Ausgabe 02/2019

Java aktuell

Praxis. Wissen. Networking. Das Magazin für Entwickler

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

4 191978 304903 02

Java aktuell

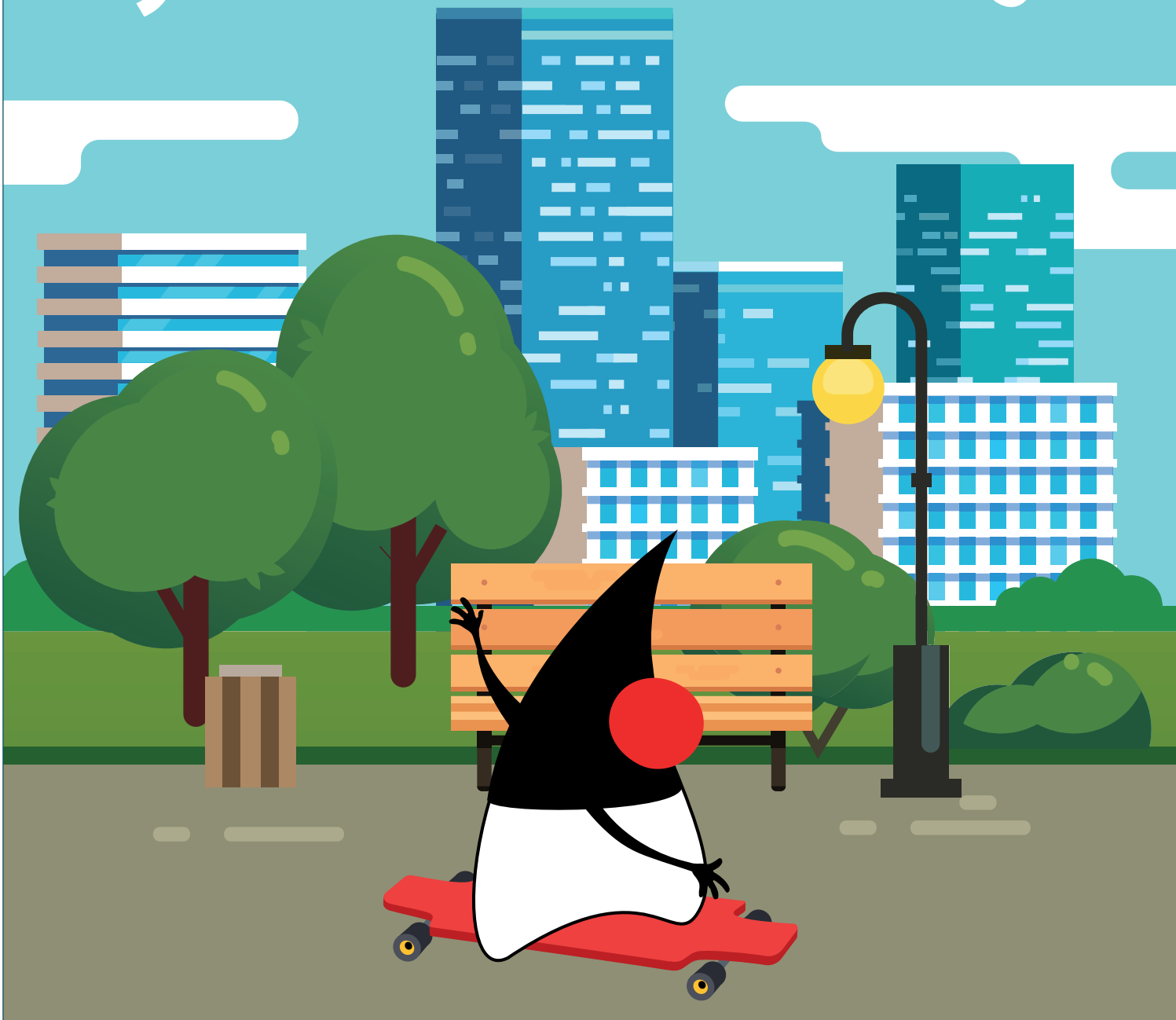
iJUG Verbund
www.ijug.eu

Open Source
Die Apache Software Foundation

Kubernetes Basics
Wie die App in die Cloud kommt

Bessere Usability
Design Thinking in der Entwicklung

Java macht Spaß



QAWARE
SOFTWARE ENGINEERING

Kommt vorbei:
JavaLand
Stand 618

IT-Probleme lösen.
Digitale Zukunft gestalten.

Mit Erfindergeist
und Handwerksstolz.

GREAT PLACE TO WORK®
2018
Beste Arbeitgeber™
Information & Telekommunikation

Von Mitarbeitenden empfohlen:
kununu.com/qaware
qaware.de/karriere

Mach's gut, Java aktuell

Im März 2019 gehe ich nach einem erfüllten Berufsleben in Rente. Meine Nachfolge in der Chefredaktion der Java aktuell übernimmt Lisa Damerow aus dem DOAG-Kommunikationsteam.

Nach Gründung der Zeitschrift im September 2010 habe ich jede Ausgabe sehr gerne gemacht – immer vor dem Hintergrund, Erfahrungen der Java-Entwicklerinnen und Entwickler an andere weiterzugeben. Das Motto „Aus der Community, für die Community" war für mich die Motivation.

Als ich im Januar 1978 nach dem Informatik-Studium meine erste Stelle bei Siemens in München als Software-Entwickler antrat, lernte ich sehr schnell die Erfahrungen meiner damaligen Kollegen schätzen, die mir beim Programmieren im 8080-Assembler hilfreich waren. Das war wohl schon der erste Anstoß, später aus dem Entwicklungslabor in die Redaktion der Zeitschrift „Chip" zu wechseln, um interessante Informationen an die Leserinnen und Lesern weiterzugeben.

Nach mehreren Zwischenstationen als Chefredakteur anderer Zeitschriften bin ich in den 1980er Jahren in die Selbstständigkeit gewechselt und habe die PR-Arbeit für Unternehmen wie Apple, Hewlett-Packard, NEC und Olivetti unterstützt. Parallel dazu folgten zwei Forks, einer als erfolgreicher Autor umfangreicher Shareware-Bücher beim Zweitausendeins-Verlag und einer mit dem Schreiben zahlreicher Wander-, Mountainbike- und Reiseführer.

Neben der Java aktuell habe ich den Interessenverbund der Java User Groups e.V. (iJUG) in allen Fragen der Presse- und Öffentlichkeitsarbeit beraten. Die Mitgliederversammlungen des iJUG haben mir immer viel Spaß gemacht, weil ich dort viel Hintergrundwissen über Java bekam. Außerdem ist es schön zu sehen, wie der Verein von anfangs sieben Gründungsmitgliedern heute die Interessen von fast vierzig User Groups aus Deutschland, Österreich und der Schweiz vertritt. Ein weiteres Highlight war für mich die Teilnahme an den Community-Veranstaltungen, angefangen bei der JavaLand über das Java Forum Stuttgart bis hin zum Java Forum Nord. Überall war das große Engagement aller Beteiligten zu spüren.

Ich freue mich, wenn ich dank tatkräftiger Unterstützung meiner Kolleginnen und Kollegen den einen oder anderen Java-Entwickler oder Oracle-Anwender bei seiner täglichen Arbeit helfen konnte. In diesem Sinne wünsche ich allen Leserinnen und Lesern viel Glück und Erfolg bei ihren Projekten.

Für ihre Geduld und Inspiration bedanke ich mich bei meiner Frau und unseren beiden Söhnen. Auch wenn das Leben manchmal schwer war, sind sie mir nie von der Seite gewichen.

Ihr

Wolfgang Taschner
Chefredakteur Java aktuell

Inhalt

Das Unit-Testen von CDI-Anwendungen ist nicht trivial

Wenn die Entwicklung eines Produkts kein Erfolg wird

Die Apache Software Foundation ist ein fester Begriff

Laut Vorurteil sind Programmierer teuer, faul und meistens unkontrollierbar

Das Java-Tagebuch

Andreas Badelt, stellv. Leiter der DOAG Java Community

Das Java-Tagebuch gibt einen Überblick über die wichtigsten Geschehnisse rund um Java.

25. Oktober 2018

Duke's Choice Awards

Die Duke's Choice Awards, jetzt auf der Code One verliehen, haben es nicht mehr in die letzte Ausgabe geschafft, die Gewinner sollen aber nicht verschwiegen werden: Ausgezeichnet wurden mehrere Projekte: ClassGraph (ein extrem schneller Classpath/Module Scanner und Annotation Processor), MicroProfile, Helidon (Oracle Microservice Framework mit MicroProfile-Unterstützung), Twitter4J sowie Apache NetBeans. Außerdem einige Firmen und andere Organisationen: Die Bulgarian JUG für ihren gesamten Einsatz in der Community, Jelastic für die Elastic JVM und JPoint für ihre „autonomous driving vehicle"-Demo, basierend auf Java/Vert.x und dem Raspberry. Herzlichen Glückwunsch!

https://blogs.oracle.com/java/announcing-2018-dukes-choice-award-winners

29. Oktober 2018

Big Red

Bumm! IBM kauft RedHat. Ganz ehrlich, wer hätte damit gerechnet? Ich war von der Übernahme generell überrascht – egal ob IBM, Google, Oracle oder sonst wer – weil ich zuletzt den Eindruck hatte, dass Red Hat mit seinem Hype-kompatiblen Portfolio offene Türen in den großen Konzernen (und vielleicht auch kleineren Unternehmen) einrennt. Offensichtlich verspricht man sich von der Übernahme und IBMs „big money" ein noch schnelleres Wachstum. IBM will mit den Red-Hat-Technologien das schaffen, was dem (einst) stolzen Riesen aus eigener Kraft nicht gelungen ist: Zu den großen Cloud-Anbietern aufzuschließen. Dabei sollen Red Hat eigenständig innerhalb von IBM weiterbestehen und insbesondere die Konzernkultur (und die mutmaßlich schlankeren Prozesse) erhalten bleiben, so zumindest die offiziellen Verlautbarungen. Mal sehen, wie lange sich das IBM-Management an den eigenen Plan hält und keine Kostensenkungs-Fantasien entwickelt. Tja, der Entwickler ist ein scheues Reh, so eine Übernahme kann sonst schnell anders laufen als gedacht. Zumindest sollte sich kurzfristig nichts am Open-Source-Engagement der beiden Firmen ändern – die ja jeweils zu den größten Unterstützern etwa von Jakarta EE und MicroProfile gehören.

https://newsroom.ibm.com/2018-10-28-IBM-To-Acquire-Red-Hat-Completely-Changing-The-Cloud-Landscape-And-Becoming-Worlds-1-Hybrid-Cloud-Provider

31. Oktober 2018

Spring Boot 2.1

Spring hat neue Versionen seiner wichtigsten Frameworks herausgebracht. Das Spring-Framework selbst war schon letzten Monat auf Version 5.1 aktualisiert worden, die nun Java 11 unterstützt. Jetzt ist auch Spring Boot 2.1 da und bindet eben dieses Spring 5.1 ein sowie eine Reihe weiterer Komponenten wie Tomcat 9 und Undertow 2. Micrometer 1.1 und unter anderem zwei neue „Actuator"-Endpunkte („/caches" und „/integrationgraph") bieten bessere Monitoring-Möglichkeiten.

https://spring.io/blog/2018/10/30/spring-boot-2-1-0

6. November 2018

TomEE 8 bald da – mit Unterstützung für EE 8

Der „Milestone 1 Build" von TomEE 8 wurde im Tomitribe-Blog angekündigt, mit Unterstützung für Java EE 8 und seinen funktionalen Klon Jakarta EE 8 sowie MicroProfile 1.3. Zum MicroProfile-Release-2.0 fehlen laut Blog-Eintrag nur noch ein paar kleinere Updates, und damit wohl auch zur Version 1.4 – im Grunde ist 2.0 ja nur die Kombination aus MicroProfile 1.4 und Java EE 8 als Grundlage. Diese Updates sollen bis zum Release-Datum erfolgen.

https://www.tomitribe.com/blog/tomee-announcing-tomee-8-m1

11. November 2018

introspected.rest

Kein reiner Java-Bezug, aber von „@java" verbreitet: „intospected.rest" soll eine Alternative zu REST und GraphQL darstellen – klingt erst mal ganz spannend. Am besten war die Antwort eines anderen Twitter-Nutzers an @java: „By the time I learn ABC of it [a] new thing comes up". Da hat er recht, meine Leseliste wird auch immer länger.

https://introspected.rest

14. November 2018

AWS und Corretto – ein OpenJDK (nicht nur) für die Cloud

Amazon bringt jetzt auch eine eigene Distribution des OpenJDK heraus: Amazon Corretto. Ohne Kaffee-Anspielung geht es halt nicht, wobei nicht erwähnt wird, welche Zugabe für den Schuss Alkohol steht – doch nicht etwa das Team für den Long Term Support? Hinter dem Projekt stehen zwei Java-Urgesteine, die beide schon für Sun arbeiteten und seit dem Jahr 2017 bei AWS sind: James „Duke, I'm your father!" Gosling und Arun Gupta. Corretto 8 soll im ersten Quartal 2019 freigegeben werden, Version 11 „vor April". Beide sollen verschiedene Plattformen unterstützen, sodass sie auch außerhalb der Cloud und auf lokalen Maschinen genutzt werden können. Corretto 8 wird dann bis mindestens Juni 2023 kostenlose Security-Updates erhalten, die Version 22 bis August 2024 – wobei mich die Einschränkung auf Security Fixes ein bisschen stutzig macht. Einem geschenkten Gaul …, das sollte bei Projekt-Entscheidungen allerdings berücksichtigt oder noch mal hinterfragt werden.

https://tinyurl.com/y75we2la

20. November 2018

Google: J2CL wird Open Source

Google hat nach vier Jahren Arbeit an dem Projekt seinen „Java to Closure Transpiler" als Open Source auf GitHub freigegeben. Dieser soll moderne Web-Entwicklung aus einer Hand erleichtern und benutzt den Google-Closure-Compiler, der „JavaScript in besseres JavaScript kompiliert" (daher „Java to Closure"). Das Ganze ist keine Konkurrenz für das Google-Web-Toolkit, sondern soll die Grundlage für GWT 3 sein.

https://github.com/google/j2cl

20. November 2018

Ktor – asynchrone Programmierung mit Kotlin

JetBrains hat die Kotlin-basierte Bibliothek Ktor 1.0 freigegeben. Ktor ist Open Source und nutzt Kotlin-Koroutinen, mit denen asynchroner Code einfach zu schreiben und (vor allem) zu lesen ist. Hauptbestandteile sind ein HTTP-Server-Framework für die JVM und ein Multi-Plattform-HTTP-Client.

https://ktor.io

20. November 2018

Java Community Process – Wahlen zum EC

Bei den diesjährigen Wahlen zum JCP Executive Committee haben immerhin etwa 40 Prozent der wahlberechtigten knapp 1.000 JCP-Mitglieder abgestimmt. Der Spezifikations-Prozess hat insbesondere durch die Übergabe von Java EE als Jakarta EE an Eclipse an Bedeutung verloren, erfüllt aber immer noch eine wichtige Aufgabe. Insgesamt sind es 25 Sitze – 12 standen diesmal zur Wahl, die anderen 12 erst wieder nächstes Jahr (plus Oracle als Nummer 25 – oder 1, wie man's nimmt). Für die nächsten zwei Jahre sind die folgenden Firmen und Personen neu beziehungsweise wiedergewählt: Alibaba, Goldman Sachs, JetBrains, MicroDoc, SAP SE, Software AG, The Bank of New York Mellon und V2COM für die „Ratified Seats" (Vorschlag durch Oracle, Zustimmung oder Ablehnung durch die Mitglieder – wobei es bei keinem der Kandidaten brenzlig wurde); Azul Systems, Eclipse Foundation und die London Java Community für die „Elected Seats" (frei wählbar, die Chicago User Group musste sich den anderen drei geschlagen geben); und Ivar Grimstad für den „Associate Seat" (von den „assoziierten" Mitgliedern, der untersten Mitgliedsstufe, frei wählbar), mit deutlichem Abstand vor Marcus Biel und zwei weiteren Kandidaten.

https://jcp.org/aboutJava/communityprocess/elections/2018.html

22. November 2018

Hystrix ist (nur noch) stabil

Netflix Hystrix, die Java-Bibliothek für Fehler- und Latenzzeit-Toleranz in verteilten Systemen, ist jetzt im Maintenance Mode und wird von Netflix nicht mehr aktiv weiterentwickelt. Für existierende Anwendungen will das Unternehmen sie weiterhin einsetzen, für neue Anwendungen jedoch auf dynamischere Ansätze bauen und konkret auf Projekte wie resilience4j, das selber von Hystrix inspiriert ist.

https://github.com/Netflix/Hystrix

28. November 2018

Gradle 5

Version 5.0 des Build-Tools Gradle ist da und unterstützt jetzt Java 11 vollständig (es bindet selbst JAXB-Packages ein, die in Java SE 11 nicht mehr enthalten sind). Außerdem ist die Kotlin-DSL jetzt „production ready" und es gibt ein paar nützliche neue Features wie Timeouts für Tasks.

https://gradle.org

28. November 2018

Java 12 mit IDEA

Das neue IntelliJ-IDEA-Release 2018.3 bietet unter anderem eine Preview-Unterstützung für die Features „Raw String Literals" und „Switch Expressions", die mit Java 12 – Zieltermin 19. März 2019 – herauskommen sollen. Switch-Expressions selber sind ja auch nur eine „Language Preview" in Java 12, die bei negativem Feedback sogar wieder entfernt werden könnte, also handelt es sich quasi um eine Preview der Preview.

https://blog.jetbrains.com/idea/tag/java-12

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

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").

Markus' eclipse-Corner

Am 14. Dezember verstrich die Frist, bis zu der Oracle und Co. im Namen der Eclipse Foundation das erste GlassFish-Release unter Führung der Stiftung veröffentlichen wollte. Gleichfalls gerissen wurde damit die lauthals verkündete Deadline für die Übergabe von Jakarta EE 8 durch den gleichen Konzern an die Community.

Ja, richtig gelesen: Die Firma Oracle hat es trotz Unterstützung potenter Freunde wie IBM und Red Hat tatsächlich nicht geschafft, binnen mehr als einem Jahr einfach den gleichen Code nochmals freizugeben! Die Reaktion der Eclipse Foundation dazu war: keine – oder sagen wir es so: „Das Release ist bis auf Weiteres verschoben." PMC-Lead Ivar Grimstadt hat mir dazu gesagt, er geht davon aus, dass es Ende Januar wird, doch es gibt weder einen neuen Termin noch einen Zeitplan noch eine Veröffentlichung der Steering Committee Minutes von Oktober und November.

Aha, anscheinend wusste man vom massiven Zeitverzug also schon seit Monaten und will es der Öffentlichkeit nicht sagen? Vielleicht schämt man sich im Firmen-Konsortium auch einfach, dass man gegenüber Oracle, die man bei Kritik gerne als „hart arbeitendes Mitglied" in Schutz genommen hat, doch – möglicherweise ob der immensen Beitragszahlungen – etwas zu nachsichtig war und nun komplett der Lächerlichkeit preisgegeben wurde?

Doch wundert uns das wirklich? Regt das echt noch einen auf? Ganz ehrlich: Erstens habe ich das sowieso von Anfang an erwartet und zweitens war es doch dank öffentlicher Quellen wie Issue-Trackern und Commit-Statistiken der Einzelprojekte durchaus absehbar.

Und jetzt? Letztendlich ist es vielen Anwendern mittlerweile egal, ob GlassFish 5.1 oder Jakarta EE 8 überhaupt noch erscheinen. Die Welt hat sich massiv gewandelt und ein API-Set, das hauptsächlich darauf abzielt, einzelne Application-Server zu betreiben, gilt vielen schon lange als hoffnungslos gestrig.

MicroProfile und proprietäre Lösungen werden von vielen als attraktivere Optionen angesehen, was zumindest teilweise erklärt, wieso die gleichen Personen, die bei Oracle eigentlich an der Migration und Freigabe von GlassFish arbeiten müssten, dort im vierten Quartal weniger Commits leisteten als beispielsweise an proprietären Cloud-Produkten wie „Oracle Helidon". Cloud geht vor, und vor allem geht hier den Industrial Playern vor allem der Konzerngewinn vor dem Nutzen für die Community. „Helodon" basiert ja letztendlich auf der ehemaligen JAX-RS-Referenz-Implementierung „Jersey" – doch dort wurden vergleichsweise wenig Ressourcen investiert und der Fortschritt war (und bleibt) schleppend. Ganz zu schweigen von der Einbindung der Community: Pull Requests liegen monatelang herum, ohne Eingang in das Produkt zu finden. So stellten wir uns Open Source nicht vor!

So bleibt auch in dieser Ausgabe wieder nur zu konstatieren: Oracle bleibt nun mal Oracle, Eclipse Foundation hin oder her. Wer Java EE, Entschuldigung, Jakarta EE benötigt, muss noch weiter warten oder sollte sich dringend einem der vielen Teilprojekte als aktiver und regelmäßiger Contributor anschließen. Die Möglichkeiten dafür sind dafür seit vielen Monaten vorhanden, und vielleicht geht dann ja das nächste Release dank vieler hiermit aktivierter Leserinnen und Leser ein klein wenig schneller über die Bühne. Die Hoffnung stirbt ja bekanntlich zuletzt. Gerne helfen wir bei der Arbeitsvermittlung, Zuschriften leiten wir gerne weiter!

Referenzen

[1] Ursprünglicher Release-Plan: *http://www.agilejava.eu/2018/09/13/eclipse-glassfish-release-plan*

[2] Aktueller Stand des Release: *https://wiki.eclipse.org/Eclipse_GlassFish_5.1_Components_Release_Tracker*

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.

Vaadin 10 – der Sprung von GWT zu WebComponents

Sven Ruppert

Um GWT durch WebComponents zu ersetzen, wurden alle Komponenten vollständig neu als WebComponents gebaut und dann das Java-API gegen diese WebComponents programmiert.

Lang ist es her, GWT wurde am 17. Mai 2006 von Google veröffentlicht und führte damals zu einem großen Umbruch. Mit GWT konnte der Java-Entwickler Web-Anwendungen sowohl auf der Client- als auch auf der Serverseite realisieren. Für den Entwickler fühlte es sich in gewisser Weise so an, als würde er eine Desktop-Anwendung entwickeln.

Sicherlich gab es auch andere Frameworks, die Ähnliches in Teilen angegangen sind, GWT war allerdings schon sehr mächtig am Markt und ist es vielerorts noch immer. Zum Beispiel gab es ein kleines Unternehmen in Finnland, Turku, mit dem Namen „IT Mill", das ebenfalls ein Framework in dieser Art hatte. Man erkannte damals, dass GWT viele Vorzüge hatte, und ersetzte die hausinterne Implementierung durch GWT. Allerdings ist das für den Entwickler sichtbare API in Java gleich geblieben. Mittlerweile hat sich die Firma umbenannt und ist den meisten unter dem Namen „Vaadin" bekannt. Dieses Jahr war es wieder soweit, GWT wurde durch WebComponents ersetzt *(siehe Abbildung 1)*.

