enterprise for Java [5]

Auf Druck durch verschiedene Seiten, etwa Spring und seine „Mix & Match"-Politik bei der Nutzung von Standards oder auch die Formierung von Alternativen wie Eclipse MicroProfile *[6]*, sowie nicht zuletzt durch die eigene Ausrichtung auf die Cloud und Lösungen wie das Fn Project *[7]* sah sich Oracle praktisch gezwungen, das Zepter über die Java-Enterprise-Plattform um die JavaOne 2017 abzugeben. Eclipse erwies sich durch eine konsequente Release-Politik mit jährlichen Zyklen sowie etlichen JSRs von JPA bis JSON-B als die beste Alternative. Auch einige von Oracle nicht mehr umsetzbare Standards zur Unterstützung von Microservices wie Konfiguration, Circuit Breaker, Health Check etc., die seit Anfang 2017 bei Eclipse MicroProfile eine Art Rapid Incubator fanden, waren ein weiterer Grund sich für Eclipse zu entscheiden.

Die Eclipse Foundation ist bekanntermaßen sehr transparent und alles geschieht sehr offen – ohne dass eine einzelne Firma oder ein Mitglied die alleinige Kontrolle hätte, was an Oracle und davor an Sun im JCP gerne kritisiert wird. Da sich die Eclipse Foundation anders als Oracle praktisch nur auf Mitgliedsbeiträge oder Spenden ihrer Mitglieder stützt und auch in einigen Bereichen wie bei Juristen, Lizenzexperten etc. nicht eine Armee von Anwälten besitzt – wie Oracle und andere Großkonzerne –, ist allerdings etwas Vorsicht bei den Erwartungen angesagt, insbesondere was den Übergang zu Eclipse anbelangt.

Oracle kündigte auf der JavaOne 2016 an, NetBeans der Apache Foundation überantworten zu wollen. Eclipse hätte dafür bekanntlich keinen echten Sinn ergeben. Erst vor wenigen Wochen, am 10. September 2017, begann laut GitHub Mirror das Repository bei Apache zu existieren. Es hatte also rund ein Jahr für NetBeans gedauert. Die Code-Basis aller Java-EE-JSRs, die nicht bereits bei Eclipse liegen (JSON-B, JPA) dürfte um einiges größer sein. Neben EE4J haben die Eclipse-Lizenz- und -Patent-Experten ja auch noch den üblichen Annual Release Train (Oxygen+1) zu bewältigen sowie mehrere Sub-Communities wie IoT, LocationTech, PolarSys oder ambitionierte und Release-freudige Projekte wie MicroProfile. Als Spec Lead von JSR 363 habe ich selbst erlebt, dass dem Wunsch der Deutschen Telekom und des SmartHome-Projekts, unser JSR nutzen zu dürfen, erst vier bis sechs Monate nach dem Antrag durch SmartHome nachgekommen wurde. Es hieß damals, die Telekom sei zwar ein Solution-Member, aber kein so großer Beitragszahler wie vermutlich Red Hat oder IBM. Daher dauerte etwas länger, selbst mit Schwergewichten, die höhere Beiträge leisten. Hinter EE4J könnte es also viele Monate dauern, bis die einzelnen JSRs dem Umbrella folgen, der ja nicht viel mehr als ein „Feature" im Eclipse-Sprachgebrauch ist oder ein Parent POM.

Das Rotationsprinzip im EE4J PMC, das den Vorsitz alle drei Monate wechselt, ist eine gute Sache. Es ähnelt etwa dem Amt des Schweizer Bundespräsidenten *[8]*, das einmal im Jahr neu besetzt wird, und bei dem nur selten dieselbe Partei zweimal den Präsidenten stellt. Außerdem ist der Vizepräsident immer von einer anderen Partei. Den unabhängigen Vertreter Ivar Grimstad als ersten Vorsitzenden zu ernennen, zeugt auch vom Wunsch, der Community eine Stimme zu geben – auch wenn er nicht in der Java-EE-8-Plattform-EG dabei war und dort, anders als im initialen EE4J PMC, deutlich mehr kleinere Unternehmen oder Einzelpersonen vertreten sind. Die Einladung an Firmen wie Ivars Arbeitgeber (eine große Beraterfirma aus Skandinavien), doch der Eclipse Foundation beizutreten, statt ihn jetzt als „Individual" kostenlos vorzuschicken, wurde auch recht offen und unverhohlen kommuniziert, was angesichts der Spendenfinanzierten Eclipse Foundation auch nicht unverständlich ist. Daher ist es jedoch möglich, dass im Gegensatz zum zuletzt auch für Firmen kostenlosen JCP dort Vertreter größerer Unternehmen, Beratungsfirmen oder IT-Anbieter selbst als Individual Eclipse Committer bevorzugt werden, weil man sich dort weitaus höhere Beiträge erhofft, wenn das Unternehmen sich engagiert und selbst beitritt.

Fazit

Es ist viel geschehen im Bereich „Enterprise Java". Auch wenn der 20. Geburtstag der Java-EE-Plattform (die im Jahr 1998 erstmals präsentiert wurde) gleichzeitig ihr Todestag sein mag, so ist das meiste positiv zu sehen. Allerdings wird es wegen der vornehmlich Beitrags-finanzierten Eclipse Foundation und ihrer teils begrenzten Ressourcen in den Bereichen „Legal", „IP" oder „Patentwesen" trotz des Überschwangs auf der JavaOne 2017 über die „letzten drei Wochen vs. 13 Jahre" sicher gut 13 Monate dauern, bis die ersten Java-EE-JSRs bei EE4J landen werden. Im Extremfall könnten es sogar zwei bis drei Jahre werden, doch muss der Ehrlichkeit halber hinzugefügt werden, dies kommt von jemandem, der schon viel Vergleichbares in eigenen Projekten erlebt hat, ebenso in der Java-EE-Umbrella-Plattform seit Version 6 und in unzähligen JSRs.

Links

[1] *https://javaee.github.io*
[2] *https://github.com/javaee/glassfish*
[3] *http://javamoney.org*
[4] *https://javaee-guardians.io*
[5] *https://projects.eclipse.org/projects/ee4j/charter*
[6] *https://microprofile.io*
[7] *http://fnproject.io*
[8] *https://de.wikipedia.org/wiki/Liste_der_Schweizer_Bundespr%C3%A4sidenten*

Werner Keil
werner@catmedia.us

Werner Keil ist zurzeit als externer Java-EE-Consultant, Microservice-Architekt und Test-Automation-Experte im Automotive/Embedded-Bereich tätig sowie als Agile Coach, Java-Embedded- und Eclipse-RCP-Experte bei einem Anbieter von Embedded und Realtime-Systemen. Er hilft Global-500-Unternehmen aus Branchen wie „Mobil/Telekom", „Web 2.0", „Finanzen", „Tourismus/Logistik", „Autobau", „Gesundheit", „Umwelt & Öffentliche Hand" sowie IT-Anbietern, darunter Oracle oder IBM. Für einen Medien-Giganten entwarf er Mikro-Formate für die Suchmaschine eines Online-Musik-Shops und für eine Privat-Universität ein Portal, das heute gemeinhin als soziales Netzwerk gilt. Werner Keil entwickelt Enterprise-Systeme mithilfe von Java, Java EE, Oracle, IBM oder Microsoft und betreibt Web-Entwicklung mit Adobe, Ajax/JavaScript sowie dynamischen oder funktionalen Sprachen.


```
public abstract class NewJob {
   private final static int JUNIOR = 1;
   private Set<Expertise> wantedExpertise;
   public NewJob() {
      wantedExpertise = new HashSet(Arrays.asList(Expertise.values()));
   }
   public void apply(short jobLevel, Expertise javaExpertise) {
      if (jobLevel >= JUNIOR && wantedExpertise.contains(javaExpertise)) {
         applyToZertificon();
         workAtZertificon();
      } else {
         tellYourFriends();
      }
   }
   public void applyToZertificon() {
      Browser.open("https://www.zertificon.com/jobs");
      MailClient.sendMail("Bewerbung als Java-Entwickler", "jobs@zertificon.com");
   }
   private Benefits workAtZertificon() {
      final String city = "Berlin";
      final Boolean permanentPosition = true;
      final String[] products = {"E-Mail-Verschlüsselung", "Dateitransfer", "Zertifikatsmanagement"};
      return new Benefits("attraktive Konditionen", "flexible Arbeitszeiten", "nettes Team");
   }
   public enum Expertise {
      JAVA_SE, JAVA_EE, Spring
   }
   public abstract void tellYourFriends();
}
```

Das Ende der Finsternis

Markus Karg, Java User Group Pforzheim

Nachdem die Anwenderschaft, neudeutsch „Community", monatelang lautstark an der offensichtlichen Untätigkeit von Oracle herumgemeckert hat, ging es plötzlich ganz schnell: Java EE wurde an die Eclipse Foundation übergeben. Doch was bedeutet das im Detail? Ist Java EE nun tot und die JCP aufgelöst? Ein bewusst provokativer Kommentar aus Anwendersicht.

Plötzlich war es da: das lange geforderte Öffnen von Test Compatibility Kits (TCK), Reference Implementations (RI) und Specifications (Specs). Was auch immer der letztendliche Auslöser war, sei es die monatelange Nörgelei oder einfach nur der Wunsch, sich einer Altlast zu entledigen, der Community soll es recht sein: Java EE ist nun endlich und vollständig Open Source. Oder doch nicht? Naja, es kommt darauf an, wen man fragt! Oracle sieht das so, die Community nicht unbedingt.

Trau, schau wem …

Die Eclipse Foundation gibt sich zumindest schon mal demonstrativ euphorisch: Mike Milinkovic, Executive Directory eben jenes Industrie-Clubs, gab jüngst in der Keynote seiner Hausmesse EclipseCon Europe stolz die Parole aus, seine Organisation sei nun „der Hort der Innovation", da neben IBM OpenJ9 nun auch EE4J (also das bisherige Java EE) dort weiterentwickelt werde. Aber werden sie das denn wirklich? Fakt ist: Bislang tut sich rein gar nichts, sieht man vom allgegenwärtigen Auf-den-Busch-Klopfen einmal ab.

Zumindest tut sich nicht mehr als vorher. So schnell wird sich das auch nicht ändern. Auf die Frage, wo Oracle EE4J in einem Jahr stünde, will sich Oracle kaum mehr abringen lassen als ein „Aller Code ist von der Rechtsabteilung geprüft, auf die Server der Eclipse Foundation kopiert und besteht als das Java EE 8 TCK." Moment mal … Java 8? Wurde denn nicht behauptet, durch den Move zur Eclipse Foundation gehe dank der breiten Community-Beteiligung nun alles viel flotter? Schade. Also ein weiteres Jahr Stillstand.

Dafür kann sich die Community endlich beteiligen. Zum Beispiel mit

Contributions. Ach, halt, nein, doch nicht. Denn erstens sind die Committer nach einem bislang nicht veröffentlichten Schlüssel handselektiert und die meisten stammen weiterhin von Oracle. Zweitens, während des Moves können keine Contributions von Dritten angenommen werden, wie auch, sonst kommt die Rechtsabteilung beim Code-Review ins Trudeln. Drittens, was „später" (also nach besagtem Wartejahr) überhaupt von Dritten angenommen wird, entscheiden Personen, die noch gar nicht festgelegt sind, also gibt es dazu bis auf Weiteres keine Aussage – der Prozess ist noch gar nicht definiert. Das muss nichts Negatives bedeuten, kann es aber.

Apropos gewählt. Deren Wahl wird, wen hätte es gewundert, keineswegs demokratisch erfolgen (wie etwa beim Oracle-gelenkten JCP), sondern die Funktionsträger bei EE4J werden nach Gutdünken durch das Project Management Committee (PMC) bestimmt, ganz im Open-Source-Sinne eines „Benevolent Dictator", der nun mal aller Benevolenz zum Trotz eben doch nur ein Diktator ist. Ist es das, was die Community wollte: einen anderen Diktator? Und übrigens, klar, das PMC (initial besetzt durch Oracle, IBM, Red Hat, Payara, Tomitribe und Ivar Grimstad) wird natürlich auch nach Gutdünken von der Eclipse Foundation dominiert. Dort bestimmt, man mag es ja kaum sagen, ausschließlich die Industrie: Ein Stimmrecht ist an die Zahlung so erheblicher Euro-Summen gekoppelt, dass sich wohl kein Anwender erdreisten wird, ein Stimmrecht kaufen zu wollen – sei es ein ambitionierter Freelancer, ein JUG-Mitglied oder ein wissenschaftlicher Mitarbeiter einer Hochschule. Im Schwabenland würden wir sagen: „Das hat ein Gschmäckle" – außer es ist eben Open Source, da wird so etwas dann plötzlich als normal angesehen. In der Summe ist kein echter Vorteil gegenüber dem bisherigen Stewart auszumachen, im Gegenteil, die JCP war zumindest demokratisch und angesichts der kürzlichen Oracle-Wahlschlappe zum Thema „Jigsaw" bewiesenermaßen auch nicht zahnlos. Also alles nix, oder?

Und es ward Licht

Naja, ganz so schwarz malen sollte man es dann doch nicht. Letztendlich hat die Eclipse Foundation in den vergangenen Jahren hervorragende Arbeit geleistet. Mit der kostenlosen und offenen Eclipse-IDE hat die Stiftung Meilensteine gesetzt und liefert eine der nach wie vor beliebtesten Werkzeuge der Branche. Auch das Lizenzmodell ist sehr liberal. Im Unterschied zum bisherigen Status quo verleibt sich die Eclipse Foundation nämlich nicht das Intellectual Property (IP) jeder Contribution ein, sondern belässt diese (ganz in Einklang mit dem deutschen Urheberrecht) beim Autor. Sie erhält lediglich (ebenfalls im Einklang mit dem deutschen Urheberrecht) ein nicht-exklusives, unbeschränktes und unterlizenzierbares Verwertungsrecht.

Das ist wesentlich besser als die bisherige Übertragung des Urheberrechts an Oracle. Darüber hinaus soll es, so der Plan, grundsätzlich jedem möglich und sehr viel einfacher werden, durch Zeichnen eines Vertrags mit der Eclipse Foundation, gefolgt vom Stellen von Pull Requests auf GitHub, Contributor zu werden. Wenn das so kommt, ist auch dies für die Community ein Vorteil, da man bislang Papier und Fax benötigte und nicht alle Projekte Pull Requests überhaupt akzeptiert hatten. Wer viele Pull Requests stellt, kann dann

auf Ermessen der Projektleitung zum Committer erhoben werden. Das wäre ein echtes Novum bei einigen Projekten sowie ein großer Schritt in Richtung Community-Einbindung und würde sicherlich viele Verbesserungen in den Code bringen, die bislang an der abschottenden Oracle-Politik scheiterten.

Die Hoffnung stirbt zuletzt

Insofern könnte der Move zur Eclipse Foundation nun wirklich die leidigen Themen „Community- Beteiligung" und „Innovationsgeschwindigkeit" voranbringen – wenn das Wörtchen „wenn" nicht wäre. Denn auf Nachfrage Ende Oktober konnten Eclipse Foundation, PMC und auch Oracle nur wenig Licht in ein paar wichtige Fragen bringen. Doch diese sind es, die letztendlich über den Erfolg des Unterfangens entscheiden. Denn je nach Antwort bleibt sonst von der Aktion kaum mehr übrig als der Tausch des Titels, des Besitzers und des Regelwerks – wie beispielsweise bei EclipseLink, das als Oracle TopLink startete, später als EclipseLink eine Reinkarnation unter Eclipse-Leitung erlebte, um dann ebenso in Untätigkeit vor sich hin zu dümpeln wie seine Java-EE-Geschwisterprojekte, die den Schritt in Richtung Eclipse gerade erst vollziehen. Die gestellten Fragen und ihre Antworten im Einzelnen stehen im Kasten.

Fazit

Abwarten und Tee trinken! Erst wenn Oracle, PMC und Eclipse Foundation ihre Hausaufgaben komplett gemacht haben, ist eine finale Bewertung möglich. Hoffen wir, dass die in Aussicht gestellte Teilhabe der Community in der besagten Form kommen wird und dass sie schneller kommt, als angesagt. Denn nichts wäre schlimmer für Java EE als weitere Jahre der Untätigkeit – egal unter welcher Flagge.
Für mich als ISV-Entwicklungsleiter, der auf Standards und Produktunabhängigkeit angewiesen ist, sowie als Expert-Group-Mitglied (JSR 370) ist die aktuelle Situation fatal: Ich bin weiterhin in meiner Arbeit gehemmt, und zwar für mindestens ein weiteres Jahr. Was danach kommt, steht in den Sternen. Das kann gut oder schlecht sein. Ich muss akzeptieren, wie es kommt, und bereite mich auf zwei Szenarien vor: Die Sache geht schief, dann werden wir die Idee eines Industriestandards beerdigen können und wieder zu Produktabhängigen APIs wechseln müssen; oder die Sache klappt im Gegensatz zu anderen Oracle-Open-Source-Aktionen der letzten Jahre – das glaube ich allerdings einfach (noch) nicht. Mal sehen …

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.

Wer standardisiert zukünftig Java EE?
Die Eclipse Foundation ist keine Standardisierungsorganisation. Das kann möglicherweise weiter der JCP bleiben, es gibt jedoch keine spruchreife Entscheidung. Es ist noch nicht einmal klar, wer das überhaupt zu entscheiden hat. Der JCP arbeitet bislang wie gewohnt weiter und hält die üblichen Wahlen ab, zeigt sich aber in seinen Meetings entsprechend verunsichert. Man könnte den JCP auch einfach reformieren, daran scheint Oracle allerdings kein Interesse zu haben. Die Eclipse Foundation möchte im Prinzip einfach nur Code produzieren und zeigt sich etwas überrascht davon, dass die Community so stark an Standards interessiert ist.

Wie und wo sollen die bisherigen JSR Expert Groups weiter an Standards arbeiten?
Oracle findet es am sinnvollsten, wenn die Expert Groups einfach weitermachen wie bisher, was natürlich nicht funktioniert, da dies voraussetzen würde, dass Oracle nun aktiv in allen JSRs mitarbeitet – das war exakt das Hauptproblem der vergangenen Monate und ist es noch immer.

Wie können sich User Groups (wie iJUG oder JUGs) beteiligen?
Jeder Contributor muss einen Vertrag mit der Eclipse Foundation (also nicht mehr mit Oracle) unterzeichnen (Contributor Agreement). Der Arbeitgeber muss der Mitarbeit zustimmen, wie auch schon beim JCP. Tut er das nicht, schaut man vom Spielfeldrand aus zu. Contributions sind per GitHub als Pull Requests zu stellen. Das ist einerseits sehr willkommen für Code, geht jedoch nicht bei Diskussionen um Spezifikationen: Dieses Modell richtet sich rein auf die Programmierung von Produkten, nicht auf das Finden von Hersteller-übergreifenden Industriestandards.

Wie ist die künftige Beziehung OpenJDK/EE4J?
OpenJDK (also Java SE 9) und Java EE 8 teilen sich derzeit einige Packages. Diese sind in Java 9 bereits als „Deprecated for Removal" markiert und werden aus Java SE verschwinden. Sehr wahrscheinlich werden sie durch EE4J übernommen – aber eben nur wahrscheinlich. Solange Oracle die Kontrolle über Java SE und Java EE hatte, war das kein Problem. Doch was passiert, wenn Oracle die Packages streicht, und die Eclipse Foundation sie nicht weiter pflegt?

Wie werden die „javax.*"-Packages zukünftig genannt?
Sie werden nicht umbenannt, auch nicht bei neuen Versionen. Nur ganz neue, bislang nicht existierende APIs werden neue Packages mit anderem Präfix erhalten.

Wird es bald Maintenance-Releases geben?
Auf keinen Fall im Laufe des kommenden Jahres, denn Oracle ist voll mit dem Move beschäftigt und kkann daher viele neue Ideen erstmal weder selbst umsetzen noch als externe Contributions annehmen.

Bleiben zukünftige Major-Releases von EE4J abwärtskompatibel?
Das erste EE4J-Release wird abwärtskompatibel zu Java EE 8 sein. Bei zukünftigen Releases behält sich das PMC vor, das WORA-Prinzip (Write Once, Run Anywhere) zu brechen, da man einige unliebsame Technologien wie Common Object Request Broker Architecture (CORBA) nicht mehr für notwendig erachtet. Hier gibt es starken Widerstand in der Community, die Bestandscode weiterbetreiben können muss, ohne auf Produkt-Updates der Runtime beziehungsweise des Application-Servers zu verzichten. Hier zeigt sich, wie Wunsch und Wirklichkeit auseinanderdriften: Die Eclipse Foundation interessiert nur Innovation; die Community interessiert hingegen, bestehende Anwendungen möglichst lange pflegen zu können – denn ohne WORA braucht man kein Java.

Should I stay or should I Go?

Ralf Wirdemann, kommitment GmbH & Co. KG

Go ist eine einfache, statisch getypte und kompilierte Programmiersprache. Einfaches Deployment, hohe Performance und sehr guter HTTP- und JSON-Support machen sie zur ersten Wahl für die Entwicklung von REST-APIs. Durch hohe Lesbarkeit ist Go zudem eine Sprache für produktive Softwareteams.

Go wurde im Jahr 2007 als Antwort auf die zunehmende Komplexität der Software-Entwicklung von Google ins Leben gerufen. Inzwischen setzt neben Google eine Reihe weiterer bekannterer Firmen Go ein. Das bekannteste Beispiel ist sicherlich Docker, aber auch Firmen wie Dropbox sind dabei, ihre Software auf Go umzustellen. Bei Dropbox sind es mittlerweile fünfzehn Go-Teams, deren Repositories mehr als 1.3 Millionen Zeilen Go-Code enthalten. Ein anderes Beispiel sind die Wunderkinder aus Berlin, die ihre sehr erfolgreiche Aufgabenverwaltung Wunderlist von einem Rails-Monolithen auf eine Microservice-Architektur umgestellt haben, bei der die meisten Services in Go programmiert sind.

Dieser Artikel führt in die Grundlagen der Programmiersprache Go ein, erklärt die zugrunde liegenden Paradigmen und macht sich auf die Suche nach Gründen für die rasche Verbreitung von Go. Darüber hinaus beleuchtet der Artikel die Aspekte von Go, die nicht so gut gelungen sind. Aufbauend auf diesen Pros und Kontras schließt der Artikel mit einem Fazit und einer Antwort auf die Frage: „Should I stay or should I Go?".

Der Go-Workspace

Der Go-Workspace ist ein zentrales Verzeichnis, das über die Umgebungsvariable „GOPATH" referenziert wird. Darunter finden sich die drei Unterverzeichnisse „bin", „pkg" und „src", die Binaries, Packages und Sourcen der im Workspace verwalteten Projekte enthalten *(siehe Listing 1)*.

Die zugrunde liegende Idee ist, dass alle Go-Projekte inklusive der benötigten Abhängigkeiten an zentraler Stelle abgelegt sind. Pro-

jekte werden so schnell gefunden und können sich benötigte Bibliotheken teilen.

Die Sprache Go

Der Einstieg in ein Go-Programm erfolgt über die Funktion „main", die bei Programmstart ausgeführt wird *(siehe Listing 2)*. Das Programm lässt sich durch Eingabe des Kommandozeilen-Befehls „go run main.go" direkt ausführen.

Packages

Go-Files werden in Packages organisiert; alle Dateien eines Verzeichnisses gehören zum selben Package. Der letzte Teil des Pfadnamens bestimmt den Package-Namen, der als erstes Statement in den Dateien des Packages deklariert wird *(siehe Listing 3)*.

Packages definieren Namensräume: Alles Kleingeschriebene bleibt innerhalb des Packages, alles Großgeschriebene ist auch außerhalb des Packages sichtbar. Packages werden per „import"-Statement importiert. Öffentliche Typen, Variablen und Funktionen werden genutzt, indem ihnen der Package-Name vorangestellt wird. So wird die öffentliche Funktion „Greet" des Packages „service" wie in *Listing 4* importiert und aufgerufen.

Das „main"-Beispiel zeigt eine Ausnahme in Bezug auf die Namenskonvention für Packages: Die Go-Datei mit der Funktion „main" gehört zum Package „main", unabhängig vom Verzeichnis, in dem sie liegt.

Funktionen und Variablen

Funktionen in Go werden mit dem Schlüsselwort „func", gefolgt vom Namen einer optionalen Parameterliste sowie keinem, einem oder mehreren Rückgabewerten deklariert *(siehe Listing 5)*. Variablen sind mit dem Schlüsselwort „var" oder per Direktzuweisung deklariert *(siehe Listing 6)*. Bei der Direktzuweisung erkennt der Compiler den Typ der Variablen, sodass die explizite Angabe des Typs „string" entfällt und die Variable direkt initialisiert wird.

Go enthält die gängigen Standardtypen, wie „string", „int", „float32/64", „byte" oder „bool". Darauf aufbauend können benutzerdefinierte Typen entweder als einfache oder als zusammengesetzte Typen deklariert sein *(siehe Listing 7)*.

Zusammengesetzte Typen werden per Literal instanziiert, indem dem Typnamen eine in geschweiften Klammern angegebene Parameterliste übergeben wird, beispielsweise „g := Greeting{message: „Hello", greetee: „Java Forum"}".

Methoden

Eine Funktion wird zu einer Methode, indem dem Funktionsnamen der Typ vorangestellt ist, auf dem die Methode implementiert werden soll. Das Beispiel in *Listing 8* implementiert eine Methode „Greet" auf dem Typ „Greeting".

Innerhalb des Methodenrumpfs wird das Objekt, auf dem die Methode arbeitet, durch die dem Typ vorangestellte Variable repräsentiert (im Beispiel „g"). Der Aufruf einer Methode erfolgt über die Punkt-Notation *(siehe Listing 9)*.

Pointer

Go behandelt Funktionsparameter nach einer „Call by Value"-Se-

```
bin/
  treubuilder
pkg/
  linux_amd64/
     bitbucket.org/rwirdemann/javaforum/
        service.a
src/
  bitbucket.org/rwirdemann/javaforum/
     main.go
     service/
        hello.go
```

Listing 1

```
package main

func main() {
  println("Hello, Java")
}
```

Listing 2

```
// /home/ralf/gows/src/hello/service/greet.go
package service

func Greet(greetee string) {
}
```

Listing 3

```
package main

import "service"

func main() {
     service.Greet("Kalle")
}
```

Listing 4

```
func swap(x int, y int) (int, int) {
     return y, x
}
```

Listing 5

```
var vorname string // oder
nachname := "Meyer"
```

Listing 6

```
// Einfacher Typ basierten auf string
type message string

// Zusammengesetzter Typ
type Greeting struct {
     message string
     greetee string
}
```

Listing 7

```
func (g Greeting) Greet() string {
  return g.message + ", " + g.greetee
}
```

Listing 8

mantik. Entsprechend arbeiten Funktionen und Methoden auf Kopien ihrer Parameter beziehungsweise Objekte. Gemäß dieser Semantik ist die Methode „Greet" Seiteneffekt-frei, da sie nicht auf dem in „main" instanziierten „greeting"-Objekt, sondern auf einer Kopie davon arbeitet.

Seiteneffekte werden in Go durch den Einsatz von Pointern erzwungen. Ein Pointer ist eine Variable, die statt eines konkreten Wertes eine Speicheradresse enthält. Diese Adresse zeigt auf die Stelle im Hauptspeicher, an der der eigentliche Wert gespeichert ist. Pointer werden mit einem dem Typ vorangestellten Stern gekennzeichnet: „func (g *greeting) Greet() string {}".

Ein Pointer wird entweder über die Funktion „new" oder den Operator „&" erzeugt: „greeting := &Greeting{message: „Hello", greetee: „Java Forum"}". Methodenaufrufe auf Pointern entsprechen denen auf normalen Variablen: „greeting.Greet()". Der entscheidende Unterschied: Die Methode „Greet" arbeitet jetzt nicht mehr auf einer Kopie von „greeting", sondern auf der in „main" instanziierten Version. Entsprechend können Methoden auf Pointer-Typen die an sie gebundenen Objekte verändern.

Ein Vorteil der standardmäßigen „Call by Value"-Semantik von Go ist, dass Funktions- und Methodenaufrufe immer immutable sind, solange sie keinen globalen Zustand verändern. Erst durch die Verwendung von Pointern wird einer Funktion beziehungsweise Methode explizit erlaubt, Seiteneffekte zu erzeugen.

Interfaces

Go-Interfaces haben einen Namen und definieren eine Reihe von Methoden *(siehe Listing 10)*. Ein Typ erfüllt ein Interface, wenn er alle Methoden des Interface implementiert. Entsprechend erfüllt Greeting das Interface Printable, wenn Greeting die Methode „print" implementiert *(siehe Listing 11)*. Interfaces können überall dort eingesetzt werden, wo zuvor der konkrete Typ verwendet wurde, zum Beispiel als Funktionsparameter *(siehe Listing 12)*.

Arrays und Slices

Neben den bekannten Basistypen enthält Go mit Arrays und Slices zwei weitere wichtige Typen. Arrays sind Listen gleichen Typs mit fester Länge: „a := [3]int{1, 2, 3}". Arrays sind immutable und können nur verändert werden, indem sie kopiert werden. Beispielsweise erzeugt die Funktion „append" ein neues Array, das aus dem übergebenen Array und dem im zweiten Parameter übergebenen Element besteht: „a = append(a, 4)".

Die eigentliche Arbeit auf Arrays erfolgt mithilfe von Slices. Im Gegensatz zu Arrays besitzen Slices keine feste Länge, sondern definieren eine dynamische Sicht auf Arrays: „var s[]int = a[1:3]". Die Slice „s" enthält die Elemente „2" und „3". Wird einem Element der Slice ein neuer Wert zugewiesen, ändert sich nicht das unterliegende Array, sondern es wird intern eine Kopie des Arrays erzeugt und der Variable „a" zugewiesen: „s[0] = 1 // a => [1, 1, 3]".

Maps

Neben Arrays sind Maps der zweite wichtige Container-Datentyp in Go. Maps enthalten Key-Value-Paare und werden von der Funktion „make" oder per Literal erzeugt: „m := make(map[string] Contact)". Die Zuweisung von Elementen erfolgt durch Angabe des

```
func main() {
  greeting := Greeting{message: "Hello2, greetee:
"Java"}
  greeting.Greet()
}
```

Listing 9

```
type Printable interface {
  print() string
}
```

Listing 10

```
func (g Greeting) print() string {
  return fmt.Sprintf("%s, %s", g.message, g.greetee)
}
```

Listing 11

```
func printOnConsole(printable Printable) {
  fmt.Printf("=> %s", printable.print());
}
```

Listing 12

```
for i := 0; i < 10; i++  // index-basierte Iteration
for i < 10               // while
for                      // for ever
```

Listing 13

```
switch r.Method {
case "GET":
  ...
case "POST":
  ...
default:
  ...
}
```

Listing 14

Keys in geschweiften Klammern: „m[„ralf"] = Contact{name: „Ralf Wirdemann"}". Bereits vorhandene Werte werden überschrieben. Die Überprüfung, ob die Map einen Wert für einen bestimmten Key enthält, erfolgt über das Ok-Idiom: „if v, ok := m[„ralf"]; ok {}". Die Variablen „v" und „ok" sind nur innerhalb des „if"-Blocks gültig.

Schleifen und Bedingungen

Go kennt nur eine Schleife, die „for"-Schleife. Je nach Parameter erfüllt die Schleife unterschiedliche Zwecke *(siehe Listing 13)*.

Für Verzweigungen kennt Go die beiden Schlüsselworte „if" und „switch". „if" haben wir bereits im Zusammenhang mit Maps kennengelernt. Das Besondere am „switch"-Statement ist der Wegfall von „break" in den einzelnen „case"-Zweigen *(siehe Listing 14)*.

Das Schlüsselwort „break" ist optional und kann verwendet werden, um einen „case"-Block vor der kompletten Abarbeitung abzuschließen. Das optionale Schlüsselwort „fallthrough" kann am Ende eines

```
switch {
case height <= 4:
  fmt.Println("Short")
case height <= 5:
  fmt.Println("Normal")
case height > 5:
  fmt.Println("Tall")
}
```

Listing 15

```
go run main.go    // führt main.go direkt aus
go build hello    // übersetzt Projekt "hello",
                  // Binary landet im aktuellen Ver-
zeichnis
go install hello  // übersetzt Projekt "hello",
                  // Binary im bin Verzeichnis
go test           // führt Tests im aktuellen Ver-
zeichnis aus
```

Listing 16

```
type Engine struct {
  hp int
}

type Car struct {
  Engine
}
```

Listing 17

```
func (e *Engine) start() {...}

func main() {
  var c Car
  c.hp = 100
  c.start()
}
```

Listing 18

```
type Animal interface {}

kalle := Dog{"Kalle"}
felix := Cat{"Felix"}

animals := [2]Animal{kalle, felix}
for _, v := range animals {
  fmt.Printf("Hey, %s\n", v.Talk())
}
```

Listing 19

```
func foo(fn func(x int) int) int {
  return fn(2)
}

func main() {
  f := func(x int) int {
     return x * x
  }
  foo(f)
}
```

Listing 20

„case"-Blocks eingesetzt werden, um die nachfolgenden „case"-Blöcke in die weitere Auswertung mit einzubeziehen. Eine weitere Verwendungsart von „switch" ist die Ersetzung der Variablen im „switch"-Teil durch einzelne Bedingungen in den „case"-Zweigen *(siehe Listing 15)*. Diese Variante wird häufig als lesbare Version hintereinandergereihter „if"-Statements verwendet.

Das Werkzeug Go

Das zentrale Go-Werkzeug heißt „go". Einmal installiert, lassen sich alle wichtigen Aufgaben auf der Kommandozeile ausführen. *Listing 16* zeigt einige Beispiele. Go ist eine Multi-Paradigmen-Programmiersprache, die sowohl Konzepte der objektorientierten als auch der funktionalen Programmierung unterstützt.