Built for the modern mobile-first web

Heute gibt es drei wichtige Zielplattformen für den Aufbau von UIs für Business-Apps: Web, Android und iOS. Zwar bieten native Android- und iOS-Apps heute wohl das ausgereifteste UX-System, doch das Web hat sich aufgrund verschiedener Eigenschaften als Hauptziel für die meisten Geschäftsanwendungen etabliert.

Es gibt zwei Technologien, die diesen Trend sehr verstärken und die Welt der Geschäftsanwendungen nachhaltig verändern werden.

Progressive Web Applications

Mit Progressive Web Applications (PWA, siehe *„https://vaadin.com/progressive-web-applications")* nähert sich die Benutzererfahrung, die der Anwender einer Web-Anwendung erfährt, immer mehr den nativen Apps auf iOS und Android an. Mit PWA verschmelzen die nativen Apps und die Web Apps in ihrem Verhalten und der Benutzer kann immer weniger Unterschiede feststellen. Die Browser Chrome, Safari, Edge und Firefox bieten die dafür notwendigen Funktionen auf dem Desktop schon heute auf jedem Betriebssystem, auf dem sie ausgeführt werden.

WebComponents

Wofür bis heute Frameworks benötigt worden sind, kommen nun WebComponents *(siehe „https://www.webcomponents.org/introduction")* ins Spiel. Sie verändern das gesamte Ökosystem rund um die Web-Entwicklung, wenn man sich ansieht, wie bisher mit Komponenten gearbeitet worden ist. Ziel ist es, zwischen den verschiedenen Frameworks und Web-Plattformen kompatibel zu bleiben. In der Praxis bedeutet das, dass in einem Projekt WebComponents verschiedener Hersteller zusammen verwendet werden können.

Hier schließt sich der Kreis wieder. Alle Vaadin-Komponenten wie zum Beispiel der Button oder das Grid können als WebCompo-

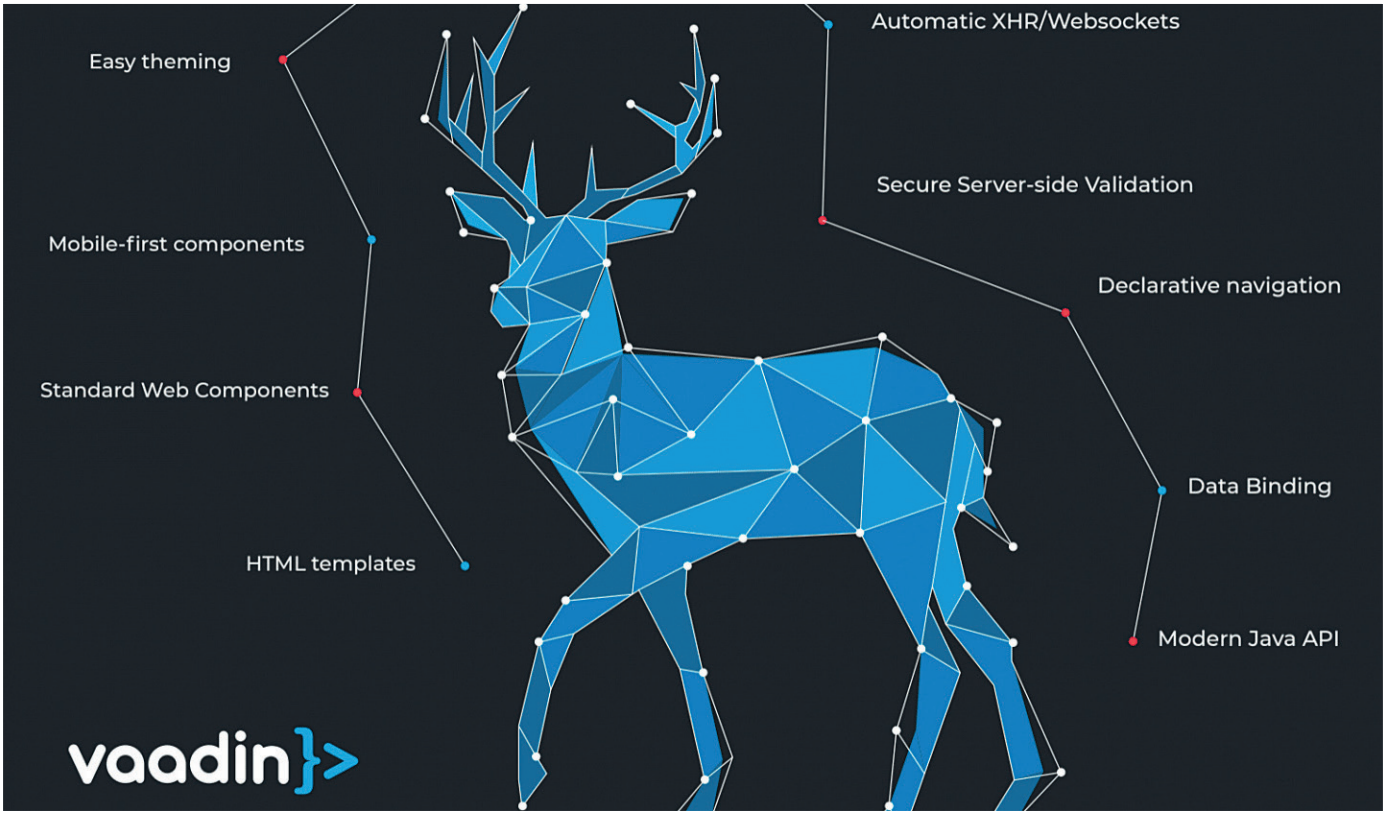


Abbildung 1: Das neue Vaadin

nent beliebig mit anderen WebComponents anderer Hersteller und Frameworks, die mit WebComponents umgehen können, kombiniert werden. Damit ist es nun seit Vaadin 10 möglich, mit Vaadin alleine eine rein clientseitige Anwendung zu entwickeln.

Client- und serverseitige Entwicklung

Bei der Entwicklung von Web-Anwendungen kann man den Fokus auf die Client- und/oder Serverseite legen. Je nachdem, worauf die Web App hin optimiert werden soll. Da es nicht die eine einzige richtige Antwort gibt, wurde Vaadin 10 so konzipiert, dass die Entwicklungsplattform sowohl die clientseitigen als auch die serverseitigen Technologien unterstützt, um Anwendungsentwicklern die Qual der Wahl zu lassen.

```xml
<vaadin-date-picker
  label="Select birth date"
  value="[[user.birthDate]]"
  required>
</vaadin-date-picker>
```

```java
DatePicker bd = new DatePicker("Select birth date");
bd.setValue(user.getBirthDate());
bd.setRequired(true);
```

Listing 1

Dies sollte nicht bedeuten, dass die Plattform so tun würde, als ob jede Anwendung die gleiche Architektur oder Teile hätte. Stattdessen stellt die Plattform Teile bereit, die gut zusammenarbeiten, allerdings unabhängig voneinander verwendet werden können; die Entwickler wählen die gewünschten Teile selbst aus.

Aufgrund dieser dualen Natur der Plattform können Teams ihre Kompetenzen nutzen und eine nahtlose Zusammenarbeit zwischen Web- und Java-Entwicklern aufbauen. *Listing 1* zeigt ein Beispiel aus beiden Welten.

Vaadin 10 – Flow

Zum Java-Teil: Das, was der Entwickler unter dem Namen „Vaadin-Framework" (Vaadin 8) kennt, heißt nun „Flow". Es ist Teil der Vaadin-Plattform und die Dokumentation dazu findet man unter *„https://vaadin.com/flow"*.

Ein obligatorisches „Hello World": Hier hat sich im Vergleich zu Vaadin 8 einiges verändert, besser gesagt, vereinfacht. Bei einer Vaa-

```xml
<properties>
    <maven.compiler.release>10</maven.compiler.release>
</properties>
```

Listing 2

din- oder Flow-Anwendung wird mindestens ein Servlet benötigt. Dieser initiale Einstiegspunkt musste unter Vaadin 8 noch selber in Java geschrieben werden. Das kann man sich nun sparen.

Beginnen wir zuerst mit der Bereitstellung eines Servlet-Containers. Für dieses Beispiel dient das Apache-Projekt „meecrowave". In diesem Beispiel kommt das OpenJDK 10 zum Einsatz, was bedingt, dass das Attribut „release" dementsprechend in der „pom.xml" gesetzt wird *(siehe Listing 2)*. Die Abhängigkeiten, um Apache „meecrowave" zu verwenden, sind ebenfalls recht überschaubar *(siehe Listing 3)*.

Um den Servlet-Container nun zu starten, benötigen wir noch eine Klasse mit einer „Main"-Methode. Darin wird der Container in diesem Beispiel von Hand konfiguriert und gestartet *(siehe Listing 4)*.

Damit ist die Infrastruktur so weit vorhanden. Kommen wir nun zu den Abhängigkeiten, die wir für die Vaadin-Anwendung benötigen. Zuerst holen wir uns die Versions-Definitionen, die in der verwendeten Vaadin-Plattform-Version gültig sind *(siehe Listing 5)*.

Hier gibt es nun etwas Neues zu beachten. Seit der Version 10 wurde das Versionsschema geändert. Bis zu der Version 8 gab es immer eine führende Hauptversion. Die Version 9 wurde aufgrund der großen Unterschiede zu Version 8 einfach übersprungen und es geht demnach gleich mit der Version 10 weiter.

Dabei handelt es sich ja um die Version der Plattform. Die Plattform selbst ist ein Konglomerat von Versionen einzelner Komponenten. So hat der Button als „WebComponent" eine eigene Version. Diese WebComponent des Buttons wird dann in der dazugehörigen Vaadin-Flow-Version verwendet. Alles zusammen ist in der Version der Vaadin-Plattform zusammengeführt. Das hat nun folgende Punkte, die für ein langlebiges Projekt wichtig sind.

Als Erstes handelt es sich bei der Version 10 um eine LTS-Version. Das bedeutet, dass diese Version über zehn Jahre gepflegt wird. Dazu erscheint alle sechs Monate eine weitere Hauptversion der Plattform. Dabei handelt es sich jedoch um eine kurzlebige Version in Bezug auf den Support. In diesen Versionen gibt es Weiterentwicklungen, die nicht unbedingt auch in die LTS-Version zurückportiert werden. Daraus ergibt sich der nächste Punkt.

Bei Verwendung der Plattform in einer Version (LTS oder nicht, ist dabei nicht von Bedeutung) kann die Version einzelner Komponenten verändert werden. Es kann demnach in der Version 10 (LTS) die Version etwa des Buttons als WebComponent erhöht werden, weil in dieser neuen Version eine neue Funktion implementiert worden ist, die für das Projekt von Bedeutung ist.

Es muss also nicht die gesamte Plattform in der Version angepasst werden. Man kann sich die Version der Plattform daher so vorstellen, dass es sich um eine getestete Kombination von Versionen einzelner Komponenten handelt. Besondere Aufmerksamkeit in Bezug auf Stabilität und Wartung einschließlich Sicherheits-Patches er-

```xml
    <!--Infrastructure-->
    <dependency>
      <groupId>org.apache.meecrowave</groupId>
      <artifactId>meecrowave-core</artifactId>
      <version>${meecrowave.version}</version>
      <scope>compile</scope>
    </dependency>
```

Listing 3

```java
public class BasicTestUIRunner {
  private BasicTestUIRunner() {
  }

  public static void main(String[] args) {
    new Meecrowave(new Meecrowave.Builder() {
      {
//        randomHttpPort();
        setHttpPort(8080);
        setTomcatScanning(true);
        setTomcatAutoSetup(false);
        setHttp2(true);
      }
    })
        .bake()
        .await();
  }
}
```

Listing 4

```xml
  <dependencyManagement>
    <dependencies>
      <!--Vaadin -->
      <dependency>
        <groupId>com.vaadin</groupId>
        <artifactId>vaadin-bom</artifactId>
        <version>${vaadin.version}</version>
        <type>pom</type>
        <scope>import</scope>
      </dependency>
    </dependencies>
  </dependencyManagement>
```

Listing 5

```xml
    <dependency>
      <groupId>com.vaadin</groupId>
      <artifactId>vaadin</artifactId>
    </dependency>

    <dependency>
      <groupId>com.vaadin</groupId>
      <artifactId>vaadin-lumo-theme</artifactId>
    </dependency>
```

Listing 6

```java
@Route("")
public class VaadinApp extends Composite<VerticalLayout> {

  private final Button btnClickMe = new Button("click me");
  private final Span lbClickCount = new Span("0");

  private int clickcount = 0;

  public VaadinApp() {
    btnClickMe.addClickListener(event -> lbClickCount.setText(valueOf(++ clickcount)));
    getContent().add(btnClickMe , lbClickCount);
  }
}
```

Listing 7

fahren die LTS-Versionen, da diese eben in dem Zeitraum von zehn Jahren gepflegt werden. Die nun verwendeten Abhängigkeiten in diesem Beispiel sind die beiden Hauptabhängigkeiten, die einem alle Open-Source-Bestandteile in dem Projekt zur Verfügung stellen *(siehe Listing 6)*.

Kommt es bei der Anwendung auf eine möglichst geringe Größe des Deployments an, kann man natürlich auch lediglich die jeweils benötigten Elemente definieren. Also zum Beispiel nur den Button sowie ein Label und das dann auch mit beziehungsweise ohne Java-API.

Die Implementierung

In Vaadin 8 war es noch notwendig, ein Servlet und eine Implementierung der „Basis UI"-Klasse selbst zu definieren. In der Realität sind diese allerdings meist immer gleich definiert; Anpassungen sind an dieser Stelle eher selten. Aus diesem Grunde wird mit Vaadin 10 nun auch eine Kombination aus Default-Implementierungen geliefert. Diese kommen zum Einsatz, wenn keine eigenen mit in den Klassenpfad gelegt werden.

Es kann also direkt mit der Definition der UI selbst begonnen werden. Neu ist auch die Definition der jeweiligen Route-Elemente. Als Erstes implementiert man eine Startseite, auf der ein Button und ein Label zu sehen sind. Dazu erzeugt man eine Klasse mit dem Namen „VaadinApp" und erbt von der Klasse „Composite<VerticalLayout>".

Bei der Klasse „Composite<T>" handelt es sich um einen Holder eines bestimmten Typs, der selber allerdings in dem zu erzeugenden DOM-Baum neutral ist. Nun stellt sich natürlich sofort die Frage nach dem „Warum?". Wenn zum Beispiel von der Klasse „VerticalLayout" geerbt wird, so ist die Komponente selbst ja auch wieder ein Layout. Allerdings ist es oft so, dass das Layout ja nur zur Strukturierung der benötigten Elemente verwendet werden soll: Die Komponente selbst ist jedoch kein Layout. Also wurde die Klasse „Composite" eingeführt. Ein ähnliches Konstrukt gibt es auch in zwei Varianten in Vaadin 8. Hier wurde demnach aufgeräumt und ein neuer, einheitlicher Weg definiert.

Innerhalb der Klasse „VaadinApp" hat man dann die Instanz des Typs, in diesem Fall die Klasse „Div", mit der Methode „getContent()" als Zugriff. Diese Instanz kann man dann verwenden, um die Kind-Elemente hinzuzufügen. *Listing 7* zeigt genau das mit einem Button und einem Span.

Das Erzeugen der Elemente erfolgt wie ein ganz normales Attribut der Klasse. Die beiden Elemente werden dann dem Layout, das die Root-Komponente darstellt, mit „getContent().add(btnClickMe , lbClickCount);" hinzugefügt.

Die Annotation der Klasse „@Route("")" definiert diese grafische Komponente als Root-Komponente. Die Attribut-Elemente der Annotation enthalten noch mehr Elemente: „@Route(value = "", layout = UI.class, absolute = false)". Der Default-Wert gibt den Navigationsteil an, unter dem diese Komponente zu erreichen sein soll. Zusammen mit dem Attribut „absolute" wird der finale Pfad bestimmt.

Als Standartwert „absulute=false" wird der Teil, der durch das Attribut „value = """ angegeben ist, als relatives Pfad-Element ausgewertet. Wenn man nun die „Main"-Methode der zuvor erzeugten Klasse „VaadinApp" aufruft, kann man nachfolgend unter der URL *„http://localhost:8080"* die Anwendung ausprobieren *(siehe Abbildung 2)*.

Das Layout

Das Layout einer Anwendung lässt sich auf viele Arten definieren und anpassen. Auch bei Vaadin ist es möglich, zum Beispiel durch CSS Anpassungen vorzunehmen. Hier wird allerdings der Weg beschrieben, der mithilfe programmatischer Layouts gegangen werden kann. Wer mehr zum Thema „CSS und Styling" lesen möchte, dem sind die entsprechenden Seiten unter *„vaadin.com"* empfohlen. Ein Einstiegspunkt könnte *„https://vaadin.com/docs/v11/flow/importing-dependencies/tutorial-include-css.html"* sein.

In der Annotation „@Route" ist auch das Attribut „layout" enthalten. Hier kann man eine Klasse angeben. Die angegebene Klasse selbst kann man sich nun wie einen Behälter oder besser gesagt wie einen Rahmen um die später anzuzeigende Komponente vorstellen.

```java
public class MainLayout extends Composite<Div> implements RouterLayout {

  private Div content = new Div();

  public MainLayout() {
    final VerticalLayout layout = new VerticalLayout(
        new Span("from MainLayout top") ,
        content,
        new Span("from MainLayout bottom")
    );
    getContent().add(layout);
  }

  //SNIPP...
}
```

Listing 8

```java
public class MainLayout extends Composite<Div> implements RouterLayout {

  private Div content = new Div();

  //SNIPP - constructor

  @Override
  public void showRouterLayoutContent(HasElement hasElement) {
    Objects.requireNonNull(hasElement);
    Objects.requireNonNull(hasElement.getElement());
    content.removeAll();
    content.getElement().appendChild(hasElement.getElement());
  }
}
```

Listing 9

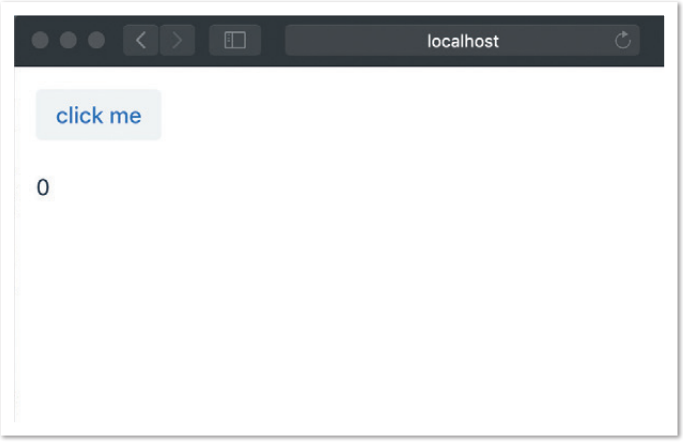


Abbildung 2: Click me

Die Klasse „MainLayout" im Beispielprojekt stellt das Hauptgerüst der Anwendung dar. Das wird hier stark vereinfacht dargestellt, indem oben und unten am Browser-Fenster jeweils ein Text angezeigt wird. Zwischen diesen beiden Texten soll nun der Rest der Anwendung dargestellt werden. Als Container für diesen Inhalt ist eine Instanz der Klasse „Div" bereitgestellt *(siehe Listing 8)*.

Was nun noch fehlt, ist der Mechanismus, um eine Komponente in das vorgehaltene „Div" zu setzen. Zum einen benötigen wir den programmatischen Teil im Layout selbst. Dazu wird die Methode „showRouterLayoutContent(HasElement hasElement)" aus dem Interface „RouterLayout" überschrieben. In dieser Methode wird das übergebene Element in den dafür vorgesehenen Container eingesetzt. Zuvor werden eventuell vorhandene Elemente entfernt *(siehe Listing 9)*.

Um nun dieses erste Basis-Layout zu verwenden, wird die Klasse „VaadinApp" mit dem neu erzeugten Layout versehen. Das erfolgt dadurch, dass nun die Klasse „MainLayout" als Wert dem Attribut „layout=MainLayout.class" innerhalb der Annotation „@Route" übergeben wird.

Wenn wir nun die Anwendung starten, erhalten wir eine Exception. Der Grund dafür ist in der Fehlernachricht zu lesen. Wenn man mit einem Layout arbeitet, so gehören die Annotationen „@Theme(..)" an die Klasse, die das Layout definiert; in diesem Fall an die Klasse „MainLayout" *(siehe Abbildung 3)*.

Kommen wir nun zu dem Punkt, in dem das Layout aus verschiedenen Elementen zusammengebaut wird. Zum Beispiel soll ein

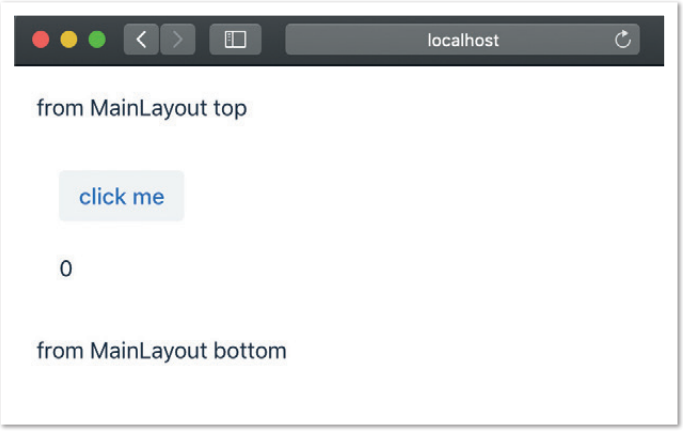


Abbildung 3: MainLayout

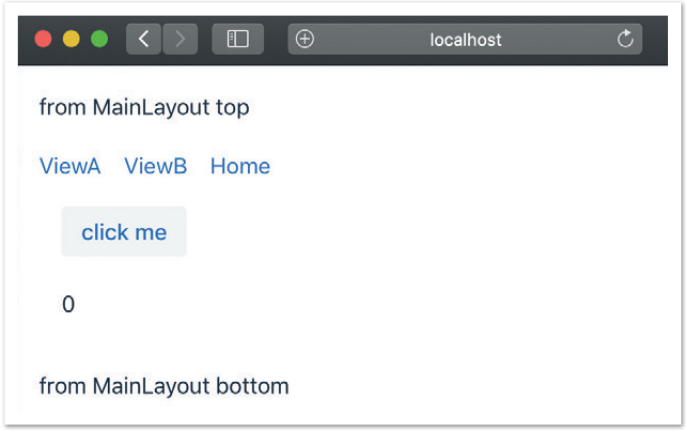


Abbildung 4: Das fertige Layout

```java
@ParentLayout(value = MainLayout.class)
public class LayoutWithMenuBar extends Composite<Div> implements RouterLayout {

  public LayoutWithMenuBar() {
    final HorizontalLayout layout = new HorizontalLayout(
        new RouterLink("ViewA" , ViewA.class) ,
        new RouterLink("ViewB" , ViewB.class) ,
        new RouterLink("Home" , VaadinApp.class)
    );
    getContent().add(layout);
  }
}
```

Listing 10

```java
@Route(value = "ViewA", layout = LayoutWithMenuBar.class)
public class ViewA extends Composite<Div> {

  public ViewA() {
    getContent().add(new Span("ViewA"));
  }
}

@Route(value = "ViewB", layout = LayoutWithMenuBar.class)
public class ViewB extends Composite<Div> {

  public ViewB() {
    getContent().add(new Span("ViewB"));
  }
}
```

Listing 11

Menü allgemein definiert werden, auch dieses ist Teil des Layouts. Hier kommen wir zu dem Begriff „ParentLayout". Mit der Annotation „@ParentLayout(..)" kann ein übergeordnetes Layout angegeben werden.