Objektorientierte Programmierung in Go

Objektorientierung ist durch drei Eigenschaften gekennzeichnet: Kapselung, Vererbung und Polymorphismus. Kapselung wird in Go durch Packages und das Konzept der Groß- (außerhalb sichtbar) und Kleinschreibung (Package-intern) realisiert. Statt auf Vererbung setzt Go auf Komposition. Im Beispiel in *Listing 17* komponiert der Typ „Car" eine Reihe von Bauteilen, aus denen ein Auto besteht.

Attribute und Methoden enthaltener Typen können direkt auf Variablen des komponierenden Typs aufgerufen werden. So lässt sich eine auf „Engine" implementierte Methode „start" direkt auf einer „Car"-Instanz aufrufen *(siehe Listing 18)*. Die Modellierung basiert auf Komposition, während die Benutzung komponierter Typen der einer vererbten Typ-Hierarchie entspricht.

Polymorphismus

Polymorphismus bezeichnet die Eigenschaften, zur Laufzeit andere Formen annehmen zu können. Erbt beispielsweise in Java die Klasse B von A, dann dürfen Objekte der Klasse A zur Laufzeit auf Objekte der Klasse B zeigen. Die Variable vom Typ A nimmt zur Laufzeit die Form B an. In Go existiert kein Vererbungs-basierter, dafür aber ein Interface-basierter Polymorphismus *(siehe Listing 19)*.

Im Beispiel wird das Interface „Animal" von den beiden Typen „Dog" und „Cat" implementiert. Entsprechend ist es möglich, ein Array vom Typ „Animal" mit Instanzen unterschiedlicher Typen zu initialisieren, sofern sie dieses Interface implementieren.

Funktionale Programmierung

Die wesentlichen Eigenschaften funktionaler Programmiersprachen sind Immutability, Seiteneffekt-Freiheit sowie die Behandlung von Funktionen als „First Class"-Objekte. Die Eigenschaften Immutability und Seiteneffekt-Freiheit haben wir bereits im Abschnitt über Pointer kurz erörtert, sodass wir in diesem Abschnitt nur noch auf den Umgang mit Funktionen eingehen.

Go-Funktionen sind „First Class"-Objekte, sie können also überall dort eingesetzt werden, wo sonst normale Variable genutzt werden. Das Beispiel in *Listing 20* weist einer Funktion der Variablen „f" zu und übergibt diese Variable und damit die Funktion einer anderen Funktion „foo". Zusätzlich lassen sich eigene Funktionstypen deklarieren *(siehe Listing 21)*.

Die Beispiele dieses Abschnitts zeigen, dass Go weder rein objektorientiert noch rein funktional ist. Dennoch lassen sich Konzepte aus beiden Welten in lesbarer Form umsetzen.

Warum Go?

Go ist eine produktive und leicht zu erlernende Programmiersprache. Erfahrene Java-Programmierer sollten Go innerhalb einer Woche lernen und produktiv einsetzen können. Standards und Konvention sorgen dafür, dass Go-Code sehr gut lesbar ist. Die reduzierte Syntax führt zudem dazu, das Programmierer dieselben Sprachkonstrukte nutzen und sich nicht gemäß ihrer persönlichen Präferenz für das ihnen liebste Sprachkonstrukt entscheiden. So gibt es in Go nur eine einzige Schleife, die „for"-Schleife, oder nur eine gültige Form der Klammersetzung in „if"-Blöcken. Go-Code ist immer gleich formatiert, ungenutzte Imports oder Variablen und Funktion sind Fehler.

Go ist eine typsichere Programmiersprache – eine Eigenschaft, die viele Programmierer aufgrund der deutlich besseren Refaktorisierbarkeit typisierter Sprachen schätzen. Das Go-Typ-System ist dabei äußerst leichtgewichtig und steht dem Programmierer nicht im Weg. Typ-Inferenz sorgt dafür, dass Programmierer den Typ nur dann angeben müssen, wenn der Compiler den Typ nicht automatisch erkennen kann. So wird der Typ der beiden Zielvariablen in *Listing 22* automatisch erkannt.

Neben Produktivität und statischer Typisierung ist Performance ein weiterer wichtiger Grund für den Einsatz von Go. Go-Programme kompilieren zu Assembler und sind in ihrer Ausführungsgeschwindigkeit vergleichbar mit C. Die Go-Standard-Bibliothek enthält derzeit an die 160 Pakete, die für viele typische Programmierprobleme Standardlösungen bieten. Ein gutes Beispiel sind die Bibliotheken „net/http" und „encoding/json", die alles für die Programmierung von REST-APIs enthalten.

```
type myfunction func(x int) int

func foo(fn myfunction) int { ... }
```

Listing 21

```
func swap(x int, y int) (int, int) {
  return y, x
}

i, j := swap(1, 2)
```

Listing 22

```
if file, err := os.Open(file); err == nil {
}
```

Listing 23

Go-Programme lassen sich für unterschiedliche Zielplattformen cross-kompilieren. Ergebnis ist ein statisch gelinktes Binary, das per einfaches Secure-Copy auf die Zielmaschine kopiert wird. Dort müs-

sen weder VMs installiert noch Bibliotheken, wie Jars oder RubyGems, in einer bestimmten Version existieren. Das Programm kann direkt ausgeführt werden. Deployments werden so zum Kinderspiel.

Was nervt?

Genau wie in jeder anderen Programmiersprache gibt es in Go eine Reihe von Kritikpunkten, die im Rahmen einer Gesamtbetrachtung nicht unter den Tisch fallen dürfen. So kennt Go keine Exceptions. Stattdessen geben Funktionen, bei deren Ausführung es zu Fehlern kommen kann, als letzten Rückgabewert eine Instanz vom Typ „error" zurück. Entsprechend ist die Zuweisung von Funktionsergebnissen bei gleichzeitiger Überprüfung des Fehlerwerts ein häufig anzutreffendes Go-Idiom *(siehe Listing 23)*.

Der Verwendung von Fehler-Rückgabewerten ist grundsätzlich keine schlechte Idee, zumindest dann nicht, wenn die Programmiersprache mehrere Rückgabewerte erlaubt. Fehler-Rückgaben sind allerdings sinnlos, wenn sie nicht direkt im Anschluss an den Funktionsaufruf geprüft werden. Das bringt uns zum eigentlichen Minuspunkt des Go-Fehlerbehandlungskonzepts: Einbußen bei der Lesbarkeit. Bei mehreren hintereinander folgenden Funktionsaufrufen mit potenziell fehlerhaftem Ausgang wird aus eigentlich gut lesbarem Code eine „if"-Kaskade, die deutlich schlechter zu lesen ist.

Andererseits lässt sich auch argumentieren, dass die Fehlerbehandlung in Go eng an die Fehler-verursachende Funktion heranrückt. So wird erstens schnell erkannt, ob Fehler behandelt werden, und zweites wird klar, welche Funktion welche Art von Fehler liefert.

Fehlende Unterstützung von Generics ist einer der am häufigsten geäußerten Kritikpunkte. Go verfügt zwar mit Arrays und Maps über zwei generische Container-Typen, erlaubt jedoch nicht die Implementierung eigener generischer Typen.

Go ist eine Sprache für die Entwicklung von Tools und Services, die schnell und robust sein müssen, dabei aber eher im Hintergrund oder auf der Kommandozeile arbeiten. Beispiele sind Webservices, DevOps-Tools oder auch die Dropbox-Bandbreitenregulierung. Go enthält gute Bibliotheken für die Kommandozeilen-Programmierung, aber keine Bibliothek für die Entwicklung grafischer Benutzeroberflächen. Neben dem Wunsch nach Generics sind GUI-Bibliotheken eine weitere häufig geäußerte Forderung an zukünftige Go-Versionen.

Einfaches, aber unzureichendes Dependency-Management

Das Go-Dependency-Management-Konzept scheint auf den ersten Blick charmant einfach. Statt der Verwendung eines zusätzlichen Tools mit komplizierter XML-Beschreibungssprache wird das benötigte Package einfach importiert und die zugehörige Bibliothek via „go get" in den Workspace geladen.

Spätestens in größeren Projekten und bei mehreren Projekten im selben Workspace zeigt sich jedoch schnell ein Problem. Angenommen, der Workspace enthält zwei Projekte, die beide dieselbe Bibliothek referenzieren, diese aber in unterschiedlichen Versionen benötigen. Oder man hat eine Bibliothek entwickelt, die eine andere Bibliothek in einer ganz bestimmten Version benötigt. Wie stellt man sicher, dass die Benutzer ihrer Bibliothek die benötigte Bibliothek genau in der erwarteten Version in ihrem Workspace haben?

Die hauseigenen Go-Werkzeuge bieten bisher keine Lösungen für die genannten Probleme. Allerdings hat die Community hier schon einen Schritt vorgelegt und mit „godep" ein Werkzeug entwickelt, um das Management von Abhängigkeit in bestimmten Versionen zu ermöglichen.

Fazit

Go besticht durch Einfachheit. Sowohl Sprache als auch Tools sind einfach erlern-und benutzbar. Standards und Konventionen erzwingen gut lesbaren Code, was die Sprache zu einer produktiven Sprache für Teams macht.

Der Go-Sprachumfang ist stark reduziert und besteht aus nur fünfundzwanzig Schlüsselwörtern. Im Vergleich dazu: Java hat fünfzig und Swift sogar achtundfünfzig Schlüsselwörter. Dennoch ist die Sprache sehr ausdrucksstark und unterstützt sowohl objektorientierte als auch funktionale Sprachkonzepte.

Eine im März 2017 veröffentlichte Umfrage hat ergeben, dass die Mehrheit der Entwickler Go für die Entwicklung von Webservices einsetzt. Die Standard-Library liefert bereits so gute HTTP-/JSON-Unterstützung, dass der Einsatz eines zusätzlichen Web-Frameworks entfallen kann. Die entwickelten Services sind schnell und robust und lassen sich durch einfaches Kopieren auf die Zielmaschine deployen.

Go-Entwickler behaupten: „Simplicity is addictive". Einfachheit und Produktivität machen Spaß. Und genau das ist der Punkt: Go-Programmierer sind zufriedene Entwickler, die ihre Werkzeuge lieben, Ergebnisse liefern und Unternehmen erfolgreich machen.

Referenzen und Links

- *https://golang.org*
- *https://tour.golang.org*
- *https://gobyexample.com*

Ralf Wirdemann
ralf.wirdemann@kommitment.biz

Ralf Wirdemann ist Agile Developer Coach und hilft Teams, sowohl technisch als auch methodisch auf Spur zu kommen. Guter Sourcecode ist ihm genauso wichtig wie gute User Stories, ein fürs Team passender Entwicklungsprozess oder eine funktionierende Deployment-Pipeline.

Vavr.io – Objekt-funktionale Programmierung leicht gemacht

David Schmitz, Senacor Techologies

Funktionale Programmierung hat in den letzten Jahren eine Art zweite Blüte erlebt und ist nun auch im Mainstream angekommen. In Folge hat Java 8 einige Konstrukte der funktionalen Programmierung in Java eingeführt, die im Tagesgeschäft jedoch Lücken offenbaren. Das Open-Source-Projekt Vavr (siehe „http://www.vavr.io“) will genau diese Lücken schließen. Sinnvolle, praktische Ideen und Konzepte aus Sprachen wie Clojure und Scala können mit Vavr nun auch in Java auf einfache Art genutzt werden.

Wir wollen hier keinen theoretischen Exkurs in die Vor- und Nachteile funktionaler Programmierung machen. Dennoch sei der Vorteil der funktionalen Programmierung anhand eines kleinen Beispiels verdeutlicht. Funktional geschriebene Programme vermeiden unnötige Seiteneffekte. Dazu zählen auch die im Java-Umfeld standardmäßig genutzten Collections, die eine beliebte Fehlerquelle sind. Der Code konzentriert sich auf das, was man eigentlich erreichen will, statt in den Details des „Wie“ zu ertrinken. Deshalb spricht man hier von deklarativer Programmierung. Ein einfaches Beispiel ist die Verwendung von „for“-Schleifen im Vergleich zu Operationen wie „map“ *(siehe Listing 1)*.

```
String[] names = new String[]{"Foo", "Bar", "Baz"};

// imperativ
String[] upper = new String[names.length];
for (int i = 0; i < names.length; i++) {
  upper[i] = names[i].toUpperCase();
}

// funktional mit Vavr
String[] upper2 = Array.of(names)
                       .map(String::toUpperCase)
                       .toJavaArray(String.class);
```

Listing 1

```
listOfUsers
  .stream()
  .filter(user -> {....})
  .collect(Collectors.toList());
```

Listing 2

```
listOfUsers
  .stream()
  .filter(user -> {
     try {
        return user.validate(); // throws Exception
     } catch (Exception ex) {
        return false;
     }})
  .collect(Collectors.toList());
```

Listing 3

Im imperativen Fall ist die eigentliche Logik „Umwandlung in Großbuchstaben“ in einem Wust von Boilerplate-Code verborgen. Der funktional geschriebene Code liest sich wie Prosa: Wir nehmen ein Array und wandeln jedes Element in Großbuchstaben um. Funktionale Programmierung hilft uns dabei, die Komplexität unserer Programme zu reduzieren.

Das Problem mit Java 8

Java 8 hat viele Sprach-Erweiterungen eingeführt, die aus anderen Sprachen lange bekannt waren und auf die viele Entwickler gewartet haben, wie die Erweiterung der Collections um Operationen wie „filter“ und „map“ oder die Einführung von Lambdas. Leider hat sich dabei auch schnell Ernüchterung breit gemacht. Die Verbindung von „Streams“ und „Collectors“, um überhaupt „filter“ nutzen zu können, führt beispielsweise oft zur „Pipeline-Hölle“ *(siehe Listing 2)*.

Auch die Verwendung von Lambdas mit Checked Exceptions ist nicht ganz so einfach, wie initial gehofft, und zwingt den Programmierer zu umständlichen Konstrukten wie im Beispiel von *Listing 3*.

Da Lambdas in Streams keine Checked Exceptions werfen dürfen, müssen diese umständlich gefangen und gekapselt werden. All dies hat zur Entwicklung einer Vielzahl von Bibliotheken geführt, die unter anderem diese Lücken schließen wollen. Wie auch vor knapp vier Jahren Javaslang, das nun als Vavr fortgesetzt wird. Im Kern konzentriert sich Vavr auf:

- Funktionale Datenstrukturen, also persistente, unveränderliche Datenstrukturen wie „List“, „Set“ oder „Seq“
- Abstraktionen für Werteklassen, beispielsweise Konstrukte wie „Option“ und „Tuple“
- Funktionalen Zucker, also „Lift“, „Curry“ und andere Konstrukte, die aus funktionalen Sprachen bekannt sind
- Funktionale Ausnahmebehandlung
- Strukturelle Dekomposition wie Pattern Matching auf Objekt-Instanzen

Wir können nicht auf alle Aspekte eingehen, sondern werden nur ein paar Highlights beleuchten, die im Tagesgeschäft direkten Nutzen bringen.

Funktionales Java, aber bitte ohne großen Ballast

Um die grundlegenden Ideen von Vavr zu verstehen, nimmt man am besten die Tastatur in die Hand und fängt an zu programmieren. Unter *„https://github.com/koenighotze/vavr-kata-demo“* findet sich ein Demo-Projekt, das wir für unsere Diskussion nutzen. Ich lade dazu ein, das Repository zu klonen und die Anpassungen schrittweise selber auszuprobieren. Die Demo ist eine Spring-Boot-Anwendung *(siehe „https://projects.spring.io/spring-boot“)*, die Sport-Teams verwalten soll. Wie bei Spring Boot üblich, startet man die Anwendung mit „gradle bootRun“.

Die Anwendung exponiert die Ressource „Team“ über einen entsprechenden Endpunkt *„http://localhost:8080/teams“*. Dieser unterstützt lesende Zugriffe über „GET“ auf *„http://localhost:8080/teams“* für alle Teams und „GET“ auf *„http://localhost:8080/teams/{id}“* für das Team mit der Id „{id}“. Über den Pfad *„http://localhost:8080/teams/{id}/logo“* kann zu dem Team mit der Id „{id}“ das zugehörige Wappen angefragt werden.

Wir nutzen die Anwendungen, um punktuell den Nutzen von Vavr zu verdeutlichen. Dabei legen wir bewusst den Blick auf einfache Vavr-Funktionen, die es erlauben, Schritt für Schritt Code zu verbessern und die Vorteile funktionaler Programmierung nutzen zu können. Die jeweiligen Anpassungen können auf den Branches „step1-*“ bis „step7-*“ nachvollzogen werden.

Optionale Ergebnisse

Als erstes Beispiel schauen wir uns die typische Nutzung von Null-Werten an. Null ist der größte Fehler aller Zeiten, wenn man der allgemeinen Meinung Glauben schenkt. Leider ist null häufig noch immer Teil der täglichen Arbeit. Dazu ein Beispiel aus der Demo-Anwendung: Die Methode „TeamInMemoryRepository#findById“ liefert „null“, wenn das Team nicht gefunden wurde. Der Controller muss entsprechend reagieren *(siehe Listing 4)*.

Funktionale Programmiersprachen umgehen diese „null-pointer“-Semantik durch den „Maybe“- beziehungsweise „Option“-Typ. Dieser repräsentiert entweder einen Wert, falls verfügbar, oder eben die Abwesenheit eines Wertes.

Auch Vavr hat seine Variante; wir nutzen hier „Option<T>“. Sie repräsentiert die Anwesenheit eines Wertes durch die Unterklasse „Some<T>“ und die Abwesenheit durch „None<T>“. Wir verbessern nun die Semantik und ändern das Repository, sodass es ein „Option<Team>“ zurückliefert *(siehe Listing 5)*.

Um nun das Ergebnis zu nutzen, verwenden wir "map", das Schweizer-Armee-Messer der funktionalen Programmierung *(siehe Listing 6)*.

Das Ergebnis liest sich wie ein einfacher Text: „Suche ein Team mit der id „id“. Falls es gefunden wird, liefere das Team und den Status „ok“, ansonsten liefere „not found“ zurück“.

Persistente Collections

Javas Collections sind ein häufiger Quell für Fehler *(siehe Listing 7)*. Die Collection mit „Collections.unmodifiableList“ unveränderbar zu machen, ist leider nicht hilfreich. Für den Nutzer der unveränderbaren Liste ist das nicht sichtbar und erst zur Laufzeit offenbart sich das Problem mit einer hässlichen Ausnahme *(siehe Listing 8)*.

```
Team team = teamRepository.findById(id);
if (null == team) {
  return notFound().build();
}
return ResponseEntity.ok(team);
```

Listing 4

```
Option<Team> team = teamRepository.findById(id);
```

Listing 5

```
teamRepository
  .findById(id)
  .map(ResponseEntity::ok)
  .getOrElse(() -> notFound().build());
```

Listing 6

```
Set<String> teams = new HashSet<>(asList("F95", "FCK"));
Set<String> moreTeams = teams;
moreTeams.add("STP");

// Ooops...Test schlägt fehlt
assertThat(teams).hasSize(2);
```

Listing 7

```
Set<String> teams = new HashSet<>(asList("F95", "FCK"));
Set<String> unmodTeams = unmodifiableSet(teams);

// Ooops...Exception!
unmodTeams.add("Boom");
```

Listing 8

```
TreeSet<String> moreteams = teams.add("STP");

assertThat(teams).hasSize(3);
assertThat(moreteams).hasSize(4);
```

Listing 9

```
String readTeamLogo (Team t) throws InterruptedExcep-
tion,… {
  return supplyAsync(() -> {
     try {
        return fetchRemoteLogo(t);
     } catch (IOException e) {
        throw new RuntimeException(e);
     }
  })
  .get(3000, MILLISECONDS);
}

String fetchRemoteLogo (Team t) throws IOException {
  ...
}
```

Listing 10

Vavr vermeidet solche Probleme und bringt eine Vielzahl eigener Collections mit. Diese reichen von einer einfachen „List“ bis hin zu eher spezialisierten Datenstrukturen wie „PriorityQueue“. Alle Vavr-Collections sind funktionale Datenstrukturen, sie sind also unveränderlich und persistent. Was das bedeutet, wird an einem einfachen Beispiel klar. Angenommen, wir haben eine „TreeSet“ von Teams

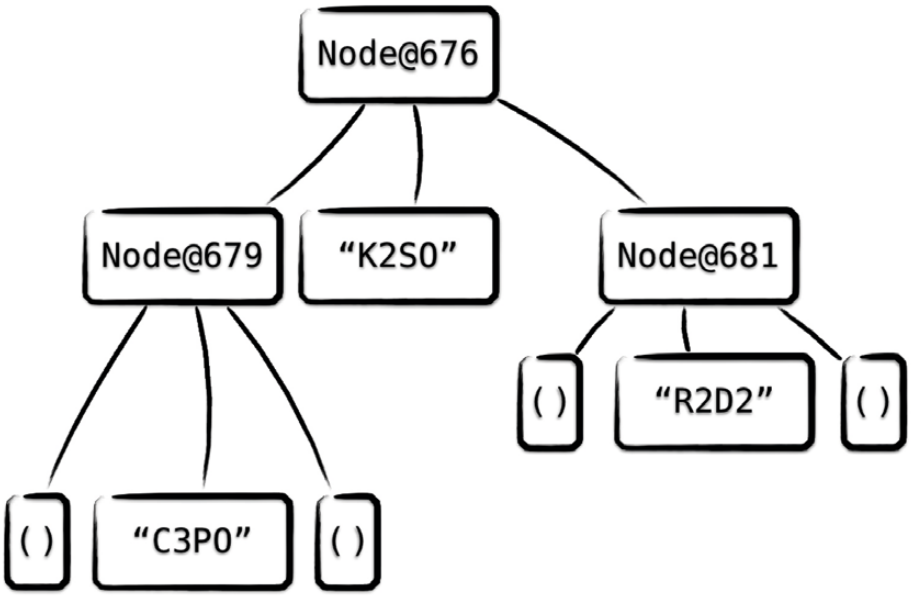


Abbildung 1: Vavrs TreeSet ist ein Baum, der aus Knoten (wie „Node@681“) und Werten (wie „R2D2“) besteht. Leere (End-) Knoten sind als „()“ dargestellt

„TreeSet<String> teams = TreeSet.of(“F95”, “FCK”, “FCB”);“ und fügen dieser Menge ein Element hinzu *(siehe Listing 9)*.

Vavr verändert nicht die ursprüngliche „TreeSet“, es wird eine neue erzeugt und durch geschickte Zeiger-Manipulation ein Großteil des Ursprungsbaums wiederverwendet. *Abbildung 1* zeigt, über die „TreeSet“ von „teams“, wie das konkret funktioniert.

Wird ein Element zugefügt, kann Vavr einen Großteil der ursprünglichen Datenstruktur weiter nutzen und muss nur geschickt das neue Element einfügen *(siehe Abbildung 2)*.

Es sollte klar sein, dass solche Operationen in der Implementierung von Vavr recht komplex sind. Für den Nutzer ist die Komplexität glücklicherweise völlig transparent. Wer sich um die Performance sorgt, sei hier auf die Micro-Benchmarks von Vavr verwiesen. Wir bevorzugen immer eine seiteneffektfreie Implementierung und optimieren nur, wo es Sinn ergibt. Im Zweifel ist der Übergang zwischen Vavr- und Java-Collections immer nur ein „toJavaSet“ entfernt.

Funktionale Ausnahmebehandlung

Exceptions und funktionale Programmierung sind in Java 8 ein trauriges Kapitel. Werfen wir einen Blick auf die Methode „TeamsController#readTeamLogo“ *(siehe Listing 10)*.

Hier wird versucht, über die Methode „fetchRemoteLogo“ das Wappen eines Teams zu laden. Dabei kann eine „IOException“ geworfen werden. Das Ganze soll asynchron über „CompletableFuture#supplyAsync“ mit einem Timeout von drei Sekunden geschehen („CompletableFuture#get“). Daher muss die „IOException“ gefangen und irgendwie sinnvoll behandelt werden: Hier wird sie als „RuntimeException“ gekapselt.

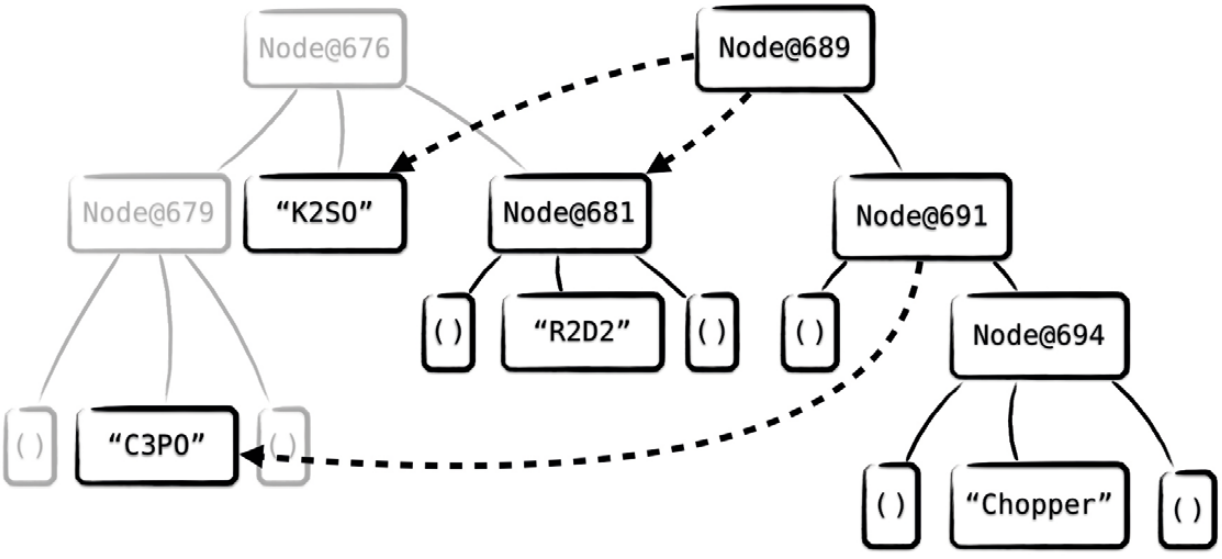


Abbildung 2: Wird der Wert „STP“ hinzugefügt, verwendet Vavr einen Großteil des Ur-Baums wieder – hier als gestrichelte Pfeile illustriert

Schließlich kann die Verwendung von „CompletableFuture“ selbst noch einen ganzen Zoo an Exceptions werfen, unter anderem „InterrupedException“. Offensichtlich ist das ganz schön viel Code für eine doch recht simple Semantik. Mit den Vavr-„Try“- und -„Lift“-Konstrukten wird die Fehlerbehandlung funktional und übersichtlich.

Als Erstes kapseln wir „fetchRemoteLogo“ so, dass keine Exception mehr geworfen wird. „fetchRemoteLogo“ ist eine partielle Funktion: Sie ist nur für Teams definiert, deren Logo geladen werden kann. Für alle anderen ist der Rückgabewert nicht definiert. Mit Vavr „CheckedFunction1.lift“ machen wir aus dieser partiellen Funktion eine totale Funktion; die obskure Syntax ist leider Java geschuldet *(siehe Listing 11)*.

„liftedReader“ liefert als Ergebnis ein „Option<String>“, sodass wir einfach im Fehlerfall ein Default-Wappen nutzen können: „liftedReader.apply(team).getOrElse(default);“. Lift ist ein wunderbares Mittel, um Code zu kapseln, den man selber nicht mehr modifizieren kann – beispielsweise wenn man in einem Brownfield-Projekt arbeitet und Legacy Code nutzen muss. Wir nutzen die geliftete Variante nun in der Implementierung von „readTeamLogo“ *(siehe Listing 12)*.

Besser, aber wir haben noch immer die ganzen Exceptions der „CompletableFuture“. Hier kommt Vavr „Try“ zu Hilfe. „Try<T>“ kapselt einen Funktionsaufruf, der eine Exception werfen kann. Das Ergebnis ist dann entweder ein „Success<T>“ mit dem Ergebnis oder ein „Failure<T>“ mit der geworfenen Exception. Sowohl „Success“ als auch „Failure“ lassen sich wie „Option“ nutzen. Das nutzen wir nun in der Implementierung von „readTeamLogo“ *(siehe Listing 13)*.

Diese Funktion drückt bereits durch die Signatur und den Typ der Rückgabe aus, was zu erwarten ist: „Try<String>“ – ein Versuch, einen String zu liefern, der entweder glückt oder eben nicht. Darüber hinaus bietet „Try“ Operationen, die man auch von „composable functions“ kennt. Man kann ohne Weiteres mit „andThen“ etc. mehrere Aufrufe nacheinander funktional kombinieren, ohne in einen Catch-Sumpf abzutauchen. „Try“ bietet aber noch viel mehr. So kann man damit die explizite Behandlung von Ausnahmen sauberer als mit „catch“-Blöcken ausdrücken *(siehe Listing 14)*.

Mit „recover(Exceptiontype, recoverFunction)“ kann auf Ausnahmen vom Typ „Exceptiontype“ ein Ersatzergebnis mithilfe der „recoverFunction“ konstruiert werden.

Rund um Vavr

Im Vavr-Ökosystem ist noch einiges mehr zu finden. Vavr-Jackson *(siehe „https://github.com/vavr-io/vavr-jackson“)* ist eine Bibliothek, um Vavr-Datentypen in REST-Schnittstellen direkt nutzen zu können. Vavr-Test ist eine Erweiterung für Property Based Testing, ähnlich wie ScalaCheck. Resilience4J *(siehe „https://github.com/resilience4j/resilience4j“)* ist eine Circuit-Breaker-Implementierung auf Basis von Vavr. Damit kann man mit einfachen Mitteln Java-Funktionen mit Circuit-Breakern ausstatten, ohne auf komplexere Frameworks wie Hystrix zurückgreifen zu müssen. Mit Spring Data 2 lassen sich Vavr-Datentypen sogar direkt in Repositories nutzen; *Listing 15* zeigt ein Beispiel. Spring Data generiert die notwendigen Implementierungen, die die gewünschten Vavr-Strukturen liefern.

Ausblick auf 1.0

Während Javaslang bereits auf dem Weg in Richtung Version 3 war, hat Vavr nun ein paar Schritte in der Versionsnummer zurückgemacht. Aktuell ist Version 0.9x verfügbar, und es wird sicher durch die Refactorings noch ein paar Wochen bis zu einer 1.0 dauern. Diese Version 1.0 steht ganz im Zeichen von Aufräumarbeiten und Vorbereitungen in Hinblick auf Java 9 – insbesondere das Modulsystem. Weiter werden einige Konzepte potenziell „deprecated“, etwa das Pattern Matching auf Objektebene *(siehe „http://blog.vavr.io/pattern-matching-starter“)*, da zukünftige Java-Versionen diese Konzepte nativ unterstützen werden *(siehe „http://cr.openjdk.java.net/~briangoetz/amber/pattern-match.html“)*.

```
Function1<Team, Option<String>> liftedReader =
lift(this:: fetchRemoteLogo);
```

Listing 11

```
String readTeamLogo(Team t) throws InterruptedExcep-
tion,… {
  Function1<Team, Option<String>> lifted = …;

  return supplyAsync(() -> lifted.apply(t)
                            .getOrElse(default))
        .get(3000, MILLISECONDS);
}
```

Listing 12

```
Try<String> readTeamLogo(Team t) {
  Function1<Team, Option<String>> lifted = …;

  return Try.of(
        () -> supplyAsync(
              () -> lifted.apply(t)
                  .getOrElse(default)
            ).get(3000, MILLISECONDS)
     );
}
```

Listing 13

```
readTeamLogo(team)
  .map( /* Erfolgsfall */ )
  // Behandlung der Ausnahme innerhalb der Completa-
bleFuture
  .recover(ExecutionException.class, logoFetch-
Failed())
  // sonstige Fehlerfälle
  .getOrElse(TeamsController::logoFetchTimedoutRespon
se);
```

Listing 14

Fazit

Vavr ist keine Allzweckwaffe. Man wird sicher nicht die Ausdruckstärke von Scala, Haskell oder ähnlichen Sprachen in Java erwarten dürfen. Dafür ist Java auch niemals gedacht gewesen. Es ist auch klar, dass für die Nutzung mit Hibernate oder MongoDB Typ-Konverter geschrieben werden müssen, was man auf punktueller Ebene abwägen muss. Am Ende macht Vavr den Übergang zwischen Java- und Vavr-Datenstrukturen extrem einfach, sodass man immer noch ein Sicherheitsnetz hat.

Schließlich muss man in seinen Projekten entscheiden, wie viele Collection-Bibliotheken parallel sinnvoll sind. Vavr, Guava, Commons Collections, Eclipse Collections etc.; irgendwann ist es halt

genug und man wählt genau eine Implementierung.

In unseren Brownfield-Projekten haben sich gerade die einfachen Konzepte von Vavr wie die hervorragend designten, funktionalen Datenstrukturen und Abstraktionen wie „Option", „Lazy" und „Try" bewährt. Diese verbessern den Code und helfen insbesondere dabei, die Komplexität typischer, verschachtelter Programme aufzubrechen und Legacy-Code funktional zu refaktorieren. Schließlich ist an jeder Stelle erkennbar, dass Einfachheit und der Fokus auf Programmierer Designziele von Vavr sind. Man kommt auch ohne einen Kurs in Monaden-Theorie rasch zu guten Ergebnissen.

In Summe lohnt sich der Blick auf Vavr auf jeden Fall. Wer die Vorteile der funktionalen Programmierung innerhalb von Java nutzen möchte, wird sicher viel Freude an Vavr haben.

David Schmitz
david.schmitz@senacor.com

David Schmitz ist Principal Architect bei Senacor Technologies und seit mehr als zwanzig Jahren in zahlreichen Projekten und Technologiestacks unterwegs. Aktuell beschäftigt er sich mit Cloud-Technologien, Serverless-Architekturen und seiner Lieblingssprache Elixir.

Schrödinger programmiert Java

gelesen von Jürgen Thierack

Wie schon der Einband vermuten lässt, geht es in diesem Buch bunt zu. Damit haben wir das Gegenteil der üblichen Bleiwüste, was die Behauptung, etwas anders zu sein, voll und ganz rechtfertigt. Wer ist Schrödinger? Auf jeden Fall hat er nichts mit dem Nobelpreisträger aus dem letzten Jahrhundert zu tun, obgleich diese Assoziation wieder da ist, wenn im Buch öfter mal eine Katze auftaucht. Unser Schrödinger ist ein aufgewecktes Kerlchen mit blonder Tolle und massivem Kinn, das mit dem Autor im Dialog Java lernt. Zusammen mit anderen Autoren lernt Schrödinger in anderen Büchern des Verlages übrigens auch C++, C#, HTML5 und JavaScript.