Das neue Layout, das wir für dieses Beispiel erzeugen wollen, erhält den Namen „LayoutWithMenuBar". Dieses erbt nicht von unserem „MainLayout", sondern erweitert dieses. Die Beziehung zwischen diesen beiden Layout-Ebenen wird mithilfe der Annotation „@ParentLayout(value = MainLayout.class)" an der Klasse „LayoutWithMenuBar" hergestellt *(siehe Listing 10)*. Bei den angegebenen Klassen „ViewA" und „ViewB" handelt es sich lediglich um Platzhalter für theoretische Navigationsziele innerhalb der Anwendung *(siehe Listing 11)*.

```java
@ParentLayout(value = MainLayout.class)
public class LayoutWithMenuBar extends Composite<Div> implements RouterLayout{

  private final Div content = new Div();

  private final HorizontalLayout menuBar = new HorizontalLayout(
      new RouterLink("ViewA" , ViewA.class) ,
      new RouterLink("ViewB" , ViewB.class) ,
      new RouterLink("Home" , VaadinApp.class)
  );

  public LayoutWithMenuBar() {
    getContent().add(new VerticalLayout(menuBar, content));
  }

  @Override
  public void showRouterLayoutContent(HasElement hasElement) {
    Objects.requireNonNull(hasElement);
    Objects.requireNonNull(hasElement.getElement());
    content.removeAll();
    content.getElement().appendChild(hasElement.getElement());
  }
}
```

Listing 12

Alle Views werden mit demselben gerade erweiterten Layout versehen. Wird die Anwendung nun gestartet, so ergibt sich das Bild in *Abbildung 4*. Wie wir sehen, sind die Elemente wie vorgesehen und ineinander geschachtelt. Wir können nun in dem angebotenen Menü navigieren und erhalten immer dasselbe Layout, allerdings mit wechselndem Inhalt, der durch die jeweilige View gesetzt worden ist.

Die Lösung ist allerdings nur zum Teil korrekt. Wenn man die erzeugten HTML-Elemente betrachtet, sieht man, dass die Menü-Elemente und die Inhalte der Arbeitsfläche zusammen in einem „Div" enthalten sind. Allerdings ist es von der Logik wohl eher so gemeint, dass die jeweiligen Arbeitsflächen innerhalb des letzten Layout-Elements enthalten sind.

Um hier genau das gleiche Verhalten zu realisieren, muss auch in dem erweiterten Layout die Methode „showRouterLayoutContent" implementiert werden. Damit ist nun sichergestellt, dass sich die Teil-Layouts neutral verhalten und keine implizierten Anforderungen enthalten sind *(siehe Listing 12)*.

Fazit

Das soll als kleine Einführung in Vaadin10 genügen; wer mehr wissen möchte, sollte einen Blick in die Vaadin-Online-Dokumentation werfen. Dort gibt es neben der Übersicht über alle verfügbaren Komponenten auch ein ausführliches Online-Handbuch.

Hinweis: Die Sourcen zu diesem Artikel sind auf GitHub unter *„https://github.com/Java-Publications/vaadin-v10-java-aktuell-intro"* zu finden.

Sven Ruppert
sven.ruppert@gmail.com

Sven Ruppert programmiert bereit seit dem Jahr 1996 in Java. Als Oracle Developer Champion und Developer Advocate bei Vaadin hilft er weltweit Entwicklern beim Wachstum ihres Geschäfts. In seiner Freizeit spricht Sven Ruppert auf internationalen und nationalen Konferenzen und schreibt für IT-Magazine sowie für Tech-Portale.

Unit-Tests für CDI und Eclipse-MicroProfile-Projekte

Gunnar Hilling, Gunnar Hilling IT Consulting GmbH

Das Eclipse-MicroProfile-Projekt macht klassische JEE-Technologien wie zum Beispiel JAX-RS fit für die Microservice- und Cloud-Welt. Dabei geht es um neue Bereiche wie „Health Check", „Fault Tolerance" und „Metrics", allerdings auch um die Verwendung von CDI zur Dependency Injection. Dieser Standard ist beim Unit-Testen nach wie vor problematisch.

Im Vergleich zum Konkurrenten Spring Boot, bei dem das Test-Framework schon immer ein „first class citizen" war, ist das Testing dem CDI-Standard herzlich egal. Es handelt sich natürlich auch – im Gegensatz zu Spring – um einen offenen Standard und eben nicht um ein Produkt. Hinzu kommt, dass das Unit-Testen von Anwendungen, die wie das MicroProfile auf CDI basieren, nicht trivial ist, da für den Test zunächst ein Container zu starten ist. Dies war für eine Stand-alone-Java-Umgebung in den ersten Versionen des Standards nicht einmal spezifiziert. Es gibt das Arquillian-Framework, das Integrationstests realisieren soll, aber nicht gerade trivial zu verwenden ist.

Ein großes Hindernis im Vergleich zu Spring ist außerdem die Tatsache, dass in Spring die Injection explizit konfiguriert werden kann. Dies ist so in CDI nicht vorgesehen, sondern die Injection erfolgt ausschließlich über Annotations und CDI-Extensions. Hierdurch wird die Verwendung von Mocks als Test-spezifischer Ersatz für die echten Implementierungen erschwert.

Ein einfacher Lösungsansatz

Um Komponenten möglichst einfach testen zu können, hat der Autor aus einem Kundenprojekt heraus die JUnit-Extension „cdi-test" entwickelt. Folgende Ziele wurden dabei verfolgt:

- Möglichst einfache Anwendung, flache Lernkurve
- Schnelle Ausführung vieler Unit- oder Komponenten-Tests
- Erweiterbarkeit
- Inzwischen basierend auf JUnit 5

Die gesamte Extension enthält im Core weniger als 1.000 Zeilen Code und funktioniert mit verschiedenen Weld- und JDK-Versionen. *Listing 1* zeigt einen einfachen Test-Case.

```
@ExtendWith(CdiTestJunitExtension.class)
public class SimpleTest {
    @Inject
    private Person person;
    @Test
    public void testInjection() {
        assertNotNull(person);
    }
}
```

Listing 1

Komplette Beispiele für Unit-Tests sind auf der Homepage des Projekts bei GitHub *(siehe „https://github.com/guhilling/cdi-test")* verfügbar. Dort steht auch die aktuelle Dokumentation, die unter anderem zeigt, wie die Abhängigkeiten für einen solchen Test zu definieren sind.

Die Extension startet bei Test-Ausführung lediglich einen CDI-Container für alle Tests. Dadurch ist die Ausführungszeit auch bei großen Projekten niedrig. Die Isolation aller Tests gegeneinander ist dennoch dadurch gewährleistet, dass alle CDI-Scopes inklusive des ApplicationScope für jeden einzelnen Test neu gestartet werden. Die Testklasse selber ist übrigens keine CDI-Bean, da sie von JUnit 5 instanziiert wird. Die Extension nimmt lediglich die Injection vor.

Verwendung von Mocks

Angenommen, man möchte einen Komponenten-Test für den Service „SampleService" erstellen, der intern einen „BackendService" verwendet. Dieser soll für unseren Test gemockt werden. Dafür gibt es eine Integration mit Mockito. Ein Mock des „BackendService" wird einfach im Test für alle Konsumenten dieses Service aktiviert, indem man in der Testklasse ein Attribut von diesem Typ anlegt und mit „@Mock" annotiert. Der Test sieht nun wie in *Listing 2* aus.

```
@ExtendWith(CdiTestJunitExtension.class)
public class ServiceMockTest {
    @Mock
    private BackendService backendService;
    @Inject
    private SampleService sampleService;
    @Test
    public void createPerson() {
        Person person = new Person();
        sampleService.storePerson(person);
        verify(backendService).storePerson(person);
    }
    @Test
    public void doNothing() {
        verifyZeroInteractions(backendService);
    }
}
```

Listing 2

Die eigentliche Magie ist in dieser Klasse unsichtbar, es wird jedoch verifiziert, dass der „SampleService" tatsächlich den Mock aufgerufen hat. Die „CdiTestJunitExtension" setzt dies dadurch um, dass jede Injection durch einen Proxy geleitet wird. Die Proxies stehen unter Kontrolle der Extension und können dadurch abhängig von der aktiven Testklasse entweder auf die Ziel-Bean oder auf einen Mock verweisen.

Test-Implementierung

Es gibt allerdings noch einen weiteren häufigen Wunsch alternativ zum Mock, nämlich eine maßgeschneiderte Test-Implementierung, die entweder in allen Tests oder wiederum pro Test eine Bean ersetzt. Auf globaler Ebene kann dies durch die Annotation „@GlobalTestImplementation" erreicht werden. Soll die Aktivierung analog zu Mocks pro Test-Case möglich sein, wird „@ActivatableTestImplementation" verwendet. Die Alternative wird einfach entsprechend annotiert, der Testfall ist analog zum vorherigen Beispiel *(siehe Listing 3)*.

Um diese Implementierung in einer Testklasse indirekt in allen CDI-Beans zu verwenden, die „BackendService" benutzen, wird die Implementierung einfach zusätzlich „injected". Dies dürfte in den meisten Fällen ohnehin notwendig sein, um die Test-Implementierung vorzubereiten und die Ergebnisse und Aufrufe zu verifizieren *(siehe Listing 4)*. Auch hier sollte die Benutzung nicht allzu viele Überraschungen bereithalten.

Erweiterbarkeit des Test-Frameworks

Häufig ist es wünschenswert, bestimmte Aspekte des eigenen Projekts auch in Test-Support-Klassen oder eben in bestimmten Test-Implementierungen abzubilden.

```
@ActivatableTestImplementation
public class BackendServiceTestImplementation extends BackendService {
     […]
     public int getInvocations() {
        return invocations;
     }
}
```

Listing 3

```
@ExtendWith(CdiTestJunitExtension.class)
public class ServiceTestImplementationTest {
    @Inject
    private SampleService sampleService;
    @Inject
    private BackendServiceTestImplementation testBackendService;

    @Test
    public void callTestActivatedService() {
        sampleService.storePerson(new Person());
        assertEquals(1, testBackendService.getInvocations());
    }
}
```

Listing 4

```
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
public @interface BackendServiceException {
    Class<RuntimeException> value();
}
```

Listing 5

„cdi-test" unterstützt dies durch zwei einfach zu nutzende Erweiterungspunkte:

- Es gibt zwei zusätzliche CDI-Scopes: „@TestScoped" und „@TestSuiteScoped". „@TestScoped"-Beans sind dabei für einen einzelnen Test gültig, der Scope wird aber vor allen anderen Scopes aktiviert und erst nach Beendigung des Tests als Letztes deaktiviert. „@TestSuiteScoped"-Beans sind während des gesamten Testlaufs gültig.
- Speziell für Test-Implementierungen von Services kann es nützlich sein, diese per Annotation an der Testmethode oder an der Testklasse zu konfigurieren. Dies lässt sich leicht durch einen CDI-Event-Listener realisieren. Die Test-Erweiterung erzeugt Events für das Starten und die Beendigung eines Tests.

Als Beispiel folgendes Szenario: Eine Test-Implementierung für einen Service soll, gesteuert durch eine Annotation, für verschiedene Tests unterschiedliche Ergebnisse liefern. Eine Annotation an der Testmethode kontrolliert, ob eine Exception geworfen wird *(siehe Listing 5)*. Um diese Annotation auszuwerten, lässt sich die vorgestellte Test-Implementierung erweitern *(siehe Listing 6)*.

Durch die Annotation „@ActivatableTestImplementation" wird nicht nur diese Klasse für die Umlenkung von Methoden-Aufrufen auf Beans berücksichtigt, sondern gleichzeitig der Scope auf „@TestScoped" festgelegt. Dieser Scope wird während der Testausführung vor allen Standard-Scopes gestartet und erst nach deren Beendigung wieder deaktiviert. Der Test mit Prüfung auf einen vorher definierten Fehler sieht dann wie in *Listing 7* aus.

Auf Basis dieser Mechanismen sind effiziente Erweiterungen mit geringem Programmier-Aufwand möglich; zum Beispiel kann man eine „EntityManagerFactory", deren Erzeugung zeitaufwendig ist, lediglich einmal für alle Tests bauen, dann aber für jeden einzelnen Test-Case einen neuen „EntityManager" aufsetzen und außerdem die Datenbank in einen definierten Zustand versetzen.

Eclipse MicroProfile

Eclipse MicroProfile spielt inzwischen eine wichtige Rolle in der ehemaligen Java-Enterprise-Welt. Zum einen wird ein Standard zur Entwicklung von leichtgewichtigen Microservices bereitgestellt, den auch eine Reihe von Firmen unterstützt. Zum anderen lassen sich hier Technologien evaluieren, die es dann vielleicht in eine noch zu definierende Version 9 von Java EE schaffen.

Um Unit-Tests für eine MicroProfile-Anwendung zu schreiben, sind zunächst die gleichen Anforderungen wie für jede Anwendung auf CDI-Basis gegeben: Im Prinzip kommt man also mit „cdi-test" weiter. Einen Großteil der MicroProfile-Standards wird man nicht unbedingt in Unit-Tests testen; Circuit-Breaker und Health-Checks sind eher ein Fall für Systemtests.

```
@ActivatableTestImplementation
public class BackendServiceTestImplementation extends BackendService {

    private int invocations;
    private RuntimeException exceptionToThrow;

    @Override
    public void storePerson(Person person) {
        if (exceptionToThrow != null) {
            throw exceptionToThrow;
        } else {
            invocations++;
        }
    }

    protected void testStarted(@Observes @TestEvent(EventType.STARTING) ExtensionContext context) {
        context.getTestMethod().ifPresent(m -> {
            // Setzen der zu werfenden Exception
        });
    }
```

Listing 6

```
@Test
@BackendServiceException(RuntimeException.class)
public void callTestActivatedServiceWithBackendException() {
    Assertions.assertThrows(RuntimeException.class, () -> {
        sampleService.storePerson(new Person());
    });
}
```

Listing 7

```
@ApplicationScoped
public class Controller {

@Inject
@ConfigProperty(name = "some.string.property")
private String stringProperty;
[…]
}
```

Listing 8

Es gibt jedoch eine zentrale Ausnahme innerhalb einer MicroProfile-Anwendung: MicroProfile-Config. Damit definiert man eine einheitliche Möglichkeit, MicroProfile-Anwendungen zu konfigurieren, und das dürfte in Zukunft in vielen MicroProfile-Beans verwendet werden. Dies erfolgt einfach durch die Annotation „@ConfigProperty" *(siehe Listing 8)*.

Das Beispiel dürfte selbsterklärend sein. Weitere Features sind automatische Umwandlungen in die gängigsten Datentypen. Eine Eigenschaft wie „stringProperty" im obigen Beispiel muss auch tatsächlich definiert sein. Wenn dies optional sein soll, muss (logisch) „Optional" verwendet werden. In einem „cdi-test" für den oben definierten Controller muss man also einen entsprechenden CDI-Producer für die im Projekt definierten Properties implementieren. Hierzu gibt es bereits Implementierungen – warum also das Rad neu erfinden.

```
<dependency>
    <groupId>de.hilling.junit.cdi</groupId>
    <artifactId>cdi-test-microprofile</artifactId>
    <version>1.0.0</version>
    <scope>test</scope>
</dependency>
```

Listing 9

Um die Arbeit zu erleichtern, hat der Autor die „cdi-test"-Erweiterung „cdi-test-microprofile" geschrieben. Beispiele und Source-Code sind wie üblich bei GitHub verfügbar *(siehe „https://github.com/guhilling/cdi-test-microprofile")*. Bei einem bereits lauffähigen Test genügt es, diese Abhängigkeit zusätzlich zu vereinbaren *(siehe Listing 9)*. Die Bibliothek beruht auf „smallrye-config", das auch von „thorntail", ehemals „wildfly-swarm", verwendet wird.

Die Properties für das Testing werden momentan noch aus der Datei „microprofile-config.properties" gelesen, die als Test-Ressource angelegt werden sollte. In Zukunft wird es auch möglich sein, die Werte aus den „project-defaults.yml" zu verwenden, wie bei „thorntail" üblich.

Leider gibt es momentan ein unschönes Problem beim Packaging durch das „thorntail-maven"-Plug-in: Falls im Test-Scope ein „deltaspike-core-api" in einer zu neuen Version, nämlich größer 1.8,

vorhanden ist, erstellt das Plug-in kein sinnvolles „uberjar" mehr. Diese Bibliothek wird allerdings von „cdi-test" in Version 1.9.0 benötigt. Das Problem lässt sich umgehen, indem für die transitive Abhängigkeit „deltaspike-core-api" explizit die Abhängigkeit „provided" vereinbart wird *(siehe Listing 10)*. Die Ursache für dieses Problem ist allerdings im „thorntail"-Projekt zu suchen: Das Packaging sollte nicht abhängig von Test-Dependencies sein.

Custom Converter

Für eigene Datentypen ist es im Zusammenhang mit „microprofile-config" oft nützlich, eigene Converter zu definieren. Das in „cdi-test-microprofile" verwendete „smallrye-config" kann zwar schon recht intelligent auch andere als Standard-Datentypen zum Beispiel über einen passenden Konstruktor mit einem String als Argument erzeugen, man stößt jedoch recht schnell auf Fälle, in denen dies nicht geht. Der Autor lässt sich gerne seine Value-Objekte von der Immutables-Compiler-Erweiterung erzeugen *(siehe Listing 11)*. Aus diesem Interface entsteht beim Compile-Vorgang eine Implementierung „ImmutableHorse", die zur Instanziierung eine innere Builder-Klasse bereitstellt. Ein Custom-Converter für MicroProfile-Config würde dann wie in *Listing 12* aussehen.

Durch die hohe Priorität wird dieser Converter stets benutzt, selbst wenn die Config-Implementierung selbst in der Lage wäre, diesen Datentyp aus String zu erzeugen. Um den Konverter in einer Standard-Implementierung zu nutzen, muss er über das „ServiceLoader"-Framework aktiviert werden, hierzu wird im Classpath eine Datei mit dem Namen „services/org.eclipse.microprofile.config.spi.Converter" angelegt. Inhalt sind die Namen der zusätzlichen Converter „de.hilling.junit.cdi.microprofile.HorseConverter". Dieser Mechanismus funktioniert genauso auch für Unit-Tests, die mit „cdi-test-microprofile" erstellt wurden, da diese letztlich eine Standard-Implementierung nutzen.

```
<dependency>
    <groupId>org.apache.deltaspike.core</groupId>
    <artifactId>deltaspike-core-api</artifactId>
    <version>1.9.0</version>
    <scope>provided</scope>
</dependency>
```

Listing 10

```
@Value.Immutable
public interface Horse {
    String getName();
}
```

Listing 11

```
@Priority(1000)
public class HorseConverter implements Converter<Horse> {
    @Override
    public Horse convert(String value) {
        return ImmutableHorse.builder().name(value).build();
    }
}
```

Listing 12

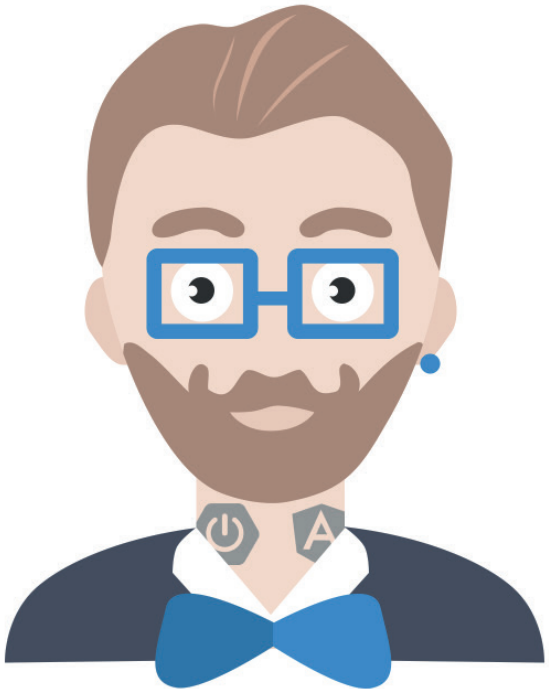
Fazit

Das Thema „Testing" ist leider in der Vergangenheit sowohl von den JEE-Anbietern als auch vor allem von den Standardisierungsgremien sehr stiefmütterlich behandelt worden. Eclipse MicroProfile bietet für die Anwendungsentwicklung sehr gute und praxisnahe Lösungen. Der Ansatz, hier offene und praxisnahe Standards zu entwickeln, die auch in Zukunft die Wahl zwischen verschiedenen Technologien lassen, ist zu begrüßen. Der Autor meint allerdings, dass auch hier das Thema „Testbarkeit" bisher nicht ausreichend behandelt wurde. Der Erfolg von Spring und Spring-Boot im Vergleich zu JEE ist nicht zuletzt darauf zurückzuführen, dass beide es den Entwicklern leicht machen, damit entwickelte Anwendungen zu testen.

Genau wie bei JEE fehlt beim MicroProfile bis heute ein starkes Commitment der Community und der Member zum Thema „Testbarkeit". Der Autor würde sich auch deshalb über Feedback und weitere Anforderungen zum Thema sehr freuen.

Gunnar Hilling
gunnar@hilling.de

Gunnar Hilling ist nach seinem Abschluss als Dipl.-Physiker direkt in die IT gewechselt. Dazu beigetragen hat sicherlich die frühe Beschäftigung im Studium mit Unix- und Shell-Programmierung sowie mit den für das 20. Jahrhundert grandiosen NeXTstations, die die Osnabrücker Uni damals angeschafft hatte. Nach einem mehrjährigen Ausflug in die Lehre an der Uni Stuttgart hat er sich selbstständig gemacht und arbeitet seitdem im Trainingsgeschäft sowie in einer Reihe von Großprojekten hauptsächlich im Bereich „Java Enterprise". In den letzten Jahren hat sich dabei der Fokus mehr in Richtung DevOps und Integration sowie Automation verschoben.



Getting Hip with JHipster

Frederik Hahne, WPS Management GmbH

Der Scrum-Master Samu hat ein Problem. Er kann nicht noch mehr Funktionen aus dem unternehmensweiten Ticketsystem herausholen. Deshalb spielt er mit dem Gedanken, eine Web-Anwendung zu schreiben, die sich in das bestehende Ticketsystem integriert, sodass er dort spezielle Anforderungen und Workflows implementieren kann. Leider liegen seine Tage als Entwickler etwas weiter zurück und er hat auch keine Zeit, sich mit dem Schreiben von Boilerplate-Code einer modernen Web-Anwendung zu befassen. Zum Glück erinnert er sich, dass die Java-Entwicklerin Jennifer in ihrer Freizeit im JHipster-Projekt aktiv ist. Dieser Artikel begleitet Samu dabei, wie er die Möglichkeiten von JHipster erkundet, um einen ersten Prototyp zu bauen und diesen am Ende in der Cloud zu betreiben.

Samu schaut sich erst mal die Dokumentation von JHipster an *[1]*. Das Programm behauptet von sich selbst, ein Framework zu sein, um Spring-Boot-basierte Web-Anwendungen und Microservices mit Angular oder React Frontend zu generieren, zu entwickeln und zu betreiben *[2]*. JHipster kombiniert unterschiedliche Technologien und Frameworks, konfiguriert diese nach aktuellen Best Practices und stellt sicher, dass die verwendeten Technologien reibungslos miteinander funktionieren.

Da Samu für einen Prototyp keine Zeit hat, um sich in die Besonderheiten von Spring Security einzulesen oder in die Konfiguration von Angular und Webpack einzuarbeiten, gibt er JHipster eine Chance. Er wird mit JHipster die Anwendungen hochfahren und das Datenmodell erzeugen, damit er sich auf das Implementieren der Businesslogik konzentrieren kann. JHipster hilft nicht nur Anfängern, sondern auch erfahrenen Entwicklern, schneller produktiv zu sein *[17]*.

JHipster in Zahlen

Samu möchte auf einem aktiven und gut gepflegten Projekt aufbauen, damit er bei Problemen und Fragen schnell Hilfe findet und Fehler durch das Team zeitnah behoben werden können, und schaut sich die Historie des Projekts an. JHipster wurde im Jahr 2013 gestartet, seitdem gab es fünf Major-Releases. Die aktuelle Version ist 5.6.1 (November 2018). Das Projekt ist sehr aktiv *[19]* und wird von einigen Firmen finanziell gesponsert. Im letzten Monat wurden mehr als 100 Pull Requests gemergt und mehr als 90 Issues geschlossen *(siehe Abbildung 1)*.

Das Kernteam besteht aus 23 Mitgliedern, darunter auch zwei Java Champions, und insgesamt 460 Contributern. Einige Teile wie zum Beispiel der Kubernetes-Support werden direkt von Google entwickelt. Unter den Nutzern von JHipster *[3]* sind neben Google, HBO und Pivotal auch Firmen wie Siemens und Bosch. JHipster hat fast

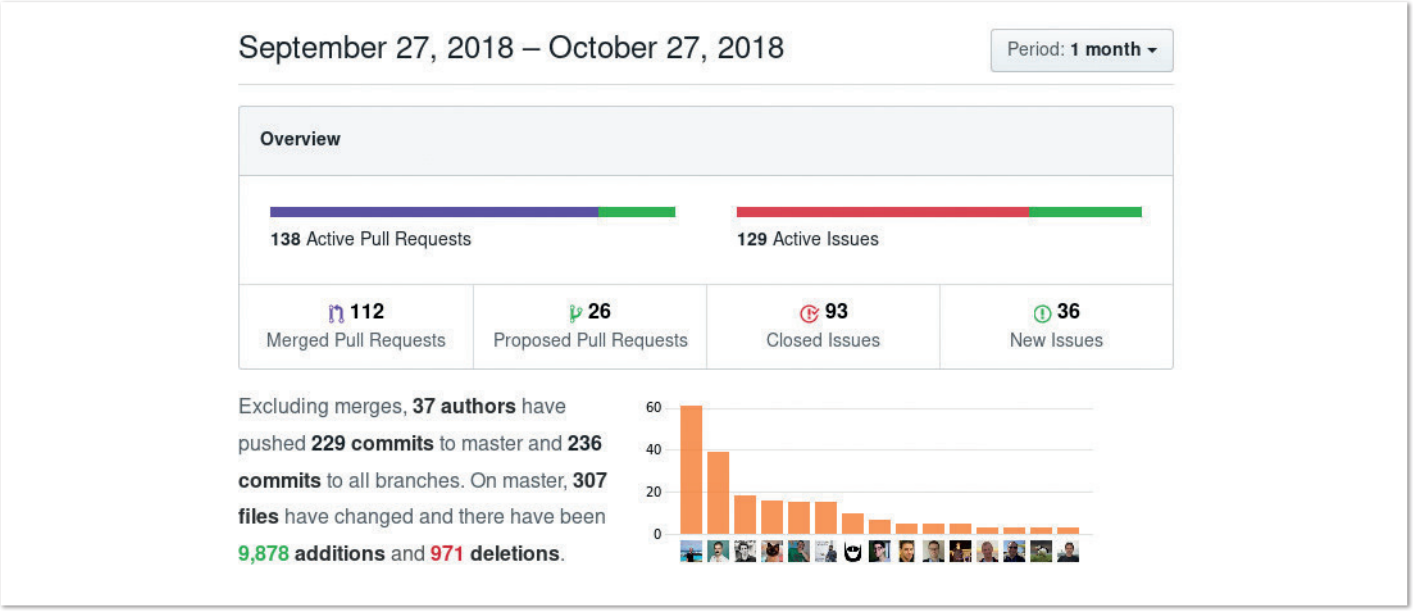


Abbildung 1: GitHub Pulse Oktober 2018

12.000 Stars auf GitHub und nach eigener Angabe mehr als 15.000 Downloads pro Woche. Samu ist überzeugt, dass ihm bei Problemen die Community hilfreich zur Seite stehen kann. Auch die Aktivität des Teams stimmt Samu positiv, dass Fehler schnell behoben werden.

Installation

JHipster bietet verschiedene Installationsarten an *[4]*. Die Dokumentation empfiehlt die lokale Installation, dennoch möchte Samu von Jennifer gerne mehr erfahren. Insbesondere JHipster Online klingt sehr spannend, daher schreibt er Jennifer eine Chat-Nachricht und fragt, wie er am besten starten sollte. Jennifer erklärt ihm, dass JHipster Online *[7]* selbst mit JHipster entwickelt wird. Damit kann man eine JHipster-Anwendung konfigurieren und als Zip-Archiv herunterladen *(siehe Abbildung 2)*.

Wenn man seinen GitHub- oder GitLab-Account mit JHipster Online verbindet, lassen sich die Projekte direkt in einem Git-Repository generieren. Zudem kann man verschiedene Entitätsmodelle verwalten, editieren und direkt in eine generierte Anwendung importieren. JHipster Online erzeugt dabei einen Pull Request auf dem Projekt, sodass man als Entwickler alle Änderungen reviewen kann. Demnach ist es sehr einfach, verschiedene Anwendungskonfigurationen und Datenmodelle zu testen.

Allerdings, so Jennifer, sollte man JHipster ebenfalls lokal installieren, da JHipster Online einige Funktionen wie beispielsweise Deployments bisher noch nicht unterstützt. Jennifer gibt Samu noch den Tipp, Docker und Docker Compose zu installieren, damit er die erzeugten Compose-Skripte für Datenbanken verwenden kann und diese nicht lokal auf seinem System installieren muss.

Module und Blueprints

Samu ist besorgt, dass JHipster es ihm schwer macht, Konfigurationen zu ändern, und Erweiterungen nicht einfach möglich sind. Jennifer kann ihn beruhigen. Da JHipster im Kern eine Spring-Boot-Anwendung ist, lassen sich alle Konfigurationen durch modifizierte Versionen überschreiben oder erweitern. Außerdem versucht JHipster, die einzelnen Technologien in der Standard-Konfiguration zu verwenden und Modifikationen nur vorzunehmen, wenn diese unbedingt notwendig sind *[13]*. Dadurch sind Anpassungen relativ einfach möglich. Im Zweifel genügt ein Blick in die Dokumentation des entsprechenden Projekts.

Falls Samu größere Anpassungen vornehmen möchte, kann er auch ein Modul *[10]* oder einen Blueprint *[11]* entwickeln und JHipster Online On-Premises installieren, sodass automatisch immer sein spezielles Modul zusätzlich bei der Generierung einer Anwendung verwendet wird. Ein Modul hat Zugriff auf die Konfiguration der JHipster-Anwendung und kann, wie zum Beispiel ein Subgenerator, Dateien anlegen. Über bestimmte Erweiterungspunkte (sogenannte „needles“) kann ein Modul bestehende Dateien erweitern oder modifizieren. So lassen sich beispielsweise neue Menüpunkte oder weitere Maven-Dependencies einfügen.

Im Unterschied zu einem Modul kann ein Blueprint existierende Dateien überschreiben oder löschen und ein eigenes Set von Dateien ausliefern. Damit ist es dem Autor eines Blueprints möglich, unter anderem Spring Boot durch ein anderes Framework zu ersetzen, die Spring-Security-Konfiguration an die Bedürfnisse des eigenen Unternehmens anzupassen oder Java durch Kotlin zu ersetzen *[14]*. Module sind seit JHipster 3 verfügbar, Blueprints erst seit Version 5. Daher ist die Auswahl an Blueprints noch nicht sehr groß. Momentan arbeitet das JHipster-Team an einem Blueprint, um „Vue.js“ als Client-Side-Framework verwenden zu können *[15]*.

```
npm install -g yo generator-jhipster
```

Listing 1

```
docker-compose -f src/main/docker/postgresql.yml up
-d 1
./gradlew -Pprod
```

Listing 2

Let's get started

Samu befolgt die Installationsanleitung *[4]* und installiert ein JDK (8), NodeJS in der aktuellen LTS-Version, Docker und Docker Compose. JHipster installiert er via NPM (ehemals Node Package Manager) *(siehe Listing 1)*.

Weitere Installationsarten wie mit Docker oder Homebrew stehen ebenfalls zur Verfügung. Falls man die benötigten Tools nicht auf seinem System installieren kann oder darf, kann man die JHipster Devbox verwenden, eine Vagrant-basierte virtualisierte Entwicklungsumgebung mit allen benötigten Werkzeugen *[20]*.

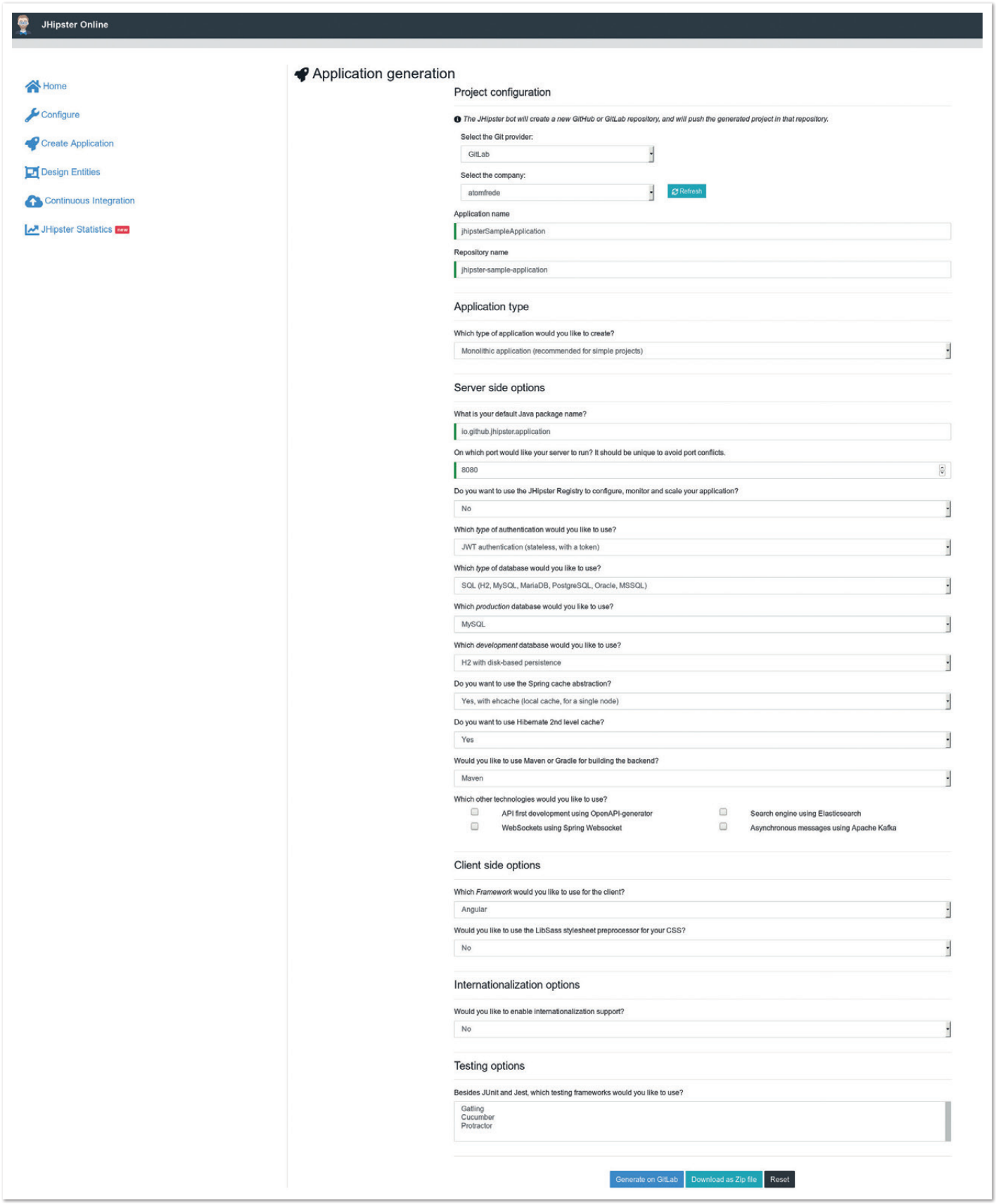


Abbildung 2: Anwendungskonfiguration in JHipster Online

Abbildung 3: Konfiguration der JHipster-Anwendung

Samu startet den Generator im Terminal durch Eingabe von „jhipster". Er muss eine Reihe von Fragen beantworten und entscheidet sich fast immer für den Vorgabewert *(siehe Abbildung 3)*. Statt MySQL als Datenbank verwendet er PostgreSQL, da er sich damit besser auskennt. Er entscheidet sich für Gradle statt Maven, da er bereits einige Gradle-Skripte geschrieben hat.

Nach der Generierung der App startet Samu die Anwendung mit „./gradlew". Nachdem der Java- und der Frontend-Build erfolgreich durchgelaufen sind, öffnet Samu die Anwendung („localhost:8080") und wird von der JHipster-Willkommensseite begrüßt *(siehe Abbildung 4)*.

Abbildung 4: Die JHipster-Willkommensseite

Da Samu bei seiner Arbeit als Scrum-Master oft von den traumhaften Turnaround-Zeiten der modernen JavaScript-Anwendungen gehört hat, probiert er das Ganze direkt aus. Er startet in einem Terminal das Java-Backend mit „./gradlew" und in einem weiteren Terminal das Frontend mit „npm run start". Samu fügt ein „div"-Tag in die Datei „src/main/webapp/app/home/home.component.html" ein und nach kurzer Zeit sind seine Änderungen im Browser zu sehen. Durch die Integration der Spring-Boot-DevTools muss auch bei Änderungen am Java-Code nicht die komplette Anwendung neu gestartet werden.

Momentan läuft seine Anwendung noch im sogenannten „Development Profile". Daher sind die JavaScript- und CSS-Dateien noch nicht optimiert und als Datenbank wird noch H2 statt PostgreSQL verwendet. Er folgt den Anweisungen der Dokumentation und startet die Anwendung im Production Profile *(siehe Listing 2)*. JHipster erzeugt für alle externen Services (wie Datenbanken) passende Docker-Compose-Skripte, damit er als Entwickler die Datenbank nicht lokal installieren muss.

Samu meldet sich mit dem hinterlegten Standard-Login „admin/admin" an und findet neben einer Benutzerverwaltung und einer Übersicht der erzeugten Metriken (Antwortzeit, Cachestatistiken, JVM-Metriken) auch die Möglichkeit, die Log-Level zur Laufzeit zu

ändern. Die Sprachumschaltung zwischen seinen gewählten Sprachen klappt reibungslos. Samu hat mit wenigen Befehlen eine Webanwendung mit Benutzerverwaltung, Security, REST-API und einem modernen Frontend generiert. Nun kann er mit dem Erstellen und Testen seines Datenmodells fortfahren.

Das Datenmodell

Samu möchte in einem ersten Schritt einen speziellen Workflow abbilden. Auch wenn alle Teams bereits relativ autonom arbeiten können, gibt es gewisse Abhängigkeiten zwischen einzelnen Aufgaben; so kann zum Beispiel das Team „Einkaufswagen" erst dann neue Produkt-Informationen verarbeiten, wenn das Team „Produkt" diese Daten auch bereitstellt. Diese Abhängigkeiten will Samu explizit darstellen und beispielsweise automatisch ein Ticket für das Einkaufswagen-Team erstellen, wenn das Produkt-Team sein Ticket abgeschlossen hat. Dazu hat er sich ein einfaches Modell überlegt *(siehe Abbildung 5)*.

Jedes Team hat einen Workspace, der mehrere Listen mit Aufgaben beinhaltet. Diese Listen sind entweder intern oder dienen als Out- beziehungsweise In-Box eines anderen Teams. Die Aufgaben möchte Samu später aus dem Ticketsystem in das neue Tool synchronisieren. Am Ende stellt er sich auch eine einfache Ansicht der Aufgaben für die Teams vor, ohne dass diese mit Informationen konfrontiert werden, die üblicherweise nur für das Projektmanagement relevant sind. Samu wirft noch einen kurzen Blick auf die Dokumentation *[5]* und legt seine erste Entität an *(siehe Listing 3)*.

```
jhipster entity team
```

Listing 3

```
application {
  config {
     applicationType monolith,
     baseName RijaFlow
     packageName com.gitlab.atomfrede.rijaflow,
     authenticationType jwt,
     prodDatabaseType postgresql,
     buildTool gradle,
     clientFramework angular,
     useSass true,
     enableTranslation true,
     nativeLanguage en,
     languages [en, de]
  }
}
```

Listing 4

Samu beantwortet alle Fragen gewissenhaft. Allerdings findet er es überhaupt nicht „hip", für jede Entität die ganzen Fragen beantworten zu müssen *(siehe Abbildung 6)*. Da auch schon Feierabend ist, setzt er seine Änderungen zurück und beschließt, am nächsten Tag Jennifer zu fragen, ob die CLI die einzige Möglichkeit ist, Entitäten anzulegen.

JDL

Jennifer schmunzelt am nächsten Morgen ein wenig und fragt, wieso er denn nicht das JDL Studio *[6]* verwende, um dort sein Entitätenmodell mit einer domänenspezifischen Sprache zu beschreiben. Als Bonus würde das Modell direkt im Browser als Klassen-Diagramm gerendert. Neben Entitäten könne man auch ganze Anwendungen und sogar Microservices mit JDL definieren und somit vollständig ohne CLI arbeiten *(siehe Listing 4)*.

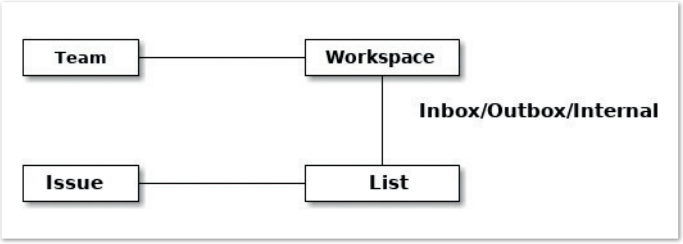


Abbildung 5: Das Datenmodell

```
The entity team is being created.


Generating field #1

? Do you want to add a field to your entity? Yes
? What is the name of your field? name
? What is the type of your field? String
? Do you want to add validation rules to your field? Yes
? Which validation rules do you want to add? Required

================ Team ================
Fields
name (String) required


Generating field #2

? Do you want to add a field to your entity? No

================ Team ================
Fields
name (String) required


Generating relationships to other entities

? Do you want to add a relationship to another entity? Yes
? What is the name of the other entity? workspace
? What is the name of the relationship? workspace
? What is the type of the relationship? one-to-one
? Is this entity the owner of the relationship? Yes
? What is the name of this relationship in the other entity? team
? When you display this relationship on client-side, which field from 'workspace' do you want to use? This
 field will be displayed as a String, so it cannot be a Blob name
? Do you want to add any validation rules to this relationship? No

================ Team ================
Fields
name (String) required

Relationships
workspace (Workspace) one-to-one
```

Abbildung 6: Konfiguration einer Entität mit dem Entity Generator

Samu liest noch ein wenig Dokumentation und schreibt sein JDL-Model. Er exportiert es und importiert es in seine Anwendung mit „jhipster import jdl jhipster-jdl.jh". Anschließend inspiziert Samu, was JHipster alles generiert hat. Neben Liquibase Changesets, um die Datenbank-Tabellen anzulegen, wurden alle nötigen Spring-Data-Repositories und Entitätsklassen mit entsprechenden JPA-/Hibernate-Annotationen erzeugt. Für jede Entität steht eine einfache CRUD-Oberfläche zur Verfügung, die über ein REST-API mit dem Backend kommuniziert *(siehe Abbildungen 7 bis 9 und Listing 5)*.