Schrödinger ist stets neugierig und zeigt viel Emotionen, ist manchmal genervt und gefrustet. Erschöpfung ist ihm auch nicht fremd. In seinem grünen Büro hört er von der Theorie, in seiner lila Werkstatt muss er dann „Unmengen von Code, der ergänzt, verbessert und repariert werden will" bearbeiten. In seinem gelben Wohnzimmer kann er dann auch mal eine ruhigere Kugel schieben.

Originalton Schrödinger, als er im Kapitel 14 „Austauschschüler – das Datenaustauschformat XML" angekommen ist: „Ich verwende also dieses XSD-XML-Dings, um zu beschreiben, was in meinen XML-Dokumenten an Elementen und Attributen erlaubt ist? Uff das muss ich erstmal verdauen. Mir schwirrt der Kopf." Daneben eine Zeichnung, die keinen Zweifel daran lässt, dass Schrödinger der Kopf schwirrt. Darauf der Lehrer: „Keine Panik. XSD gehört auch zu den eher schwer zu verstehenden Dingen. Die XSD im Beispiel ist nach dem sogenannten „Russian Doll Design" aufgebaut, weil die einzelnen Elementdefinitionen ineinander verschachtelt sind (erst terminkalender, dann termine, dann termin), wie russische Matruschkas eben." Dazu wieder eine Illustration, nämlich eine Gruppe Matruschka-Puppen.

Wir haben mehr Sinne am Lernen beteiligt, als es bei normalen Lehrbüchern üblich ist. Da ist die Optik, aber auch die Besonderheit der verschiedenen Dialogsituationen, in denen es mal einfacher und schneller, mal schwieriger und langsamer zu Fortschritten kommt. Die Sinne für Gehör, Geruch und Temperatur können natürlich nicht bedient werden.

Es ist erstaunlich, wie viel Stoff auf den mehr als 700 Seiten untergebracht werden konnte, schließlich ist die fachliche Informationsdichte pro Papierfläche gegenüber Bleiwüsten deutlich reduziert. Neben den Basics – primitiven Datentypen, Objektorientierung und Kontrollstrukturen – finden wir Stringverarbeitung, Formatierung von Strings und Zahlen, abstrakte Klassen, Interfaces, typisierte Collections, Generics (übrigens sehr schön erläutert), Exceptions, Streams, Lambdas, Serialisierung, Threads, XML-Format und -Bindung, Datenbanken mit JDBC, Web Start und Unit-Tests. Bei den GUI-Oberflächen kommt auch JavaFX zur Sprache. Es werden Regular Expressions, Patern Matching, Internationalisierung und Lokalisierung abgehandelt. Auf acht Seiten hat das Modulsystem von Java 9 Platz gefunden. „Schrödinger programmiert Java" ist ein neuer, nett bereiteter Weg für Anfänger und auch für Leute, die schon ein gewisses Grundlagenwissen haben, das sie festigen und ausbauen wollen.

Jürgen Thierack
thierack@igfm-muenchen.de

Titel:	**Schrödinger programmiert Java – Das etwas andere Fachbuch**
Autor:	**Philip Ackermann**
Verlag:	**Rheinwerk Computing, 2017**
Umfang:	**712 Seiten**
Preis:	**44,90 Euro, eBook 39,90 Euro, Bundle Buch + eBook 49,90 Euro**
ISBN:	**978-3-8362-4583-8**

Jenkins

Coding Continous Delivery – Grundlagen des Jenkins-Pipeline-Plug-ins

Johannes Schnatterer und Daniel Behrwind, Triology GmbH

Wer schon einmal eine Continuous-Delivery-Pipeline mit einem herkömmlichen CI-Tool durch Verketten einzelner Jobs und ohne direkten Pipeline-Support eingerichtet hat, der weiß, wie unübersichtlich ein solches Unterfangen sein kann. Diese Artikelserie zeigt, wie sich eine Pipeline mithilfe des Jenkins-Pipeline-Plug-ins an zentraler Stelle als Code definieren lässt. Im ersten Teil geht es um die Grundlagen und um praktische Tipps für den Einstieg.

Continuous Delivery hat sich im Umfeld agiler Software-Entwicklung als adäquates Vorgehen erwiesen, qualitativ hochwertige Software in kurzen Zyklen zuverlässig und wiederholbar zu veröffentlichen. Dabei durchlaufen Änderungen an der Software eine Reihe von qualitätssichernden Schritten, bevor sie in Produktion gehen. Eine typische Continuous-Delivery-Pipeline könnte beispielsweise so aussehen:

- Checkout aus der Versionsverwaltung
- Build
- Unit-Tests
- Integrationstests
- Statische Code-Analyse
- Deployment in einer Staging-Umgebung
- Funktionale und/oder manuelle Tests
- Deployment in Produktion

Dabei ist es essenziell, all diese Schritte zu automatisieren, was typischerweise mit Continuous-Integration-Servern wie Jenkins geschieht.

Zwar eignen sich herkömmliche Jenkins-Jobs gut, um einzelne Schritte einer Continuous-Delivery-Pipeline zu automatisieren. Da die Schritte aber aufeinander aufbauen, ist deren Reihenfolge zwingend einzuhalten. Mit herkömmlichen Jenkins-Jobs (oder anderen Continuous-Integration-Tools ohne direkten Pipeline-Support) kann das schnell unübersichtlich sein. Einzelne Jobs, oft angereichert um etliche Pre- und Post-Build-Schritte, werden verkettet. Dies führt dazu, dass man sich von Job zu Job hangeln muss, um zu verstehen, was passiert. Zusätzlich sind solche komplexen Konfigurationen weder test- noch versionierbar und müssen für jedes Projekt neu aufgesetzt werden.

Hier schafft das Jenkins-Pipeline-Plug-in Abhilfe. Es bietet die Möglichkeit, die gesamte Pipeline mithilfe einer Groovy-DSL an zentraler Stelle, einer versionierten Skript-Datei („Jenkinsfile"), als Code zu definieren. Dabei stehen dem Anwender zwei Varianten der DSL zur Auswahl: ein imperativer, tatsächlich eher geskripteter Stil (nachfolgend als „scripted Syntax" bezeichnet) und seit Februar 2017 auch eine deklarative Variante (nachfolgend „declarative Syntax" genannt).

Die deklarative Syntax ist dabei eine Teilmenge der „scripted"-Syntax und bietet mit einem vorgegebenen Aufbau und mehr beschreibenden Sprachelementen eine Grundstruktur (ähnlich einer „maven-pom"-Datei), die den Einstieg vereinfachen kann. Damit führt sie, auch wenn sie umfangreicher und unflexibler ist, zu Build-Skripten, die intuitiver verständlicher sind als solche, die in „scripted"-Syntax verfasst sind. Diese bietet dafür fast alle Freiheiten, die die Sprache Groovy mit sich bringt (Einschränkungen siehe *[1]*), erfordert aber gegebenenfalls auch eine tiefergehende Auseinandersetzung mit Groovy.

Die Entscheidung für die eine oder die andere Variante kann aktuell noch als Geschmacksfrage angesehen werden; es ist nicht absehbar, ob sich eine der beiden Richtungen durchsetzen und die andere letztlich verdrängen wird. Neuere offizielle Beispiele sind jedoch meist in der „declarative"-Syntax verfasst und es gibt einen visuellen Editor, der nur auf diese ausgelegt ist. Um einen direkten Vergleich zu bieten, sind die Beispiele in diesem Artikel in beiden Stilen formuliert.

Kernkonzepte

Die Beschreibung einer Build-Pipeline mit der Jenkins-Pipeline-DSL gliedert sich im Wesentlichen in Stages und Steps. Stages sind frei wählbare Gruppierungen der Schritte einer Pipeline. Die Punkte der

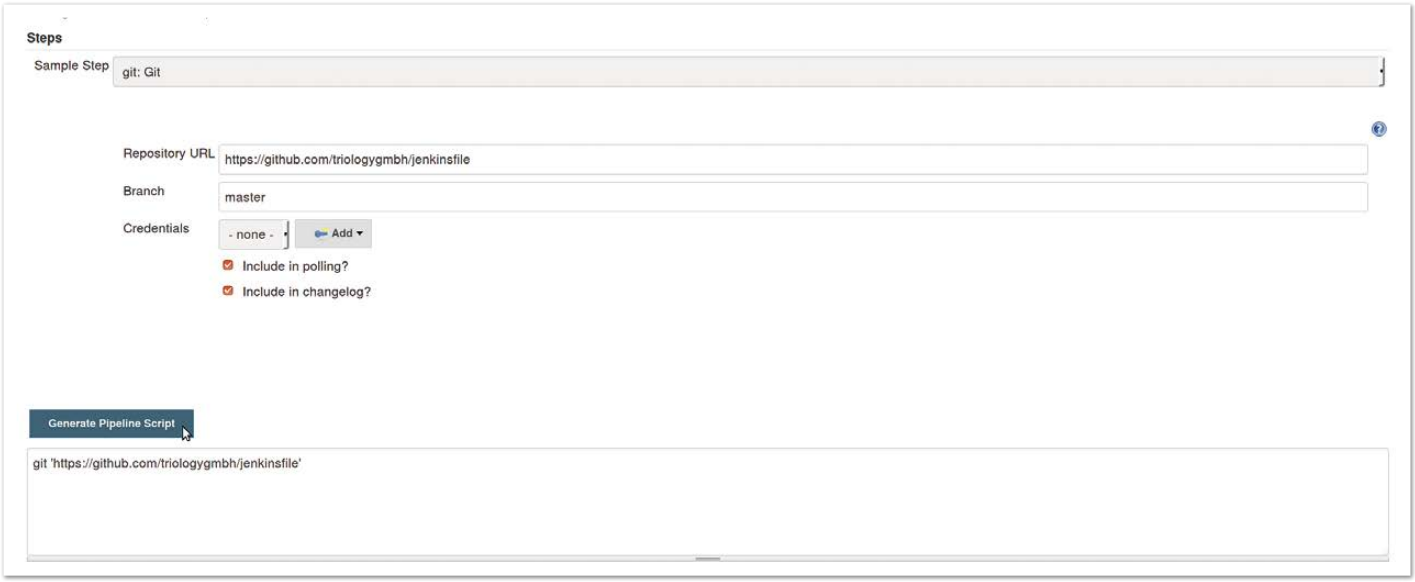


Abbildung 1: Der Snippet-Generator

```
pipeline {
  agent any

  tools {
     maven 'M3'
  }

  stages {
     stage('Checkout') {
        steps {
           git 'https://github.com/triologygmbh/jen-
kinsfile'
        }
     }

     stage('Build') {
        steps {
           sh 'mvn -B package'
        }
     }
  }
}
```

Listing 1

```
node {
  def mvnHome = tool 'M3'

  stage('Checkout') {
     git 'https://github.com/triologygmbh/jenkinsfile'
  }

  stage('Build') {
     sh "${mvnHome}/bin/mvn -B package"
  }
}
```

Listing 2

oben genannten Beispiel-Pipeline könnten beispielsweise jeweils eine Stage darstellen. Ein Step ist dabei ein Befehl, der einen konkreten Build-Schritt beschreibt und der letztlich von Jenkins ausgeführt wird. Eine Stage umfasst also einen oder mehrere Steps.

Für eine minimale Pipeline-Definition muss neben mindestens einer Stage mit einem Step noch ein Build-Executor allokiert sein, also etwa ein Jenkins-Build-Slave. Dies geschieht in der „declarative"-Syntax mithilfe der „agent"-Section, in der „scripted"-Syntax per „node"-Step. In beiden Varianten kann der Executor mit Labels weiter qualifiziert werden, sodass sichergestellt ist, dass er bestimmte Bedingungen erfüllt (etwa eine bestimmte Java-Version oder eine Docker-Installation zur Verfügung stellt).

Bevor es an der Zeit ist, die erste Pipeline einzurichten, noch ein Hinweis auf die verschiedenen Arten von Pipeline-Jobs, die Jenkins anbietet:

- *Pipeline*
  Ein einfacher Pipeline-Job, der die Script-Definition direkt über die Weboberfläche von Jenkins oder in einem „Jenkinsfile" aus dem Source Code Management (SCM) erwartet.
- *Multibranch-Pipeline*
  Erlaubt es, ein SCM-Repository mit mehreren Branches anzugeben. Steht in einem Branch ein „Jenkinsfile", wird die darin definierte Pipeline bei Änderungen am Branch ausgeführt. Pro Branch wird dabei on-the-fly ein Jenkins-Job angelegt.
- *GitHub-Organization*
  Eine Multibranch-Pipeline für eine GitHub-Organization beziehungsweise einen GitHub-User. Dieser Job scannt alle Repositories einer GitHub-Organization und legt für alle, deren Branches ein „Jenkinsfile" enthalten, einen Folder an, der eine Multibranch-Pipeline enthält. Es handelt sich also quasi um eine geschachtelte Multibranch-Pipeline *[2]*.

Erste Schritte

Um sich mit den Möglichkeiten des Pipeline-Plug-ins vertraut zu machen, bietet sich das simpelste Setup an: Man baut ein Projekt ohne vorhandenes „Jenkinsfile", indem man das Pipeline-Script direkt in einem Pipeline-Job über die Weboberfläche von Jenkins beschreibt.

Zum Einstieg ist es wichtig zu wissen, dass es in jeder Art von Pipeline-Job auf der Weboberfläche von Jenkins Links zur Dokumentation der auf der aktuellen Jenkins-Instanz verfügbaren Pipeline-Features gibt. Die in allen Jenkins-Instanzen verfügbaren Basic Steps *[3]* lassen sich durch zusätzliche Plug-ins erweitern.

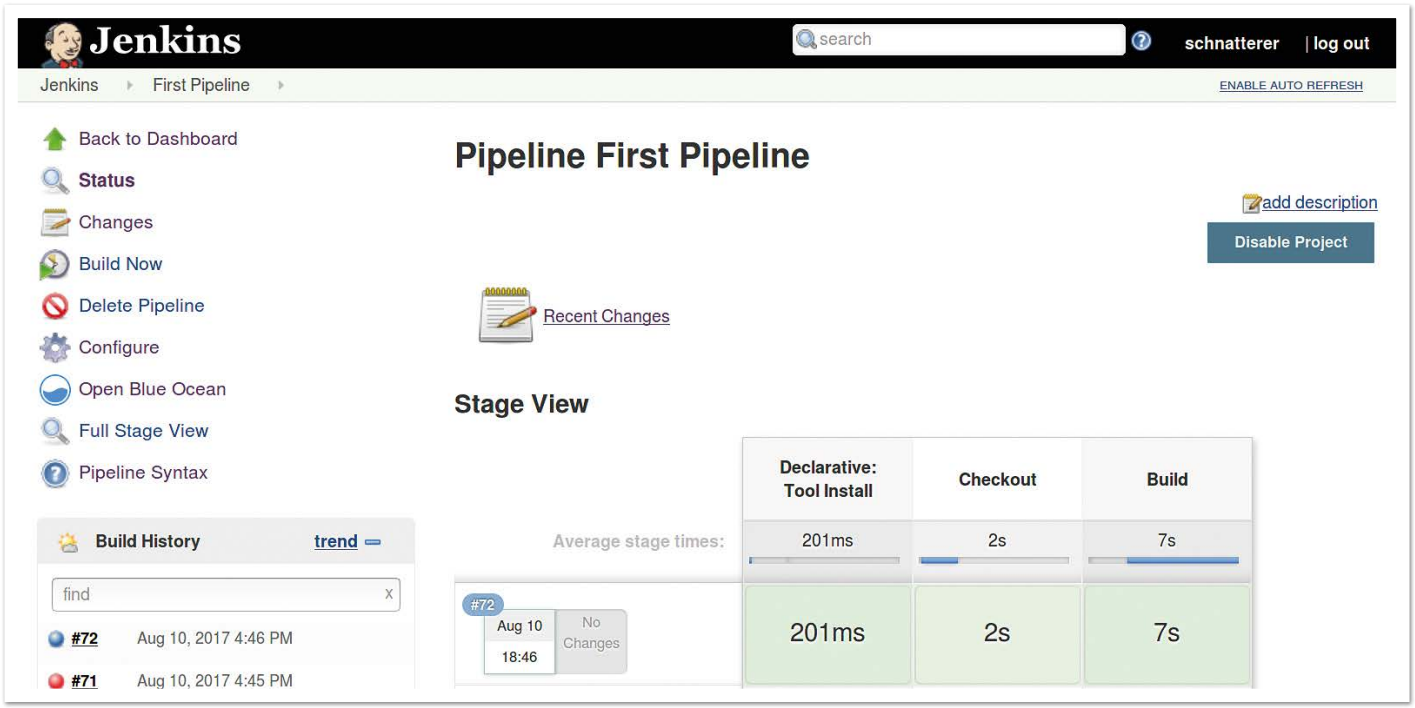


Abbildung 2: Jenkins-Stage-View im „classic Theme“

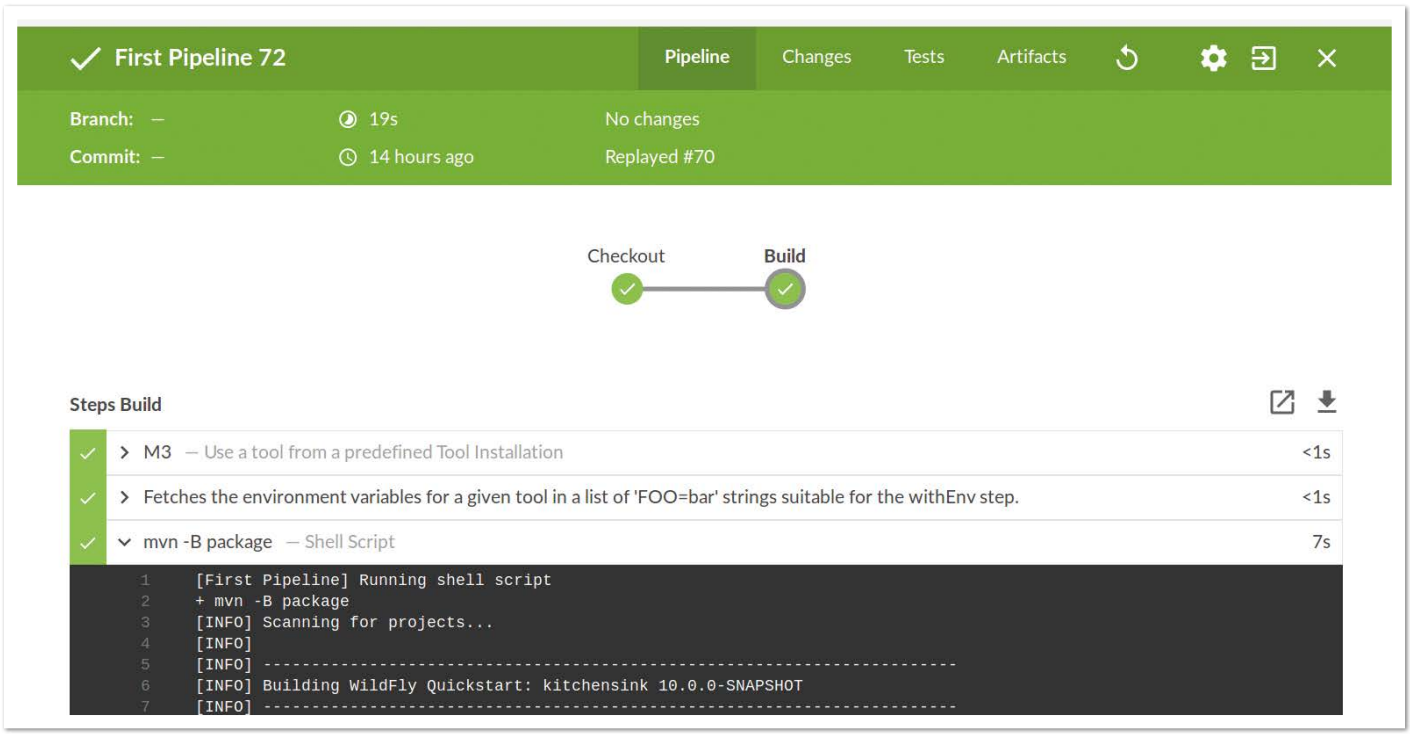


Abbildung 3: Ansicht eines Step-Logs im Blue-Ocean-Theme

Klickt man im Job auf Pipeline-Syntax, gelangt man auf den Snippet-Generator. Zusätzlich gibt es einen globalen Link in jeder Jenkins-Instanz. Die URL lautet *„https://jenkins/pipeline-syntax“*, also zu Beispiel: *„https://opensource.triology.de/jenkins/pipeline-syntax“*.

Der Snippet-Generator ist ein hilfreiches Tool für den Übergang vom bisherigen Jenkins-Job „Zusammenklicken“ zur Pipeline-Syntax. Hier kann man wie gewohnt Teile seines Build-Jobs mit der Maus zusammenstellen und sich daraus ein Snippet in Pipeline-Syntax generieren lassen *(siehe Abbildung 1)*. Der Snippet-Generator kann aber auch später hilfreich sein, um die Syntax neuer Plug-ins kennenzulernen oder wenn man keine Autovervollständigung in seiner IDE hat. Apropos Autovervollständigung: Im Snippet-Generator gibt es weitere Links auf hilfreiche Informationen:

- Die für diese Instanz verfügbaren globalen Variablen. Dazu gehören environment variables, Parameter des Build-Jobs und Informationen zum aktuellen Build-Job, zum Beispiel *„https://opensource.triology.de/jenkins/pipeline-syntax/globals“*
- Die ausführliche Dokumentation aller für diese Instanz verfügbaren Steps und Classes sowie ihre zugehörigen Parameter, zum Beispiel *„https://opensource.triology.de/jenkins/pipeline-syntax/html“*

Abbildung 4: Pipeline-Script from SCM

- Eine IntelliJ-Groovy-DSL-Datei (GDSL), mit der man Autovervollständigung aktivieren kann. Wie das funktioniert, zeigt ein Blog Post *[4]*

Pipeline-Scripts

Mit diesem Wissen kann das Schreiben von Pipeline-Scripts beginnen. Die grundlegenden Features des Jenkins-Pipeline-Plug-ins werden anhand eines typisches Java-Projekts gezeigt. Als Beispiel dient hier der „kitchensink"-Quickstart von WildFly, einer typischen JEE-Webanwendung mit CDI, JSF, JPA, EJB, JAX-RS und Integrationstests mit Arquillian. *Listing 1* zeigt die „declarative"-Syntax für ein minimales Pipeline-Script zum Bauen dieses Projekts auf Jenkins.

Dieses Script allokiert einen beliebigen Build-Executor, holt sich die unter „Tools" konfigurierte Maven-Instanz, checkt den Default-Branch der Git-URL aus und führt einen nicht-interaktiven Maven Build aus. Hier zeigt sich der einheitliche Aufbau der „declarative"-Pipeline. Jede Pipeline ist umschlossen vom „pipeline"-Block, der wiederum aus „Sections" beziehungsweise „Directives" *[5]* besteht. Diese spiegeln unter anderem die beschriebenen Kernkonzepte Stage und Step wider. *Listing 2* zeigt zum Vergleich die gleiche Pipeline in „scripted"-Syntax.

Auch hier sieht man das Konzept der Stage, dies beinhaltet dann direkt die Steps. Das Konzept der Sections beziehungsweise Directives gibt es hier nicht. Bei „node", „tool", „stage", „git" etc. spricht man hier jeweils nur von Steps. Diese Syntax bietet deutlich mehr Freiheiten. Anders als bei der „declarative"-Syntax, die immer von einem „pipeline"-Block umgeben sein muss, könnte man hier auch Steps außerhalb des „node"-Blocks ausführen. Dabei zeigt sich, warum die deklarative Syntax eine Teilmenge der „scripted"-Syntax ist: Im Prinzip ist der die „declarative"-Pipelines umschließende „pipeline"-Block ein Step der „scripted"-Syntax.

Damit diese Scripts auf einer Jenkins-Instanz ausführbar sind, muss auf einem Jenkins 2.60.2 auf Linux im Auslieferungszustand unter „Global Tool Configuration" nur eine Maven-Installation mit dem Namen „M3" angelegt sein. Danach kann dieses mithilfe der „tools"-Declarative beziehungsweise Step im Pipeline-Job bekannt gemacht und ausgeführt werden.

Im Hintergrund führt die Angabe des Tools „M3" dazu, dass Maven auf dem aktuellen Build-Executor zur Verfügung gestellt wird. Falls nötig, wird es dazu installiert und im „PATH" bekannt gemacht. Das Ergebnis des Builds ist im „classic Theme" von Jenkins in einer Stage-View angezeigt, in der je Build die Stages und deren Ausführungszeiten visualisiert sind *(siehe Abbildung 2)*.

```
stage('Checkout') {
checkout scm
}
```

Listing 3

Eine spezielle, für Pipelines designte Ansicht bietet das offizielle Jenkins-Theme „Blue Ocean" *[6]*. Mit deutlich modernerem UX kann man hier sehr übersichtlich mit Pipelines arbeiten. Man sieht auf einen Blick, welche Steps pro Stage ausgeführt wurden, und kann beispielsweise das Console-Log bestimmter Steps mit einem Klick einsehen. Für „declarative"-Pipelines gibt es außerdem einen visuellen Editor *[7]*. Wenn das Plug-in installiert ist, kann man über Links in jedem Build-Job zwischen Blue Ocean und Classic wechseln. *Abbildung 3* zeigt die Pipeline aus *Abbildung 2* in Blue Ocean.

Die gezeigten ersten Pipeline-Beispiele werden im Folgenden sukzessive erweitert, um die grundlegenden Features der Pipeline zu zeigen. Dabei sind die Änderungen jeweils sowohl in „declarative"- als auch in „scripted"-Syntax zu sehen. Den aktuellen Stand jeder Erweiterung kann man bei *[8]* nachverfolgen und ausprobieren. Hier gibt es für jeden Abschnitt unter der in der Überschrift genannten Nummer jeweils für „declarative" und „scripted" einen Branch, der das vollständige Beispiel umfasst. Das Ergebnis der Builds jedes Branch lässt sich außerdem direkt auf der Jenkins-Instanz der Autoren *[9]* einsehen.

Umzug ins SCM-/Jenkins-File

Einer der größten Vorteile von Jenkins-Pipelines ist, dass man sie unter Versionsverwaltung stellen kann. Dazu ist es Konvention, eine Datei „Jenkinsfile" ins Root-Verzeichnis des Repository zu legen. Dann kann der beschriebene Pipeline-Job auf „Pipeline-Script from SCM" *(siehe Abbildung 4)* umgestellt oder ein Multibranch-Pipeline-Job angelegt werden.

Die URL des SCM (in diesem Fall Git) wird im Job konfiguriert. Man könnte die Pipeline-Scripts wie gezeigt einchecken, allerdings würde man dann die URL des Repository im Repository wiederholen. Dies würde allerdings das DRY-Prinzip *[10]* verletzen. Die Lösungen sind je nach Syntax unterschiedlich:

- *Declarative*
  Der Checkout erfolgt standardmäßig durch die „agent"-Section; die Checkout Stage kann hier also komplett entfallen.
- *Scripted*
  Der Checkout wird nicht standardmäßig ausgeführt; es gibt jedoch die Variable „scm", die die im Job konfigurierte Repository-URL enthält, sowie den Step „checkout", der den im Job konfigurierten SCM-Provider (in diesem Falle Git) enthält *(siehe Listing 3)*.

Lesbarkeit mit eigenen Steps verbessern

Groovy macht es als Grundlage der Pipeline sehr einfach, die vorhandenen Steps zu erweitern. Am bestehenden Beispiel kann der Aufruf von Maven ausdrucksstärker gemacht werden, indem er in eine eigene Methode gekapselt wird *(siehe Listing 4)*.

Diese erlaubt es, sowohl in „declarative"- als auch in „scripted"-Syntax Maven mit „mvn 'package'" aufzurufen. Die Definition der Tools ist in die Methode verlagert. Damit werden für die Ausführung auf Jenkins sinnvolle Maven-Parameter (Batch Mode, Maven-Version ausgeben, Snapshots updaten, fehlgeschlagene Tests auf Konsole ausgeben) von den im Kontext des jeweiligen Aufrufs interessanten Maven-Parametern (hier „package"-Phase) getrennt. Dies erhöht die Lesbarkeit, weil man beim Aufruf nur die wesentlichen Parameter übergibt. Zudem muss man nicht die Parameter wiederholen, die ohnehin bei jedem Aufruf mitgegeben werden sollten.

Zusätzlich kommt hier ein spezifisches JDK zum Einsatz. Dies erfordert zwar, dass analog zu Maven unter „Global Tool Configuration" nun eine JDK-Installation mit dem Namen „JDK8" installiert ist. Dadurch wird jedoch der Build deterministischer, da nicht implizit das JDK von Jenkins verwendet wird, sondern ein explizit benanntes.

Diese Maven-Methode ist den offiziellen Beispielen entnommen *[11]*. Eine Methode wie „mvn" ist ein guter Kandidat für die Auslagerung in eine Shared-Library. Diese ist in einem folgenden Teil dieser Artikelserie beschrieben.

Unterteilung in Stages

Ähnlich wie beim Schreiben von Methoden oder Funktionen in der Software-Entwicklung ist es auch für die Wartbarkeit von Pipeline-Scripts sinnvoll, kleine Stages zu verwenden. Diese Unterteilung erlaubt es, bei scheiternden Builds mit einem Blick zu erkennen, wo etwas schiefging. Außerdem werden die Zeiten pro Stage gemessen, wodurch schnell ersichtlich wird, welche Teile des Builds am meisten Zeit benötigen.

Der Maven Build in diesem Beispiel lässt sich weiter unterteilen in „Build"- und „Unit"-Test. An dieser Stelle werden zudem erstmals Integration-Tests ausgeführt. In diesem Beispiel erfolgt die Ausführung mit Arquillian und WildFly Swarm. In „declarative"-Syntax sieht das wie in *Listing 5* aus. In „scripted"-Syntax entfallen die „steps"-Sections.

Als Nachteil ist zu nennen, dass der gesamte Build dadurch etwas langsamer wird, da verschiedene Maven-Phasen in mehreren Stages durchlaufen werden. Dies ist hier schon optimiert, indem „clean" nur einmal am Anfang aufgerufen wird und die „pom.xml" um ein Property „skipUnitTests" erweitert ist, was die erneute Ausführung der Unit-Tests in der Integration-Test-Stage verhindert.

Bei Integration-Tests besteht generell die Gefahr von Port-Konflikten. Beispielsweise kann die Infrastruktur gleichzeitig laufender Builds an die gleichen Ports binden. Dadurch kommt es zu unerwartet fehlschlagenden Builds. Dies lässt sich durch die Verwendung von Docker effektiv umgehen. Dies ist in einem folgenden Teil dieser Artikelserie beschrieben.

Ende des Pipeline-Laufs und Fehlerbehandlung

Üblicherweise gibt es Schritte, die immer am Ende eines Pipeline-Laufs ausgeführt werden sollen, unabhängig davon, ob der Build erfolgreich war oder nicht. Das beste Beispiel dafür sind Test-Ergebnisse. Schlägt ein Test fehl, wird auch der Build fehlschlagen. In jedem Fall sollten aber die Test-Ergebnisse in Jenkins erfasst werden.

Zudem sollte bei fehlschlagenden Builds oder wenn sich der Status des Builds ändert, eine spezielle Reaktion erfolgen. Üblich ist hier das Versenden von E-Mails, denkbar wären aber auch Chat-Benachrichtigungen oder Ähnliches. Beide Fälle sind in Pipelines mit den ähnlichen syntaktischen Konzepten spezifiziert. Allerdings unterscheiden sich hier die Ansätze zwischen „declarative"- und „scripted"-Syntax.

Generell stünden in beiden Fällen die Sprachmittel von Groovy, also „try-catch-finally"-Blöcke, zur Verfügung. Diese sind allerdings nicht ideal, da gefangene Exceptions dann keine Auswirkungen auf den Build-Status haben. In der „declarative"-Syntax stehen hier die „post"-Section mit

```
def mvn(def args) {
  def mvnHome = tool 'M3'
  def javaHome = tool 'JDK8'

  withEnv(["JAVA_HOME=${javaHome}",
"PATH+MAVEN=${mvnHome}/bin:${env.JAVA_HOME}/bin"]) {
     sh "${mvnHome}/bin/mvn ${args} --batch-mode -V -U
-e -Dsurefire.useFile=false"
  }
}
```

Listing 4

```
stages {
  stage('Build') {
     steps {
        mvn 'clean install -DskipTests'
     }
  }

  stage('Unit Test') {
     steps {
        mvn 'test'
     }
  }

  stage('Integration Test') {
     steps {
        mvn 'verify -DskipUnitTests -Parq-wildfly-
swarm '
     }
  }
}
```

Listing 5

```
post {
  always {

     junit allowEmptyResults: true,
        testResults: '**/target/surefire-reports/TEST-
*.xml, **/target/failsafe-reports/*.xml'
  }
  changed {
     mail to: "${env.EMAIL_RECIPIENTS}",
        subject: "${JOB_NAME} - Build #${BUILD_NUMBER}
- ${currentBuild.currentResult}!",
        body: "Check console output at ${BUILD_URL} to
view the results."
  }
}
```

Listing 6

```
node {
  catchError {
     // ... Stages ...
  }

  junit allowEmptyResults: true,
        testResults: '**/target/surefire-reports/TEST-*.xml, **/target/failsafe-reports/*.xml'
  statusChanged {
     mail to: "${env.EMAIL_RECIPIENTS}",
        subject: "${JOB_NAME} - Build #${BUILD_NUMBER} - ${currentBuild.currentResult}!",
        body: "Check console output at ${BUILD_URL} to view the results."
  }
}
def statusChanged(body) {
  def previousBuild = currentBuild.previousBuild
  if (previousBuild != null && previousBuild.result != currentBuild.currentResult) {
     body()
  }
}
```

Listing 7

```
node {
  // ... catchError und nodes
  mailIfStatusChanged env.EMAIL_RECIPIENTS
}

def mailIfStatusChanged(String recipients) {

  if (currentBuild.currentResult == 'SUCCESS') {
     currentBuild.result = 'SUCCESS'
  }
  step([$class: 'Mailer', recipients: recipients])
}
```

Listing 8

```
pipeline {
  agent any

  options {
     disableConcurrentBuilds()
     buildDiscarder(logRotator(numToKeepStr: '10'))
  }

  stages { /* .. */ }
}
```

Listing 9

```
node {
  properties([
     disableConcurrentBuilds(),
     buildDiscarder(logRotator(numToKeepStr: '10'))
  ])

  catchError { /* ... */ }
}
```

Listing 10

```
stage('Build') {
steps {
mvn 'clean install -DskipTests'
archiveArtifacts '**/target/*.*ar'
}
}
```

Listing 11

den Conditions „always“, „changed“, „failure“, „success“ und „unstable“ zur Verfügung. Damit lässt sich ausdrucksstark definieren, was am Ende jeder Ausführung erfolgen soll. *Listing 6* zeigt das beschriebene Szenario.

Erwähnenswert ist hier, dass man mit wenig Aufwand auch den bestehenden E-Mail-Mechanismus von Jenkins verwenden kann, was später in diesem Abschnitt erklärt ist. Zudem ist die Herkunft der E-Mail-Adresse der Empfänger hier von Relevanz. Im Beispiel werden diese aus der Environment-Variable „EMAIL_RECIPIENTS“ geladen. Diese muss von einem Administrator in der Jenkins-Konfiguration festgelegt sein. Alternativ kann man die Empfänger natürlich auch direkt ins „Jenkinsfile“ schreiben. Dann werden sie allerdings mit ins SCM eingecheckt.

In der „scripted“-Syntax steht hier nur der „catchError“-Step zur Verfügung. Dieser verhält sich im Wesentlichen wie ein „finally“-Block. Um obiges Szenario abzubilden, muss mit „if“-Bedingungen gearbeitet werden *(siehe Listing 7)*. Auch hier empfiehlt es sich, aus Gründen der Wartbarkeit einen eigenen Step zu definieren.

Wie erwähnt, lässt sich das Thema „E-Mails“ sowohl in der „declarative“- als auch in der „scripted“-Syntax vereinfachen, indem der bestehende E-Mail-Mechanismus von Jenkins zum Einsatz kommt. Hier werden die bekannten „Build failed in Jenkins“-und „Jenkins build is back to normal“-E-Mails versendet. Dazu wird die Klasse „Mailer“ verwendet, für die es keinen dedizierten Step gibt. Dies ist über den generischen Step „step“ möglich.

Wenn man auch die „back to normal“-E-Mails erhalten möchte, ist noch eine Besonderheit zu beachten: Die Klasse „Mailer“ liest den Wert aus der Variable „currentBuild.result“ aus. In der Pipeline wird diese im Erfolgsfall erst ganz am Ende der Pipeline gesetzt, dadurch erfährt die Klasse „Mailer“ es nie. Also bietet sich auch hier die Implementierung als eigener Step an. In „scripted“-Syntax lässt sich dies wie in *Listing 8* realisieren, die Lösung lässt sich aber auch mit „declarative“-Syntax verwenden. Abschließend sei in Bezug auf Notification mit Hipchat, Slack etc. ein Blogbeitrag von Jenkins empfohlen *[12]*.

Properties und Archivierung

Bei herkömmlichen Jenkins-Jobs gibt es viele kleinere Einstellungen, die man über die Oberfläche vornimmt, wie beispielsweise die Größe der Build-Historie, Verhindern paralleler Builds etc. Selbstverständ-

lich sind diese bei Verwendung des Pipeline-Plug-ins im „Jenkinsfile" beschrieben. In der „declarative"-Syntax heißen diese Einstellungen „options" *(siehe Listing 9)*, in der „scripted"-Syntax „properties". Sie werden über den gleichnamigen Step gesetzt *(siehe Listing 10)*.

Ein weiterer nützlicher Step ist „archiveArtifacts". Er speichert durch den Build erstellte Artefakte („.jar", „.war", „.ear" etc.), sodass diese über die Weboberfläche von Jenkins eingesehen werden können. Dies kann für Debugging nützlich sein oder tatsächlich zur Archivierung von Versionen, wenn man kein Maven Repository verwendet. In „declarative"-Syntax lässt sich dies wie in *Listing 11* formulieren. In „scripted"-Syntax entfallen die „steps"-Sections. Sie speichert alle „.jar", „.war", „.ear" etc., die in einem der Maven-Module erzeugt wurden.

Tipps für den Einstieg

Es gibt einige weitere grundlegende Directives wie beispielsweise „parameters" (deklariert Build-Parameter) und „script" (zum Ausführen eines Blocks in „scripted"-Syntax innerhalb der „declarative"-Syntax). Hier ist die Lektüre *[5]* empfehlenswert. Darüber hinaus gibt es viele weitere Steps, die im Wesentlichen durch Plug-ins bereitgestellt werden, siehe offizielle Übersicht *[13]*. Damit diese in der Pipeline verfügbar sind, müssen Entwickler ein entsprechendes API verwenden. Die Pipeline-Kompatibilität einzelner Plug-ins ist hier zusammengefasst *[14]*. Zum jetzigen Zeitpunkt haben die meisten gängigen Plug-ins Unterstützung für das Jenkins-Pipeline-Plug-in. Eine weitere empfehlenswerte Literatur sind die Top 10 Best Practices for Jenkins Pipeline *[15]*.

Abschließend noch einige nützliche Tipps für die Arbeit mit dem „Jenkinsfile". Beim ersten Aufsetzen einer Pipeline empfiehlt es sich, mit einem normalen Pipeline-Job zu starten und das „Jenkinsfile" erst unter Versionsverwaltung zu stellen, wenn der Build läuft. Ansonsten läuft man Gefahr, seine Commit-Historie zu verunreinigen. Bei Änderungen einer bestehenden Multibranch-Pipeline bietet sich das „Replay"-Feature an, mit dem temporär für die nächste Ausführung die Pipeline über die Weboberfläche von Jenkins editieren kann, ohne das „Jenkinsfile" im SCM zu verändern.

Ein letzter Tipp: Auch bei Pipelines kann man den Workspace über die Weboberfläche von Jenkins einsehen. Durch die „agent"-Section beziehungsweise den „node"-Step ist es möglich, mehrere Build-Executors zu belegen. Das ist mit dem Thema „Parallelisierung" in einem folgenden Teil dieser Artikelserie näher beschrieben. Deshalb kann es auch mehrere Workspaces geben. Diese kann man im „classic Theme" jeweils einsehen, indem man in einem Build-Job auf „Pipeline Steps" klickt und dort auf „Allocate node : Start". Hier gibt es dann den bekannten „Workspace"-Link auf der linken Seite.

Fazit und Ausblick

Dieser Artikel zeigt die Grundlagen des Jenkins-Pipeline-Plug-ins. Er beschreibt die Grundkonzepte und Begriffe, die verschiedenen Job-Arten, führt theoretisch sowie anhand von Beispielen in die Syntax des „Jenkinsfile" ein und gibt Praxistipps für das Arbeiten mit Pipelines. Das beschriebene Beispiel konfiguriert Jenkins, baut den Code, führt darauf Unit- und Integration-Tests aus, archiviert Test-Ergebnisse sowie Artefakte und versendet E-Mails – das alles mit einem ungefähr dreißig Zeilen langen Skript.

Um von Continous Delivery zu sprechen, fehlen hier natürlich noch Schritte wie statische Code-Analyse, beispielsweise mit SonarQube, und natürlich Deployments auf Staging- und Produktiv-Umgebungen. Um dies zu realisieren, bietet sich die Verwendung einiger Werkzeuge und Methoden wie Nightly Builds, Wiederverwendung über verschiedene Jobs hinweg, Unit Testing, Parallelisierung und Docker an. Dies ist in folgenden Teilen dieser Artikelserie beschrieben.

Weitere Informationen

[1] *https://jenkins.io/doc/book/pipeline/syntax/#differences-from-plain-groovy*
[2] *https://opensource.triology.de/jenkins/job/triologygmbh-github*
[3] *https://jenkins.io/doc/pipeline/steps/workflow-basic-steps*
[4] *https://www.triology.de/blog/jenkins-pipeline-plugin-code-vervollstaendigung*
[5] *https://jenkins.io/doc/book/pipeline/syntax/#declarative-pipeline*
[6] *https://jenkins.io/doc/book/blueocean*
[7] *https://www.cloudbees.com/blog/getting-started-blue-oceans-visual-pipeline-editor*
[8] *https://github.com/triologygmbh/jenkinsfile*
[9] *https://opensource.triology.de/jenkins/job/triologygmbh-github/job/jenkinsfile*
[10] *https://pragprog.com/the-pragmatic-programmer/extracts/tips*
[11] *https://github.com/jenkinsci/pipeline-examples*
[12] *https://jenkins.io/blog/2017/02/15/declarative-notifications*
[13] *https://jenkins.io/doc/book/pipeline/syntax*
[14] *https://github.com/jenkinsci/pipeline-plugin/blob/master/compatibility.md*
[15] *https://www.cloudbees.com/blog/top-10-best-practices-jenkins-pipeline-plugin*

Johannes Schnatterer
johannes.schnatterer@triology.de

Johannes Schnatterer ist Solution Architect bei der TRIOLOGY GmbH in Braunschweig. Technologisch ist er dort in den Bereichen „Java EE" und „Web" tätig und versucht, mit besonderem Fokus auf Qualität, Open-Source-Enthusiasmus, einem Hauch von Pedantismus und der Pfadfinderregel die IT-Welt jeden Tag ein bisschen besser zu machen.

Daniel Behrwind
d.behrwind@gmail.com

Daniel Behrwind ist als Software-Entwickler bei der TRILOGY GmbH tätig. Dort befasst er sich überwiegend mit der Entwicklung von Individual-Software, wobei er schwerpunktmäßig an Java-Webprojekten arbeitet. Als leidenschaftlicher Clean-Code-Verfechter ist er begeistert von kreativen Lösungen für komplexe Probleme, die so offensichtlich aussehen, als seien sie von selbst entstanden.

Ach, wenn Einhorn-Entwickler auf Bäumen wüchsen!

Timothée Bourguignon, MATHEMA Software GmbH

„Schreibe deinen Code, als ob derjenige, der danach jahrelang damit leben muss, ein gewalttätiger Psychopath ist, der auch weiß, wo du wohnst ...", sagte mein Mentor – und nach einer kurzen Pause fügte er hinzu: „ ... und jetzt denk bitte über deinen ersten Commit nach."

Gemäß dieser Definition hätte ich bestimmt schon hundertmal sterben müssen. Mein erster ernsthafter Commit war der praktische Teil meiner Masterarbeit. Ziel war es, eine C#-Regel-Engine zur Zeitplan-Erstellung zu schreiben. Es gab Milliarden von möglichen Kombinationen und dreißig Sekunden Zeit, um eine gute Lösung zu erzeugen. Dreißig Sekunden. Ziel erreicht. Es hat funktioniert. Es gab sogar ein paar Unit-Tests. Aber von Qualität konnte man nicht wirklich reden. Die Dokumentation war schlicht nicht vorhanden. Außer, man verstand meine Masterarbeit selbst als Dokumentation. Es gab mehr Vererbung als in der Trump-Familie. Clean Code – was damals noch nicht geschrieben war – kam nicht infrage, denn ich war weit entfernt, davon ein Verständnis zu haben. Dieses Stück Software wurde so gut wie nie produktiv eingesetzt. Das war für mich eine Katastrophe, vielleicht sogar ein Fiasko.

Jedes Mal, wenn ich diese Geschichte erzähle, sehe ich grinsende Gesichter und bekomme ähnliche Geschichten erzählt. Als ob nur die wenigsten von uns einen halbwegs richtigen Start in das Berufsleben bekommen hätten. Deshalb formuliere ich folgende Hypothese: Die ersten Jahre unserer Karriere verbringen wir damit, überhaupt zu verstehen, was von uns erwartet wird und was ein guter Entwickler sein könnte.

Was ist überhaupt ein guter Entwickler?

Wissen wir es Jahre später besser? Was würde man heutzutage darauf antworten? Etwa ein Cowboy-Coder, der eine Web-App wie Chuck Norris allein in zwei Stunden runterprogrammieren kann? Ein „10x"-Rockstar-Entwickler, der alles immer besser kann als alle anderen? Ein Hacker, der nachts magische Skripte aus dubiosen Seiten irgendwie zum Leben erweckt?

Jeff Atwood hat es in seinen Blog-Eintrag „We hire the best, like everyone else" *[1]* sehr gut beschrieben. Schlechte Programmierer sind nur einer von vielen Gründen, warum neun von zehn Startups nicht erfolgreich sind. Um Teil der erfolgreichen zehn Prozent zu sein, müssen die Entwickler eine deutlich breitere Palette an Fähigkeiten besitzen. Sie müssen verstehen, was ein „Minimum Viable Product" ist oder wie Business-Entscheidungen getroffen werden. Sie müssen ein Verständnis vom Markt entwickeln, Marketing-affin sein, User Experience beherrschen, People Skills meistern, „Pivot"-Möglichkeiten erkennen, Networking betreiben etc.

Wir reden hier nicht mehr von einfachen Programmierern, wir reden von eierlegenden Wollmilchsäuen, wir reden von Einhorn-Entwicklern. Spoiler Alert: Diese wachsen leider nicht auf Bäumen! Zu dieser Erkenntnis bin ich nach einer Vielzahl von Vorstellungsgesprächen als Interviewer gekommen. Wenn ich Obelix zitieren könnte: „Die spinnen, die Leute".

Wie suche ich Einhorn-Entwickler?

Hier ist meine Vorstellungsgespräch-Geheimsauce: keine Trickfragen, keine Whiteboard-Programmieraufgaben, keine Brain-Teaser … ich suche einfach Vernunft. Ich erwarte zuerst einmal, dass ihr eure Geschichte erzählen könnt: Euer Leben ist euer Thema überhaupt. Das kennt ihr am besten. Ich kann euch nicht widersprechen. Ihr könnt nicht falsch liegen. Könnt ihr über dieses „einfache Thema" erzählen? Könnt ihr darüber klar kommunizieren? Habt ihr die Abzweigungen in euren Geschichten kritisch hinterfragt? Wisst ihr, wie ihr an dem heutigen Tag angekommen seid? Denn: Kann ich wirklich erwarten, dass ihr eine komplexe architekturelle Entscheidung kommunizieren könnt, wenn ihr es über ein Thema, das ihr komplett beherrscht, schon nicht schafft? Kann ich erwarten, dass ihr Prozesse und Entscheidungen kritisch hinterfragen werdet, wenn ihr nicht mal eure eigene Geschichte unter die Lupe genommen habt?

Ihr wärt überrascht, wie niedrig die Anzahl von Leuten ist, die diesen einfachen Test bestehen. Ich passe natürlich meine Erwartungen an externe Faktoren an: Welche Rolle will ich gerade besetzen? Interviewe ich einen Kandidaten für einen Feuerwehreinsatz, also jemand, der sofort einsatzbereit ist? Oder suche ich ein High-Potenzial, gegebenenfalls frisch von der Uni, das ich ausbilden werde? Wie viel Erfahrung hat der Kandidat überhaupt? Hat er oder sie schon mehrmals diese stressige Interview-Situation erlebt? Die Kernfrage bleibt jedoch die Konstante in allen Interviews, die ich leite. Damit habe ich den Kandidaten in Bezug auf Kommunikation, Logik und die Möglichkeit, sich selber zu kritisieren, erforscht.

Das kann aber doch nicht alles sein? Ich lege natürlich den Fokus auch auf technische Fähigkeiten. Jemand, der seine Geschichte gut erzählen, aber nicht programmieren kann, wäre mir keine große Hilfe. Ich suche allerdings auch hier etwas Besonderes. Ich suche Leute mit besonderen Stärken, nicht Leute, die keine Schwächen haben. Diese Stärken werden meine Mannschaft unterstützen. Die Schwächen kann ich dann mit den Stärken der anderen Team-Mitglieder ausgleichen. Ich setze also auf Lernfähigkeit.

Dennoch stellt sich die Frage: Gibt es so etwas wie eine zu starke Schwäche? Ja klar, wenn es den Lernprozess behindern würde. Soft-Skills zum Beispiel sind enorm wichtig, um Coaching zu ermöglichen. Dale Carnegie hat es im Buch „How to win friends and influence people" so ausgedrückt, dass zu 85 Prozent der Erfolg einer Person an die Soft-Skills angelehnt ist. Ich stimme ihm zu.

Wie erfolgreich war ich bisher mit diesem Vorgehen? Sehr. Oder sehr wenig. Je nachdem. Die False-Positive-Quote blieb sehr gering, die Rejection-Rate war hingegen sehr hoch. Können wir das als Erfolg deklarieren? Leider nur bedingt.

Das hat dazu geführt, dass ich mich selbst infrage gestellt habe. Dass ich diesen gerade eben beschriebenen Prozess hinterfragt und verbalisiert habe. Dass ich Leute interviewt habe, um ihre Geschichten zu hören. Mein Fazit ist dasselbe wie das in Chad Fowlers Blog *[2]*: „There's a person I want to work with. I can't find this person. I've literally searched the world, and I can hardly find a trace. I'm not talking about someone specific. In fact, that's the problem. I'm talking about a set of traits and an attitude which is more scarce than I realized until recently."

Sucht Chad auch nach Einhorn-Entwicklern? Es scheint so. Wie sieht dieser Entwickler überhaupt aus? Ich empfehle sehr stark, diesen Blog-Eintrag zu lesen. Dort findet ihr eine gute Auflistung der Eigenschaften, die er sucht. Diese Eigenschaften sind jedoch sehr „High-Level", etwa „Everyone knows that when you take on a task whether it's huge and scary or tiny and boring, you're going to see it through to the best of your ability." Ich teile seine Meinung und habe mir mein eigenes Bild gemacht. Wie also sieht für mich der Einhorn-Entwickler aus?

Der Einhorn-Entwickler schont seinen 640K-Gehirn-Arbeitsspeicher

„640K is more memory than anyone will ever need", hat vermeintlich Bill Gates gesagt. Über wie viel Arbeitsspeicher unsere Gehirne verfügen, ist unbekannt, jedoch handelt es sich um eine endliche (kleine) Menge. Ich suche zuerst einmal Entwickler, die das Schonen dieser Ressourcen gelernt haben.

Welchen Anteil eurer Zeit verbringt ihr an einen normalen Tag beim Codelesen statt -schreiben? Für Robert Martin, Autor des Buchs „Clean Code", sind 90 Prozent die Norm. Die zwei Minuten, die ihr braucht, um einen Kaffee zu holen, sind schon lang genug, um die letzten Code-Zeilen, die ihr geschrieben habt, zu vergessen. Zurück an der Tastatur werdet ihr Minuten brauchen, um das Problem wieder in euren Gehirn-RAM zu laden. Dazu müsst ihr euren eigenen Code lesen und die Intention verstehen. Das waren nur zwei Minuten und der eigene Code. Wie ist es mit Legacy-Code, der fünf Jahre alt ist und von jemand anderem geschrieben wurde?

Ich suche Leute, die diese Diskrepanz verstanden haben. Denn die Kunst des Programmierens liegt nicht in der Übersetzung unserer Ideen in Einsen und Nullen, sondern in der Fähigkeit, unseren Code morgen noch zu verstehen. Dies ist wichtig, weil wir einen Kampf gegen Komplexität führen. Komplexität hindert uns daran, schnell und agil zu reagieren. Kennt ihr die Begriffe „essenzielle Komplexität" und „akzidentelle Komplexität"? Essenzielle Komplexität ist die beschriebene Komplexität im Begriff „Rocket Science", während akzidentelle Komplexität eine Messung davon ist, wie schlecht wir programmieren (Zitat von J. B. Rainsberger: „How much we suck"). Essenzielle Komplexität können wir nur bedingt ändern, aber akzidentelle Komplexität sollte Richtung null tendieren.

Clean Code wird uns die nötige Flexibilität für Experimente geben, um unseren Code zu erweitern und unsere Business-Anforderungen umzusetzen. Nehmen wir an, dass unser Code jetzt lesbar, erweiterbar

und sauber ist. Fühlen wir uns wohl dabei, alles auf den Kopf zu stellen? Fühlen wir uns bereit dazu, Teile des Projekts anzufassen, die wir noch nie gesehen haben? Wahrscheinlich nicht. Dafür brauchen wir ein Sicherheitsnetz in Form von Tests. Diese Tests müssen wiederholbar, nachhaltig, zuverlässig und schnell, also automatisiert sein. Es ist wieder derselbe Gedanke. Ohne dieses Netz behalten wir Bedenken im Kopf, haben Angst, Dinge zu ändern. Diese Bedenken fressen unsere Ressourcen. Mit diesen Bedenken sind wir nicht in der Lage, schnell und agil zu reagieren. Wir können keine Experimente durchführen.

Ich suche also Leute, die verstanden haben, dass Tests nicht wegen Metriken geschrieben werden. Ich brauche Leute, die begriffen haben, dass „Test first or debug after" nicht nur ein lustiger Spruch ist. Ich benötige Leute, die verinnerlicht haben, dass Tests zur Entwicklung gehören und die beides nie gedanklich trennen. Ich brauche Entwickler, die dieses Sicherheitsnetz effizient und effektiv bauen, aus eigenem Wohlgefühl.

Das Letzte, was uns daran hindern wird, schnell und agil zu agieren, sind Werkzeuge. Erinnert euch daran, wie es war, Autofahren zu lernen. Ihr wart bestimmt wie ich, voll auf die Bewegungen konzentriert: Kopplungspedal drücken, vom zweiten zum dritten Gang (hoch-rechts-hoch) schalten, dann Kopplungspedal langsam heben. Als ihr vollkonzentriert an diese Bewegungen gedacht habt, habt ihr weniger auf den Verkehr geachtet. Eine Vollbremsung in diesen Moment hätte möglicherweise böse Auswirkungen gehabt. Um souverän fahren zu können, habt ihr diese Bewegungen ins Unbewusstsein verschoben. Sie sind zu einem Automatismus geworden. Ihr habt das „Fahren vom Fahren entfernt".

Wer sich nicht traut, Code zu löschen, obwohl er im Quellverwaltungssystem gespeichert ist, oder sich nicht traut, Generatoren zu verwenden, und stattdessen alles per Hand schreibt, wer Tastatur-Abkürzungen nicht kennt oder wer mit weniger als sechs Fingern, ohne zu schauen, tippt, meistert wichtige Werkzeuge nicht. Ohne diese Werkzeuge hat euer Umfeld nicht eure volle Aufmerksamkeit. Ohne diese Aufmerksamkeit seid ihr allerdings nicht in der Lage, frei zu experimentieren.

Neben diesen technischen Teilen suche ich im Bereich „Soft Skills und Achtsamkeit" bestimmte Fähigkeiten.

Der Einhorn-Entwickler jagt im Rudel

Ich glaube nicht an den Mythos des einsamen Wolfs, der allein sein Wissen akquiriert hat und ganz allein, von der Welt abgeschottet, Probleme löst. Ich habe mit voller Begeisterung die Kommunikation in Open-Source-Projekten beobachtet. Hinter ihren Rechnern haben die Teilnehmer sehr hohe Erwartungen und sind messerscharf in ihrer Kritik, aber auch empathisch und voller Begeisterung für den Menschen. Ich glaube an die Energien, die wir in Paaren, Mobs oder Gruppen erzeugen, und die Synergie, die es uns bringt. Ich suche insofern Motoren dafür, also Leute, die aktiv danach suchen.

„Was ist mit ausgewogenen Teams?", werdet ihr fragen. Es klingt, als ob ich nur extrovertierte Leute suchen würde. Ihr habt Recht. Es kann nicht nur Alpha-Tiere in einem Team geben. Wir brauchen auch introvertierte und zurückhaltende Leute. Das ist richtig, solange wir uns nicht selbst täuschen. Wir nutzen zu oft diese Idee, um giftiges Verhalten im Team zu akzeptieren. Zurückhaltendes und introvertiertes Verhalten ist okay, solange derjenige – im geschützten Kontext des Teams – auch aktiv, initiativfreudig, neugierig, engagiert, mit dem Status-quo nie zufrieden, selbstständig und selbstkritisch ist.

„Man lernt, wenn man lehrt", hat Seneca geschrieben. Lehren ist ein wichtiges Werkzeug, um Neugier zu befriedigen. Habt ihr schon mal Code-Reviews durchgeführt? Habt ihr vielleicht ein Training vorbereitet und gehalten? Eine Frage auf Stack-Overflow gestellt? Der kleinste gemeinsame Nenner zwischen all diesen Tätigkeiten ist die Ausformulierung einer Idee und dadurch die Festigung eures Wissens. Damit entdecken wir Löcher in unserem Wissen. Wir werfen ein neues Licht auf Fragen, die wir längst als „beantwortet" begraben hatten. Wer lehren kann, kann kommunizieren, ist empathisch, kann sich projizieren und hat Geduld. Viele Fähigkeiten, die ich zu lieben gelernt habe.

Wo kann man heutzutage lehren? Überall. In der Stille des eigenen Zimmers, durch das Schreiben von Blogs oder Artikeln. Im Büro unter anderen durch Code-Reviews und „Brown Bag Lunch"; außerhalb der Firma durch Book-Clubs, User-Groups oder Konferenzen.

„Sich projizieren können" habe ich als nötige Fähigkeit zum Lehren genannt. Um sich projizieren zu können, muss man selber eine Introspektion führen. Erst wenn ihr verstanden habt, wie ihr selbst tickt, könnt ihr es an anderen Leuten untersuchen. Diese Introspektion ist das japanische „Hansei", was in etwa dem Sprichwort „Selbsterkenntnis ist der erste Schritt zur Besserung" entspricht. Wir müssen uns verstehen, um weiterzukommen.

Konfuzius hat es so ausgedrückt: „Wähle einen Beruf, den du liebst, und du brauchst keinen Tag in deinem Leben mehr zu arbeiten." Was liebt ihr? Warum seid ihr jetzt da? Warum lest ihr überhaupt diesen Artikel? Wo wollt ihr hin? Oder noch materialistischer: Wie arbeitet ihr am besten? Im Sitzen? Im Stehen? Open Space? Mit viel Energie? Lieber ganz ruhig in einem Einzelbüro oder sogar von daheim aus? Kühl? Warm? Früh? Spät? Schnell? Hektisch? Langsam? Posiert? Kennt ihr euer psychologisches Profil? Wissen ihr, mit welcher Art von Leuten ihr am besten umgehen könnt? Mit welchen nicht? Wer sich kennt, kann entscheiden, in welchem Umfeld er „rocken" wird. Denn es muss nicht mehr bewiesen werden, dass motivierte und glückliche Mitarbeiter immer effektiver sind.

Das Wort „effektiv" im letzten Absatz habe ich nicht zufällig ausgesucht. Ich meine nicht effizient, sondern in der Tat effektiv. Effizienz ist die Kunst, irgendetwas schnellstmöglich zu machen. Effektivität ist die Kunst, das Richtige zu tun. Ich brauche keine „Cyborgs", die Effizienz zur Hochkunst entwickelt haben, sondern Profis, die das Richtige für ihre Firmen, Projekte oder Kunden machen. Das heißt auch, dem Kunden (der König ist) contra zu geben, wenn er Bullshit macht und irgendetwas verlangt, das gegen unsere Ethik verstößt.

Ich brauche Profis, die ihre Work-Life-Balance wahrnehmen und Spaß nie aus den Augen verlieren. Scott Hanselman hat es so in einem Tweet geschrieben: „If your code doesn't save babies, you need to chillout." Sind es eure Banking-Software oder Versicherungsrechner wert, eure Kinder jeden zweiten Tag nicht ins Bett bringen zu können?

Drei Werte für den Alltag

Was ich im Endeffekt suche, ist das Bestreben (weil gleich nochmal „Lust" kommt), folgende drei Werte im eigenen Leben zu maximieren: Sorgfalt, Lust zu bauen und Kritikfähigkeit. Das ist die Kunst der Einhorn-Entwicklerschaft. Einhorn-Entwickler, wie hier beschrieben, existieren selbstverständlich nicht. Aber diese Werte im eigenen Leben zu berücksichtigen und stets zu versuchen, ihnen treu zu bleiben, ist das Mindset, das ich suche.

Ihr müsst diesen Weg jedoch zum Glück nicht alleine zurücklegen. Eure beste Trumpfkarte nennt sich „Mentoring". Vergesst das Internet, unsere moderne Bibliothek von Alexandria, wo alles und das Gegenteil (nichts?) gleichzeitig zu finden ist. Wendet euch stattdessen lieber an eure Mitmenschen. Werdet Mentoren und Mentees und helft euch gegenseitig. Interessiert euch für „den Menschen". Zeigt diesen Vertrauenspersonen eure Schwäche und arbeitet gemeinsam daran, diese zu verbessern. Fördert euch, motiviert euch gegenseitig und lernt, Leute zu begleiten. Das ist das wahre Leben. Oder wie Winston Churchill gesagt hat: „We make a living by what we get. We make a life by what we give."

Also bleibt technisch auf Zehenspitzen, qualitätsfokussiert und sorgfältig, sodass eure Zukunft nicht künstlich verkompliziert wird. Jagt im Rudel, offen, freizügig und behaltet im Kopf, dass der Mensch doch die wichtigste Variable in unserem Leben ist. Stellt euch selber infrage und versucht, ein besserer Mensch zu werden. Mit diesem Mindset, und egal ob ihr ein Ergebnis erreicht oder nicht, habt ihr die erste Stufe zur Einhorn-Entwicklerschaft erklommen. Vielleicht sehen wir uns ja in meinem Team.

Quellen

[1] Jeff Attwood „We hire the best, just like everyone else": *https://blog.codinghorror.com/we-hire-the-best-just-like-everyone-else*

[2] Chad Fowler „Who I want to hire": *http://chadfowler.com/2013/04/09/who-i-want-to-hire.html*

Timothée Bourguignon

tim@mathema.de

Timothée Bourguignon ist als Chief Learning Officer (CLO) und Agile Coach bei MATHEMA Software GmbH tätig. Seine Themenschwerpunkten umfassen Agilität, Lean, Scrum & Kanban, Mentoring, Teambuilding und noch mehr. Daneben hält er regelmäßig Vorträge auf Fachkonferenzen und schreibt Artikel in Fachmagazinen. In seiner Freizeit, wenn seine kleine Familie schläft, kümmert er sich lieber um die Umwandlung seiner Kollegen in mythische Wesen durch seinen Developer's-Journey-Projekt („www.devjourney.info").

Neuer Teilnehmerrekord auf dem JUG Saxony Day

Highlights, Rekorde und Spätsommersonne

Caroline Titze, JUG Saxony

Am 6. Oktober 2017 fand bereits zum vierten Mal in Radebeul die jährliche IT-Konferenz des JUG Saxony e.V. statt. Das vielseitige Vortragsprogramm, das in fünf Tracks Themen von Microservices über agile Softwareentwicklung bis hin zu Technologien wie Spring und Kotlin abdeckte, lockte in diesem Jahr 500 Teilnehmer auf den JUG Saxony Day – ein neuer Rekord und ein toller Erfolg für die Veranstalter.

Zu den Highlights zählte neben der Keynote von Stefan Zörner zum Thema „Softwarearchitektur für alle" der Vortrag „Rundungsfehler" von Michael Wiedeking, der sich dem Aufbau von Gleitkommazahlen und den Problemen, die BigDecimals mit sich bringen, widmete. Das begeisterte Publikum wählte ihn per App zum besten Speaker des Tages. Den zweiten Platz holten sich Marcel Schiffe und Mirko Seifert, die die Teilnehmer mit einem interaktiven Workshop überzeugen konnten.

Die Pausen wurden genutzt, um Wissen auszutauschen, Kontakte zu pflegen und zu knüpfen sowie sich mit heißem Kaffee für die nächste Session zu stärken. Nicht nur für die teilnehmenden Studierenden, denen dank der Sponsoren kostenfreie Tickets zur Verfügung gestellt wurden, bot sich die Gelegenheit, mit den Ausstellern ins Gespräch zu kommen und sich über Karrieremöglichkeiten zu informieren.

Das letzte Highlight der Konferenz stellte die große Abschlussverlosung dar, bei der unter anderem zahlreiche aktuelle Fachbücher verlost wurden. Unter den letzten Strahlen der sächsischen Spätsommersonne ließen die Teilnehmer anschließend bei kühlen Getränken, Leckereien vom Grill und angeregten Diskussionen den Tag ausklingen. Alles in allem ein erfolgreicher Konferenztag!

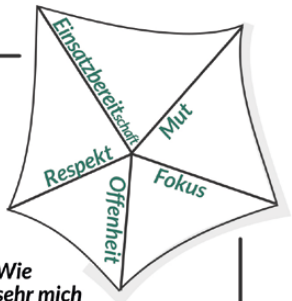
Der Termin für den nächsten JUG Saxony Day steht bereits fest. Am 28. September 2018 begrüßen wir euch zur nächsten Konferenz.