Alle diese Schritte hätten Samu sehr viel Zeit gekostet und für einen Prototyp hätte er vermutlich keine Datenbank-Migrationen eingerichtet, sodass er dies später hätte nachholen müssen. Samu pusht seinen aktuellen Stand ins Versionskontrollsystem

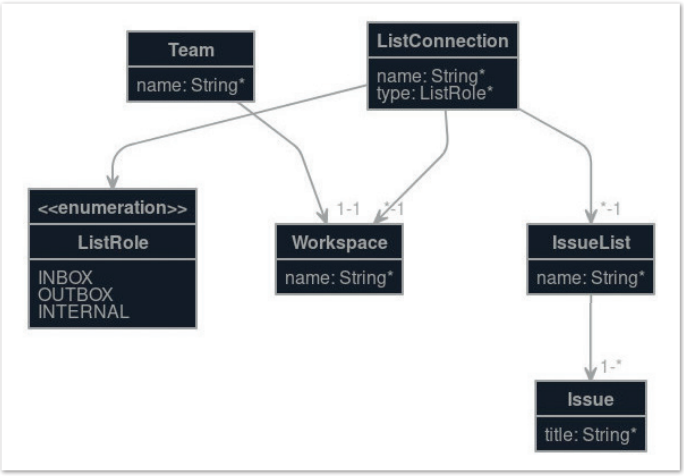


Abbildung 7: Samuś JDL-Modell

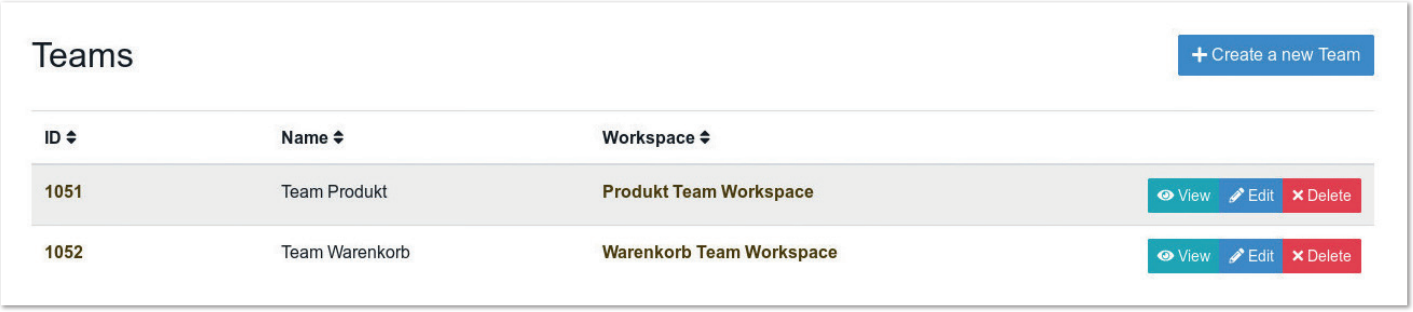


Abbildung 8: Generierte CRUD-Oberfläche (Übersichtstabelle)

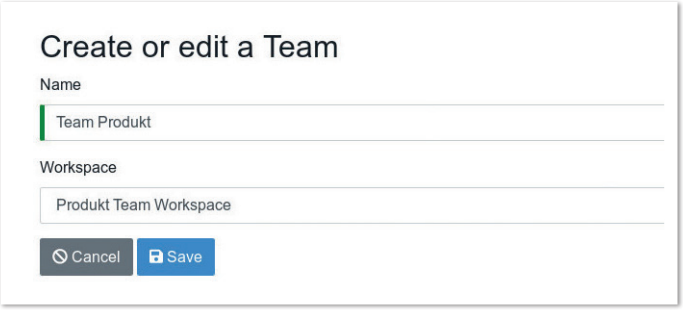


Abbildung 9. Generierte CRUD-Oberfläche (Detailansicht)

und macht erst mal Mittagspause, um sich später mit dem Betrieb in der Cloud sowie automatischen Builds zu beschäftigen.

Betrieb und CI/CD

Samu möchte sich nicht mit dem Betrieb von Servern und Datenbanken beschäftigen. Daher verwendet er Plattformen wie Heroku oder Cloud Foundry, um die Konfiguration von Zertifikaten, Datenbanken und Webservern zu erledigen. JHipster unterstützt neben Heroku und Cloud Foundry auch AWS, Kubernetes und Azure *[9]*. Samu entscheidet sich für Heroku, da er dort bereits ein Konto hat und sein lokales System alle Voraussetzungen erfüllt, um mit Heroku zu kommunizieren *[12]*.

```
entity Team {
  name String required minlength(3)
}
entity Workspace {
  name String required
}
entity IssueList {
  name String required
}
entity Issue {
  title String required
}
entity ListConnection {
  name String required
  type ListRole required
}
enum ListRole {
  INBOX, OUTBOX, INTERNAL
}
relationship OneToOne {
  Team{workspace(name)} to Workspace{team(name)}
}
relationship OneToMany {
  ListConnection to IssueList{listConnection(name)}
  ListConnection to Workspace{listConnection(name)}
}
relationship OneToMany {
  IssueList to Issue{issueList(name)}
}
paginate Team, Workspace, IssueList with pagination
paginate Issue infinite-scroll
```

Listing 5

```
Heroku configuration is starting
? Name to deploy as: rijaflow
? On which region do you want to deploy ? eu
? Which type of deployment do you want ? Git (compile on Heroku)

Using existing Git repository

Heroku CLI deployment plugin already installed

Creating Heroku application and setting up node environment
https://rijaflow.herokuapp.com/ | https://git.heroku.com/rijaflow.git

Provisioning addons
Created Database addon

Creating Heroku deployment files
   create src/main/resources/config/bootstrap-heroku.yml
   create src/main/resources/config/application-heroku.yml
   create Procfile
   create gradle/heroku.gradle
 conflict build.gradle
? Overwrite build.gradle? overwrite
    force build.gradle
```

Abbildung 10: Erzeugen einer Heroku-Konfiguration

Abbildung 11: Erfolgreicher Ablauf der erzeugten GitLab-CI-Pipeline

Mit „jhipster heroku" startet Samu den Heroku-Subgenerator. Er muss einen Namen für die Anwendung vergeben und auswählen, in welcher Region („us" oder „eu") die Anwendung betrieben werden soll. Er entscheidet sich für Europa. Um nicht immer die Anwendung auf seinem Rechner bauen zu müssen und die „.war"-Datei hochzuladen, entscheidet er sich dafür, die Anwendung von Heroku bauen zu lassen *(siehe Abbildung 10)*.

Der Subgenerator erstellt eine Anwendung auf Heroku, erweitert die Build Files („pom.xml" oder „build.gradle") um die nötigen Plugins, erstellt automatisch die passende Datenbank auf Heroku, konfiguriert die Datenbank-Verbindung der Anwendung und deployt den aktuellen Stand. Nach ein paar Minuten ist die Anwendung unter *„https://rijaflow.herokuapp.com/"* erreichbar.

Samu ist ein großer Automatisierungsfan. Natürlich möchte er daher auch seinen Prototyp automatisch bauen, testen und im besten Fall auch direkt in der Cloud deployen lassen. Da er GitLab nutzt, verwendet er den JHipster-CI/CD-Subgenerator *[8]*, um automatisch eine passende Konfiguration für sein Projekt zu erzeugen. Neben GitLab sind auch Jenkins-Pipelines, Travis und das recht junge Azure-Pipelines unterstützt. Nach dem Starten des CI/CD-Generators mit „jhipster ci-cd" muss er noch einige Details auswählen.

Samu entscheidet sich, seine Anwendung mit Sonar zu analysieren und von GitLab Aktualisierungen direkt auf Heroku zu installieren. Insgesamt erzeugt JHipster eine Pipeline, die aus mehreren Stages besteht. Der Deploy-Schritt ist dabei im Standard optional. Er committet und pusht seine Konfiguration und GitLab startet direkt die Pipeline mit der erzeugten Konfiguration *(siehe Abbildung 11)*.

Fazit und Ausblick

Samu konnte dank JHipster eine moderne Web-Anwendung innerhalb eines Tages erzeugen und in der Cloud betreiben. Er kann sich auf sein eigentliches Ziel konzentrieren und muss sich nun noch um die Synchronisation der Aufgaben aus dem bestehenden Ticketsystem kümmern sowie eine eigene UI für die Benutzer der Anwendung bauen. Alle Schnittstellen und Datenbank-Entitäten wurden von JHipster erzeugt und sind einsetzbar. Dank der JDL kann Samu sowohl die Konfiguration der Anwendung (etwa OAuth2 statt JWT) als auch sein Datenmodell leicht anpassen.

Neben dem obligatorischen Update auf Spring Boot 2.1 und vollem Java-11-Support wird es in JHipster 6 einige Erweiterungen der JDL geben, sodass diese mehr ins Zentrum rückt und die CLI sogar überflüssig werden könnte. Es ist geplant, neben Anwendungskonfiguration und Entitätsdefinition ebenfalls CI/CD und Deployment-Optionen via JDL zu definieren. Außerdem beabsichtigt das JHipster-Team, Aktualisierungen von JHipster zu vereinfachen, wenn viel eigener Code geschrieben wurde. Dadurch wird JHipster noch mehr zu einem Framework, das nicht nur das „Scaffolding" einer Anwendung vereinfacht, sondern den kompletten Lebenszyklus und Entwicklungsprozess abdeckt.

Quellen

[1] *https://jhipster.tech*
[2] *https://github.com/jhipster/generator-jhipster*
[3] *https://www.jhipster.tech/companies-using-jhipster*
[4] *https://www.jhipster.tech/installation*
[5] *https://www.jhipster.tech/creating-an-entity*
[6] *https://start.jhipster.tech/jdl-studio/*
[7] *https://start.jhipster.tech*
[8] *https://www.jhipster.tech/setting-up-ci*
[9] *https://www.jhipster.tech/production*
[10] *https://www.jhipster.tech/modules/creating-a-module*
[11] *https://www.jhipster.tech/modules/creating-a-blueprint*
[12] *https://www.jhipster.tech/heroku*
[13] *https://www.jhipster.tech/policies*
[14] *https://github.com/jhipster/jhipster-kotlin*
[15] *https://github.com/jhipster/jhipster-vuejs*
[16] *http://www.jhipster-book.com*
[17] *https://www.packtpub.com/application-development/full-stack-development-jhipster*
[18] *http://yeoman.io*
[19] *https://www.openhub.net/p/generator-jhipster*
[20] *https://github.com/jhipster/jhipster-devbox*

Frederik Hahne
frederik.hahne@wescale.com

Frederik Hahne arbeitet als Software-Entwickler bei der WPS Management GmbH in Paderborn an der offenen B2B-Integrationsplattform „wescale" (siehe „wescale.com"). Er ist Organisator der JUG Paderborn und hat in Paderborn die lokale Devoxx4Kids-Gruppe ins Leben gerufen. Seit dem Jahr 2015 ist er Mitglied des Java-Hipster-Core-Teams und betreut momentan hauptsächlich den Gradle-Build des Projekts. Er twittert unter „@atomfrede".

Anbindung der Hardware-Sicherheitsmodule mit Java

Christoph Pfeifer, Robotron Datenbank-Software GmbH

Hardware-Sicherheitsmodule (HSM) mit eigenen Java-Anwendungen zu verknüpfen kann über einen eigenen JCE-Provider erfolgen. Der Artikel zeigt, was dabei grundsätzlich umgesetzt werden muss und welche Probleme dabei auftreten können.

```
SUN version 1.8, SunRsaSign version 1.8, SunEC version 1.8,
SunJSSE version 1.8, SunJCE version 1.8, SunJGSS version 1.8,
SunSASL version 1.8, XMLDSig version 1.8, SunPCSC version 1.8,
SunMSCAPI version 1.8
```

Listing 1

Die JVM unterstützt bereits nach der Installation viele einfache kryptografische Funktionen. Diese werden durch bereits integrierte JCE-Provider („Java Cryptography Extension") angeboten. Die konkrete Implementierung ist für den Programmierer komplett transparent. Über statische Instanz-Methoden auf den jeweiligen Klassen (etwa „Keystore" oder „Signature") wird das entsprechende Objekt unter Nutzung der Provider zurückgegeben.

Der Vorteil für den Nutzer des Objekts ist, dass er sich nicht um die konkrete Instanziierung kümmern muss und unabhängig davon entwickeln kann. Dies ermöglicht es ebenfalls, die JCE-Provider einfach auszutauschen und damit die kryptografischen Funktionen grundlegend zu verändern. Mit den eigenen Instanzen im Rahmen eines eigenen JCE-Providers werden diese kryptografischen Funktionen dann komplett transparent für den Aufrufer in Verbindung mit HSMs durchgeführt.

Wie Provider funktionieren

Die zentrale Klasse für die Verwaltung der Provider ist die „Security"-Klasse. Darüber können neue Provider registriert und die bereits registrierten Provider abgefragt werden. Folgende Provider können zum Beispiel beim Start einer JVM bereits registriert sein *(siehe Listing 1)*.

Ruft man nun die Instanz-Methode für eine kryptografische Funktion auf, werden die Provider der Reihenfolge nach auf entsprechend unterstützte Klassen durchsucht. Dabei enthält jeder eine Zuordnung von Services zu konkreten Klassen. Ein Service definiert sich als einfacher Text, der allerdings einer konkreten Bildungsvorschrift folgen muss. Dabei steht zu Beginn immer der Name der Klasse, die überschrieben werden soll, dann folgt ein Punkt und danach wird eine Kennung (wie der Algorithmus) vergeben. Welche Standardnamen es gibt, kann man in der Dokumentation bei Oracle *[1]* nachlesen. Insbesondere wenn der Provider für den Programmierer transparent zur Verfügung stehen soll, sollten die Standardnamen genutzt werden.

Der Service „KeyPairGenerator.EC" beschreibt zum Beispiel einen „KeyPairGenerator", der den EC-Algorithmus unterstützt. Dem Service muss dann eine Klasse zugeordnet sein, die die entsprechende „ServiceProviderInterface"-Klasse überschreibt. Im Fall des „KeyPairGenerator" ist das dann die „KeyPairGeneratorSpi"-Klasse.

Sind am Ende alle Services im eigenen Provider hinterlegt, ist der Provider noch über die „Security"-Klasse zu registrieren.

```
KeyPairGeneratorSpi
SignatureSpi
SecureRandomSpi
KeyAgreementSpi
CipherSpi
```

Listing 2

Dabei ist darauf zu achten, an welcher Position der neue Provider eingefügt wird. Soll der Provider mit der Oracle JVM genutzt werden, muss dieser zusätzlich noch mit einem von Oracle signierten Zertifikat unterschrieben werden. In der OpenJDK-JVM ist dies nicht notwendig.

Funktionen von Hardware-Sicherheits-Modulen in Java

Die Verwaltung von Schlüsseln in Java ist über Objekte der Klasse „Keystore" möglich. In der eigenen Erweiterung der Klasse „KeystoreSpi" werden dann die Methoden so überschrieben, dass sie mit den HSM arbeiten. Außerdem sind die „ServiceProviderInterfaces" folgender Klassen noch zu überschreiben, sofern diese jeweilige Funktion auf einem HSM durchgeführt werden soll *(siehe Listing 2)*.

Sollen zusätzlich noch TLS-Verbindungen mithilfe von HSM aufgebaut werden, sind noch folgende Klassen zu überschreiben *(siehe Listing 3)*. Sind all diese Klassen überschrieben und als Service im Provider registriert, arbeitet dieser nun mit der eigenen Anbindung an die HSM.

Mandantentrennung

Zur Trennung unterschiedlicher Mandanten in HSM werden häufig Entitäten genutzt. Eine Entität entspricht in etwa einem Ordner, in dem Schlüssel abgelegt werden können. Nutzer, die sich auf dem HSM anmelden, können dann nur die Schlüssel in ihren freigegebenen Entitäten sehen. Beim Zugriff auf die Schlüssel muss dann nicht nur der Name des Schlüssels angegeben werden, sondern auch die zugehörige Entität. Dies stellt für die Verwendung von Schlüsseln auf HSM in Java ein kleines Problem dar.

„Keystore" und „KeyManager"-Objekte verwalten Schlüssel nur mit einem Alias. Die Angabe einer Entität ist nicht vorgesehen. Für dieses Problem gibt es allerdings zwei offensichtliche Lösungsvarianten. Die erste, einfache Möglichkeit besteht darin, Schlüsselname und Entität in den Alias zu kodieren. Damit können Schlüssel-Objekte („PrivateKey") über den normalen Weg erzeugt werden. Die zweite Variante ist, eine eigene Schlüsselklasse zu erzeugen, die das Interface „Key" beziehungsweise „PrivateKey" implementiert. Im Konstruktor muss dann die Entität mit angegeben werden. Wird dann während der Operation mit einem HSM dieser Schlüssel genutzt, muss das Objekt hart auf die eigene Schlüsselklasse gecastet werden, um die Entität abrufen zu können.

Utimaco-eID-Schnittstelle

In den ausgelieferten Bibliotheken von Utimaco ist bereits ein JCE-Provider enthalten, der für einfache Anforderungen leicht genutzt werden kann. Sollen allerdings verschiedene HSM-Schnittstellen bedient werden, muss ein eigener Provider implementiert sein, der dann entweder den ausgelieferten Provider nutzt oder direkt die eID-Schnittstelle zur Kommunikation mit dem HSM nutzt.

```
KeyManagerFactorySpi
TrustManagerFactorySpi
```

Listing 3

Worldline-JSS-Schnittstelle

Worldline liefert ebenfalls einen eigenen JCE-Provider in ihren Bibliotheken mit aus. Der „JSSJCAProvider" kann genutzt werden, um einfach Zugriff auf das HSM zu gewährleisten. Der Vorteil an dieser Schnittstelle ist die Konfiguration über eine eigene Konfigurationsdatei. Der Programmierer muss sich keine Gedanken über Ausfallsicherheit machen, da dies direkt über die Konfigurationsdatei und die Bibliothek für den Zugriff auf das HSM erledigt wird.

Der Nachteil dieser Schnittstelle liegt bei der Generierung neuer Schlüssel. Diese müssen teilweise direkt bei Worldline erstellt und ins HSM importiert werden. Dies widerspricht dem Grundgedanken eines HSM, dass die Schlüssel nie das HSM verlassen dürfen.

Zu beachten

Insbesondere in moderner Kryptografie werden unterschiedliche Kurven-Parameter für Kryptografie basierend auf elliptischen Kurven genutzt. Die Standard-JVM-Provider unterstützen nur einen einfachen Satz an Kurven-Parametern. Will man beispielsweise Brainpool-Kurven-Parameter einsetzen, weil das HSM diese nutzt, so sind weitere Anpassungen oder Bibliotheken (wie Bouncy-Castle) notwendig.

Ebenfalls ist in der JVM bis Java 8 u151 standardmäßig (aufgrund eines US-Exportgesetzes) starke Kryptografie deaktiviert. Diese muss erst durch die Installation der Erweiterung „Unlimited Strength Cryptography" aktiviert werden. Ab Java 8 u151 kann diese erweiterte Kryptografie über das Property „crypto.policy" erfolgen. Mit Java 8 u162 ist dies dann standardmäßig aktiviert.

Quellen

[1] *https://docs.oracle.com/javase/8/docs/technotes/guides/security/StandardNames.html*

Christoph Pfeifer
rds@robotron.de

Christoph Pfeifer studierte an der TU Dresden Informatik mit dem Schwerpunkt Datensicherheit. Seit dem Jahr 2014 arbeitet er als Software-Entwickler bei der Robotron Datenbank-Software GmbH im Bereich der Energiewirtschaft. Dort ist er für die Anbindung von sogenannten „Smart-Meter-Gateways" an bestehende Systeme zuständig.

kubernetes

Kubernetes Basics – wie die App in die Cloud kommt

Christian Kühn, synyx GmbH

Kubernetes ist eines der bekanntesten und am stärksten wachsenden Frameworks zur Container-Orchestrierung. Einige der meistbeworbenen Features sind die einfache Automatisierbarkeit, Selbstheilung, Load Balancing, Skalierbarkeit und die Möglichkeit, Infrastruktur als Code zu beschreiben und zu versionieren. Viele Manager fordern daraufhin die Migration auf eine Kubernetes-Plattform, viele Entwickler stellen sich allerdings die Fragen: Brauchen wir das überhaupt? Wie kommt unsere App da jetzt rein? Wie stelle ich sicher, dass die nötigen Abhängigkeiten und Ressourcen für meine Applikation bereitstehen?

Kubernetes ist im Jahr 2015 aus einem Google-Projekt hervorgegangen und wird als Open-Source-Projekt unter der Cloud Native Computing Foundation (CNCF) entwickelt. Als Plattform ist Kubernetes modular aufgebaut und über ein Plug-in-System einfach erweiterbar. Sie erlaubt das Deployment containerbasierter Applikationen und kann deren kompletten Lifecycle abbilden und steuern.

Die Plattform bietet ein einfach konfigurierbares Monitoring, mit dem sich sämtliche Komponenten und Apps überwachen und bei gewissen Kriterien automatisch neu starten lassen. Zusätzlich können Cluster-Komponenten und die angebotenen Dienste automatisch skalieren. Anwendungen und Ressourcen lassen sich innerhalb eines Clusters in gekapselten Namespaces logisch voneinander trennen und gegen unbefugte Zugriffe absichern. Ein rollenbasiertes Autorisierungssystem steuert entsprechende Berechtigungen für die verschiedenen Namespaces und Komponenten. Applikationen werden mithilfe einer Container-Runtime betrieben (die nachstehend beschriebenen Beispiele laufen auf Basis von Docker).

Steuerung

Kubernetes lässt sich über einen API-Server steuern. Dieser bietet eine REST-Schnittstelle an, über die man den Cluster und alle zu betreibenden Dienste steuern und konfigurieren kann. Der API-Server bietet eine OpenAPI-kompatible Beschreibung mit einem Swagger-Endpoint *(siehe „https://swagger.io")* an. In der Regel wird über eine Definition im YAML- oder JSON-Format der gewünschte Zustand beschrieben.

Kubernetes folgt hier einem komplett deklarativen Ansatz. Der Cluster beziehungsweise die Steuer-Komponenten kümmern sich darum, dass dieser definierte Zustand erreicht und erhalten wird. Als Best Practice hat sich hier etabliert, entsprechende Definitionen zu versionieren und innerhalb von CI/CD-Pipelines zu verwenden, um sicherzustellen, dass man jederzeit mit wenig Aufwand jeden gewünschten Zustand in der Historie (wieder-)herstellen kann.

Beispiel-Ressourcen und Übungsumfeld

Die nachfolgend verwendeten Beispiel-Ressourcen kann man auch im GitHub-Repository unter *„https://github.com/cy4n/java-aktuell-kubernetes"* finden und nachstellen. Zum ersten Kennenlernen von Kubernetes empfiehlt es sich, ein lokales Kubernetes-Demosystem aufzusetzen. Die folgenden Beispiele wurden mithilfe von Minikube *(siehe „https://github.com/kubernetes/minikube")*, das als virtuelle Maschine auf dem eigenen Entwickler-Rechner läuft, aufgebaut. Zur Kommunikation mit dem API-Server wird kubectl, das offizielle Kubernetes-Command-Line-Tool, verwendet. Hier eine kurze Übersicht über die wichtigsten kubectl-Befehle:

- **kubectl get <typ>**
  Ressource(n) anzeigen
- **kubectl apply -f <Dateiname>**
  Definition aus „yaml"-/„json"-Datei anwenden
- **kubectl delete <typ> <name>**
  Ressource entfernen
- **kubectl logs <name>**
  Logs („stdout") eines Pod anzeigen

- **kubectl describe <typ> <name>**
  Details einer Ressource anzeigen
- **kubectl exec <name>**
  Befehle innerhalb eines Pod ausführen

Kubernetes-Basis-Ressource für die Applikation

Die kleinste deploybare Einheit in Kubernetes ist ein „pod". Sie enthält in der Regel einen Applikations-Container und repräsentiert einen Prozess beziehungsweise eine Instanz einer Anwendung. Die Definition eines Pod enthält unter anderem Informationen über das zu verwendende „container"-Image, den Port, den dieser Container anbietet und (Speicher-) Volumes, die der Applikation zur Verfügung gestellt werden sollen. Zusätzlich können für diesen Container „compute"-Ressourcen definiert werden, etwa wie viel CPU und RAM mindestens zur Verfügung stehen sollen, aber auch wieviel maximal genutzt werden darf. Eine Definition eines Pod könnte wie in *Abbildung 1* aussehen und mithilfe von kubectl angewendet werden *(siehe Abbildung 2)*.

```yaml
1  apiVersion: v1
2  kind: Pod
3  metadata:
4    name: hello
5  spec:
6    containers:
7    - name: hello
8      image: cy4n/hello:0.0.4
9      ports:
10     - containerPort: 8080
```

Abbildung 1: Ressourcen-Definition „pod.yml"

```
~/demo> kubectl apply -f pod.yml
pod/hello created
~/demo> kubectl get pods -o wide
NAME      READY     STATUS    RESTARTS   AGE       IP           NODE
hello     1/1       Running   0          8s        172.17.0.6   minikube
```

Abbildung 2: Anwenden der Definition und Anzeige der laufenden Pods

```
~/demo> kubectl exec hello -- curl -s localhost:8080/host
hello
```

Abbildung 3: Ausführung eines Befehls auf der Shell des Pod

Kubectl dient hier nicht nur zum Schreiben beziehungsweise Senden neuer Definitionen an Kubernetes, sondern auch zum Anzeigen des aktuellen Zustands. In diesem Fall werden die laufenden Pods mit einigen Zusatz-Informationen angezeigt, so zum Beispiel auf welchem Cluster-Node der Container mit der Applikation gestartet und welche IP-Adresse zugewiesen wurde.

Die Anwendung in diesem Beispiel ist eine Spring-Boot-App, die ein REST-API mit einem Endpoint zur Anzeige des Server-Hosts anbietet. Da innerhalb von Kubernetes jeder Pod seinen eigenen Namen als Hostname erkennt, wird entsprechend hier immer der Pod-Name angezeigt. Da ein einfacher Pod noch nicht von außerhalb von Kubernetes angesprochen werden kann, kann man mithilfe von „kubectl exec" einen Test mit „curl", einem „commandLine"-HTTP-Client, durchführen *(siehe Abbildung 3)*.

Deployment-Applikation mit mehr Zustands-Informationen

Um die geplanten Anwendungen mit etwas mehr Zusammenhang zu versorgen, empfiehlt es sich, anstelle einzelner Pods ein „deployment" anzulegen *(siehe Abbildung 4)*. Es enthält unter anderem ein „pod"-Template (Zeile 12), das die Basis-Einstellungen eines Pod definiert. Dieses Template erweitert den Pod um ein „label", über das

```yaml
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: hello
5    labels:
6      app: hello
7  spec:
8    replicas: 2
9    selector:
10     matchLabels:
11       app: hello
12   template:
13     metadata:
14       labels:
15         app: hello
16     spec:
17       containers:
18       - name: hello
19         image: cy4n/hello:0.0.4
20         ports:
21         - containerPort: 8080
```

Abbildung 4: Ressourcen-Definition „deployment.yml"

```
~/demo> kubectl get deployment,replicaset,pod
NAME                           DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
deployment.extensions/hello    2         2         2            2           1m

NAME                                      DESIRED   CURRENT   READY   AGE
replicaset.extensions/hello-746844fd6c    2         2         2       1m

NAME                           READY     STATUS    RESTARTS   AGE
pod/hello-746844fd6c-452pr     1/1       Running   0          1m
pod/hello-746844fd6c-vdglk     1/1       Running   0          1m
```

Abbildung 5: Anzeigen des laufenden Deployments, der „replica"-Sets und der Pods

```
1   kind: Service
2   apiVersion: v1
3   metadata:
4     name: hello-service
5   spec:
6     selector:
7       app: hello
8     type: NodePort
9     ports:
10    - port: 8080
```

Abbildung 6: Ressourcen-Definition „service.yml"

```
1   apiVersion: extensions/v1beta1
2   kind: Ingress
3   metadata:
4     name: hello-ingress
5   spec:
6     rules:
7     - host: example.com
8       http:
9         paths:
10        - path: /
11          backend:
12            serviceName: hello-service
13            servicePort: 8080
```

Abbildung 7: Ressourcen-Definition „ingress.yml"

man später alle Instanzen dieser Anwendung selektieren kann. Zusätzlich sind weitere Zustands-Informationen enthalten, etwa die Anzahl der Instanzen, mit denen diese Anwendung betrieben werden soll (in Zeile 8 als „replicas" bezeichnet.

Kubernetes generiert bei einem „apply" mehrere Ressourcen, um diesen Zustand zu beschreiben *(siehe Abbildung 5)*. Zusätzlich zum erwarteten Deployment wird ein „replica"-Set generiert, das für den Lifecycle der Pods verantwortlich ist, die zu diesem Stand des Deployments gehören. Um die Zugehörigkeit darzustellen, wird der Name des „replica"-Sets aus dem Namen des Deployments (blau) und einer generierten Id (grün) aufgebaut. Die Pods wiederum beziehen ihren Namen aus dem Namen des „replica"-Sets mit einer weiteren Id (rot), die innerhalb der Pods eines „replica"-Sets eindeutig ist. Bei jeder Änderung des Pod-Templates wird ein neues „replica"-Set erstellt, das den neuen Stand abbildet und neue Pods startet, die der jeweils gültigen Template-Definition genügen.

Fester Applikations-Endpunkt

Da Pods beim Start dynamisch eine IP-Adresse zugewiesen bekommen, die auch nach deren Beenden wieder neu vergeben wird, ist es nicht sinnvoll möglich, eine Anwendung rein auf Pod-Ebene bereitzustellen. Zu diesem Zweck bietet Kubernetes die Möglichkeit, einen „service" zu definieren.

Ein Service stellt einen Endpunkt als Abstraktion der Anwendung dar, der mit einer festen IP-Adresse versehen ist und per DNS auch von anderen Anwendungen innerhalb von Kubernetes gefunden und angebunden werden kann. In einem Service wird mit einem „selector" (siehe *Abbildung 6*, Zeilen 6 und 7) zugewiesen, für welche Pods dieser Dienst angeboten wird.

Im gezeigten Beispiel in *Abbildung 6* werden alle Pods selektiert, denen das Label „app:hello" zugewiesen wurde (siehe *Abbildung 4*, Zeile 15). Kubernetes registriert im Service allerdings nur Pods, die im Zustand „READY" sind, und leitet Requests für diesen Service nur an diese Pods weiter. Diese aktuell eingebundenen und bereiten Pods kann man sich mit dem Befehl „kubectl get endpoints <service-name>" anzeigen lassen.

Mithilfe der abgebildeten „service"-Definition ist es bereits möglich, den Dienst auch schon von außerhalb des Kubernetes-Clusters zu erreichen. Der Typ „NodePort" richtet auf dem jeweiligen Worker im Cluster eine (TCP-)Port-Weiterleitung ein.

Load Balancer und Reverse-Proxy

In der Regel wird man keine Anwendung direkt bereitstellen wollen, sondern noch einen Load Balancer bzw. Proxy vorschalten. Im Falle der Web-Anwendung in der vorliegenden Demo wird ein HTTP-Proxy verwendet. Minikube bringt hier die Möglichkeit mit, einen Ingress-Controller auf Basis des bekannten Web-Servers NGINX *(siehe „https://github.com/kubernetes/ingress-nginx")* zu installieren. Dieser Server kann von allen Anwendungen in Minikube benutzt werden. Hierzu definiert man eine „ingress"-Ressource *(siehe Abbildung 7)*.

```
17        containers:
18        - name: hello
19          image: cy4n/hello:0.0.4
20          ports:
21          - containerPort: 8080
22          livenessProbe:
23            httpGet:
24              path: /actuator/health
25              port: 8080
26            initialDelaySeconds: 20
27          readinessProbe:
28            httpGet:
29              path: /actuator/health
30              port: 8080
31            initialDelaySeconds: 20
```

Abbildung 8: Ressourcen-Definition „deployment-health.yml" (Ausschnitt)

```
1   apiVersion: v1
2   kind: ConfigMap
3   metadata:
4     name: applicationyaml
5   data:
6     application.yml: |
7       management:
8         endpoints:
9           web:
10            exposure:
11              include: health,env
12      spring:
13        security:
14          user:
15            roles: ACTUATOR
```

Abbildung 9: Ressourcen-Definition „config.yml"

Die vorliegende Definition leitet HTTP-Anfragen, die die Request-URL *„example.com"* enthalten, an den Service „hello-service" auf Port 8080 weiter. Dieser Mechanismus dient dazu, Anfragen von außen weiterzuleiten. Im Ingress ist es zum Beispiel möglich, ein SSL-Zertifikat zu referenzieren, um den Dienst per HTTPS anzubieten. Wie in herkömmlichen „ReverseProxy"-Deployments ist es meist einfacher, ein Zertifikat und einen Key zur Verschlüsselung im Proxy zu definieren als in der Anwendung selbst, wo man zunächst (im Falle von Java) einen Keystore erstellen und diesen dann einbinden müsste.

Um das gezeigte Beispiel im Browser auszuprobieren, muss zunächst noch ein DNS-Eintrag angelegt werden, der die URL *„example.com"* auf die IP der Minikube-VM verweist. Über „curl" kann man die Anfrage auch ohne DNS-Mapping verschicken: „curl -H 'Host: example.com' http://192.168.99.100/host", wobei als IP-Adresse die IP der Minikube-VM einzutragen ist, hier der Standard-Wert. Bis hierhin hat man ein einfaches Deployment einer Web-Applikation erstellt, die verschiedenen Instanzen zu einer Abstraktion – einem Service – zusammengefasst und mit einem Ingress zur Verfügung gestellt.

Erweiterung 1: Readiness- und Liveness-Checks

Im Beispiel eines Pod hat *Abbildung 2* gezeigt, dass ein Pod sofort nach seinem Start in den Zustand „READY" geht. Der Pod ist laut Anzeige acht Sekunden alt, es sind bereits 1/1 Container bereit. In einer Spring-Boot-Anwendung ist das so nicht immer möglich. Hier sind Startzeiten von fünfzehn Sekunden und länger keine Seltenheit. An dieser Stelle kommen zwei neue Checks ins Spiel, ein Readiness- und ein Liveness-Check *(siehe Abbildung 8)*.

Der Liveness-Check prüft, ob ein neuer Pod beziehungsweise die enthaltene Anwendung initial gestartet wurde. Falls dies bis zu einer definierbaren Zeit nicht erfolgt, wird dieser Pod erneut gestartet. Ab Zeile 22 wurde dazu die bestehende „deployment.yml" um zwei Probes erweitert.

Sobald sich der Pod einmal im Zustand „RUNNING" befindet, wird infolge eines fehlerhaften Liveness-Checks der Pod als ungesund angesehen, abgeschaltet und komplett gelöscht. An seiner Stelle wird dann gleichzeitig ein neuer Pod gestartet.

Anhand des Readiness-Checks wird entschieden, ob ein Pod als Endpunkt für einen Service registriert wird. Im Unterschied zum Liveness-Check kann es sein, dass ein Pod mehrfach den Status

```
29              path: /actuator/health
30              port: 8080
31            initialDelaySeconds: 20
32          volumeMounts:
33          - name: applicationyaml-volume
34            mountPath: /app/config
35            readOnly: true
36        volumes:
37        - name: applicationyaml-volume
38          configMap:
39            name: applicationyaml
```

Abbildung 10: Ressourcen-Definition „deployment-volume.yml"

```
32          volumeMounts:
33          - name: applicationyaml-volume
34            mountPath: /app/config
35            readOnly: true
36          env:
37          - name: SPRING_SECURITY_USER_NAME
38            valueFrom:
39              secretKeyRef:
40                name: actuator-user
41                key: user
42          - name: SPRING_SECURITY_USER_PASSWORD
43            valueFrom:
44              secretKeyRef:
45                name: actuator-user
46                key: password
47        volumes:
```

Abbildung 11: Ressourcen-Definition „deployment-env.yaml"

```
1   apiVersion: v1
2   kind: Secret
3   metadata:
4     name: actuator-user
5   data:
6     user: amF2YQ==
7     password: YWt0dWVsbA==
```

Abbildung 12: Ressourcen-Definition „secret.yml"

wechselt. Die sogenannten „Probes" werten hier den HTTP-Status aus; als „gut" wird hier ein Status ab 200 und unter 400 angenommen, als „schlecht" beziehungsweise „ungesund" ein Status von 400 und höher.

In Spring Boot könnte man für den Readiness-Check etwa den „health"-Actuator *(siehe „https://spring.io/guides/gs/actuator-service")* benutzen. Hier wird der Service als „DOWN" (Status 503) angezeigt, wenn beispielsweise eine Abhängigkeit (wie eine Datenbank) nicht verfügbar ist. In diesem Fall könnte man sich entscheiden, diesem Pod keine Requests zuzuführen. Durch die Nutzung dieses Actuator für den Readiness-Check würde dann im Status „DOWN" der Pod als Endpunkt im Service entfernt, der Service würde also keine Requests an diesen Pod leiten.

Für den Liveness-Check eignet sich der „health"-Actuator nur bedingt. Im genannten Fall könnte ein Ausfall einer Abhängigkeit des Pod etwa in einem Drittsystem dazu führen, dass ein Pod komplett gelöscht und ein neuer als Ersatz gestartet wird, was möglicherweise gar nicht zum Erfolg führt. Der Entscheidung, wie man die Checks verwendet, liegen hier natürlich die individuelle System-Architektur der Anwendung und deren Abhängigkeiten zugrunde.

Erweiterung 2: Konfigurationsdateien

In realistischen Deployment-Szenarien lässt es sich nicht umgehen, die gleiche Anwendung mit verschiedenen Konfigurationen zu starten, etwa um zwischen Entwicklungs- und Produktions-Systemen zu unterscheiden und je nach Gebrauch unterschiedliche Parameter zu hinterlegen. Am Beispiel der Spring Boot Externalized Configuration *(siehe „https://docs.spring.io/spring-boot/docs/current/reference/html/boot-features-external-config.html")* lässt sich anschaulich die Verwendung von Konfigurationsdateien in Kubernetes lernen. Im Beispiel wird in Kubernetes eine „configMap" angelegt *(siehe Abbildung 9)*. Der Inhalt dieser Konfiguration (Zeilen 8 bis 16) wird beim Start jedes Pod als Datei mit dem Namen „application.yml" (Zeile 7) auf ein virtuelles Dateisystem innerhalb des Pod geschrieben.

Dazu wird innerhalb des Pod-Templates im Deployment ein „volume" angelegt und im Container-Dateisystem auf Höhe der „.jar"-Datei im Unterordner „config" gemountet (siehe *Abbildung 10*, Zeile 32 als Erweiterung des bekannten Deployments). Dieser Mountpoint wurdest hier nach Spring-Boot-Spezifikation *(siehe „https://docs.spring.io/spring-boot/docs/current/reference/html/boot-features-external-config.html")* ausgesucht. Die Beispiel-Konfiguration würde in Spring Boot den Actuator „/env" aktivieren. Zusätzlich wird mithilfe von Spring Security einem eingeloggten Benutzer die Rolle „ACTUATOR" zugewiesen (mehr dazu in Erweiterung 3).

Erweiterung 3: Umgebungsvariablen

Umgebungsvariablen bieten eine weitere Möglichkeit, die Anwendung zu erweitern. Im Beispiel sollen so ein Benutzername und ein Passwort für einen Benutzer abgelegt werden, auch bei der Spring Boot Externalized Configuration, die es ermöglicht, der genannten „application.yml" auch einzelne Properties als Environment-Variable zu definieren, in diesem Falle ist das der Spring-Security-Parameter „SPRING_SECURITY_USER_NAME und SPRING_SECURITY_USER_NAME".

Die Beispiel-Anwendung ist mithilfe von Spring Security so konfiguriert, dass der oben definierte Actuator-Endpunkt „/env" abgesichert wird und nur von einem eingeloggten Benutzer abgerufen werden darf, der die Rolle „ACTUATOR" besitzt. Diese Rolle ist ja bereits in der Konfigurationsdatei in Erweiterung 2 definiert. Um für einen Pod neue Umgebungsvariablen zu definieren, wird auch hier das Pod-Template im Deployment erweitert (siehe *Abbildung 11*, Zeilen 36 bis 46).

Die hier verwendeten Zugangsdaten müssen in Kubernetes als „secret" definiert sein. Hier wird die jeweilige Umgebungsvariable, etwa „SPRING_SECURITY_USER_NAME", die im „deployment" definiert wurde, mit dem entsprechenden Wert versehen *(siehe Abbildung 12)*.

Secrets werden in Kubernetes in „base64" codiert. Dies ist natürlich kein Sicherheits-Feature, sondern dient lediglich dazu, Probleme mit Sonder- und Steuerzeichen zu umgehen. Mit diesen Erweiterungen wurde die Anwendung mit einem Health-Check gestartet, sodass Kubernetes automatisch erkennt, ob ein Problem mit einer Instanz der Anwendung vorliegt und mit dem Löschen der defekten Instanz und dem Neustart weiterer Instanzen reagieren kann. Außerdem wurde die Anwendung erweitert, sodass laufzeitspezifische Konfigurationen übergeben werden können, die nicht mit im Container oder der Anwendung paketiert werden müssen.

Christian Kühn
kuehn@synyx.de

Christian Kühn ist System-Entwickler bei synyx. Zu seinen Aufgabenbereichen zählen Software-Entwicklung, Beratung und System-Administration in agilen, cross-funktionalen Teams. Seine Erfahrung beinhaltet Einflüsse aus mehr als zehn Jahren im Bereich „Operations" und mehreren Jahren in der Software-Entwicklung. Er ist Mitorganisator des DevOps-Meetups Karlsruhe und hält Vorträge zu Software-Entwicklung, Cloud und DevOps.

```java
final Programmer programmer = new Programmer()
     .with(Skill.JAVA)
     .with(Skill.SQL)
     .with(Skill.JSP)
     .with(Skill.TOMCAT)
     .with(Locale.GERMAN, Proficiency.FLUENT);
if (programmer.isSeekingForJob()) {
     final JobPosting jobPosting = programmer
             .research("https://www.gus-group.com/karriere/").stream()
         .filter(JobPosting::isPermanentPosition)
         .filter(JobPosting::hasFlexibleWorkingHours)
         .filter(posting -> posting.getDaysOff() == 30)
         .filter(posting -> posting.checkRequirements(programmer.getSkills()))
         .findFirst().orElse(null);
     if (jobPosting != null) {
         programmer.apply("bewerbung@gus-group.com", jobPosting);
     }
}
```

Warum man mit Visionen nicht zum Arzt gehen sollte – der Weg zur strategischen Produktentwicklung

Jörg Domann, Ervolution

Man stelle sich vor, die Entwicklung des Produkts läuft perfekt in Scope, Time und Budget, aber es wird trotzdem kein Erfolg. In vielen Fällen wird es daran liegen, dass die Product Vision nicht existent, fehlerhaft, veraltet oder unverstanden ist. Dieser Artikel zeigt einen Weg, die richtigen lang-, mittel- und kurzfristigen Ziele herzuleiten, wie sie aufeinander aufbauen und dauerhaft aktuell gehalten werden. Unterstützt wird dies durch Beispiele aus der imaginären Stromvolt GmbH.

Es liegt in unserer Natur, dass wir uns mit unseren alltäglichen Aufgaben allzu gern nur an den operativen Zielen orientieren. Zu selten wird hinterfragt, welche mittel- und langfristigen Ziele zu erreichen sind und wie die operativen Aufgaben und Ziele mit ihnen verknüpft sind. Für strukturiertes, zielorientiertes Denken und eine einheitliche Kommunikation hat es sich daher bewährt, den Horizont in lang-, mittel- und kurzfristige Ziele zu untergliedern. In der Produktentwicklung mit Ervolution werden hierfür folgende Stufen verwendet und in seiner Gesamtheit als „Product Vision" bezeichnet *(siehe Abbildung 1)*.

Bevor auf die einzelnen Stufen genauer eingegangen wird, sei an dieser Stelle noch auf ein paar Grundlagen im Umgang mit der Pyramide hingewiesen:

Referentielle Integrität: Kurzfristige Ziele können mittel- oder langfristigen Zielen scheinbar widersprüchlich gegenüberstehen. Viele Software-Entwickler kennen dies nur allzu gut aus dem Zielkonflikt zwischen Test-Durchführung, -Automatisierung und -Abdeckung. Um eine gute Balance zwischen lang-, mittel- und kurzfristigen Zielen herzustellen, sollten die Beziehungen zwischen den einzelnen Zielen mindestens inhaltlich nachvollziehbar, idealerweise strukturell darstellbar sein. Mit anderen Worten: Die referenzielle Integrität der Pyramide muss sichergestellt werden. Es sollte also keine Requirements geben, die nicht auf irgendeinem Weg auf die Vision einzahlen.

Pull vs. Push: Sicherlich können Produkte entwickelt werden, indem man alle operativen Tätigkeiten plant und einem Team vorgibt („Push"). Diese Projekt-Management-Methodik aus dem letzten Jahrhundert ist aber nicht mehr zeitgemäß. Wesentlich effektiver und nachhaltiger geht die Entwicklung eines Produkts vonstatten, wenn alle Projektbeteiligten ihre Aufgaben selbstständig organisieren („Pull"). Eine wesentliche (wenn auch nicht die einzige) Voraussetzung dafür ist die Kenntnis der Product Vision. Sie gehört daher an die Wände der Projekträume und auf die Startseite des Wikis, mit Sicherheit aber nicht in den Panzerschrank.

Fokussierung: Aus dem Physikunterricht kennen wir das Kräfte-Parallelogramm. Darin addieren sich einzelne Kräfte nur dann zu 100 Prozent, wenn ihr Richtungsvektor der gleiche ist. Die resultierende Kraft kann sich allerdings auch auf null reduzieren, wenn sie in entgegengesetzte Richtungen wirken. Gleiches gilt für die Umsetzung der Product Vision. Wenn die Projektbeteiligten die Produkt-Vision und ihre Priorisierung verstanden haben, können sie den gleichen Richtungsvektor einschlagen. Solche Teams erreichen leicht die doppelte Leistung gegenüber dem Durchschnitt. Im Idealfall wird die Product Vision gemeinsam mit allen Projektbeteiligten entwickelt *(siehe Abbildung 2)*.