Caroline Titze
team@jugsaxony.org
www.jug-saxony-day.org

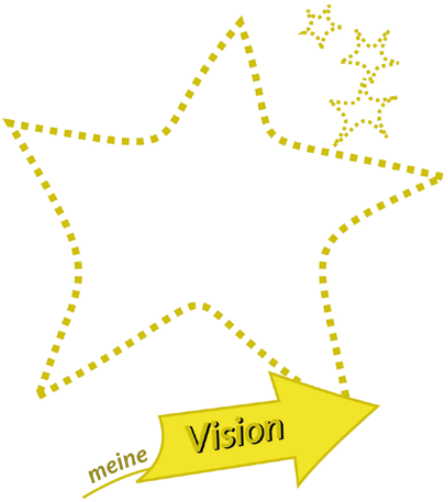
Dein agiles Selfie

Name *Julia Dellnitz, Dina Sierralta und Kerstin Wehner, smidig-Netzwerk*

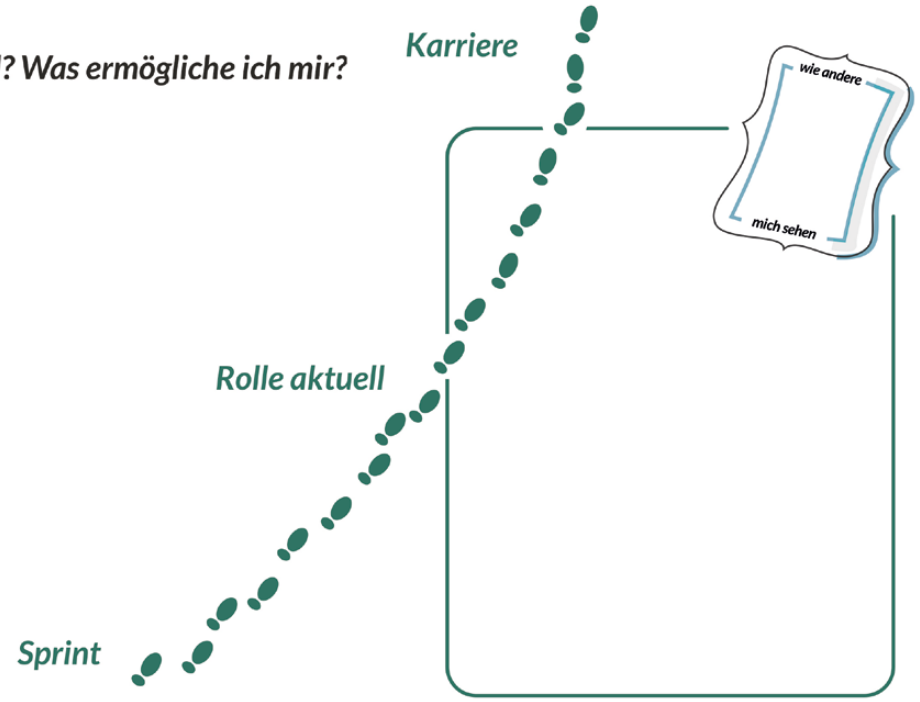
SCHWERPUNKTE meiner Arbeit



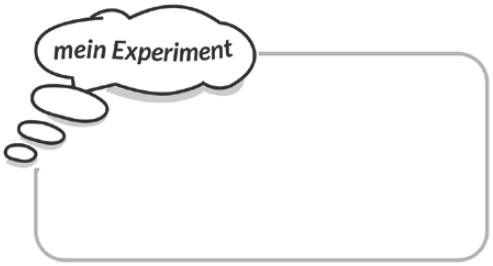
Wie sehr mich die agilen Werte leiten

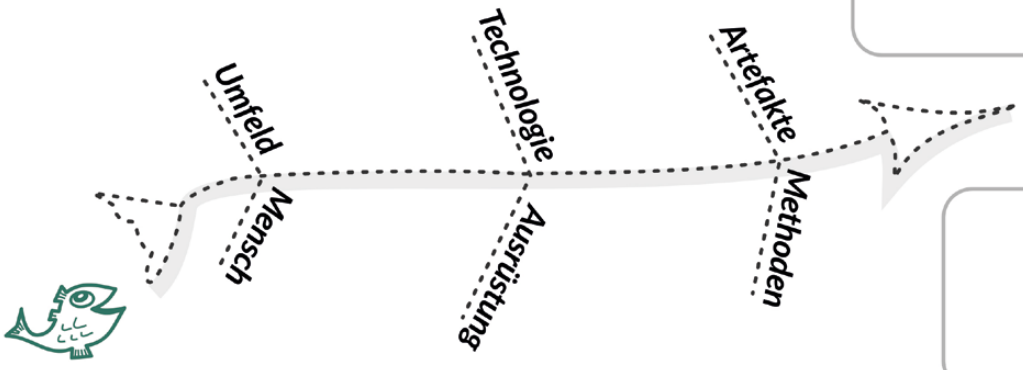


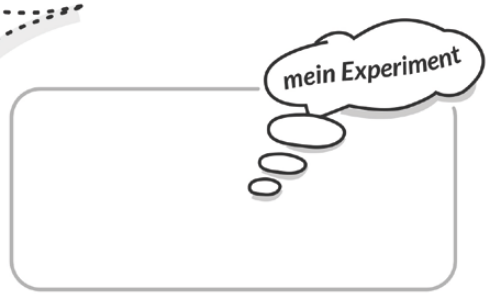
Warum arbeite ich agil? Was ermögliche ich mir?



Wie fange ich an? Wie mache ich weiter?







Das agile Selfie ist ein visuelles Selbstmanagement-Werkzeug. Es dient zum Sammeln von Anregungen für die berufliche und persönliche Entwicklung, macht deine eigenen Gedanken sichtbar und lädt zum Austausch mit anderen ein. Du kannst diesen Artikel wie eine Retrospektive mit dir selbst lesen. Dazu brauchst du etwa dreißig Minuten Zeit, ein großes Blatt Papier und einen Stift. Auf geht's!

Ist dir schon einmal aufgefallen, dass das Wörtchen „selbst" in der agilen Welt einen sehr prominenten Platz hat? „Wir erschließen bessere Wege, Software zu entwickeln, indem wir es selbst tun und anderen dabei helfen …". Es steht schon im agilen Manifest ziemlich weit vorn und findet sich an vielen anderen Stellen wieder – als selbstorganisierendes Team in Scrum, als Respekt für sich selbst in XP oder getarnt als Selbst-Appell in den agilen Prinzipien: „Heiße Änderungen willkommen …"

Agil ist nichts, worauf du warten musst

Du kannst selbst entscheiden, ab sofort agil(er) zu arbeiten, deine eigenen Aufgaben aus agiler Perspektive zu betrachten und auf andere mit einer agilen Haltung zuzugehen. Bei der Umsetzung kann diese Entscheidung allerdings zu einer echten Herausforderung werden. Auf dich selbst zu vertrauen, darauf zu achten, was dich motiviert, und auch Verantwortung für ein gemeinsames Ergebnis zu übernehmen, fällt insbesondere dann schwer, wenn Unternehmenskultur und -struktur in eine völlig andere Richtung ziehen. Der Rückfall in alte Muster und Verhaltensweisen ist so gut wie vorprogrammiert.

Aus agiler Sicht ist es angeraten, dich nicht in eine neue Situation zu zwingen, sondern nach Möglichkeiten Ausschau zu halten, in denen du dich jeweils in neuen Rollen und Zusammenhängen erproben und neue Aspekte an dir entdecken kannst. Die Software-Entwicklung bietet dir dazu drei unterschiedliche Felder, in denen du neue Fähigkeiten erproben und Kompetenzen erwerben kannst. Das funktioniert übrigens auch dort, wo die anderen (noch) gar nicht agil sind:

- Die Arbeit am Produkt, also iterativ-inkrementell Produkte entwickeln, die Nutzen für Kundinnen und Kunden schaffen und die wertschöpfend sind
- Die Arbeit am Unternehmen, also Orientierung geben, um einen organisatorischen Rahmen für agiles Arbeiten zu schaffen und die Ergebnisse und Erkenntnisse aller (agilen) Teams immer wieder neu zu verweben
- Die Arbeit an dir selbst, also deine eigene berufliche Entwicklung bezogen auf technische, soziale und persönliche Kompetenzen, die du bei deiner Arbeit einsetzen willst

Das agile Selfie unterstützt dich bei dieser Suche nach Anregungen für deine Entwicklung, es macht deine Gedanken sichtbar und lädt zum Austausch mit anderen ein. Wie bei einem echten Selfie hast du es in der Hand, wie du erscheinen möchtest und mit wem du dieses Bild teilst. Du kannst in einzelne Bereiche hineinzoomen und dir die Aspekte deiner persönlichen Entwicklung genauer anschauen, die gerade interessant für dich sind. Und wenn du etwas Spannendes entdeckst, kannst du auswählen, ob und mit wem du diese Entdeckungen teilst, Fragen erörterst oder dir Feedback einholst.

So funktioniert das agile Selfie

Wir haben das Selfie in zwei Teile aufgeteilt, weil es Erkenntnisse aus der Neurowissenschaft gibt, dass sich unser Gehirn schwertut, gleichzeitig über das „Warum?" und über das „Wie?" nachzudenken. Die Fragen im ersten Teil geben dir also einen Einblick in dein persönliches „Warum?": Wo stehst du in deiner Entwicklung, wo willst du vielleicht in Sachen Agilität noch hin, welchen Weg willst du beruflich gehen? Im zweiten Teil geht es um das „Wie?" und wir lenken deine Aufmerksamkeit in Richtung Handeln: Was willst du konkret ausprobieren, um dich weiterzuentwickeln, was brauchst du dafür, wer kann dir eine Hilfe sein?

Du kannst das Selfie übrigens auch füllen, wenn du nicht agil arbeitest. Jetzt brauchst du nur noch ein großes Blatt Papier, einen Stift, ein wenig Zeit und Muße sowie einen Ort, an dem du gut nachdenken kannst. Mehr Spaß macht das Ganze übrigens mit unserem A3-Poster. Du kannst es dir unter *„https://www.smidig.de/agile-selfie.de"* herunterladen und dann selbst ausdrucken. Bist du bereit? Dann los!

Schwerpunkte deiner Arbeit

Wir starten mit deiner täglichen Arbeit. Nimm dir ein paar Minuten Zeit und überlege mal, womit du dich den lieben langen Arbeitstag so beschäftigst *(siehe Abbildung 1)*:

- Was sind die fünf Schwerpunkte deiner Arbeit?
- Was tust du gern? Welche Tätigkeiten machen dich aus?
- Was tust du oft oder viel?
- Wofür wirst du eigentlich wirklich bezahlt?

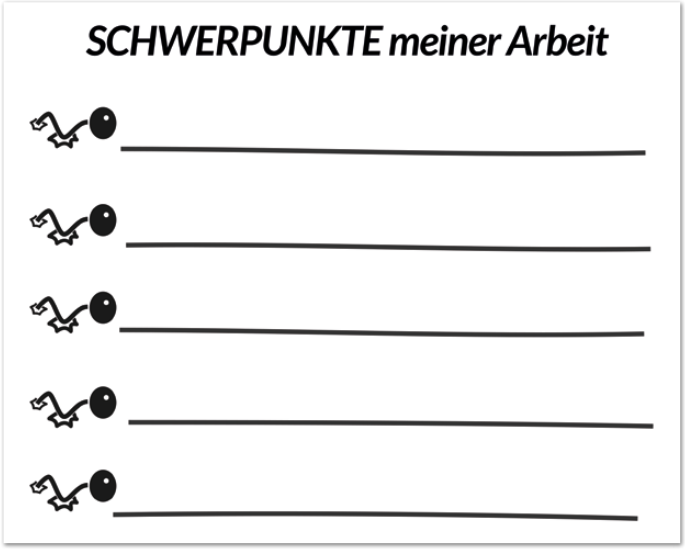


Abbildung 1: Schwerpunkte meiner Arbeit

Wenn du magst, bringst du alles in eine Reihenfolge. Was steht wirklich ganz oben bei deiner Arbeit? Du kannst auch markieren, wovon du gern mehr (+) und wovon du gern weniger (-) machen möchtest. Häufig finden sich so schon Anknüpfungspunkte für deine nächsten Schritte.

Agile Werte zur Orientierung

In der Agilität geht es darum, eine Kultur des systematischen Experimentierens und des Vertrauens zu etablieren. Unsicherheit soll konstruktiv genutzt werden. Das kann persönlich sehr fordern. Agile Werte wie Einsatzbereitschaft, Mut, Fokus, Offenheit und Respekt bieten dabei Unterstützung und Orientierung.

Wie wirksam du zur Arbeit in einem agilen Team beiträgst, hängt nämlich davon ab, wie gut es dir gelingt, nach diesen Werten zu arbeiten. Deswegen findest du sie auch auf dem agilen Selfie *(siehe Abbildung 2)*. Überleg doch mal, was sie für dich in deinem Arbeitskontext bedeuten und wie du sie lebst:

Was bedeuten dir Werte wie Mut, Offenheit oder Respekt?
Wie sehr bist du dir der Werte in deinem täglichen Tun bewusst?
Wie sehr nützen sie dir? Wie stark strebst du ihnen nach?

Markiere deine erste Einschätzung in dem Spinnennetz-Diagramm auf deinem Selfie-Bogen.

Deine Vision

Ein gutes agiles Team hat zumindest eine grobe Vision von seinem Produkt. Sie hilft, Entscheidungen zu treffen, den Weg zu finden und die eigene Motivation aufrechtzuerhalten. So eine Vision hilft natürlich auch, wenn es um die eigene berufliche Entwicklung geht *(siehe Abbildung 3)*:

- Welche Vision hast du vielleicht schon für deine eigene Entwicklung?
- Wo willst du beruflich ungefähr hin?
- Was soll agiles Arbeiten dir dabei ermöglichen?

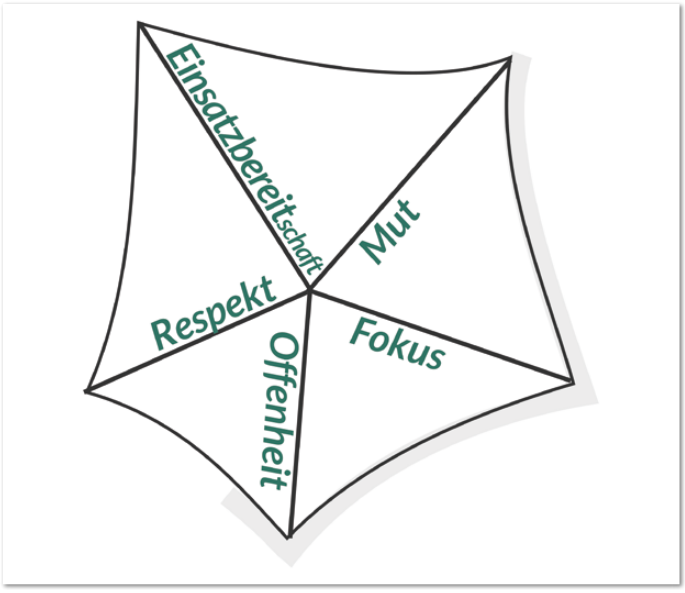


Abbildung 2: Agile Werte zur Orientierung

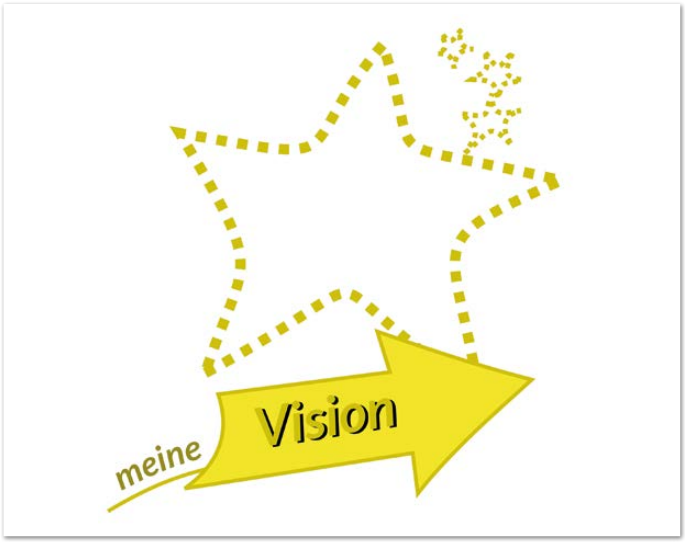


Abbildung 3: Meine Vision

Lass dir die Antworten zu den Fragen durch den Kopf gehen und versuche, ein paar Aussagen zu deiner Vision für dich selbst schriftlich festzuhalten.

Deine Stärken

Erinnerst du dich noch an die agilen Prinzipien? Ist schon eine Weile her? „Errichte Projekte rund um motivierte Individuen …" – klingelt da was? Gut! Agiles Arbeiten funktioniert nämlich dann am besten, wenn du motiviert bist, die Dinge selbst zu tun. Dazu ist es gut zu wissen, was du schon gut kannst und im Idealfall auch gern tust. Um diesen Aspekt deines Selfies zu beleuchten, kannst du dir Fragen wie diese hier stellen *(siehe Abbildung 4)*:

- Was kannst du wirklich gut?
- Was nimmst du dir aktiv, weil du es gut kannst oder gern tust?
- Was landet oft bei dir, weil du es gut kannst?
- Was lernen andere von dir?
- Worüber erzählst du gern?

Du kannst den Blick hier ruhig weiter fassen – ganz gleich, ob es um agile Qualitäten, technische Exzellenz oder Design geht. Für agiles Arbeiten brauchst du von all dem etwas. Deswegen lohnt es sich, bewusst aufzuschreiben, was du kannst und was dich ausmacht. Häufig finden sich in deinen Stärken auch Anregungen für deine Weiterentwicklung: Was davon möchtest du vertiefen oder weiter ausbauen?

Wie andere dich sehen

Gerade bei einer Arbeitsweise, die so stark auf Teamarbeit, Vielfalt und Interdisziplinarität setzt wie Agilität, ist es sinnvoll, nicht nur durch die eigene Brille zu schauen. Es lohnt sich, Kolleginnen und Kollegen zu fragen, wie sie deine Arbeit sehen *(siehe Abbildung 5)*. Du könntest sie zum Beispiel fragen:

- Wofür schätzt du mich?
- Welche Aufgaben vertraust du mir gern an?
- Bei welchen Themen bist du zögerlich? Weshalb?
- Was wünscht du dir für unsere Zusammenarbeit?

Abbildung 4: Meine Stärken

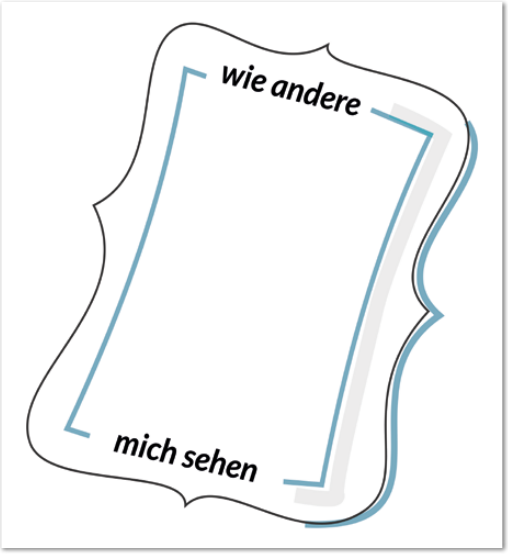


Abbildung 5: Wie andere mich sehen

Vielleicht hast du in letzter Zeit auch Feedback von jemandem bekommen, das du hier nochmal anschauen willst. Halte die Ergebnisse in dem kleinen Spiegel auf dem Selfie fest.

Es ist übrigens leichter, Feedback anzunehmen, das du dir aktiv einholst. Das liegt daran, dass du selbst offener bist, wenn du dies freiwillig tust, dass du dir verschiedene Perspektiven anschauen kannst und dass es vermutlich gerade einen guten Anlass für dich gibt, ein bestimmtes Thema anzugehen.

Die Etappen auf deinem Weg

Vielleicht entsteht langsam ein Bild in deinem Kopf, welche Themen dich besonders interessieren und wo deine agile Reise hingeht. Manche Themen mögen unmittelbar bevorstehen, etwa die komplexe Anforderung aus der User Story, an der du gerade arbeitest. Andere liegen noch in weiterer Ferne, etwa die Erweiterung, die du schon lange entwickeln willst, oder der Job in einer anderen Stadt, der dich schon länger reizt:

- Wo willst du dazu lernen?
- Was steht unmittelbar in deinem Job an? Was könntest du beispielsweise in eurem nächsten Sprint anders machen?
- Was willst du in deiner aktuellen Rolle an Fähigkeiten entwickeln?
- Was willst du für die nächsten Schritte in deiner Karriere ausbauen?

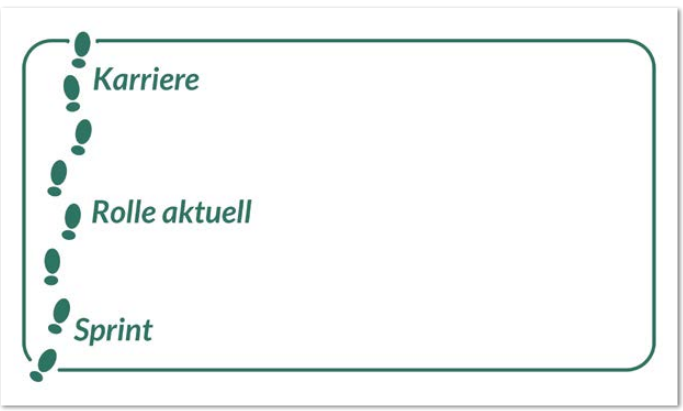


Abbildung 6: Die Etappen auf meinem Weg

Dabei geht es nicht nur um konkrete Pläne. Es können auch vage Idee, wilde Pläne oder auf den ersten Blick spinnerte Träume sein, die ein Zwischenschritt zu etwas Handfesterem sind. Wichtig ist, dass du sie auch mal formulierst und für den Moment festhältst *(siehe Abbildung 6)*.

Dein Experiment

Du hast jetzt sichtbar aufgeschrieben, was du tust, wo deine Stärken liegen, wie andere dich sehen. Hast dir auch klar gemacht, wohin deine agile Reise geht und welche Etappen auf dem Weg dorthin im Moment interessant oder möglich erscheinen. Jetzt lenke deinen Blick langsam aufs Tun. Halt einen Moment inne und frage dich:

- Was würdest du gern ausprobieren?
- Wozu hast du Lust?

Wenn dir spontan nichts einfällt, können dich auch ganz konkrete Anlässe auf die Spur bringen:

- Was stört dich – ganz massiv oder im täglichen Klein-Klein?
- Wo verlierst du Lust und Zeit?
- Was wolltest du schon immer mal tun – findest aber keine Zeit oder keinen Anlass?

Überlege dir ein oder zwei Experimente, die dich auf neues, unbekanntes Terrain bringen – raus aus deiner Komfortzone *(siehe Abbildung 7)*. Dann probiere sie aus. Schau, was gut für dich funktioniert und was nicht. Du kannst auch darauf achten, wie es dir unterwegs geht: Ist das ein gutes Tempo? Wie wirkt das Ganze auf deine Motivation? Reflektiere die kleinen Schritte, nutze weiter, was gut war, und passe an, wo etwas nicht gleich klappt. Inspect & adapt! Je zeitiger etwas nicht funktioniert hat, desto besser!

Zu einem echten Experiment gehört auch, dir klar zu machen, was genau du beobachten willst und woran du den Erfolg deines Experiments erkennst. Schreibe dir das ruhig dazu.

Impulse für deinen Weg

Deine eigene Entwicklung ist ein komplexes Unterfangen. Mal probierst du Dinge aus und sie funktionieren wunderbar. Du bist sicher,

dass etwas erfolgversprechend ist, und plötzlich passiert das Gegenteil. Manche Fähigkeiten entwickelst du schnell und an anderen Stellen fragst du dich, wie oft du das eigentlich noch machen willst, bis du es endlich begreifst. Für deine Experimente kann es also nicht schaden, nach ein paar Menschen und Dingen zu suchen, die dir helfen können.

Dazu haben wir das gute alte Fishbone-Diagramm leicht für dich abgewandelt. Schau dir die Überschriften der Gräten kurz an und beginne mit der, die dir spontan am leichtesten fällt. Hier sind ein paar Ideen für den Einstieg:

- *Menschen*
  Kolleginnen und Kollegen, von denen du dir Fähigkeiten abgucken könntest, Sprecherinnen und Sprecher oder Autorinnen und Autoren, die zu dem Thema veröffentlichen, Impulsgeberinnen und Impulsgeber in sozialen Netzen

- *Methoden*
  Scrum, Pair Programming, Design Thinking, testgetriebene Entwicklung, agile Architektur

- *Umfeld*
  In deinem Unternehmen, auf Konferenzen oder Meetups, in sozialen Medien oder Foren

- *Ausrüstung*
  Hardware, Möbel, digitale und analoge Tools zur Zusammenarbeit

- Technologie
  Konnektivität, Big Data, Machine Learning, BYOD, Digitalisierung

- *Artefakte*
  Backlog, Ausbaustufen eures Produkts, Sprintplanung

Notiere zu jeder Gräte zwei oder drei Ideen, die sich gut auf dein Experiment auswirken könnten. Dazu ein kurzes Beispiel: Stell dir vor, du möchtest mutiger werden. Wie könnte das aussehen? Wo kannst du deinen Mut testen? Du ärgerst dich ja seit Wochen über die Tests eines Kollegen; eure Zusammenarbeit und die Qualität eurer Ergebnisse leiden schon darunter. Es wäre gut, wenn du es mal ansprichst, aber schwierige Themen anzusprechen, fällt dir generell schwer. Dein Experiment könnte lauten: In der nächsten Retro einen guten Rahmen für ein offenes Gespräch Tests schaffen, konkret sagen, was dich ärgert, und eine Vereinbarung für den nächsten Sprint treffen.

Zur Vorbereitung gehst du jetzt einmal durch das Fishbone-Diagramm und sammelst Impulse:

- Menschen, die das gut können, beobachten, Perspektiven von Kolleginnen einholen
- Methoden: gewaltfreie Kommunikation, testgetriebene Entwicklung
- Umfeld: Scrum-Master ansprechen für eine offene Atmosphäre in der Retro
- Ausrüstung: Laptop dabeihaben, um gemeinsam etwas durchzutesten
- Technologie: Tools zur Testautomatisierung
- Artefakte: Konkrete Tests auswählen, an denen du dein Feedback festmachen kannst

Fazit

Wenn du alle Elemente des Selfies zusammenfügst, entsteht Schritt für Schritt dein agiles Selfie – eine Momentaufnahme deiner persönlichen und beruflichen Entwicklung unter agilen Gesichtspunkten.

Es kann hilfreich sein, dir von außen Rückmeldungen zu deinem Selfie zu holen. Du kannst zum Beispiel das Selfie mit Kolleginnen und Kollegen, deiner Führungskraft oder Mitarbeiterinnen und Mitarbeitern von dir durchgehen und ihre Sicht auf die Dinge einfügen. Es ist sinnvoll, auch jemanden einzubeziehen, der dich gut kennt, etwa deinen Partner oder einen Freund. Ebenso spannend ist es, wenn du dir einen Impuls von weiter weg holst, also jemanden befragst, den du auf einem Meeting oder auf einer Konferenz getroffen hast oder mit dem du über ein soziales Netz verbunden bist.

Ihr könnt die Selfies auch im Team nutzen und anreichern, indem ihr euch gegenseitig Feedback gebt, Ideen miteinander austauscht und

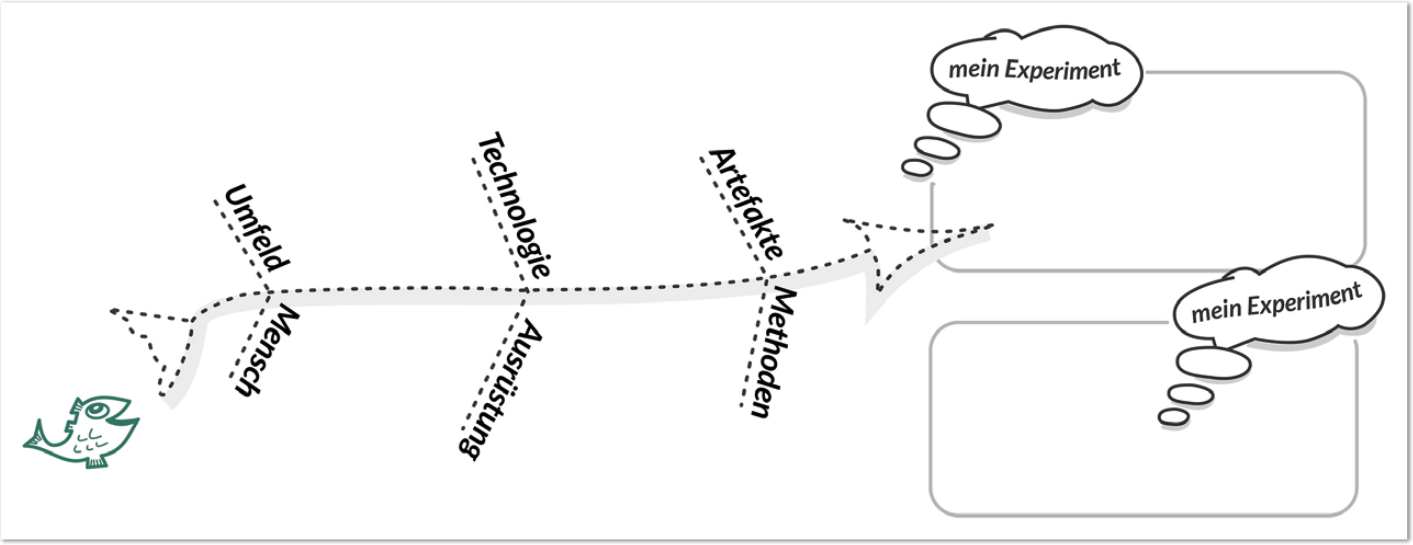


Abbildung 7: Mein Experiment

das Bild betrachtet, das sich zeigt, wenn man die unterschiedlichen Bereiche der Selfies aus einem Team gemeinsam betrachtet. Ganz im Sinne einer Retrospektive, die sich damit beschäftigt, was ihr gelernt habt und was ihr weiter verbessern könnt.

Wie beim echten Selfie lohnt sich eine erneute Aufnahme von Zeit zu Zeit, um die bereits stattgefundene Entwicklung sichtbar zu machen und dein Vorgehen anzupassen. So bekommst du dauerhaft einen fundierten Überblick über deine Lernerlebnisse, deine persönliche Entwicklung und die Früchte deiner Arbeit. In diesem Sinne: Cheeeeze!

Hinweis: Wir bedanken uns bei der COOK Trading Ltd aus der BCorp-Community für die Inspiration. COOK hat ein ähnliches Format zum selbstgesteuerten Performance-Review entwickelt, der uns gleich agil erschien und weiterentwickelt werden wollte.

Julia Dellnitz
julia@learnical.com

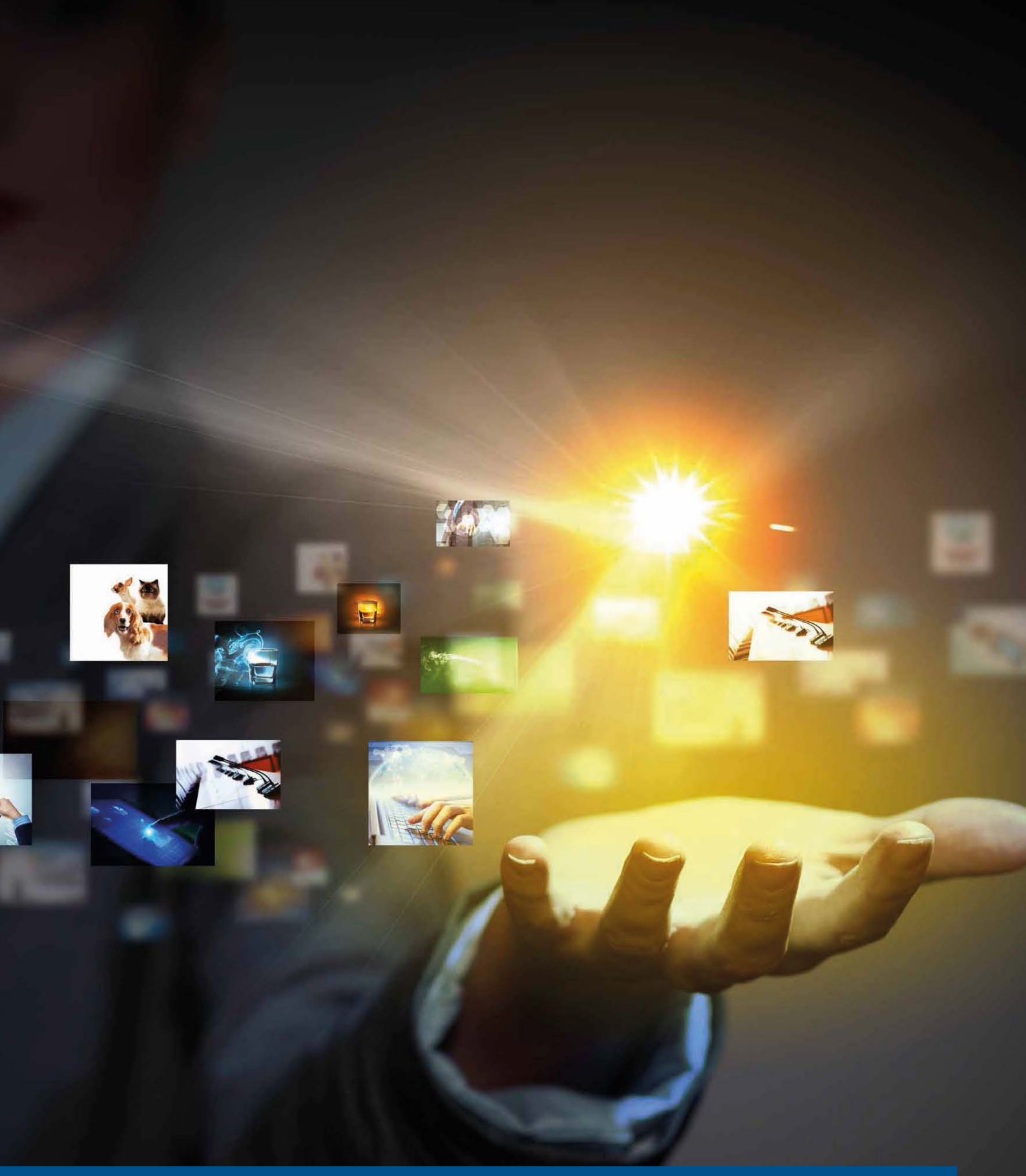
Julia Dellnitz ist Co-Gründerin von Learnical und dem smidig-Netzwerk. Die Ozeanographin und Betriebswirtin berät Unternehmen, die Agilität nicht nur in der IT einsetzen wollen. Sie bloggt und spricht auf internationalen IT-Konferenzen zu Agilität und Vielfalt.