Vision

„Wer Visionen hat, sollte zum Arzt gehen", sagte Helmut Schmidt in einem Interview im Jahr 1980. Bei allem Respekt vor dem ehemaligen Bundeskanzler der BRD, hier lag er falsch. Die Vision ist unsere Vorstellung von einer möglichen Zukunft. Sie bestimmt die Richtung, in die wir gehen. Für die Entwicklung eines erfolgreichen Produkts ist es daher essenziell, sich Gedanken darüber zu machen, wie die Welt in fünf bis zehn Jahren aussieht. Der Zeithorizont ist dabei abhängig von der Lebensdauer des Produkts und den zu erwartenden Veränderungen in dessen Umfeld. In der aktuellen Phase der Digitalisierung werden sich die meisten Produkt-Visionen auf einen eher kurzen Horizont fokussieren müssen.

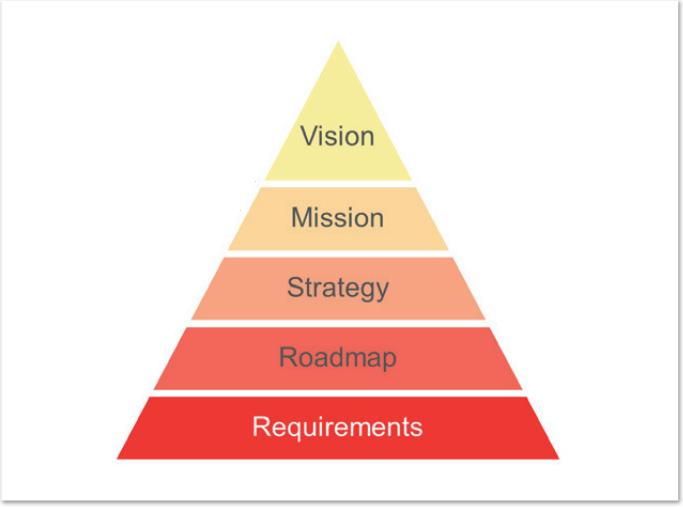


Abbildung 1: Product Vision

Wie kommt man nun zu einer geeigneten Beschreibung der Vision? Hier hat jede Organisation ihr Geheimrezept. Zwei Methoden seien an dieser Stelle genannt: Bei der KPI-Methode werden messbare Schlüssel-Indikatoren ermittelt, die das Produkt betreffen. Diese entstehen aus einem hinreichend langen Zeitraum der Vergangenheit. Daraus wird eine Funktion abgeleitet und für die Zukunft fortgeschrieben. Es sollte darauf geachtet werden, dass Funktionen dieser Art in der Regel nicht linear, sondern exponentieller Natur sind. Der Zeitraum muss daher ausreichend groß sein, damit der exponentielle Charakter erkennbar wird.

Spätestens an der Stelle eines unrealistischen Anstiegs ist ein Paradigmenwechsel zu erwarten. Hier wird sich der Indikator anders entwickeln oder durch einen anderen ausgetauscht werden. Ein gutes Beispiel ist die Entwicklung der Prozessor-Performance auf Basis der KPIs „Taktfrequenz" und „Anzahl der Prozessorkerne". Zunächst war die Taktfrequenz über viele Jahre der maßgebliche KPI der Prozessorleistung. Mitte der 2000er gesellte sich die Anzahl der Rechenkerne als marktrelevanter KPI hinzu, die Taktfrequenz verlor an Aussagekraft.

Die Was-wäre-wenn-Methode hilft besonders dann, wenn KPIs als konstant angesehen werden, dies aber nicht sind. Besonders langjährige Insider einer Branche und Experten auf den betreffenden Gebieten sind hier gefährdet. Um dieser Denkfalle zu entrinnen, sollte die Was-wäre-wenn-Frage gestellt werden. So führte die Infragestellung des vermeintlichen Fakts, dass in einer Speicherzelle nur ein Bit gespeichert werden kann, dazu, dass wir heute mit 3 bis 4 Bit pro Zelle den Inhalt mehrerer Festplatten auf einer Micro-SD unterbringen können. Aber auch gesellschaftliche Konstanten können und sollten infrage gestellt werden, wenn sie einen Bezug zum Produkt haben: „Was wäre, wenn es von der Gesellschaft nicht mehr akzeptiert wird, Wälder und Gemeinden für die Gewinnung von Kohle zu opfern?"

Mit dieser Methode kann in einer fehlertoleranten Projektkultur gern auch mal über das Ziel hinausgeschossen werden. Das regt

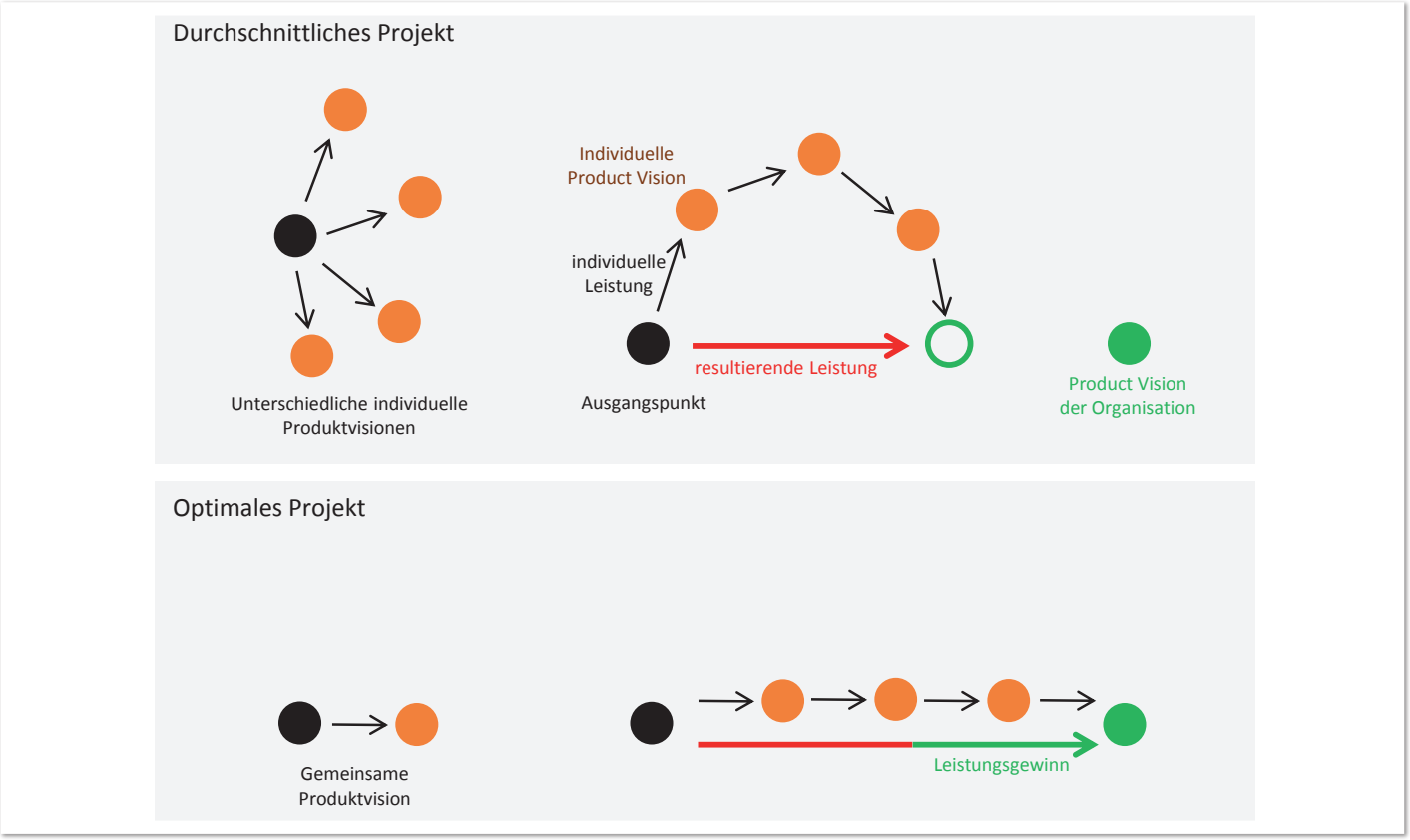


Abbildung 2: Leistungsgewinn durch eine gemeinsame Produkt-Vision

die Kreativität an und hilft, Denkblockaden zu lösen: „Wie müsste eine Welt aussehen, in der PI nicht 3,14159265… sondern 3 wäre?" Hat man ein klares Bild von der Zukunft in Bezug auf das zu entwickelnde Produkt, gilt es, dieses so zu formulieren und idealerweise grafisch zu verstärken, dass es in 15 bis 20 Sekunden einem potenziellen Nutzer erklärt werden kann.

Die imaginäre Stromvolt GmbH hat für ihre Vision als KPIs die Preise für den Bezug, die Erzeugung und die Speicherung elektrischer Energie analysiert und daraus abgeleitet, dass bereits in drei bis vier Jahren der Break-Even für regionale Elektroenergie-Erzeugung und -Speicherung erreicht werden kann *(siehe Abbildung 3)*.

Im Ergebnis kommt sie zu folgender Vision: „Elektrische Energie wird künftig in der Nähe der Verbraucher erzeugt und gespeichert. Erzeuger und Verbraucher stimmen sich autonom und dezentral ab."

Mission

Die Mission beschreibt die Position, die eine Organisation in dieser Vision einnimmt, oder den Beitrag, den sie dazu leistet, diese

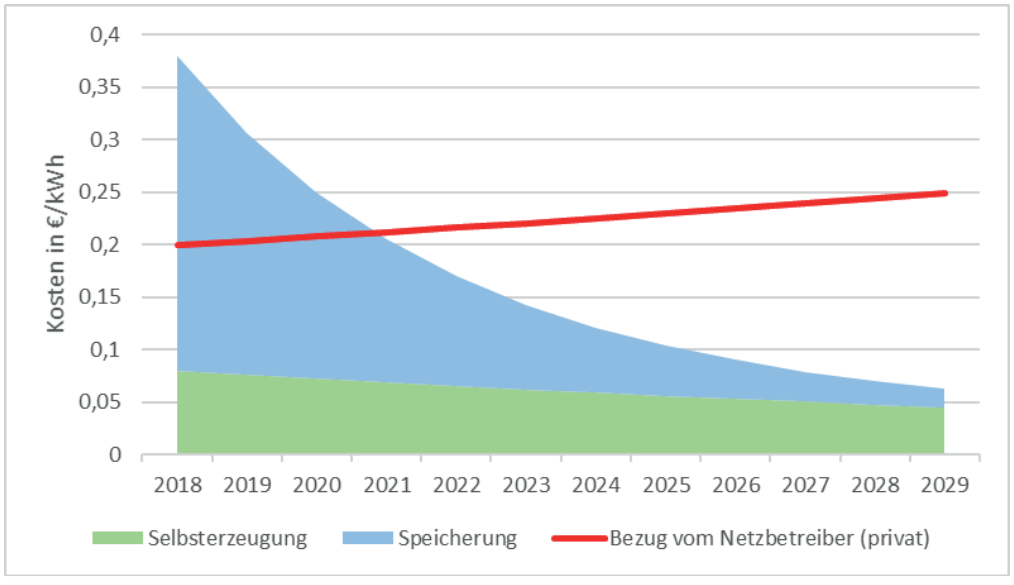


Abbildung 3: KPI-Diagramm der Stromvolt GmbH

Vision real werden zu lassen. Aus ihr sollte der Mehrwert für den Kunden erkennbar sein. Auch die Mission sollte in 15 bis 20 Sekunden jedem potenziellen Kunden erklärt werden können. Für die Stromvolt GmbH lautet sie: „Mit unserer eGRID-Solution werden unsere Kunden unabhängig vom Energieversorger, optimieren ihre Erträge und sorgen für ein ausgeglichenes und stabiles Stromnetz."

Gemeinsam mit der Vision und einer auf den Punkt gebrachten Mission kann man sich bereits an den ersten Elevator Pitch wagen. Verläuft dieser positiv, findet man sich schnell in der Chefetage wieder und wird mit Fragen zur Realisierung konfrontiert. Dafür liefert die nächste Ebene die richtigen Antworten.

Strategie

Die Strategie beschreibt Mittel und Wege, mit der eine Mission erfüllt werden kann. Den Aufwand zur Herleitung einer wirksamen Strategie sollte man nicht unterschätzen. Sie erfordert viel Aufmerksamkeit für interne und externe Faktoren sowie ihre Veränderungen. Ein guter Ansatz und Einstieg in die Strategie ist die Durchführung einer SWOT-Analyse. Das englische Akronym steht für „Strength, Weaknesses, Opportunities and Threats". Im betreffenden Kontext lassen sich die Wörter teilweise entgegen der wörtlichen Übersetzung am besten mit „Stärken, Schwächen, Chancen und Risiken" übersetzen. Die SWOT-Analyse ist nicht sehr beliebt und wird oft bekrittelt. Es hat jedoch noch keinem geschadet, die Chancen und Risiken seiner Umwelt wahrzunehmen und sich seiner eigenen Stärken und Schwächen bewusst zu werden.

Vielleicht rührt die Kritik von dem Umstand her, dass der Begriff etwas unglücklich gewählt wurde, denn die SWOT-Analyse beinhaltet neben der Analyse auch die Ableitung konkreter Maßnahmen. Dieser Schritt wird gern einmal vergessen.

Eine vollständige SWOT-Analyse kann so erfolgen: Im ersten Schritt werden die inneren Faktoren (Stärken, Schwächen) sowie externe Faktoren (Chancen, Risiken) bezogen auf Vision und Mission ermittelt. Dabei unterfüttert man die Punkte soweit wie möglich mit Fakten. Idealerweise lassen sich KPIs ermitteln, die anschließend zur Messung des Erfolgs der gewählten Strategie herangezogen werden.

Fällt es schwer, die richtigen Faktoren zu finden, hilft das Businessmodell des Produkts. Existiert es noch gar nicht, ist bereits die erste große Schwäche gefunden. Ist es vorhanden, können folgende Fragestellungen dabei unterstützen, Stärken, Schwächen, Risiken und Chancen zu ermitteln:

- Welche Mehrwerte verspreche ich dem Kunden (Value Proposition)?
- Welche Kunden spreche ich an?
- Welche Form der Kundenbeziehung strebe ich an?
- Über welche Kanäle spreche ich den Kunden an?
- Wie sieht der Business Case aus?
  - In welcher Höhe werden wann welche Kosten/Ressourcen anfallen?
  - In welcher Höhe werden wann welche Einnahmen/Werte erzielt?

SWOT-Analyse der Stromvolt GmbH Schritt 1	
Stärken 1)Agiles Unternehmen 2)International aufgestellt	**Schwächen** 1)Klein im Vergleich zu Mitbewerbern 2)keine Lobby in den Parlamenten
Chancen 1)Break Even ist nah 2)Konzerne sind auf anderer Strategie unterwegs	**Risiken** 1)Konzerne schwenken um 2)Rohstoffpreise für Energiespeicher steigen

Abbildung 4: SWOT-Analyse der Stromvolt GmbH, Schritt 1

SWOT-Analyse Schritt 2		Intern	
		Stärken	Schwächen
Extern	Chancen	• Schneller internationaler Marktstart	• Vernetzung mit lokalen Partnern • Zusammenarbeit mit Kommunen
	Risiken	• Frühestmögliche Kundenbindung durch ein Global Dashboard	• Strategische Partnerschaft mit Batterieherstellern

Abbildung 5: SWOT-Analyse der Stromvolt GmbH, Schritt 2

- Mit welchen Partnern arbeite ich zusammen?
- Was werden meine Mitbewerber anbieten?

Für die Stromvolt GmbH sieht der erste Teil einer stark verkürzten SWOT-Analyse wie in *Abbildung 4* aus.

Im zweiten Schritt der SWOT-Analyse werden die internen und externen Faktoren als Kreuzprodukt gegenübergestellt, um für alle relevanten Kombinationen Maßnahmen zu definieren. Daraus ergeben sich folgende konkrete Fragestellungen:

- Was ist zu tun, um mit der Stärke Sn die Chance Om zu ergreifen?
- Was ist zu tun, um mit der Stärke Sn dem Risiko Rm zu begegnen?
- Was ist zu tun, damit die Schwäche Wn nicht die Chance Om zerstört?
- Was ist zu tun, damit die Schwäche Wn nicht durch das Risiko Rm verstärkt wird?

Die Erkenntnisse aus dem zweiten Schritt der SWOT-Analyse können unmittelbaren Einfluss auf die Stärken, Schwächen, Chancen und Risiken haben. Daher sind die Schritte 1 und 2 in der Regel mehrfach zu wiederholen, bis sich ein stabiles Gleichgewicht eingestellt hat, in der die strategischen Maßnahmen ein erfolgreiches Produkt versprechen. Für die Stromvolt GmbH ist in der *Abbildung 5* eine Teilmenge der strategischen Maßnahmen abgebildet.

Roadmap

Die Liste der strategischen Maßnahmen kann recht umfangreich und der Umsetzungszeitraum recht lang sein. Es empfiehlt sich daher, Teilziele zu formulieren und diese auf der Roadmap in eine zeitliche Reihenfolge zu bringen. Für die Herleitung einer solchen Roadmap werden die strategischen Maßnahmen zunächst nach folgenden Kriterien priorisiert:

- Wirksamkeit für die Erfüllung der Mission
- Abhängigkeit von anderen strategischen Maßnahmen
- Finanzielle Aspekte (Kosten, Nutzen, Liquidität etc.)
- Erfolgswahrscheinlichkeit

In einem zweiten Schritt werden die Maßnahmen zu Teilprodukten gruppiert und anschließend auf der Zeitachse abgetragen. Selbstverständlich sind hier die wichtigsten Maßnahmen mit den geringsten Wartekosten so früh wie möglich positioniert. Durch Versionierung von Teilprodukten lässt sich die Zeit bis zu einem Marktstart weiter verkürzen.

Die so entstandenen Teilprodukte werden auf der Zeitachse abgetragen und mit wesentlichen Meilensteinen versehen. Diese sind zunächst nur semantisch und haben noch kein festes Datum, maximal einen groben Zeitrahmen. Die schrittweise Präzisierung der Meilenstein-Termine erfolgt später entlang der agilen Entwicklung anhand der Release-Burndown-Charts. *Abbildung 6* zeigt die Roadmap für die Stromvolt GmbH.

Requirements

Die unterste Ebene der Product Vision beschreibt konkret und messbar das Minimum an Anforderungen für jedes Element der Roadmap, in der agilen Welt als „Minimal Viable Product" (MVP) bezeichnet. Die Requirements sind somit die Quelle für das Product Backlog. Daher ist es für den Product Owner hilfreich, wenn die Definition der Anforderungen bereits analog zu User Stories und Epics formuliert werden. Sie sollten also folgende Informationen beinhalten:

- Wer hat eine Anforderung?
- Was konkret erwartet er?
- Welchen Zweck verfolgt er damit?
- Gibt es konkrete Randbedingungen oder Akzeptanzkriterien?

Eine exemplarische Anforderung an das Grid API V1.0 der Stromvolt GmbH lautet: Als Stromvolt-Partner für operative Energiespeicherung möchte ich Zugang zu den anonymisierten historischen Verbrauchsdaten bekommen, damit ich meine Anlage optimal auslegen kann. Dazu gehören:

- Menge der erzeugten elektrischen Energie
- Menge der konsumierten Energie
- Flexible Wahl der Zeit-Intervalle (Minute bis Quartal)
- Regionale Verteilung

Agile Product Vision

Wurde eine Product Vision mit den vorangegangenen Schritten erstellt, ist die Arbeit noch nicht beendet. Genau genommen war die initiale Erstellung einer Produkt-Vision erst der Anfang. Die Welt verändert sich täglich und mit ihr die Wünsche der Kunden. Damit ändern sich die Anforderungen an das Produkt. Die Änderungen können sich wiederum auf die Roadmap, Strategie, Mission und Vision auswirken. Je höher die Ebene, desto schleichender und unauffälliger kommt der Anpassungsbedarf daher. So kann es passieren, dass die Vision von der Realität überholt wird.

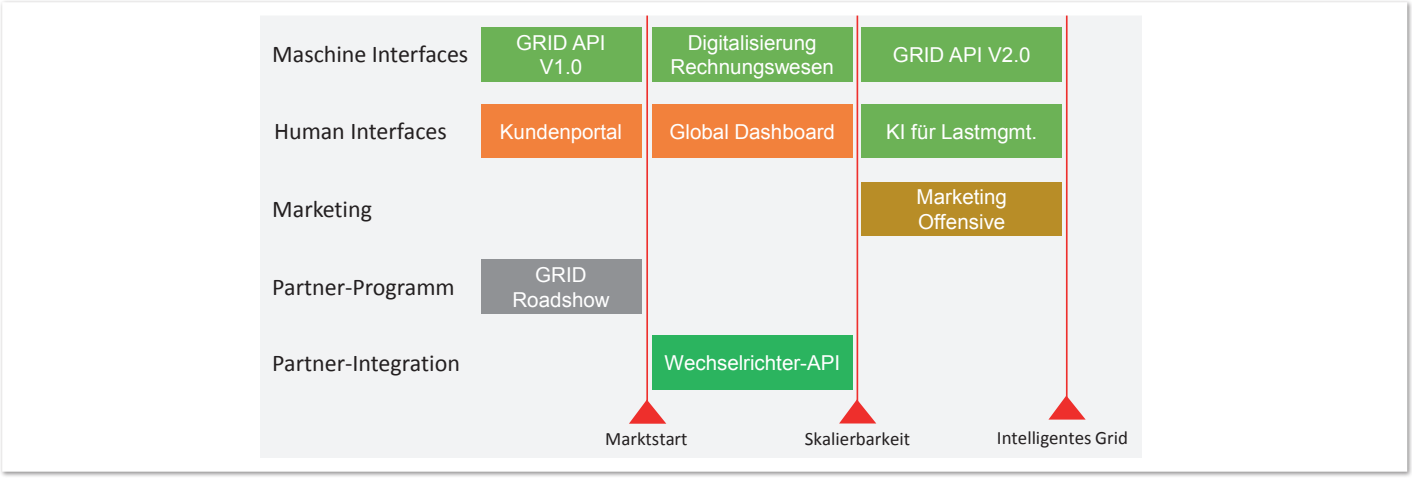


Abbildung 6: Roadmap der Stromvolt GmbH

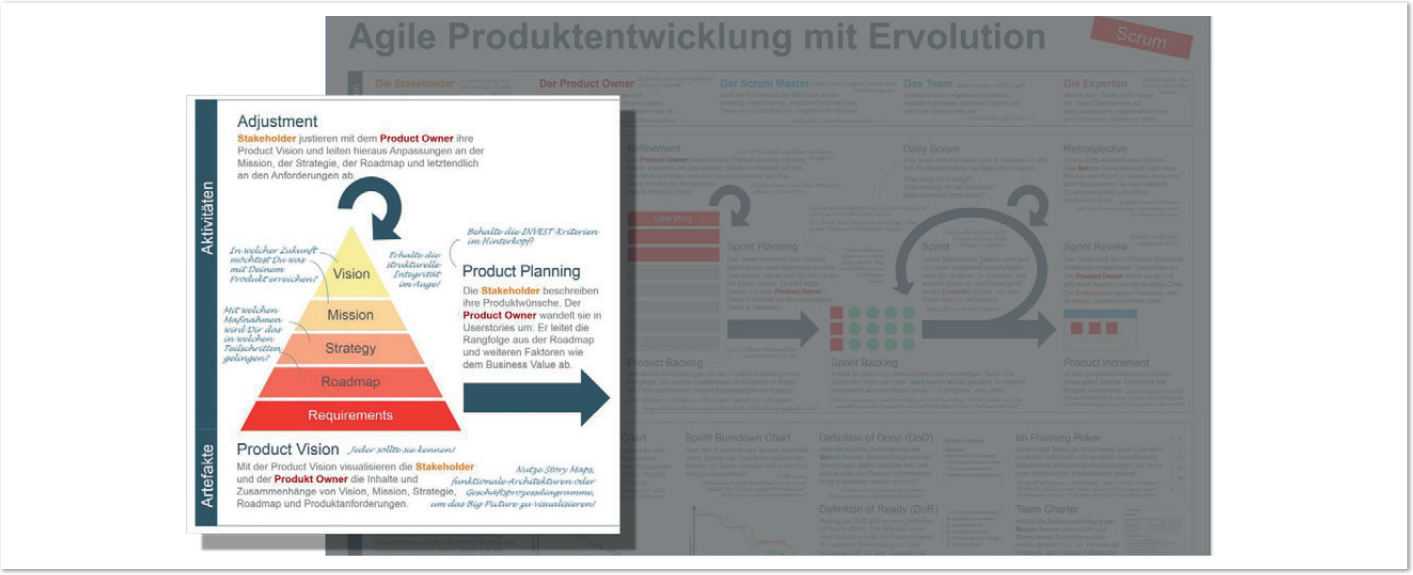


Abbildung 7: Agile Produkt-Entwicklung mit Ervolution

Firmen wie Kodak, Nokia etc. zeigen, dass eine fehlende Anpassung der Product Vision existenzbedrohend sein kann. Aus diesem Grund sollte die kontinuierliche Weiterentwicklung der Product Vision in den Entwicklungsprozess integriert werden. In der „agilen Produkt-Entwicklung mit Ervolution" ist die Product Vision ein weiteres Artefakt, das durch die iterativen Aktivitäten „Adjustment" und „Product Planning" aktualisiert beziehungsweise verarbeitet wird *(siehe Abbildung 7)*.