Kerstin Wehner
kerstin@graphictranslation.com

Kerstin Wehner ist Führungskraft im Finanzdienstleistungssektor. Mit Stift und Papier facht sie mit Bildern und Worten Neugier an und macht sichtbar, wo Neues entsteht, etwa während einer agilen Transition. Kerstin hat einen MBA in England gemacht.

Dina Sierralta
dina@learnical.com

Dina Sierralta schafft im smidig-Netzwerk Räume für Menschen. Ihre Leidenschaft ist es, Gastgeberin für alle möglichen Situationen zu sein, in denen Menschen gemeinsam agil arbeiten und lernen können. Dina hat Pharmazie studiert und viele Jahre als Lean Managerin gearbeitet.



Design – das unbekannte Wesen

Alexander (Sascha) Klein, codecentric AG

Ein gutes Software-Produkt benötigt neben seiner Funktionalität auch eine gute Schnittstelle zum Benutzer. Dafür werden häufig Grafikdesigner engagiert. Doch Entwickler und Designer sprechen nicht immer die gleiche Sprache. Es entstehen Missverständnisse und Unstimmigkeiten.

In der Regel werden Designs umgesetzt, ohne dass die Entwickler verstehen, welchen Zweck sie verfolgen und warum die Dinge so sind, wie sie sind. Spätestens dann, wenn das Produkt die initiale Entwicklungsphase hinter sich gelassen hat, verlassen die Designer meist das Team und die Entwickler sind auf sich allein gestellt. Die ursprünglichen Design-Ziele werden dann schnell aus dem Auge verloren. Dieser Artikel hilft Entwicklern, die Sprache der Designer zu verstehen und Einblick in deren schillernde Welt zu erhalten.

„Features sind wichtiger als Bedienbarkeit"

Oft hört man die Überzeugung von Entwicklern und Geldgebern, dass es nur darauf ankommt, welche Funktionen eine Software hat, und es egal ist, wie sie aussieht und zu bedienen ist. Was stimmt an dieser Aussage? Wenn wir ehrlich sind, wird etwas, das schwer zu bedienen ist, nicht oder nur ungern benutzt – außer wir müssen es tun. Features, die nicht genutzt werden, bieten keinen Mehrwert und sind damit wertlos.

„Aus gutem Anwendungsdesign folgt gutes UI-Design"

Menschen denken in Funktionen und nicht in Abläufen. Statt „Die Maus in die Menüzeile bewegen und auf ‚Datei' klicken, dann die Maus über den Eintrag ‚Speichern' bewegen und die linke Maustaste drücken" wollen wir „das Dokument speichern". Wie dies zu bewerkstelligen ist, ist zweitrangig – sofern es für uns intuitiv und einfach zu erreichen ist. In der Softwareentwicklung allerdings sind die Abläufe unser täglich Brot, denn nur diese können direkt umgesetzt werden.

Zudem haben wir einen Interessenkonflikt zwischen Anwendungs- und UI-Design. Ziele im Anwendungsdesign sind beispielsweise Abstraktion und Wartbarkeit. Beim UI-Design zählen Intuitivität, kurze Wege und Übersichtlichkeit. Somit ist etwa Redundanz im Anwendungsdesign zu vermeiden, um Daten oder Code nicht mehrfach pflegen zu müssen. Allerdings ist Redundanz im UI-Design oft ein wichtiges Element – wer möchte schon am Anfang einer langen Webseite nach unten scrollen müssen, um auf die nächste Seite zu gelangen, wenn er den Inhalt schon gelesen hat, oder im anderen Falle nach oben scrollen, wenn er unten angelangt ist?

„Funktionalität ist die Summe der Möglichkeiten"

Funktionalität ist immer benutzerbezogen. Nur was dem Benutzer dient, ist für ihn Funktionalität. Kaum ein Benutzer nutzt alle Funktionen, die Excel bereitstellt, und nur was er nutzt, ist für diesen Benutzer die Funktionalität von Excel. Überspitzt gesagt, ist es für den einen ein Projektsteuerungstool, für einen anderen eine Reisekostenabrechnung und für den Dritten ein Rechenwerkzeug.

„Design ist kein Entscheidungskriterium"

Das menschliche Gehirn bildet sich in unter einer Sekunde einen ersten Eindruck, und nach etwa sieben Sekunden hat es eine Entscheidung darüber getroffen, ob man das Produkt kauft bezie-

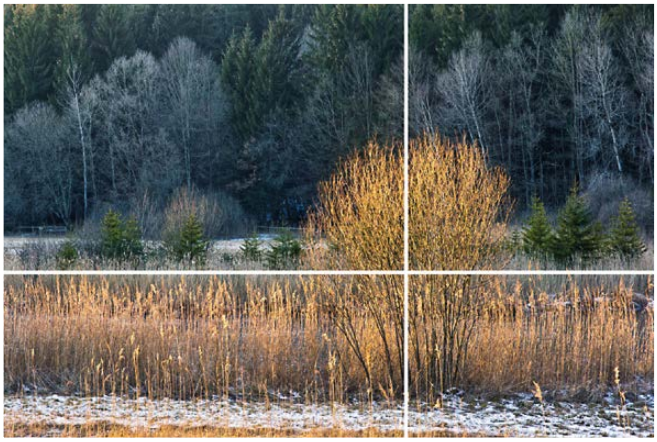


Abbildung 1: Der goldene Schnitt, Quelle „http://www.freeimages.com/photo/1414250"

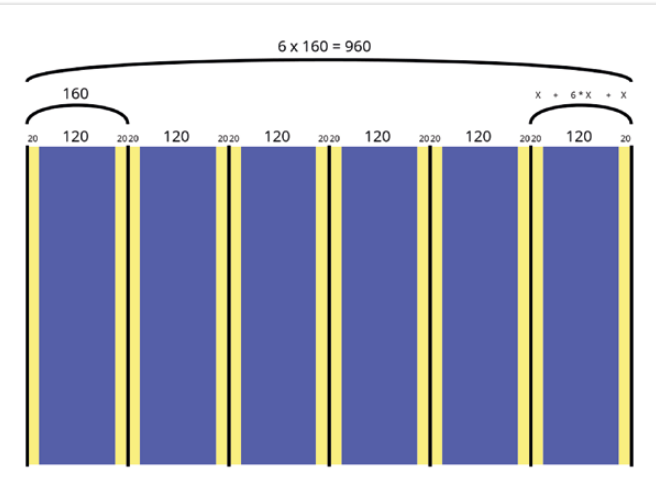


Abbildung 2: Das 960-Grid-System

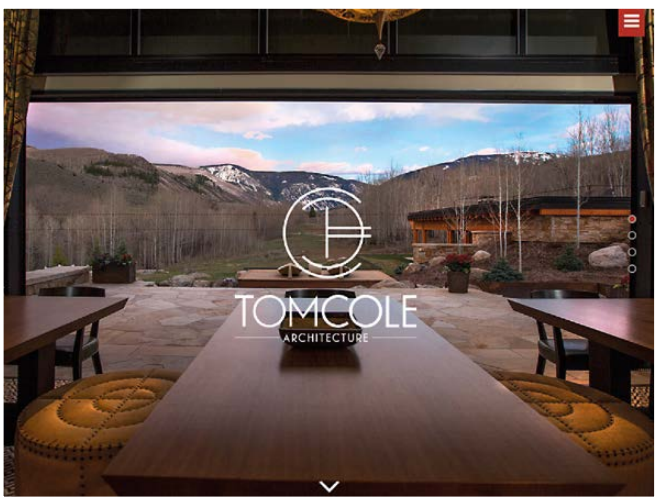


Abbildung 3: Ein Beispiel von symmetrischem Gleichgewicht, Quelle: „http://tomcolearchitect.com"

hungsweise verwenden möchte. Die gesamte restliche Zeit bis zu einer bewussten Entscheidung verbringt unser Unterbewusstsein damit, Gründe zu sammeln, um im Bewusstsein diese Entscheidung zu untermauern. Doch in sieben Sekunden hat man sich in der Regel noch nicht einmal einen Überblick über die Features eines Produkts verschafft – geschweige denn sie evaluiert.

Die erste Entscheidung wird auf Basis von Emotionen getroffen, die von Design und Bedienkonzept ausgelöst werden. Eine negative erste Entscheidung zu revidieren, ist nur in seltenen Fällen möglich und sehr schwierig. Demnach entscheiden Aussehen und Bedienbarkeit darüber, ob ein Produkt eingesetzt wird. Die Features hingegen entscheiden nur darüber, ob es dann doch nicht eingesetzt wird – falls zwar das Design, aber nicht die Features passen.

Ein paar Grundlagen

„Design" heißt auf Deutsch „Gestaltung" und bedeutet in der Regel „Entwurf" oder „Formgebung". Es setzt sich mit den Funktionen eines Objekts und der Interaktion mit einem Benutzer auseinander. Natürlich gilt dies nicht nur für physikalische Objekte, sondern auch für Software, Oberflächen und alle anderen Arten von Entitäten.

Eine Funktion ist eine zu erfüllende Aufgabe und im Allgemeinen in Form einer Schnittstelle zwischen zwei Objekten beschreibbar. Im Software-Bereich ist eines dieser Objekte in der Regel die Anwendung selbst, das andere ein Benutzer. Hier wird deutlich, dass der Benutzer also eine zentrale Rolle einnimmt und man grob sagen kann, dass jede Funktion eine Benutzerschnittstelle besitzt.

Ohne Benutzer gibt es keine Funktion und damit keine Anwendung. Ein Benutzer muss allerdings nicht zwingend ein Mensch sein. Im Falle von REST-Schnittstellen oder APIs sind dies andere Systeme – nichtsdestotrotz benötigen sie eine definierte Schnittstelle. Auch wenn diese anders aussieht und andere Techniken nutzt, die grundlegenden Regeln für Design gelten hier ebenso.

Es ist zu bedenken, dass es in den meisten Fällen mehr als einen Benutzer für eine Anwendung und meist auch für eine Schnittstelle gibt. Diese haben unterschiedliche Eigenschaften, Bedürfnisse, Restriktionen und Ziele, die es zu berücksichtigen gilt. Ein paar Beispiele von unterschiedlichen Benutzergruppen sind Erst- und Gelegenheitsnutzer, Power-User, Mausbediener, Tastaturkürzel-Liebhaber, Fremdsprachler, Menschen mit Behinderung und viele mehr. Daraus ergibt sich, dass im Design im Allgemeinen und im Grafik- und UI-Design im Speziellen der Fokus auf den Benutzer gelegt werden sollte. Ein paar goldene Regeln dafür sind:

- 20 Prozent der Funktionen werden in 80 Prozent der Zeit genutzt. In der Entwicklung und im Design sollten für diese 20 Prozent der Funktionen 80 Prozent des Aufwands aufgebracht werden.
- Über die Navigation beziehungsweise das Bedienkonzept sollten die 20 Prozent der wichtigsten Funktionen möglichst direkt, einfach und intuitiv zu erreichen sein.
- Ähnliche Funktionen sollten in der Bedienung auch ähnlich aussehen.
- Unterschiedliche Funktionen sollten sich in der Bedienung auch deutlich unterscheiden.
- Alles, was keinen Zweck verfolgt oder von ihm ablenken kann, sollte weggelassen werden.

Ästhetik

Die Ästhetik ist die Lehre der Schönheit und Harmonie. Doch Schönheit liegt bekanntlich im Auge des Betrachters. Trotz aller unterschiedlichen Auffassungen von Schönheit gibt es Gemeinsamkeiten,

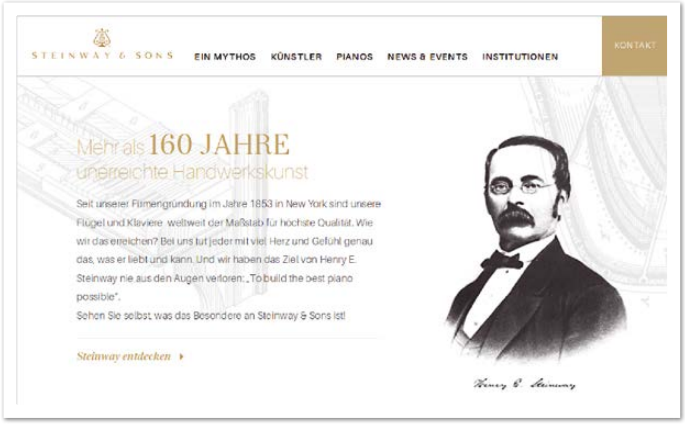


Abbildung 4: Ein Beispiel von asymmetrischem Gleichgewicht, Quelle: „http://eu.steinway.com/de"

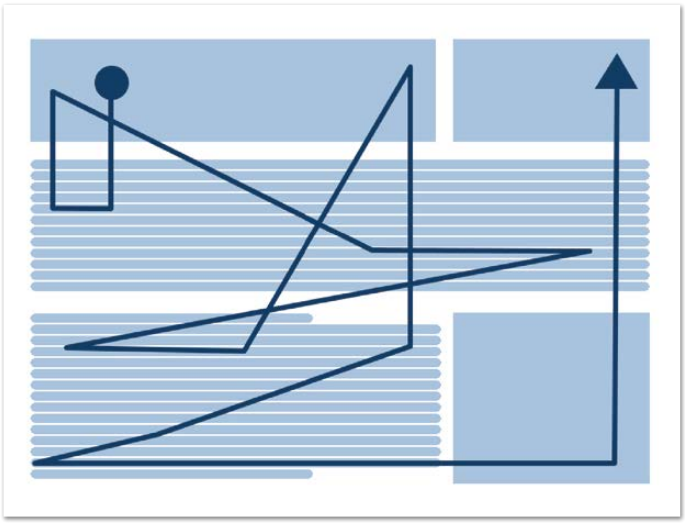


Abbildung 5: Häufiges Augenbewegungsmuster beim Betrachten einer Webseite

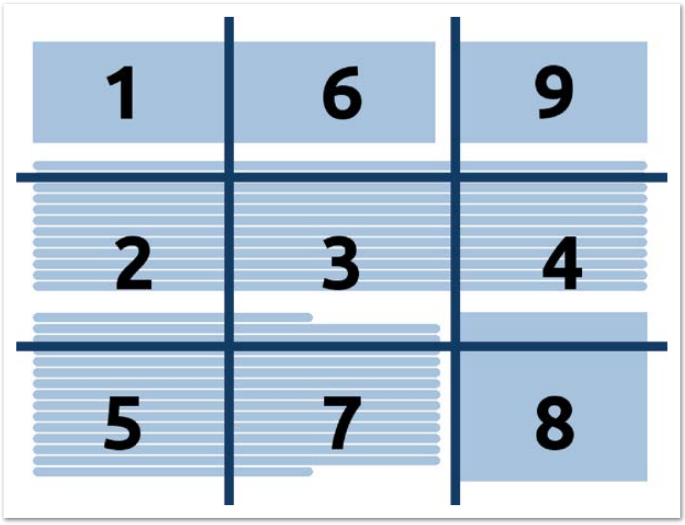


Abbildung 6: Reihenfolge, in der die Bereiche einer Webseite betrachtet werden

die darauf beruhen, wie unser Gehirn funktioniert. Am wichtigsten sind hier die idealen Proportionen, also das Verhältnis, das unser Gehirn als am harmonischsten empfindet: den goldenen Schnitt. Er ist das Verhältnis 1:Φalso ungefähr 1:1,618. Das Verhältnis 1:1

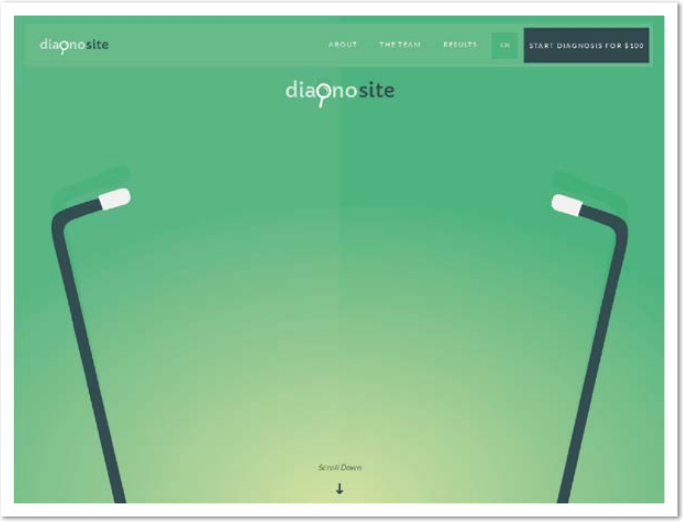


Abbildung 7: Echte Linien, Quelle: „http://diagnosite.com"

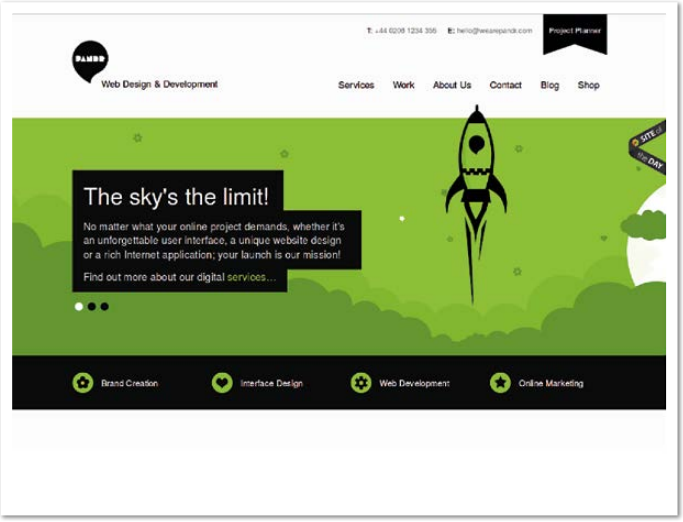


Abbildung 8: Gedachte Linien, Quelle: „http://www.wearepandr.com"

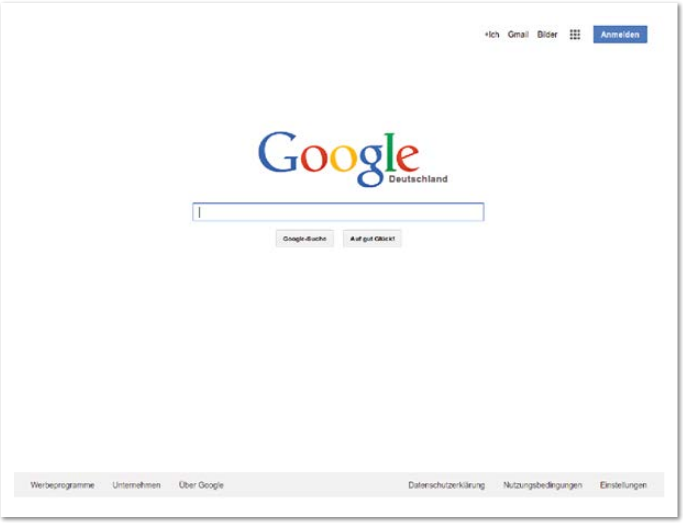


Abbildung 9: Isolation, Quelle: „http://www.google.de"

kennzeichnet die Symmetrie.

In *Abbildung 1* steht das Zentrum des Weidenstrauchs sowohl in der Horizontalen als auch in der Vertikalen genau im Verhältnis 1:Φ von rechts gesehen. Ebenso enden die Gräser im selben Verhältnis. Deshalb empfinden wir im Allgemeinen dieses Bild als harmonisch, entspannend und schön.

Dieses Verhältnis begegnet uns in der Natur sehr häufig: beim Menschen zum Beispiel das Verhältnis der Abstände vom Scheitel zum Bauchnabel, vom Bauchnabel zur Fußsohle; oder auch zwischen Scheitel und Halsansatz, Halsansatz und Bauchnabel, Bauchnabel und Knie sowie Knie und Fußsohle. Auch das Verhältnis der Blütenblätter einer Sonnenblume zu ihrem Samenkreis ist 1:Φ

Häufig wird 1:1,618 durch 1:2 ersetzt – was wir immer noch als recht harmonisch empfinden und was leichter zu implementieren ist. Damit können wir eine Fläche in ein Raster von sechs auf sechs Teile in beide Richtungen teilen. In diesem Raster lassen sich nun Elemente sowohl im Verhältnis 1:1 als auch 1:2 platzieren und so harmonisch wirken.

Hieraus entwickelte sich im Webdesign das 960-Grid-System. Dieses fügt Abstände zwischen den sechs Spalten ein und basiert auf einer Breite von 960 Pixeln. 960 wurde gewählt, weil 960 durch 2, 3, 4, 5, 6, 8, 10, 12, 15, 16, 20, 24, 30, 32, 40, 48, 60, 64, 80, 96, 120, 160, 192, 240, 320 und 480 teilbar und somit auch für komplexe Designs gut geeignet ist. Viele Webseiten und CSS-Frameworks basieren auf dem 960-Grid-System, das unter *„http://960.gs"* zu finden ist und viele Vorlagen und Helferlein bietet *(siehe Abbildung 2)*.

Visuelles Gleichgewicht

Neben der Position der Elemente und ihrer Verhältnisse zueinander ist auch ihre Gewichtigkeit zueinander bedeutsam für ein Gefühl der Harmonie im Design. Hier unterscheidet man zwischen symmetrischem und asymmetrischem Gleichgewicht. Leider lässt sich visuelles Gleichgewicht nicht so gut beschreiben – versuchen wir es einmal optisch.

Das Foto in *Abbildung 3* ist rechts und links sehr ähnlich: Neben dem großen Tisch in der Mitte stehen dieselben Hocker und zwei weitere Tische. Alle hier zu findenden Linien laufen auf die Spiegelachse zu. Ebenso die Bäume im Hintergrund. Es kommt nicht darauf an, dass alles symmetrisch ist, die vorherrschenden Elemente sollten es jedoch sein.

In *Abbildung 4* sind zwar die Elemente „Text" und „Büste" unterschiedlich groß, doch da der größere Text eine dünne, leichte Schriftart verwendet und damit dezent wirkt, die kleinere Büste dagegen mit dem intensiven, fetten Schwarz sehr dominant hervorsticht, sind beide Teile visuell im Gleichgewicht.

Benutzerführung

In aller Regel soll das Design einen Zweck erreichen: Ein Onlineshop möchte die Benutzer animieren, Waren zu kaufen, eine Webseite eines Tierschutzvereins möchte auf die Situation von Tieren hinweisen und eine Verwaltungsanwendung möchte dem Benutzer ermöglichen, intuitiv und direkt Daten zu verwalten. Um dies zu er-

reichen, will das Design den Betrachter im übertragenen Sinne an die Hand nehmen und dort hinführen, wo er richtig ist.

Untersuchungen mithilfe der Methode des Eyetracking zeigen, welche Teile eines Designs in welcher Reihenfolge und Häufigkeit betrachtet werden. Dazu verfolgt man mit einer Kamera die Augenbewegungen während der Betrachtung oder Benutzung eines Designs oder einer Anwendung nach und zeichnet sie auf. *Abbildung 5* zeigt ein häufiges Muster, in dem sich die Augen über eine Webseite bewegen.

Daraus lässt sich nun ableiten, in welcher Reihenfolge die Bereiche der Webseite mit den Augen betrachtet werden, und somit, welche Bereiche wichtiger sind *(siehe Abbildung 6)*. Informationen, die der Benutzer schnell sehen soll, sollten sich also in den zuerst betrachteten Bereichen befinden.

Wenn dies allerdings nicht möglich oder gewünscht ist, kann man die Augen an eine andere Position führen. Es gibt verschiedene Möglichkeiten, die Aufmerksamkeit auf andere Bereiche zu locken. Da das Auge gewohnt ist, Linien zu folgen, wird es dies auch tun, sobald es auf eine Linie stößt. Zudem ist das Auge bestrebt, eine bestehende Bewegung einzuhalten und somit auch die Richtung beizubehalten, wenn eine Linie beendet ist. Dabei ist es egal, ob eine Linie tatsächlich vorhanden ist, wie die Mittellinie und die Stethoskop-Bügel in *Abbildung 7*, die zum eigentlichen Inhalt unten führen, oder nur gedacht ist, wie die startende Rakete in *Abbildung 8*, die direkt zur Navigation führt.

Eine andere Möglichkeit ist die Isolation wichtiger Elemente. Das beste Beispiel dafür ist die Webseite von Google *(siehe Abbildung 9)*. Der größte Teil der Seite ist leer, isoliert von allem anderen steht in der Mitte das Eingabefeld unter dem Logo. Der Blick des Benutzers wird in die Mitte gezogen, weil er keinen anderen Aufhänger findet.

Die dritte wichtige Möglichkeit ist der Kontrast. Er ist im Allgemeinen ein stark ins Auge fallender Unterschied oder auch ein Nebeneinander unähnlicher Elemente. Beispiele dafür sind beispielsweise Farben, Ruhe/Unruhe oder Proportionen. In *Abbildung 10* sticht der leuchtend grüne Knopf aus dem sonstigen Schwarz-Weiß hervor und lädt zum Daraufklicken ein. Vor dem unruhigen Hintergrund in *Abbildung 11* liegt die einfarbige, ruhige, orangefarbene Fläche und hebt sich dadurch hervor. Der Tisch im Vordergrund von *Abbildung 12* hat einen ganz anderen Maßstab als die Skyline im Hintergrund. Deshalb wirkt der Tisch nicht zur Skyline gehörig und hebt sich dadurch ab.

Farben

Wo wir gerade von Farben gesprochen haben: Welche Farben man verwenden darf und soll, welche Farben zusammenpassen und welche nicht etc., ist einer der missverständlichsten Punkte zwischen Designer und Entwickler – gerade weil man darüber streiten kann. Letztendlich kommt es beim Design auf Emotionen an, und Farben lösen bei uns Menschen immer Emotionen aus – nur nicht immer die gleichen. Es kommt darauf an, was der Betrachter mit der Farbe assoziiert, und dies ist von Mensch zu Mensch unterschiedlich. Auch von Kulturkreis zu Kulturkreis unterscheiden sich Assoziationen. So verbinden wir im westlichen Kulturkreis grün/rot mit Start/Stopp oder OK/Falsch, im asiatischen Kulturkreis wird dies mit blau/rot verknüpft. Schon viele Menschen haben sich, auch wissenschaftlich, mit diesem Thema auseinandergesetzt. Ein paar Assoziationen, die in unserem Kulturkreis gelten, sind:

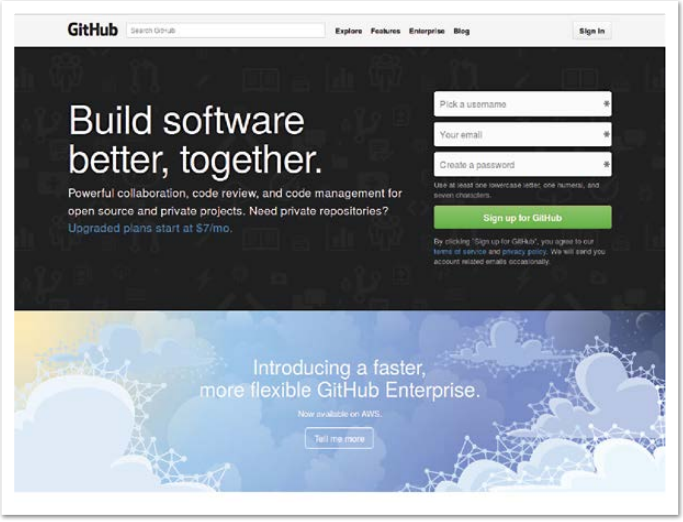


Abbildung 10: Farbkontraste, Quelle: „http://github.com"

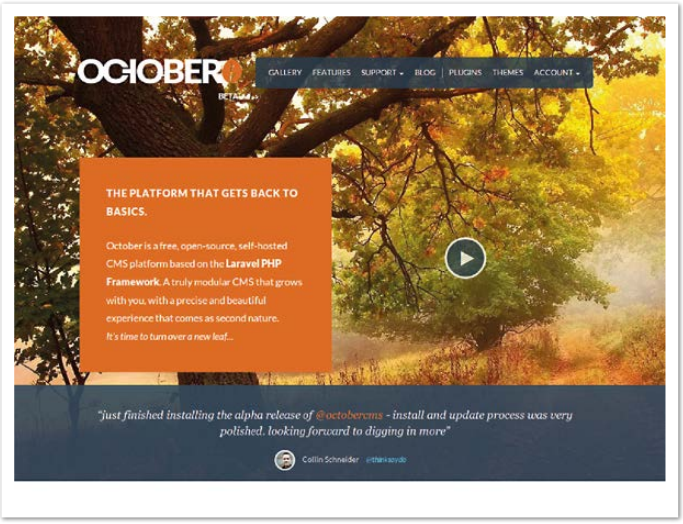


Abbildung 11: Ruhige Fläche auf unruhigem Hintergrund, Quelle: „http://octobercms.com"

Abbildung 12: Nicht passende Proportionen, Quelle: „http://wallcreations.com.au"

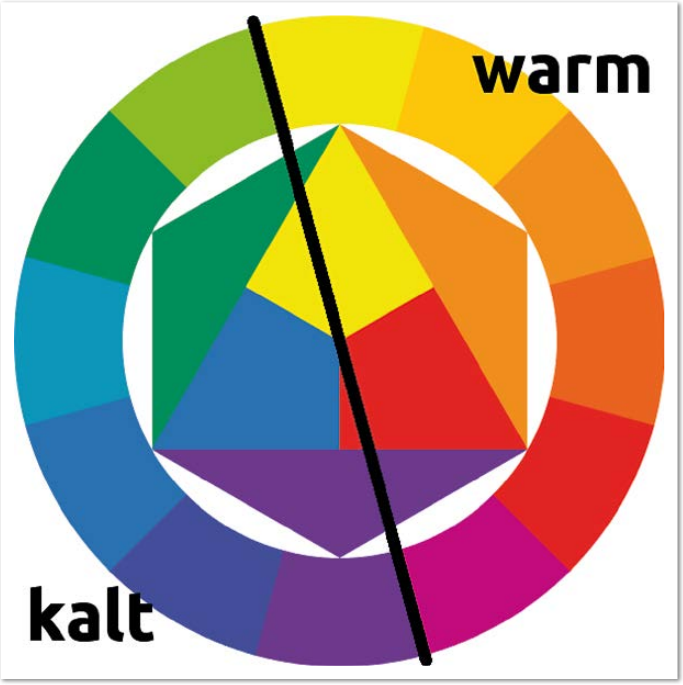


Abbildung 13: Farbkreis nach Itten mit Farbtemperatur. Quelle: „http://de.wikipedia.org/wiki/Farbkreis#/media/File:Farbkreis_Itten_1961.svg"

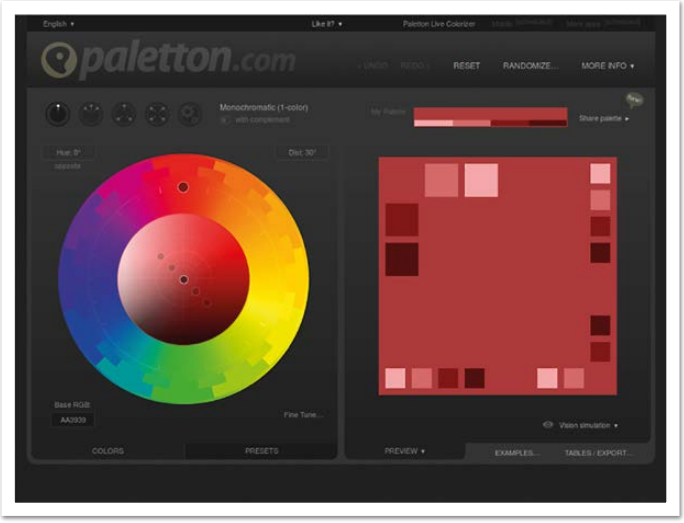


Abbildung 14: Monochromes Farbschema in Paletton. Quelle: „http://paletton.com"

Abbildung 15: Beispiel für monochromes Farbschema. Quelle: „http://www.artinmycoffee.com"

- Rot: Aktive Farbe, kraftvoll
  - Hellrot: Liebe, Leidenschaft, Blut, Wut
  - Dunkelrot: Gutmütig, arrogant, anmaßend
  - Erdiges Rot: Herbst, Erntezeit
- Orange: Aktive Farbe, Glück, Sonnenschein, Kreativität
  - Ist Appetit anregend
- Gelb: Aktive Farbe, Fröhlichkeit, kann überfordern
- Grün: Beruhigende Farbe, Natur, Wachstum, Stabilität, Bildung
- Blau: Beruhigende Farbe, Himmel, Meer, Intelligenz, Klarheit, Unglück, Kummer
  - dämpft den Appetit
- Purpur: Beruhigende Farbe, König, Macht, Wohlstand, Extravaganz, Edelsteine, Wein
  - wird selten verwendet
- Weiß: Sauberkeit, Perfektion, Reinheit, Licht
- Schwarz: Tod, Böse, Macht, Eleganz, Stärke

Farb-Eigenschaften

Die Grundfarben Rot, Gelb und Blau werden in der Farblehre mit ihren Mischfarben in einem Kreis dargestellt. Am bekanntesten ist der Farbkreis nach Itten *(siehe Abbildung 13)*. Die Farben links der Linie werden als kalte Farben bezeichnet, die rechts davon als warme Farben; man spricht hier von der Farbtemperatur. Die warmen Farben sind die dominanten und aktiven Farben, die kalten Farben die beruhigenden und devoten Farben.

Farben besitzen auch einen Farbwert. Dieser wird auch „Sättigung" genannt und stellt die Reinheit beziehungsweise Intensität der Farbe dar. Klarere Farben wirken dominanter, trübere Farben wie Pastelltöne treten eher in den Hintergrund und wirken träumerischer.

Farb-Schemata

Nicht alle Farben passen harmonisch zusammen, doch glücklicherweise gibt es auch hier ein paar Regeln, an denen man sich festhalten kann. Es existieren drei Grundformen an harmonischen Farbschemata und ihre Mischformen, die praktischerweise mit Tools visualisierbar sind. Ein solches kostenloses Tool ist unter *„http://paletton.com"* zu finden.

Beim monochromen Farbschema wird eine Farbe gewählt, und es dürfen alle Farbtöne eines Farbwertes sowie schwarz und weiß verwendet werden *(siehe Abbildung 14 und 15)*. Das komplementäre Farbschema ergänzt zu den Farben des monochromen Schemas den im Farbkreis gegenüberliegenden (komplementären) Farbwert mit seinen Farbtönen *(siehe Abbildung 16 und 17)*. Die *Abbildungen 18 und 19* zeigen das analoge Farbschema, das ein Tortenstück von maximal 60 Grad, besser nur 45 Grad, um die gewählte Farbe aus dem Farbkreis ausschneidet. Auch hier darf schwarz und weiß dazukommen.

Das triadische Farbschema ist eine Mischform der komplementären und analogen Schemata, in der nicht nur die Komplementärfarbe, sondern ein Kuchenstück um die Komplementärfarbe gewählt wird *(siehe Abbildung 20)*. Das tetradische Farbschema wählt von der Grundfarbe ein halbes Kuchenstück in die eine Richtung und von der Komplementärfarbe ein halbes Kuchenstück mit demselben Winkel in die andere Richtung aus. Das ist schwer zu beschreiben, mit *Abbildung 21* sollte es jedoch deutlich werden.

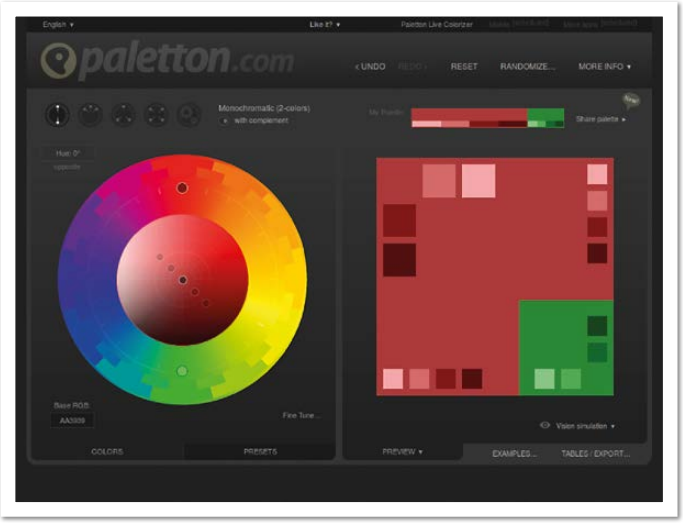


Abbildung 16: Komplementäres Farbschema in Paletton, Quelle: „http://paletton.com"

Abbildung 17: Beispiel für komplementäres Farbschema, Quelle: „http://www.naturestable.com"

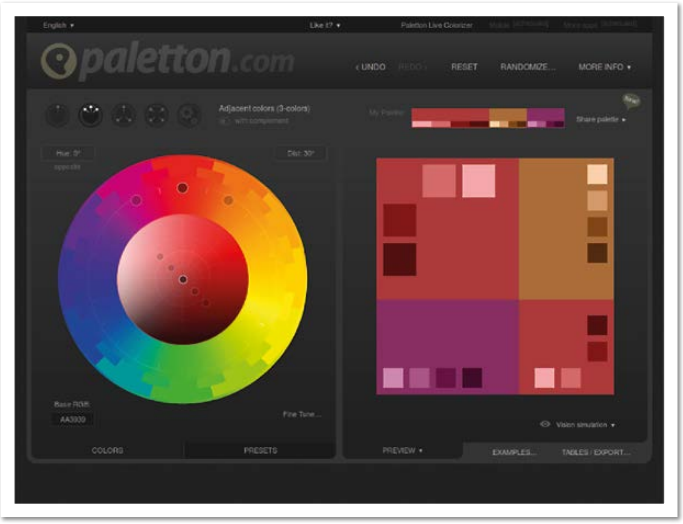


Abbildung 18: Analoges Farbschema in Paletton, Quelle: http://paletton.com

Abbildung 19: Beispiel für analoges Farbschema, Quelle: „http://www.worldcatshow.pl"

Abbildung 20: Triadisches Farbschema in Paletton, Quelle: „http://paletton.com"

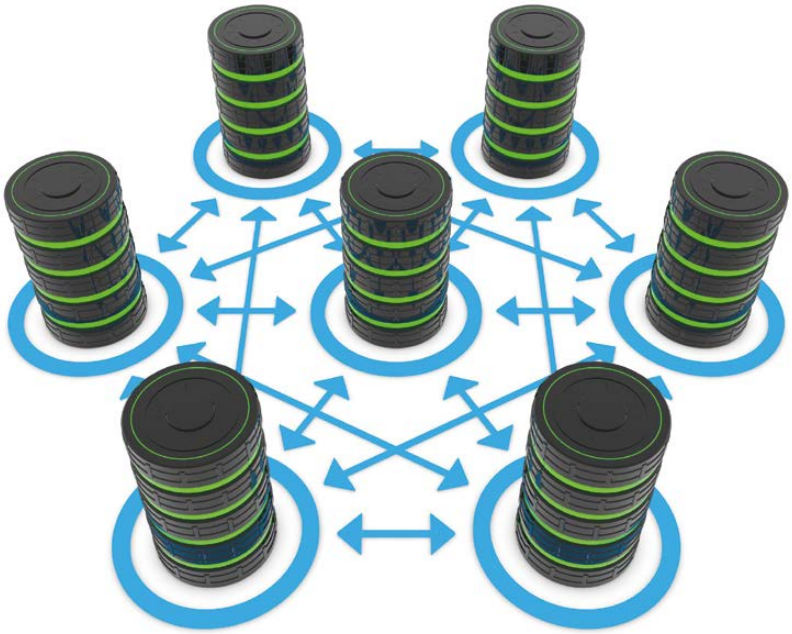
Abbildung 21: Tetradisches Farbschema in Paletton, Quelle: „http://paletton.com"

Fazit

Dieser Artikel wird noch keinen Entwickler zum Grafikdesigner machen. Aber der Autor hofft, er konnte hier einen ersten Eindruck und einen Einstieg geben. Wenn das ein oder andere Missverständnis in der Kommunikation mit Grafikdesignern in Zukunft leichter auszuräumen ist oder es gar nicht erst auftritt, sind wir einen großen Schritt weiter, um gemeinsam eine bessere Anwendung zu schaffen. Vielleicht möchte sich ja auch der eine oder andere tiefer mit diesem Thema beschäftigen.

Alexander (Sascha) Klein
alexander.klein@codecentric.de

Alexander (Sascha) Klein ist Niederlassungsleiter der codecentric AG am Standort Stuttgart. Er ist seit etwa zwanzig Jahren im Java-Umfeld als Entwickler, Architekt und Coach tätig. Seine Interessengebiete sind UI-Entwicklung, Ergonomie, DSLs und Produktivität in der Software-Entwicklung. Er ist bekennender Groovy-Jünger und Committer beim Open-Source-Projekt Griffon.



Resilient Software Design Patterns

Thorsten Maier, Orientation in Objects GmbH

Dieser Artikel beschreibt ausgewählte Entwurfsmuster zur Erstellung einer Software, die möglichst widerstandsfähig (resilient) gegenüber dem Ausfall einzelner Teilsysteme ist. Die vorgestellten Muster sind anhand einer kurzen Beispiel-Implementierung konkretisiert.

Software-Systeme werden immer komplexer. Eine klare Strukturierung und damit einhergehend eine Modularisierung ist die wichtigste Grundvoraussetzung, um komplexe Systeme auch langfristig warten zu können. Mit steigender Komplexität steigt auch die Zahl der einzelnen Teile, und der Ausfall eines einzelnen Subsystems wird immer wahrscheinlicher. Ein Ausfall kann beispielsweise durch einen Programmierfehler bei einem Versionsupdate, die Störung des Netzwerks oder aber auch durch die Überlastung der Datenbank entstehen.

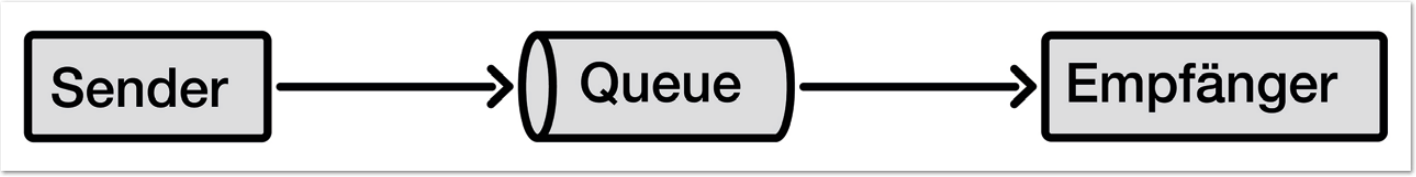


Abbildung 1: Asynchrone Kommunikation

Traditionell haben wir versucht, die Verfügbarkeit einer Software-Lösung durch eine möglichst hohe Verfügbarkeit der einzelnen Teile zu maximieren. Ab einer gewissen Anzahl von Einzelteilen ist dieses Vorgehen allerdings extrem aufwendig, da sich die Wahrscheinlichkeit für einen Ausfall mit jedem Teil erhöht.

Warum entwickeln wir unsere Software zukünftig nicht unter der Annahme, dass der Ausfall eines Teilsystems nicht die Ausnahme, sondern der Normalfall ist? Statt wie bisher zu versuchen, einen Fehler um jeden Preis zu verhindern, versuchen wir stattdessen, möglichst gut auf die auftretenden Fehler und Ausfälle zu reagieren. Genau das ist die Grundidee des Resilience Software Design. Wir entwerfen ein Software-System, das möglichst widerstandsfähig gegenüber Fehlern ist. Dabei helfen uns verschiedene wiederkehrende Muster.

Grundprinzipien

Betrachten wir zunächst das grundsätzliche Vorgehen für den Entwurf eines widerstandsfähigen Systems. Grundlage sind dabei die vier wesentlichen Prinzipien des Resilience Software Designs:

- Isolation
- Lose Kopplung
- Redundanz
- Fallback

Eine widerstandsfähige Anwendung kann nur entstehen, falls wir möglichst isolierte Fehlereinheiten entwerfen, die auch in einer instabilen Umgebung ihren Dienst möglichst lange aufrechterhalten. In jeder Fehlereinheit gelten alle umliegenden Teile per Definition als instabil; daher sollte die Kommunikation nach außen auf ein Mindestmaß reduziert werden. Ist eine Kommunikation notwendig, so sollte jede Einheit jederzeit mit einem Kommunikationsfehler rechnen und eine geeignete Reaktion vorsehen. Die angenommene Instabilität der Subsysteme bedeutet außerdem, dass bei einem Aufruf von außen alle Aufrufparameter validiert werden müssen. Daraus potenziell entstehende Fehler lassen sich so möglichst früh unterbinden.

Damit das Gesamtsystem trotz der Trennung in abgeschottete Einheiten seine Aufgabe erfüllen kann, sind definierte Schnittstellen zwischen den Teilen von entscheidender Bedeutung. Das Ziel der Schnittstellen ist die lose Kopplung der Teile, um deren Unabhängigkeit zu bewahren. Sie sollten möglichst freundlich mit ihren Aufrufern umgehen, sodass das Auftreten von Fehlern im Client möglichst unwahrscheinlich wird. Falls beispielsweise eine grafische Benutzeroberfläche mehrere Millionen Datensätze aus einer Datenbank-Tabelle anfordert, sollte die Menge der ausgelieferten Datensätze entgegen dem Willen des Aufrufers auf einen sinnvollen Wert beschränkt werden. Nur so kann die Reaktionsfähigkeit des Gesamtsystems gewährleistet werden, da der Aufrufer mit hoher Wahrscheinlichkeit ohnehin nicht alle Datensätze in einer sinnvollen Zeit verarbeiten kann.

```
JmsTemplate jmsTemplate = context.getBean(JmsTemplate.
class);
jmsTemplate.convertAndSend("coffeeOrderQueue", new
CoffeeOrder("horsten", "Cappuccino"));
```

Listing 1

```
@JmsListener(destination = "coffeeOrderQueue")
public void receiveMessage(CoffeeOrder coffeeOrder) {
  System.out.println("Received <" + coffeeOrder + ">");
}
```

Listing 2

Um die Verfügbarkeit des Gesamtsystems weiter zu erhöhen, können die lose gekoppelten Teile zudem redundant ausgelegt sein. Da in einer widerstandsfähigen Anwendung ohnehin alle Prozesse mit dem Ausfall der anderen Prozesse und somit der kurzzeitigen Nichtverfügbarkeit rechnen, ist die lastverteilte Umschaltung zwischen mehreren redundanten Kopien der logische nächste Schritt. Zu guter Letzt ist mit einem Fallback-Verhalten auf der Seite des Aufrufers sichergestellt, dass zumindest ein wesentlicher Teil der Funktionalität auch dann noch aufrechterhalten werden kann, wenn trotz Redundanz Teilsysteme vollständig ausfallen.

Design-Muster

Nachdem wir uns die abstrakte Vorgehensweise für die Erstellung einer widerstandsfähigen Anwendung angeschaut haben, folgt eine Auswahl der wichtigsten Muster zur Umsetzung dieses Konzepts. Diese lassen sich grob in zwei Kommunikationsarten einteilen: Wir unterscheiden zwischen asynchroner und synchroner Kommunikation. Asynchrone Aufrufe führen an sich bereits zu einer widerstandsfähigen Anwendung. Bei synchroner Kommunikation kann man die Widerstandsfähigkeit mit Mustern wie Verzeichnisdienst, Circuit Breaker oder clientseitigem Load-Balancing erhöhen.

Alle Konzepte dieses Artikels sind mit einem kurzen Code-Beispiel verdeutlicht. Für die technische Umsetzung bietet sich Spring *[1]* an, da in diesem Framework sowie den zahlreichen Erweiterungen bereits all die vorgestellten Konzepte integriert wurden.

Asynchrone Kommunikation

Vergleichbar mit den Schotten im Schiffsbau soll die Trennung zwischen den isolierten Teilsystemen dafür sorgen, dass niemals das System als Ganzes, sondern höchstens einzelne Teile funktionsunfähig werden. Diese geforderte Isolation kann im einfachsten Fall erreicht werden, indem Sender und Empfänger lediglich indirekt über Nachrichten kommunizieren *(siehe Abbildung 1)*.

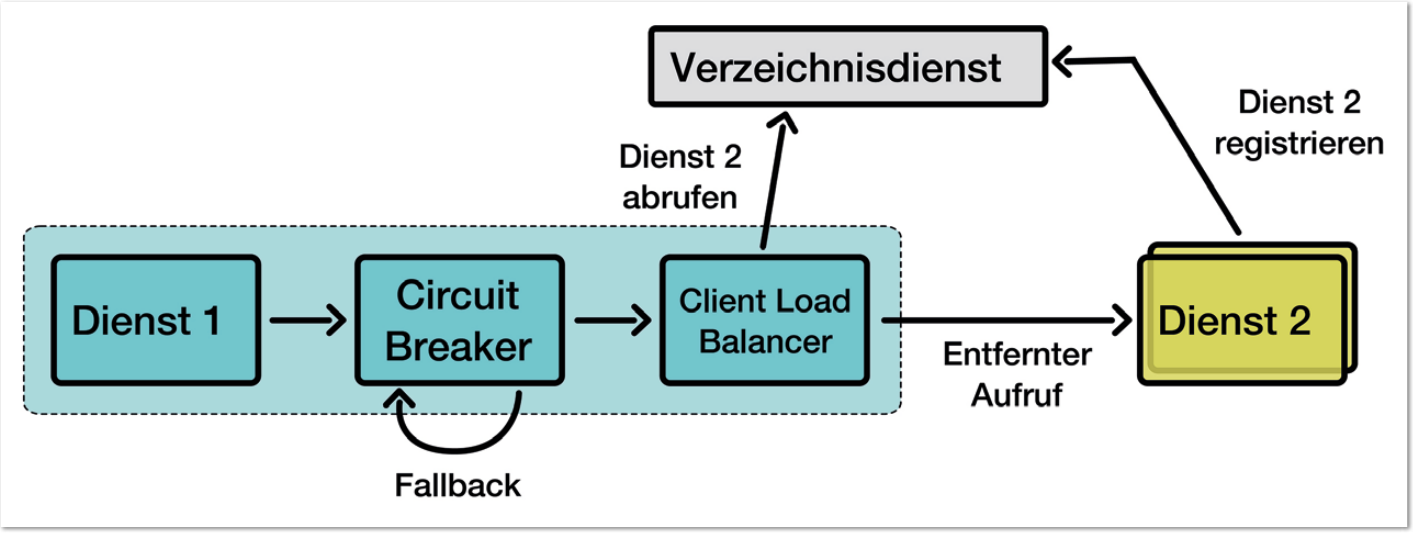


Abbildung 2: Widerstandsfähige synchrone Kommunikation

Die Nachrichten werden in der Queue zwischengespeichert und somit kann der Sender unmittelbar nach dem Versand einer Nachricht mit seiner Arbeit fortfahren; er muss nicht auf die Verfügbarkeit eines Empfängers warten. Gleichzeitig ist auch der Empfänger nicht auf die Verfügbarkeit des Senders angewiesen und kann die Nachricht zu einem für ihn passenden Zeitpunkt verarbeiten.

Kommen wir zur konkreten Implementierung. Der Versand einer asynchronen Nachricht in eine Queue wird bei Spring durch die Klasse „JmsTemplate" unterstützt *(siehe Listing 1)*. Als Empfänger für die Nachrichten einer Queue kann man sich mit der Annotation „@JmsListener" registrieren *(siehe Listing 2)*.

Obwohl die Vorteile der asynchronen Kommunikation über Queues den meisten bekannt sein dürften, wird diese Art der Kommunikation dennoch in traditionellen Systemen verhältnismäßig selten eingesetzt. Ein Grund dafür ist die unter Software-Entwicklern weitverbreitete „transaktionale Denkweise". Damit ist gemeint, dass wir darauf bedacht sind, unseren Datenbestand zu jedem beliebigen Zeitpunkt konsistent zu halten, und dass wir dies über Transaktionen sicherstellen müssen. Genau dies lässt sich mit asynchroner Kommunikation über mehrere Prozesse hinweg allerdings nur schwer erreichen, denn wir müssten unsere Transaktionsgrenze über die beteiligten Prozesse hinweg aufspannen, was technisch zwar nicht unmöglich, aber durchaus anspruchsvoll ist.

```
@SpringBootApplication
@EnableEurekaServer
public class Application {
  public static void main(String[] args) {
     SpringApplication.run(Application.class, args);
  }
}
```

Listing 3

```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-eureka</artifactId>
</dependency>
```

Listing 4

Bei genauerer Analyse der Anwendungsfälle einer Anwendung sollte man diese transaktionale Denkweise allerdings hinterfragen. Oft ist es ausreichend, wenn die Konsistenz der Daten nicht sofort, sondern erst zu einem späteren Zeitpunkt hergestellt werden kann. Im Gegensatz zu „Strict Consistency", bei der die Konsistenz am Ende jeder Aktion sichergestellt wird, reicht es bei „Eventual Consistency", dass die Daten schlussendlich in naher Zukunft konsistent sind.

„Eventual Consistency" lässt sich mithilfe asynchroner Kommunikation sicherstellen, während für „Strict Consistency" auf synchrone Kommunikation gesetzt werden muss. Im Blog-Artikel „Starbucks Does Not Use Two-Phase Commit" *[2]* ist diese Thematik anhand einer Kaffeebestellung sehr anschaulich erläutert. Zusammenfassend ist beschrieben, dass aus Gründen der Durchsatzoptimierung und der Fehlertoleranz ein asynchroner Kommunikationsansatz mit einer Becherwarteschlange beim Bestellvorgang eines Kaffees deutliche Vorteile bietet.

Was können wir daraus lernen? Wir sollten für jeden Anwendungsfall analysieren, ob „Eventual Consistency" gegebenenfalls ausreichend ist. Dann können wir durch den Einsatz von asynchroner Kommunikation ein deutlich widerstandsfähigeres Gesamtsystem entwerfen.

Widerstandsfähige synchrone Kommunikation

Selbstverständlich ist es nicht immer möglich, auf asynchrone Kommunikation zu setzen. Daher müssen wir in diesen Fällen auf synchrone Aufrufe zurückgreifen. Der entscheidende Nachteil im Sinne einer widerstandsfähigen Anwendung ist dabei, dass für eine erfolgreiche Kommunikation beide Partner gleichzeitig kommunikationsbereit sein müssen. Dies wiederum ist in einem System, in dem wir alle Kommunikationspartner per Definition als unzuverlässig einstufen, nicht immer gegeben. Wir benötigen daher Mechanismen, mit denen wir auch diese Art der Kommunikation widerstandsfähig gegen Ausfälle machen können.

Abbildung 2 zeigt den schematischen Aufbau einer fehlertoleranten synchronen Kommunikation. Dienst 1 möchte mit Dienst 2 kommu-

nizieren. Die Widerstandsfähigkeit wird hierbei durch den Einsatz eines Verzeichnisdienstes, eines Circuit Breakers und eines clientseitigen Load-Balancer erreicht.

Verzeichnisdienst

Der Verzeichnisdienst sorgt dafür, dass Dienst 1 nicht wissen muss, wo Dienst 2 gerade ausgeführt wird. Soll eine Kommunikation aufgebaut werden, wendet sich Dienst 1 über den clientseitigen Load-Balancer – der im Anschluss noch vorgestellt wird – an den Verzeichnisdienst und ruft von dort eine oder auch mehrere Instanzen von Dienst 2 ab. Auch wenn Dienst 2 seit der letzten Kommunikation umgezogen ist oder durch eine andere Instanz ersetzt wurde, kann der anschließende, entfernte Aufruf trotzdem fehlerfrei ausgeführt werden.

Mit Eureka *[3]* können wir auf einen fertigen Verzeichnisdienst zurückgreifen, der bei Netflix zum Einsatz kommt und somit bereits gezeigt hat, dass er auch für sehr große Systeme bestens geeignet ist. Mit Spring Cloud Netflix können wir Eureka auf einfache Art und Weise in unsere bestehende Infrastruktur einbinden. *Listing 3* zeigt die Erstellung eines Verzeichnisdienst-Servers als Spring-Anwendung.

Die einzelnen Dienste registrieren sich anschließend automatisch während des Startvorgangs beim Eureka-Verzeichnisdienst. Damit dies funktioniert, muss lediglich die passende Eureka-Client-Implementierung im Klassenpfad des Dienstes vorhanden sein. *Listing 4* zeigt die entsprechende Maven-Dependency.

Circuit Breaker

Durch den Einsatz eines Verzeichnisdienstes erreichen wir eine Ortstransparenz für entfernte Dienste. Problematisch ist dies allerdings, wenn ein solcher Dienst unmittelbar nach dem Auffinden ausfällt oder umzieht. Für diesen Fall benötigen wir einen zweiten Sicherungsmechanismus. Zwischen den beiden Diensten kann zusätzlich ein sogenannter „Circuit Breaker" *[4]* eingesetzt werden. Dieser überwacht die Kommunikation zwischen beiden Kommunikationsseiten und entbindet den Aufrufer somit von der Fehlerbehandlung. Falls der entfernte Dienst zu spät oder überhaupt nicht

```
@HystrixCommand(fallbackMethod = "fallback")
public String readString() {
  return remoteServiceCall();
}

public String fallback() {
  return "Fallback result";
}
```

Listing 5

reagiert, kann der Circuit Breaker diese Fehlersituation erkennen und eine Fallback-Antwort an den Aufrufer liefern. Für die praktische Umsetzung dieses Konzepts bedienen wir uns wieder einer von Netflix zur Verfügung gestellten Open-Source-Bibliothek namens Hystrix *[5]*. *Listing 5* zeigt, wie wir damit eine solche Sicherung per Annotation konfigurieren können.

Die Annotation „@HystrixCommand" erstellt automatisch einen Proxy, der zur Absicherung des entfernten Methodenaufrufs eingesetzt wird. Falls der Aufruf „remoteServiceCall()" nach einem konfigurierbaren Timeout (Default: 1 Sekunde) kein Ergebnis liefert, wird automatisch die Methode „fallback" aufgerufen und deren Rückgabewert an den Aufrufer geliefert. Die mit dieser Annotation konfigurierte Sicherung verwaltet einen internen Zustand und verfügt somit über eine gewisse Intelligenz. *Abbildung 3* zeigt die möglichen Zustände und die zugehörigen Zustandsübergänge.

Im Zustand „Geschlossen" werden alle Aufrufe an den entfernten Dienst weitergeleitet und dessen Reaktion abgewartet. Falls mehr als 50 Prozent der letzten 20 Aufrufe länger als eine Sekunde gedauert haben, springt die Sicherung auf und wechselt in den Zustand „Offen". Dieser Zustandsübergang findet bewusst nicht bei der ersten Überschreitung des Timeouts statt, damit einzelne Ausreißer keine Auswirkung auf das Verhalten des Systems haben.

Im Zustand „Offen" wird versucht, den entfernten Dienst nicht unnötig mit Aufrufen zu belasten; daher werden fünf Sekunden lang alle Anfrage mit dem Fallback-Wert beantwortet. Nach dieser War-

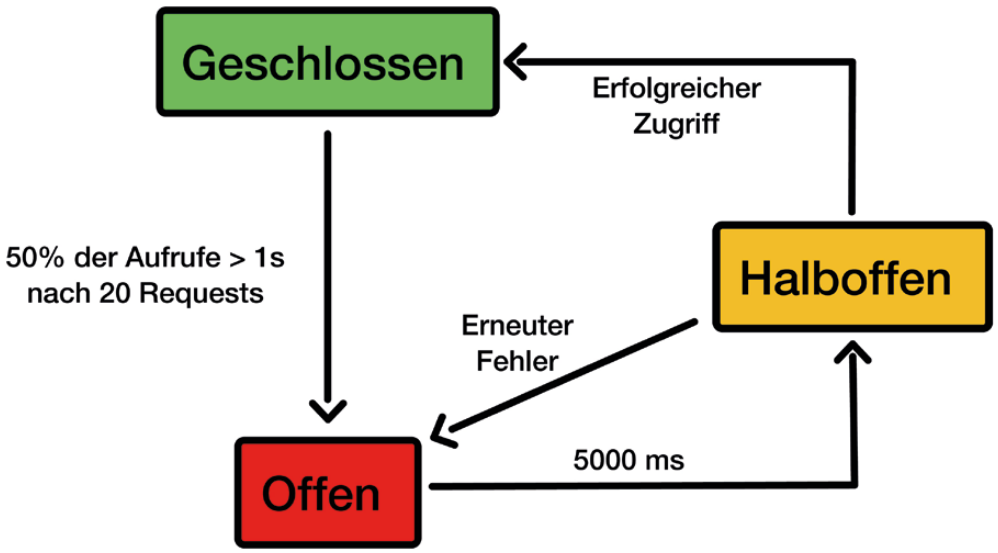


Abbildung 3: Hystrix-Zustände

tezeit findet ein automatischer Übergang in den Zustand „Halboffen" statt. Nun stellt ein erneuter Zugriff auf den entfernten Dienst fest, ob dieser wieder verfügbar ist. Ist dies der Fall, wird die Sicherung geschlossen; andernfalls erneut geöffnet und es findet nach weiteren fünf Sekunden Wartezeit der nächste Testaufruf statt. Die Default-Werte für die Timeouts sowie für den prozentualen Anteil der erlaubten Timeout-Überschreitungen lassen sich per Konfiguration anpassen.

Clientseitiges Load-Balancing

Durch den Einsatz des Verzeichnisdienstes und des Circuit Breaker konnten wir bereits eine gewisse Ausfallsicherheit erreichen. Allerdings können wir damit noch nicht auf unterschiedliche Lastanforderungen reagieren. Die Last kann auf mehrere redundante Instanzen des entfernten Dienstes verteilt werden. Ist er zustandslos implementiert, lässt er sich problemlos mehrfach starten. Klassischerweise hat man eine solche Lastverteilung durch die Zwischenschaltung eines getrennten Load-Balancer erreicht. Bei vielen einzelnen Diensten führt dies allerdings zu einem hohen Konfigurationsaufwand. Daher wollen wir uns hierzu eine leichtgewichtige und dennoch widerstandsfähige Alternative anschauen. Dabei verwenden wir einen im Aufrufer eingebauten, clientseitigen Load-Balancer. Dieser ruft sich vom Verzeichnisdienst alle zur Verfügung stehenden Instanzen des Dienstes ab und verteilt dann selbstständig die Aufrufe an die vorhandenen Instanzen. *Listing 6* zeigt den Einsatz dieses Konzepts bei einem weiteren Netflix-Projekt mit dem Namen Ribbon *[6]*.

„RestTemplate" ist eine Hilfsklasse von Spring, mit der per HTTP-Aufrufen ein Rest-Service abgefragt werden kann. Die beiden Annotationen „@LoadBalanced" und „@RibbonClient" sorgen nun dafür, dass alle Instanzen von „dienst2" über den Verzeichnisdienst abgerufen werden und der Aufruf der URL *„http://dienst2/"* automatisch auf diese Instanzen verteilt wird. Wann welche Instanz aufgerufen wird, lässt sich über den konfigurierbaren Load-Balancer-Algorithmus beeinflussen.

Die „RoundRobinRule" verteilt die Anfragen beispielsweise reihum; „RandomRule" wählt dagegen bei jedem Aufruf zufällig eine Instanz aus. Am interessantesten für den Einstieg ist allerdings die „WeightedResponseTimeRule". Dabei werden die Antwortzeiten der verschiedenen Instanzen analysiert und die Aufrufhäufigkeiten diesen Zeiten angepasst. Die schnellste Instanz bekommt verhältnismäßig die meisten Aufrufe. Aber auch langsame Instanzen werden hin und wieder aufgerufen, um die schnellen Instanzen nicht zu überlasten.

```
@RibbonClient(name = "dienst2")
public class Application {
  @LoadBalanced
  @Bean
  RestTemplate restTemplate() {
     return new RestTemplate();
  }

  public String remoteServiceCall() {
     return restTemplate().getForObject("http://di-
enst2/", String.class);
  }
}
```

Listing 6

Durch die ständige Analyse der Antwortzeiten reguliert sich das System selbstständig und kann daher auf eine zentrale Steuerung verzichten.

Fazit

Eine Anwendung ist mithilfe der in diesem Artikel gezeigten Muster widerstandsfähig gegenüber Fehlern. Am einfachsten gelingt dies durch den Umstieg auf asynchrone Kommunikation, da hierdurch automatisch isolierte und nur lose miteinander gekoppelte Teilsysteme entstehen. Aber auch synchrone Kommunikation kann wie gezeigt unter Einsatz eines Verzeichnisdienstes, eines Circuit Breaker und durch clientseitiges Load-Balancing resilient gemacht werden.

Neben den hier beschriebenen wichtigsten Mustern gibt es viele weitere, die uns bei der Implementierung eines möglichst stabilen Systems helfen können. Dazu gehören beispielsweise ein Konfigurationsserver, ein API Gateway und ein externer HTTP-Session-Speicher. Zu bedenken ist allerdings wie immer, dass jedes neu verwendete Muster Einarbeitungs- und Implementierungsaufwände zur Folge hat und dass wir uns daher in jeder Anwendung auf ein Neues für die richtigen Muster entscheiden müssen.

Weitere Informationen

[1] http://projects.spring.io/spring-framework
[2] *http://www.enterpriseintegrationpatterns.com/ramblings/18_starbucks.html*
[3] https://github.com/Netflix/eureka
[4] https://martinfowler.com/bliki/CircuitBreaker.html
[5] https://github.com/Netflix/Hystrix
[6] https://github.com/Netflix/ribbon

Thorsten Maier
Thorsten.Maier@oio.de

Thorsten Maier arbeitet bei OIO Orientation in Objects GmbH in Mannheim. Er erschließt kontinuierlich bessere Wege, Software zu entwickeln, indem er selbst als leidenschaftlicher Software-Entwickler mit Java und JavaScript unterwegs ist und anderen als Berater, Trainer und Autor dabei hilft. Trotz seiner Begeisterung für Neues sind ihm Menschen stets wichtiger als Technologien. Sein Hauptaugenmerk liegt daher auf der Frage, wie sich modernste Technologien in gewachsene Umgebungen einbinden lassen und wann man besser auf Bestehendes zurückgreifen sollte.

„Wir sind nicht dogmatisch, was die Wahl der Programmiersprache angeht …“

Usergroups bieten vielfältige Möglichkeiten zum Erfahrungsaustausch und zur Wissensvermittlung unter den Java-Entwicklern. Sie sind aber auch ein wichtiges Sprachrohr in der Community und gegenüber Oracle. Wolfgang Taschner, Chefredakteur der Java aktuell, sprach darüber mit Manuel Mauky von der JUG Görlitz.

Wie ist die JUG Görlitz organisiert?

Manuel Mauky: Die JUG Görlitz ist ein loser Verbund von Entwicklern und Java-Interessierten aus Görlitz. Wir besitzen keine formale Organisationsform (etwa als Verein), sind aber als Gruppe Mitglied im iJUG-Verband. Den Großteil der Arbeit für die JUG teilt sich eine Gruppe von aktuell fünf Leuten mit zusätzlicher, gelegentlicher Unterstützung von weiteren Helferinnen und Helfern.

Was sind die Ziele JUG Görlitz?

Manuel Mauky: Für mich persönlich ist eine Hauptmotivation, ein interessantes Unterhaltungsangebot für an Programmierung interessierte Leute zu schaffen und somit auch zur Vielfalt der Veranstaltungen in unserer Region beizutragen. Ein weiteres Ziel ist die Vernetzung der IT-Unternehmen untereinander und mit der Hochschule. Natürlich geht es immer auch darum, den Besuchern einen Einblick in andere Themen zu geben, mit denen sie sonst im Berufsleben oder im Studium nicht in Berührung kommen. Daher versuchen wir auch, ein recht breites Themenspektrum abzubilden, gerne auch mal abseits von reinem Java.

Wie viele Veranstaltungen gibt es pro Jahr?

Manuel Mauky: Aktuell gibt es pro Monat eine Veranstaltung, mit Ausnahme einer Sommer- und einer Winterpause – insgesamt also rund zehn Veranstaltungen pro Jahr. Die Frequenz kann sich aber auch ändern, je nachdem, wie viele Speaker wir finden. In früheren Jahren hatten wir beispielsweise nur jeden zweiten Monat eine Veranstaltung.

Was bedeutet Java für euch?

Manuel Mauky: Für den Großteil unserer Besucher und Organisatoren ist Java die Sprache/Technologie, mit der sie am meisten zu tun haben, und daher beziehen sich natürlich auch die meisten unserer Vorträge auf Java. Wir sind jedoch nicht dogmatisch, was die Wahl der Programmiersprache angeht. Java an sich hat also keine so herausragende Bedeutung für uns. Es geht hauptsächlich um das Programmieren, egal mit welcher Sprache. Deshalb sind wir auch anderen Themen gegenüber aufgeschlossen, beispielsweise hatten wir auch schon Vorträge zu Haskell oder JavaScript-Frameworks im Programm. Das „Java“ im Namen der Usergroup hat vor allem historische Gründe und weil es ein etabliertes Format ist. Würden wir die JUG heute nochmal neu gründen, würden wir sie vielleicht „Coding-Usergroup“ oder so nennen, um auch andere Software-Entwicklerinnen und -Entwickler anzusprechen, die mit anderen Programmiersprachen arbeiten.

Welchen Stellenwert besitzt die Java-Community für euch?

Manuel Mauky: Als JUG profitieren wir natürlich von einer starken Java-Community, aus der sich immer wieder Speaker finden, die bei uns vortragen möchten. Wir versuchen auch selbst, in dieser Richtung aktiv zu werden und den Kontakt zu anderen JUGs zu knüpfen, was für uns der Hauptgrund war, dem iJUG-Verband beizutreten. So haben mein Kollege Max Wielsch und ich im letzten Jahr viele JUGs besucht und dort Vorträge gehalten und ich möchte das auch in der Zukunft gerne wieder machen. Ansonsten sind wir als JUG aber nicht sehr aktiv in der Java-Community und beispielsweise nicht am JCP oder ähnlichen Aktivitäten beteiligt.

Wie sollte sich Java weiterentwickeln?

Manuel Mauky: Das ist natürlich schwierig zu sagen, vor allem weil Java ja mittlerweile weit mehr als nur eine Programmiersprache ist und es so viele verschiedene Anwender und Anwendungsgebiete gibt. Ich persönlich bin der Meinung, dass Java als Sprache leider nicht mehr besonders fortschrittlich ist und die Entwicklung der Sprache auch kaum mit der von anderen Sprachen mithalten kann. Auf der anderen Seite ermöglicht Java als Plattform aber auch die Entwicklung von modernen Sprachen wie beispielsweise Kotlin oder den Haskell-Dialekt Frege und einige weitere. Als Ganzes kann Java also sehr wohl noch innovativ sein. Ich würde mir also wünschen, dass die Modernisierung von Java als Sprache weiterhin aktiv betrieben wird. Allerdings wird man natürlich auch in Zukunft nicht alle Wünsche erfüllen können.

Wie sollte Oracle eurer Meinung nach mit Java umgehen?

Manuel Mauky: Oracle hat seit der Übernahme von Sun vieles vorangetrieben. Auf der anderen Seite hat Oracle aber auch vor allem in Bezug auf Java EE in der letzten Zeit für viel Verunsicherung gesorgt, weil an wichtigen Themen nicht mehr mit dem nötigen Engagement gearbeitet wurde. Die Öffnung von Java EE als Open Source ist vor diesem Hintergrund schwierig zu beurteilen. Auf der einen Seite ist das natürlich ein positives Signal, es könnte aber auch als Rückzug aus der Java-Entwicklung seitens Oracle interpretiert werden. Ich würde mir wünschen, dass Oracle auch weiterhin ein starker Trei-

ber hinter Java bleibt. Vor allem sollte Oracle aber in seiner Kommunikation klarer werden und transparenter machen, was denn die Pläne und Visionen in Bezug auf Java sind. Dann ließen sich deren Tätigkeiten und Aktionen besser beurteilen und einordnen sowie die Zusammenarbeit mit der Community leichter planen.

Wie sollte sich die Community gegenüber Oracle verhalten?

Manuel Mauky: Die Community sollte auch weiterhin selbstbewusst und kritisch die Entwicklung von Java begleiten und hinterfragen. Wir sollten aber auch wohlwollend und kompromissbereit sein. Ich glaube, dass ein übermäßiges Misstrauen gegenüber allem, was Oracle macht, das ich in Teilen der Community wahrnehme, uns nicht weiterbringt.

Manuel Mauky
manuel.mauky@saxsys.de

Manuel hat in Görlitz Informatik studiert und arbeitet seitdem bei der Saxonia Systems AG als Software-Entwickler und Architekt. Er beschäftigt sich mit allen Aspekten der Anwendungsentwicklung, mit einem Fokus auf das Frontend. Daneben interessiert er sich vor allem für die funktionale Programmierung und ist in zahlreichen Open-Source-Projekten aktiv. Er ist einer der Gründer und Organisatoren der JUG Görlitz und hält regelmäßig Vorträge auf verschiedenen Konferenzen und bei Usergroups.

Crypto 101

Oliver Milke, TRIOLOGY GmbH

Kryptographie ist ein sehr umfangreiches und vielschichtiges Thema, das viel Erfahrung und Wissen erfordert. Entwickler kommen im Alltag immer mal wieder mit Kryptographie in Berührung, insbesondere im DevOps-Umfeld. Um die Artikellänge nicht zu sprengen und beim Wesentlichen zu bleiben, möchte ich mit diesem Artikel praxisrelevante Grundlagen für die tägliche Entwicklungsarbeit schaffen, ohne dabei in die Tiefen von Security abzutauchen. Ich habe mir speziell jene Dinge vorgenommen, über die ich selbst während meiner Einarbeitungszeit in der Abteilung für Security der mobilen Online-Dienste bei VW gestolpert bin.

Kryptographie befasst sich mit Informationssicherheit und ist ein Teilbereich der Kryptologie. Ein weiterer Teilbereich ist die Krypto-Analyse, deren Ziel es unter anderem ist, Schwächen in bestehenden kryptographischen Verfahren aufzudecken oder den gebotenen Schutz zu quantifizieren. Sicherheit („Security", „secure") wiederum umfasst neben Kryptographie noch weitere Aspekte wie Prozesse und den Umgang mit den Methoden der Kryptographie sowie die Bewusstheit solcher Prozesse. Der Form halber ergänze ich hier noch, dass Sicherheit „(Safety", „safe") eher Unversehrtheit beschreibt und demnach zu einem ganz anderen Bereich gehört. Die Unterscheidung

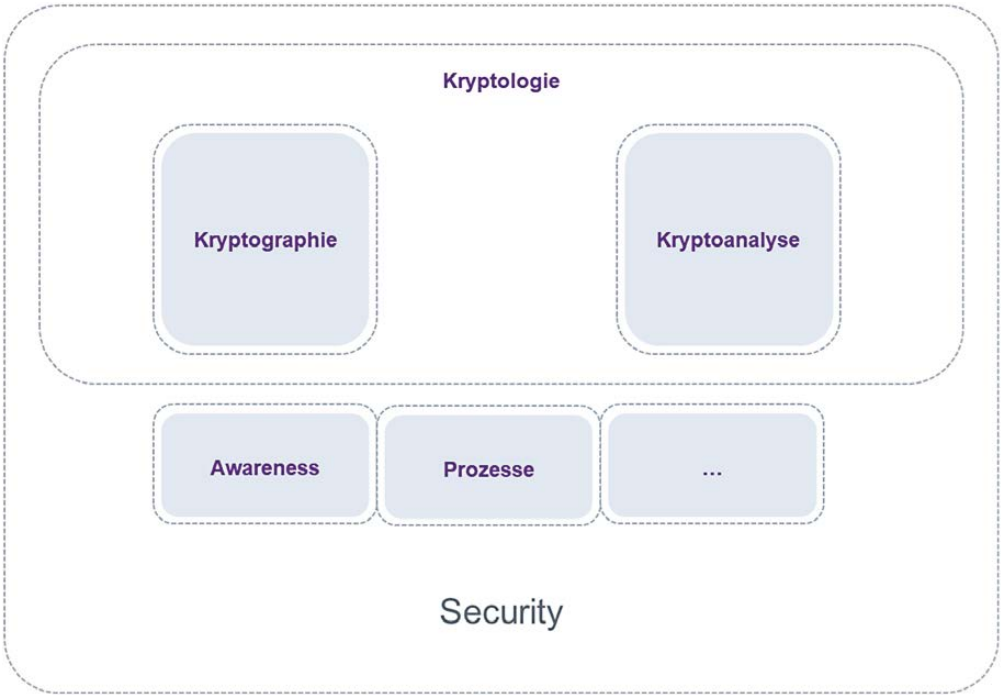


Abbildung 1: Einordnung der Kryptographie

ist im Deutschen leider nicht so deutlich *(siehe Abbildung 1)*.

Kryptographie wird eingesetzt, um drei Ziele zu erreichen:

- *Vertraulichkeit*
  Die Gewissheit, dass Nachrichten nicht mitgelesen werden
- *Integrität*
  Die Gewissheit, dass Nachrichten nicht verändert sind
- *Authentizität*
  Die Gewissheit, dass Nachrichten von einem mir bekannten Absender stammen

Kryptographische Primitive

Die im vorhergehenden Abschnitt dargestellten Ziele werden durch verschiedene Operationen, die sogenannten kryptographischen Primitive, ermöglicht. Man kann sie sich als die Grundrechenarten der Kryptographie vorstellen:

- *Kryptographischer Hash*
  Einwegfunktionen, die neben weiteren Eigenschaften vor allem kollisionsresistent sind.
- *Symmetrische Verschlüsselung*
  Verschlüsselungsverfahren, bei denen derselbe Schlüssel sowohl für die Verschlüsselung als auch für die Entschlüsselung verwendet wird. Diese Verfahren sind typischerweise sehr schnell.
- *Asymmetrische Verschlüsselung*
  Verschlüsselungsverfahren, bei denen es zwei voneinander abhängige Schlüssel gibt, wobei der jeweils andere Schlüssel die Umkehr einer Operation erlaubt, auch Public-Key-Verfahren genannt. Diese Verfahren sind typischerweise langsamer als symmetrische Verfahren.
- *Digitale Signatur*
  Zur Sicherstellung von Authentizität und Integrität einer Nachricht. Vereinfacht dargestellt handelt es sich um die asymmetrische Verschlüsselung des Hashs einer Nachricht.
- *Kryptographisch sichere Zufallszahlengenerierung*
  Man kann sich leicht vorstellen, dass echter Zufall durch eine deterministische Maschine nicht leistbar ist. Dennoch benutzen kryptographische Verfahren Noncen (Zahlen, die nur ein einziges Mal eingesetzt werden dürfen) oder Initialisierungsvektoren (Startblock bei Block-Chiffren). Sie werden typischerweise zufällig generiert und haben somit einen Einfluss auf die Sicherheit anderer Operationen.
- *Weitere*
  Es gibt noch weitere Primitive, auf die ich hier nicht eingehen will. Einen Einstiegspunkt für Interessierte bietet der gleichnamige Wikipedia-Artikel *(siehe „https://en.wikipedia.org/wiki/Cryptographic_primitive")*.

Die Kombination solcher Primitive nennt man „Krypto-System". Die Art und Weise der Kombination hängt vom Anwendungsfall ab, zum Beispiel Transport-Layer-Security (TLS) oder Secure Shell (SSH).

Sicherheit von Kryptographie

Die Sicherheit von kryptographischen Verfahren basiert zu großen Teilen auf Einwegfunktionen. Wie der Name bereits suggeriert, handelt es sich um Funktionen, zu denen (noch) keine effiziente Umkehrfunktion vorhanden ist. Beispiele für Einwegfunktionen sind neben den kryptographischen Hash-Funktionen auch die Multiplikation von Primzahlen (Grundlage für RSA) sowie die diskrete Exponentialfunktion in endlichen Körpern oder die Multiplikation von Punkten auf elliptischen Kurven (Grundlage des ElGamal-Verfahrens und des Diffie-Hellman-Verfahrens).

Das Prinzip von Kerckhoff besagt, dass die Sicherheit von kryptographischen Verfahren nicht von der Geheimhaltung des Verfahrens abhängen darf. Vielmehr muss es von der Geheimhaltung des Schlüssels abhängen. Dadurch entsteht die Möglichkeit zur Widerlegung der systematischen Sicherheit. Unsichere Verfahren können infolgedessen vermieden werden. Das Gegenteil bezeichnet man landläufig als „Security By Obscurity".

Schwachstellen in Krypto-Systemen entstehen entweder direkt in der Spezifikation (BEAST, siehe *„https://de.wikipedia.org/wiki/Transport_Layer_Security#BEAST")* eines Verfahrens oder in einer konkreten Implementierung eines Verfahrens (Heart Bleed, siehe *„https://de.wikipedia.org/wiki/Heartbleed").* Implementierungen können außerdem Seitenkanal-Angriffe *(siehe „http://www.cryptofails.com/post/70097430253/crypto-noobs-2-side-channel-attacks")* ermöglichen, indem zum Beispiel aus der benötigten Verarbeitungszeit einer Operation Schlussfolgerungen hergeleitet werden können.

Passwörter sicher speichern

Die von einem Benutzer gewählten Passwörter in Klartext zu speichern, ist ein „No-Go" – das wissen die meisten Entwickler. Aber wie genau speichert man denn Passwörter? Oft habe ich schon den Begriff „Verschlüsselung" in diesem Zusammenhang gehört, wobei das typischerweise eines von zwei Dingen bedeutet. Entweder verkompliziert man die Passwort-Speicherung, weil man jetzt auch noch „das Passwort" für eine Verschlüsselung speichern muss oder Verschlüsselung wird fälschlicherweise verwendet und meint eigentlich „Hashing".

Aufgrund der Tatsache, dass Hashes deterministische Einwegfunktionen sind, sind sie hervorragend geeignet, um Passwörter abzusichern. Allerdings kann man daher einfach Hash-Tabellen bilden, um einmal berechnete Hashes im Zusammenhang mit ihrem Klartext-Ursprung zu speichern. Auf diese Weise entsteht eine Zeit-effiziente Umkehrfunktion. Für den Fall, dass man zum Beispiel eine ganze Datenbank mit gehashten Passwörtern vorliegen hat (etwa durch den Einbruch in ein System), entsteht ein zusätzliches Problem. Rät man Passwörter und hasht sie (teuer), kann man alle vorhandenen Passwörter dagegen prüfen (günstig) und somit in einem Schritt alle Passwörter angreifen.

Abhilfe schafft ein sogenannter „Salt". Salts sind zufällig und pro Passwort individuell zu generieren, zusätzlich mit dem Passwort zu speichern und sie reichern das Passwort vor dem Hashen an. Dadurch unterscheiden sich Passwörter mindestens um ihren Salt, was zwei wesentliche Effekte hat. Zum einen sieht man zwei Datensätzen nicht mehr sofort an, ob es sich um das gleiche Passwort handelt. Zum anderen ist es dadurch nicht mehr möglich, alle Passwörter einer Datenbank gleichzeitig zu überprüfen.

Hash-Funktionen werden jedoch auch zu anderen Zwecken auf deutlich größere Datenmengen angewendet, weshalb eines der verfolgten Entwicklungsziele für manche Hash-Funktionen die Performance ist. Das ist unglaublich praktisch, um den Hash von einem Linux-Image zu berechnen und Passwörter mittels Brute Force stumpf auszuprobieren. Deswegen ist eine wichtige Anforderung an Hash-Funktionen für Passwörter die Langsamkeit. Eine einfache Lösung ist die wiederholte Anwendung einer Hash-Funktion. Die bessere Alternative ist eine Hash-Funktion, bei deren Entwicklung darauf geachtet wurde, dass sie konfigurierbar langsam ist. Wenn eine Hash-Funktion gleichzeitig noch sehr viel Arbeitsspeicher benötigt, sind die Parallelisierung oder der Einsatz von spezieller Hardware unwirtschaftlich.

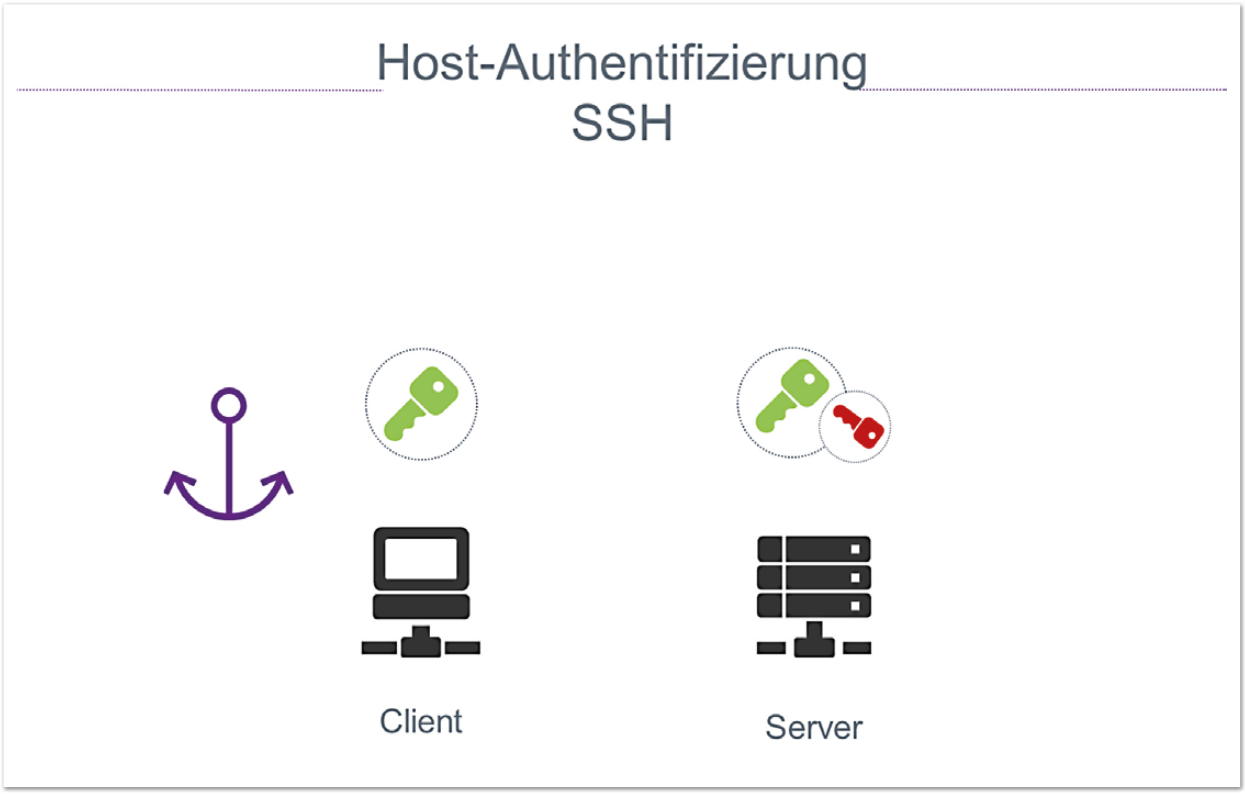


Abbildung 2: Vertrauensanker bei der Host-Authentifizierung bei SSH

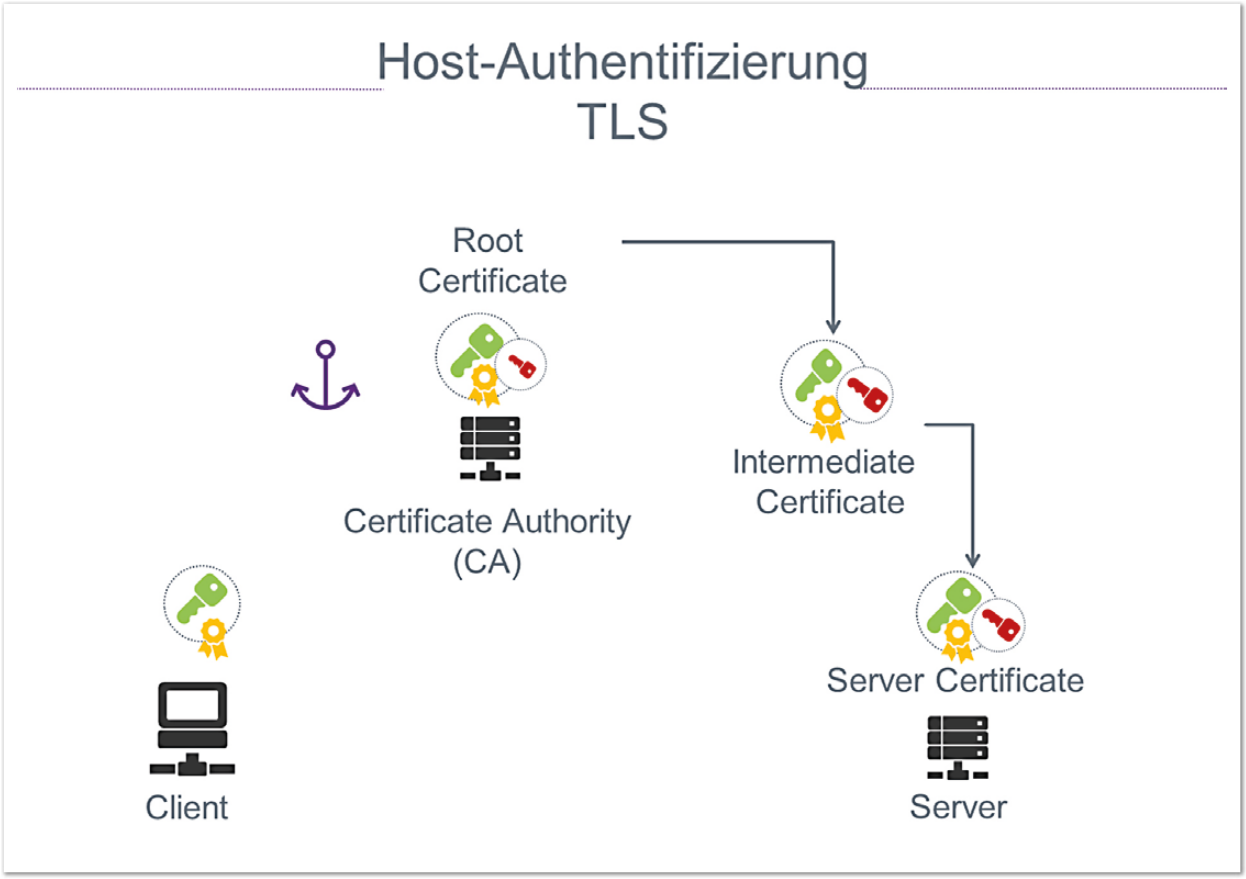


Abbildung 3: Vertrauensanker bei der Host-Authentifizierung bei TLS

Eine für Passwörter geeignete Hash-Funktion ist „bCrypt". Sie erzeugt aus einem Salt einen Kostenfaktor und aus dem Passwort einen String, der sowohl den Salt als auch den Hash enthält. „12" ist ein guter Kostenfaktor für den Moment. Im Security Stack Exchange *(siehe „https://security.stackexchange.com/questions/211/how-to-securely-hash-passwords/31846#31846")* findet ihr eine sehr ausführliche Antwort, die weitere gute Hash-Funktionen vorstellt und eine ausführliche Darstellung bietet.

Host-Authentifizierung bei SSH

Die meisten Entwickler haben sich schon mal per SSH auf einem Server eingeloggt. Bei der ersten Verbindung zu einem SSH-Server wird man typischerweise gefragt, ob man dem Server vertrauen möchte – na klar möchte man das; oder vielleicht doch lieber nicht?

Bei der Verbindungsinitiierung stellt sich der Server vor, indem er seinen öffentlichen Schlüssel bereitstellt. Anhand dessen kann der Client entscheiden, ob er den Verbindungsaufbau fortführen will, weil er den Schlüssel kennt – im Grunde wie die Gästeliste für eine VIP-Party. Im weiteren Verlauf des Handshake beweist der Server, dass er im Besitz des zugehörigen privaten Schlüssels ist *(siehe Abbildung 2)*.

Idealerweise erfolgt der Austausch des öffentlichen Schlüssels über einen anderen Kanal als bei der ersten Verbindung, denn dieser öffentliche Schlüssel ist der Vertrauensanker für die Authentifizierung des Servers gegenüber dem Client. Eine Möglichkeit ist ein USB-Stick. Ansonsten sollte man sich vom Administrator beispielsweise per Telefon den Fingerprint (Hash) des öffentlichen Schlüssels bestätigen lassen, um sicherzugehen, sich nicht mit einem Fake-Server zu verbinden. Das ist beim heimischen Raspberry Pi sicherlich wenig kritisch als bei der Verbindung zu einem Remote-Backup-Server, zu dem das eigene Backup übertragen wird.

Host-Authentifizierung bei TLS

Die Anzahl der SSH-Server, zu denen man sich regelmäßig verbindet, ist überschaubar. Etwas anders gelagert ist die Situation bei Webseiten. Jeden einzelnen öffentlichen Schlüssel einer Webseite vorab zu speichern, ist einfach unpraktikabel. Dennoch besteht der Bedarf zu entscheiden, ob eine Webseite diejenige ist, für die sie sich ausgibt, insbesondere dann, wenn Kreditkarten-Daten oder andere sensible Informationen ausgetauscht werden. An dieser Stelle wird das Konzept von Schlüsselpaaren einfach mehrstufig erweitert und das Ergebnis ist eine Zertifikatskette.

An die Stelle eines öffentlichen Schlüssels für jeden Server tritt ein öffentlicher Schlüssel einer Zertifizierungsstelle („Certificate Authority", CA). Dieser öffentliche Schlüssel ist mit dem zugehörigen privaten Schlüssel signiert und wird „Root-Zertifikat" genannt. Mit genau diesem privaten Schlüssel sind weitere öffentliche Schlüssel signiert, die „Intermediate Certificates". Es gibt typischerweise mehrere Stufen von Intermediate Certificates. Im Wesentlichen wird dadurch das Ziel verfolgt, den zentralen privaten Schlüssel so selten wie möglich aus dem Tresor zu holen. Ein Schlüsselpaar ist entweder zur Ausstellung weiterer Intermediate Certificates oder zur Verwendung für eine Webseite gedacht *(siehe Abbildung 3)*.

Durch diese Verkettung muss der Client nur dem Root-Zertifikat vertrauen und kann dennoch sichergehen, dass der Webserver der richtige ist. Zusammengefasst bedeutet das, dass ein Zertifikat ein öffentlicher Schlüssel eines asymmetrischen Verfahrens ist, der seinerseits mit einem asymmetrischen Verfahren signiert wurde. Mithilfe eines Certificate Signing Request (CSR) wird um diese Signierung eines öffentlichen Schlüssels gebeten und gleichzeitig der Beweis geführt, dass der Antragsteller im Besitz des zugehörigen privaten Schlüssels ist. Der CSR enthält daneben noch weitere Informationen, beispielsweise die URL der Webseite, für die das Zertifikat genutzt werden soll, und darüber, wer es beantragt.

Cipher Suites am Beispiel von TLS

Anleitungen zu Konfigurationen von Web-Servern enthalten regelmäßig auf den ersten Blick unleserliche Zeichenfolgen, wie zum Beispiel „ECDHE_RSA_WITH_AES_256_CBC_SHA384". Es ist allgemein bekannt, dass TLS-Verbindungen verschlüsselt sind. Das konkrete Verschlüsselungsverfahren wird jedoch durch die Cipher Suite festgelegt. Im gewählten Beispiel erfolgt die Verschlüsselung mit AES (CBC-Modus, 256 Bit Schlüssel). Zusätzlich wird ein HMAC auf Basis von SHA-384 gebildet, der die Integrität und Authentizität der vertraulichen Inhalte sichergestellt.

Woher kommt jedoch der verwendete Schlüssel, der dem Server und dem Client vorliegen muss? Er wird mithilfe des Diffie-Hellman-Verfahrens auf Basis der diskreten Exponentialfunktion auf elliptischen Kurven bestimmt. Das mathematische Verfahren ermöglicht es, dass sich zwei Parteien über einen unsicheren Übertragungsweg dennoch auf einen wirklich geheimen Schlüssel einigen können. Weiterhin überprüft der Browser mit dem RSA-Verfahren, ob der Server wirklich derjenige ist, für den er sich mit seinem Zertifikat ausgibt. An dieser Stelle muss der Server unter Beweis stellen, dass er im Besitz des passenden privaten Schlüssels ist. Schlüsselgenerierung und Zertifikatsprüfung finden als Teil des TLS-Handshake statt und sind die Grundlage für die anschließend symmetrisch verschlüsselte Kommunikation.

Ganz allgemein ausgedrückt, legt eine Cipher Suite fest, welche kryptographischen Primitive in einem Protokoll zum Einsatz kommen. Welche Elemente nötig sind, hängt dann wiederum vom Protokoll ab, deswegen unterscheiden sich zum Beispiel die Cipher Suites zwischen TLS 1.2 und TLS 1.3.

Weiterführendes Material und Tools

Unabhängig von der Frage, ob Passwörter durch die Anwendung hinreichend sicher gespeichert sind, besteht meine Empfehlung, keine schwachen Passwörter zu erlauben und starke Passwörter zu ermöglichen (unter anderem nicht auf zwölf Zeichen zu begrenzen). Für eine einfache Realisierung von Passwort-Policies in Java eignet sich Passay *(siehe „http://www.passay.org")*. Wenn Kryptographie Teil des Anwendungsfalls ist, ist man mit Bouncy Castle *(siehe „http://www.bouncycastle.org/java.html")* gut bedient. Darin ist auch eine Implementierung von bCrypt enthalten.

Einen niederschwelligen Einstieg in die sichere Konfiguration von gängigen Webservern bietet der Mozilla-Config-Generator *(siehe „https://mozilla.github.io/server-side-tls/ssl-config-generator")*. Die Konfiguration kann anschließend mit dem Qualys-SSL-Lab-Server-Test *(siehe „https://www.ssllabs.com/ssltest")* bewertet werden. Diese Überprüfung lässt sich übrigens auch dank des offiziellen API *(siehe „https://www.ssllabs.com/projects/ssllabs-apis")* prima mit dem gewünschten Monitoring-Tool automatisieren.

Bei der Entscheidung, welches Verfahren eingesetzt werden kann und wie viele Bit ein Schlüssel haben sollte, hilft keylength.com *(siehe „https://www.keylength.com")*. Hier sind unter anderem die Empfehlungen von NIST und BSI zusammengetragen.

Im Blog von Bruce Schneier *(siehe „https://www.schneier.com")* findet ihr viel zum Thema „Security" aus der Perspektive eines sehr erfahrenen Kryptographen.

Fazit

Die grundlegenden Konzepte der hier vorgestellten kryptographischen Aspekte lassen sich gut überblicken. Ich habe bewusst viele Details ausgelassen, um einen Überblick zu ermöglichen. Eine detaillierte Betrachtung wird schnell sehr umfangreich. Die Folien zum Vortrag für diesen Artikel sind auf meinem Speaker Deck *(siehe „https://speakerdeck.com/omilke/crypto-101")* verfügbar.

Für den Umgang mit Kryptographie möchte ich euch noch zwei Direktiven ans Herz legen: Erstens, selbstgemacht ist schlecht, und zweitens, neu ist schlecht. Dabei geht es vor allem darum, dass die eingesetzten Verfahren gründlich geprüft sind, was typischerweise weder bei neuen noch bei selbstgemachten Verfahren der Fall ist. Die zur Verfügung gestellten Links helfen, mit Aufwand sichere Webserver bereitzustellen.

Oliver Milke
dev@oliver-milke.de

Oliver Milke (siehe „http://oliver-milke.de") arbeitet als Software Craftsman bei der TRIOLOGY GmbH und entwickelt dort am Cloudogu EcoSystem. Seit mehr als zehn Jahren ist Software-Entwicklung sein tägliches Brot und mehr als nur ein Beruf. Mit Security und Kryptographie ist er intensiv im Bereich „Mobile Online-Dienste" bei Volkswagen in Berührung gekommen. Er ist Mit-Organisator der JUG-Ostfalen und verbringt seine Freizeit am liebsten mit Sport.

iJUG Verbund

Alle Mitglieder auf einen Blick

Der iJUG möchte alle Java-Usergroups unter einem Dach vereinen. So können sich alle Java-Usergroups in Deutschland, Österreich und der Schweiz, die sich für den Verbund interessieren und ihm beitreten möchten, gerne unter office@ijug.eu melden.

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: Stefan Kinnen. 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
Chefredakteur (ViSdP): Wolfgang Taschner
Kontakt: redaktion@doag.org

Redaktionsbeirat:
Ronny Kröhne, IBM-Architekt; Daniel van Ross, FIZ Karlsruhe; André Sept, Freiberufler; Jan Diller, Triestram und Partner

Titel, Gestaltung und Satz:
Caroline Sengpiel,
DOAG Dienstleistungen GmbH

Fotonachweis:
S. 16: © Allan Swart/123RF
S. 19: © Jesadaphorn Chaiinkeaw/123RF
S. 37: © kheat/Fotolia
S. 48: © Sergey Nivens/123RF
S. 55: © conceptw/123RF
S. 61: © Alexander Atkishkin /123RF

Anzeigen:
Simone Fischer, DOAG Dienstleistungen GmbH
Kontakt: anzeigen@doag.org

Mediadaten und Preise unter:
www.doag.org/go/mediadaten

Druck:
adame Advertising and Media GmbH,
www.adame.de

Inserentenverzeichnis

<DevCamp 2018 />

30. - 31. Januar in Hamburg

DOAG

devcamp.doag.org ->


JavaLand 2018
13. - 15. März 2018 in Brühl bei Köln
Ab sofort Ticket & Hotel buchen!
www.javaland.eu
Programm online!
JavaLand