Im Adjustment wird die Product Vision überprüft und gegebenenfalls an veränderte Rahmenbedingungen angepasst. An dieser Stelle kommen die in der Vision und Strategie verwendeten KPIs zum Einsatz. Ihre Veränderung dient als eine Eingangsgröße für das Adjustment. Im Product Planning werden die Änderungen an der Product Vision besprochen und die notwendigen Anpassungen am Product Backlog vorgenommen.

Hinweis: Eine vollständige Übersicht über die Rollen, Aktivitäten, Artefakte und Instrumente der agilen Produkt-Entwicklung mit Ervolution kann bis zum 23. März 2019 als PDF-Version eines DIN-A1-Posters unter *„https://ervolution.de/aktion-java-aktuell-2018"* kostenlos heruntergeladen werden.

Jörg Domann
joerg.domann@ervolution.de

Jörg Domann unterstützt seit dem Jahr 2006 Konzerne und mittelständische Unternehmen bei der Agilisierung ihrer Produkt-Entwicklung, beim Management komplexer Programme/Projekte sowie bei der Optimierung der Geschäftsprozesse. Zuvor war er nach erfolgreichem Abschluss seines Informatikstudiums an der TU Dresden als Software-Entwickler, Entwicklungsleiter und Manager einer Unternehmensberatung tätig.

Design Matters: Design Thinking in der Entwicklung für bessere Usability

Katarina Poplawski, itelligence

Das Design ist ein wichtiger Teil jedes Produkts, dies gilt ebenso für Software-Lösungen. Das Design der Benutzeroberfläche und damit die User Experience im Bereich von Geschäftsanwendungen werden oft vernachlässigt. Dabei kann es als Alleinstellungsmerkmal oder als Entscheidungskriterium agieren.

Während das Interface-Design für Anwendungen im Privatkunden-Segment immer weiter und rasant verbessert wird, stockt dessen Weiterentwicklung im Bereich von Geschäftsanwendungen enorm und der Unterschied wird zunehmend deutlicher. Um die beschriebene negative Entwicklung zu stoppen oder eine bestehende Anwendung benutzerfreundlicher zu gestalten, kann die Lösung der Einsatz des Design Thinking im Rahmen des Entwicklungsprozesses sein.

Die Realität

Viele Menschen müssen täglich die Herausforderung veralteter Geschäftsanwendungen bewältigen. Diese frustrieren den Endnutzer, denn die Entwicklerteams schenken der Bedienerfreundlichkeit ihrer Programmoberflächen immer noch zu wenig bis keine Beachtung. Sie entwickeln ihre Geschäftsanwendungen für einen IT-Spezialisten, der eine intuitive und einfache Usability nicht als Hauptkriterium ansieht. So muss der IT-Spezialist heute immer

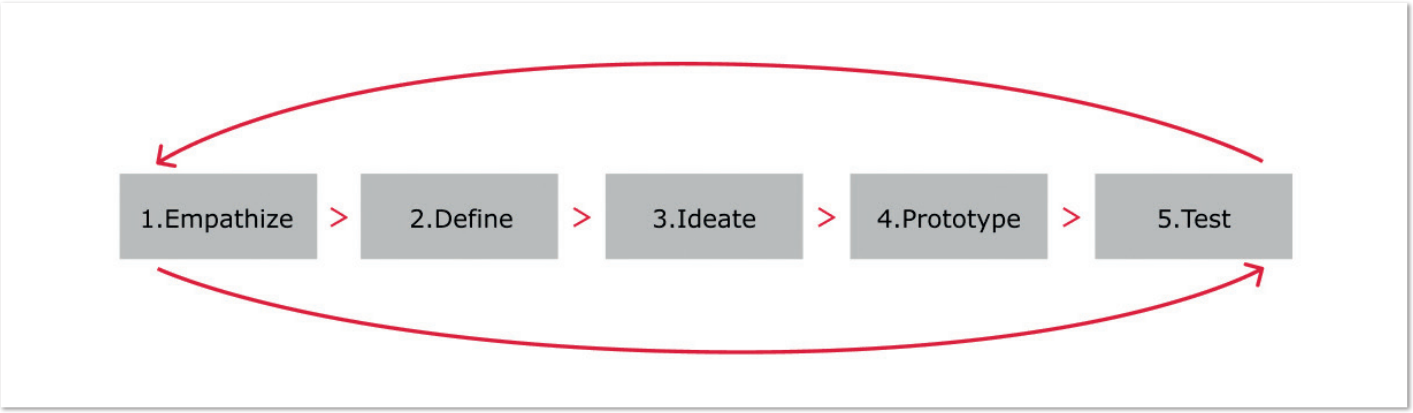


Abbildung 1: Design Thinking von d.school

noch mit Anwendungen arbeiten, deren UI-Oberfläche an die Relikte aus den 1990er-Jahren erinnert und deren Usability nach wie vor auf langen Listen, Form- und Tabellen-Views aufbaut. Egal ob ERP-, CRM- oder MRP-Anwendungen, diese erfordern zahlreiche, immer wiederkehrende manuelle Eingaben.

Der Endnutzer lernt mit der Zeit, mit der Anwendung zu arbeiten. Da viele Geschäftsanwendungen Prozess- und nicht Benutzer-orientiert gestaltet sind, ist deren Handhabung so kompliziert, dass eine oft mehrtägige Schulung unumgänglich ist. Dabei nutzt der gleiche „IT-Spezialist"-Endanwender privat sehr gerne eine Vielzahl unterschiedlicher Anwendungen, die sowohl nutzerfreundlich als auch einfach verständlich sind. Sie sollen nicht nur Spaß machen, sondern tatsächlich das Leben ihres Endanwenders bei der Lösung einer gestellten Aufgabe erleichtern.

In der Realität fragt sich der IT-Spezialist, warum seine Software-Tools so umständlich sind und ihm die Lösung seiner alltäglichen Arbeitsaufgaben erschweren. Aufgrund der privaten Erfahrungen steigen seine Anforderungen an Business-Anwendungen stetig, während die Endnutzer-Zufriedenheit zunehmend sinkt. In Wirklichkeit sind die Anforderungsunterschiede an die Usability und somit auch an das Interface-Design zwischen Geschäfts- und Privatanwendungen enorm.

Zwar nennen die meisten Softwarehersteller die Kunden- und/oder Endnutzer-Zufriedenheit bezüglich Usability als wichtige Zielstellung, in Wirklichkeit geht es in der Business-Anwendungswelt jedoch vor allem um schnelle Lösungen; getreu dem Motto „Hauptsache es funktioniert" werden altbewährte Lösungen weiterentwickelt. Die Innovation findet zumeist im Backend der Anwendung statt und kommt somit nicht beim Endnutzer an. So scheitern die guten Vorsätze schlussendlich an den Entwicklern, die oft selbstständig nach bestem Wissen und Gewissen die Anwendungen weiterentwickeln. Es gibt eine mögliche Lösung des Problems: Die Applikationen mithilfe des Design Thinking zu realisieren.

Die Theorie

Es gibt keine einheitliche Definition für das Design Thinking, da es vielseitig eingesetzt und somit je nach Anwendungsgebiet auch unterschiedlich beschrieben wird. Für einige ist es ein Prozess, für andere wiederum eine Methode. Folgende zwei Definitionen beschreiben gemeinsam das Design Thinking als einen Entwicklungsprozess am besten:

- Design Thinking ist eine kreative und kollaborative Problemlösungsmethode aus der Nutzerperspektive.
- Design Thinking ist ein Innovationsprozess, der die Kunden und deren Bedürfnisse in den Mittelpunkt setzt

Folgende Grundeigenschaften von Design Thinking lassen sich anhand der genannten Definitionen ableiten, die im Rahmen der Entwicklung einer Softwareanwendung zu beachten sind. Der Design-Thinking-Prozess ist:

- **Nutzerzentriert**
  Die Wünsche des Endanwenders stehen immer im Mittelpunkt des Prozesses
- **Ein Teamprozess**
  Basierend auf dem Wissen eines Teams von Fachexperten aus unterschiedlichen Bereichen
- **Ein Lernprozess**
  Bei jedem Schritt des Prozesses werden Erkenntnisse gewonnen und Neues erlernt, im nächsten Schritt wird das neuerworbene Wissen vom Team weiterverarbeitet
- **Iterativ und agil**
  Die einzelnen Schritte sind keine Meilensteine wie bei anderen agilen Prozessen (etwa Scrum). Somit kann man die Reihenfolge der Schritte ändern und wiederholen, solange das gewünschte Ergebnis nicht erzielt ist

Es gibt auch kein einheitliches Prozess-Schema. Die zwei grundlegenden Prozess-Schemata wurden von zwei Bildungseinrichtungen entwickelt, die sich komplett dem Design Thinking widmen: der d.school an der Stanford University und dem Hasso-Plattner-Institut in Potsdam. Beide Bildungseinrichtungen wurden vom deutschen Unternehmer Hasso Plattner, einem der SAP-Gründer, gestiftet. Die d.school untergliedert den Prozess in fünf Schritte *(siehe Abbildung 1)*, das Hasso-Plattner-Institut sieht sechs Schritte *(siehe Abbildung 2)*.

Im Grunde splittet das Hasso-Plattner-Institut den ersten Schritt „Empathize" von d.school in zwei Schritte auf, und zwar in „Verstehen" und „Beobachten". Wie bereits erwähnt, ist Design Thinking ein dynamischer und iterativer Prozess, dabei kann sich die Reihenfolge der einzelnen Schritte ändern oder einige Schritte können mehrmals wiederholt werden. So haben manche Unternehmen, wie beispielsweise IBM oder Google, den Prozess an die individuellen Gegebenheiten angepasst.

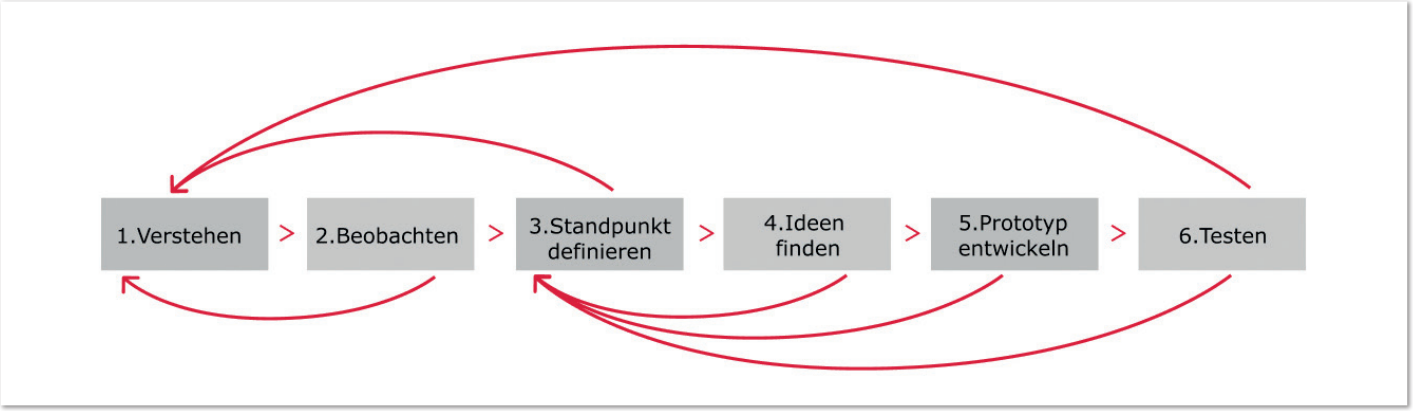


Abbildung 2: Design Thinking vom Hasso-Plattner-Institut

Wichtig ist also zu verstehen, dass, wenn der Prozess zur Entwicklung einer Softwareapplikation angewandt wird, dieser nur in Teamarbeit funktioniert und nur wenn von Anfang an der Endnutzer und seine Bedürfnisse in den Mittelpunkt des Prozesses gestellt werden. Im Grunde lehrt das Design Thinking, wie ein (Produkt-)Designer zu denken, so nach einer Lösung zu suchen und diese zu realisieren.

Die Praxis

Wie lässt sich das Design Thinking auf die Entwicklung einer Geschäftssoftware anwenden? Im ersten Schritt „Verstehen" versucht das Team, das Problem zu erfassen und einzugrenzen. Was ist das Problem und wie wird es aktuell vom Endnutzer gelöst? Diese Frage ist im ersten Schritt zu beantworten. Dafür müssen die Bedürfnisse des Nutzers erkannt und verstanden werden. Dabei kommen unterschiedliche Methoden zum Einsatz, wie beispielsweise Personas erstellen und davon die möglichen Nutzerszenarien ableiten.

Mithilfe von Personas werden möglichst detaillierte Profile des Endanwenders ausgearbeitet, die mit den Profilen in den sozialen Netzwerken vergleichbar sind und eine Charakterisierung des Endanwenders zum Ziel haben. Relevante Fragen und Informationen, die bei der Ausarbeitung von Personas Anwendung finden, sind:

- Wie alt ist mein Endnutzer?
- Wie nutzt er normalerweise die Anwendung und in welchem Kontext?
- Wie kann die Aufgabe für ihn einfacher gestaltet werden?

Je detaillierter die Beschreibung, desto einfacher lassen sich die Endnutzer-Bedürfnisse erkennen und desto schneller wird die Empathie zum Endnutzer vom Team aufgebaut. Normalerweise wird eine Software-Anwendung zumeist von mehreren unterschiedlichen Anwendergruppen genutzt. Somit müssen mehrere Personas erstellt werden, da sich beispielsweise die Bedürfnisse eines Administrators stark von den Bedürfnissen eines einfachen Endanwenders unterscheiden. Die Profile von Personas werden im weiteren Verlauf des Design Thinking verfeinert und ausgebaut.

Beim zweiten Schritt „Beobachten" werden die Annahmen aus dem ersten Schritt „Verstehen" praktisch untersucht. Auf die folgenden Schlüsselfragen werden Antworten gesucht:

- Sind die Annahmen korrekt?
- Existiert das angenommene Problem tatsächlich?

Auch hier stehen dem Team viele unterschiedliche Methoden zur Verfügung. So kann sich das Entwicklerteam selbst in die Situation des angenommenen Nutzers versetzen oder Interviews mit den Endanwendern führen, um eine Lösung für das angenommene Problem zu entwickeln. Allgemein gilt: Je früher der Endanwender einbezogen wird, desto besser. Die Mitglieder des Teams müssen bewusst die Erfahrungen des Nutzers machen, um dessen Probleme schneller definieren zu können.

Eine effektive Methode ist die Beobachtung des Endnutzers bei der Lösung des Problems: Idealerweise lässt das Design-Thinker-Team sich vom Nutzer zeigen, wie er bisher vorgegangen ist. Diese Vorgehensweise funktioniert besonders gut bei existierenden Geschäftsanwendungen. Dabei ist es sehr wichtig, die Erkenntnisse zu dokumentieren und zu visualisieren. Die Antworten werden aufgezeichnet, das Besondere fotografiert oder der Anwender gefilmt. So kann der Endnutzer besser kennengelernt und Empathie aufgebaut werden. Im Idealfall werden zwei extrem unterschiedliche Nutzer und ihre Bedürfnisse miteinander verglichen, etwa die Bedürfnisse von Gelegenheits- und Power-Usern.

Anschließend werden im nächsten Schritt „Standpunkt definieren" die aus der Nutzersicht gesammelten Erkenntnisse zum Problem analysiert, interpretiert und gewichtet. Durch die folgende Synthese lassen sich außerdem unentdeckte Nutzer-Bedürfnisse herausfinden, darauf basierend wird der Standpunkt definiert. Bei dem Schritt sollte das Projektteam eine gemeinsame Wissensbasis aufbauen, die Anforderungen und Wünsche müssen also von allen Teammitgliedern verstanden werden. Zum Festhalten von Bedürfnissen und der Aufgabe können folgende Methoden angewandt werden: Customer Journey, User Stories, Definition von Point of View etc.

Mit den gewonnenen Ergebnissen und dem definierten Standpunkt werden nun im nächsten Schritt die Ideen generiert. Aber wie generiert man Ideen? Auch in dieser Phase können je nach Aufgabe unterschiedliche Verfahren angewandt werden, um die Kreativität des Teams zu steigern und so eine Vielzahl von Ideen zu definieren. Fakt ist: Gute Ideen kommen selten im Büro am Arbeitsplatz. Kreative Firmen schaffen daher kreative Räume für ihre Mitarbeiter, die für die Laufzeit des Projekts dem Team zur Verfügung stehen. Außerdem sollten gezielt Methoden zur strukturierten Ideenfindung in der Teamarbeit angewandt werden.

Die beliebteste und die bekannteste Kreativitätstechnik zur Generierung von Ideen ist das Brainstorming, individuell oder im Team.

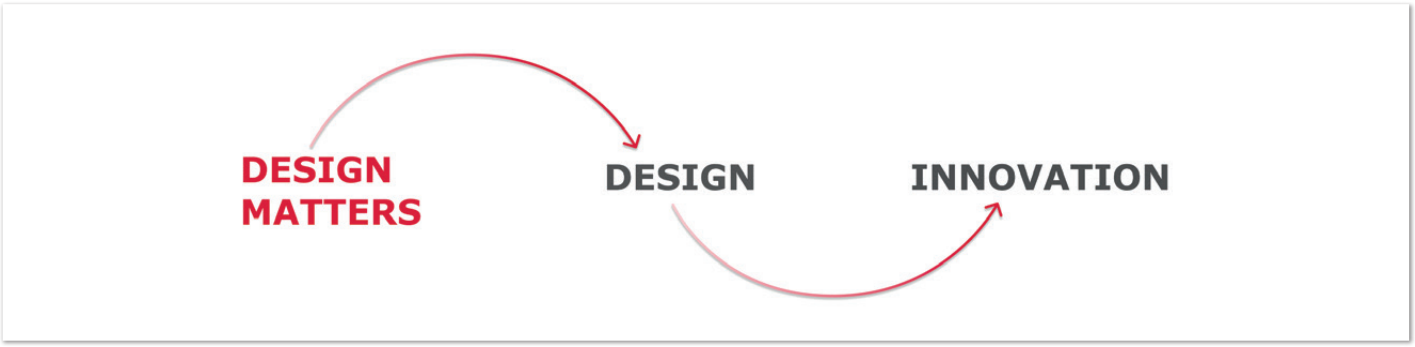


Abbildung 3: „Design Matters" – erst durch das Design wird Innovation greifbar

Nachdem eine Vielzahl von Lösungsansätzen beim freien Brainstorming entstanden sind, kann beispielsweise ein gezieltes Brainstorming zur kritischen Funktionalität der Software-Anwendung durchgeführt werden. Anschließend erfolgen die Beurteilung und das Clustering der Ideen.

Oft werden Ideen mit Anforderungen vertauscht. Insbesondere bei technischen Produkten und auch Geschäftsanwendungen fällt es den Teams schwer, die Ideen von den Anforderungen zu unterscheiden. Dazu folgendes Beispiel: Die neue Geschäftsanwendung sollte als Web-Applikation entwickelt werden und daher responsiv sein. Bei dem aufgeführten Beispiel handelt es sich nicht um eine Idee, sondern um eine Anforderung. Die Idee wäre, ein Framework vorzuschlagen, das die Responsivität unterstützt, oder direkt einen Wireframe-Prototyp eines responsiven Layouts zu erstellen.

Die Ideen sollten nicht nur schriftlich festgehalten, sondern je früher, desto besser ebenfalls visualisiert werden, um die Kreativität zu steigern. Bei dieser Vorgehensweise folgen zumeist sofort weitere Ideen. Durch die Visualisierung werden auch für die Teamkollegen die Ideen greifbar gemacht.

Um die Kreativität des Teams zu fördern, bietet es sich an, für die Applikationen dreidimensionale Prototypen zu basteln. Aus diesem Grund basteln selbst die Design-Thinker-Teams der NASA, um ihre Ideen zu visualisieren. Im Allgemeinen ist die Visualisierung beim Design Thinking sehr wichtig, denn erst dadurch lassen sich ungewöhnliche Verbindungen, Muster und Lösungen erkennen. Eine gemeinsame Gedankenkette eines Teams kann nur durch das Mitteilen entstehen, durch die Visualisierung dokumentiert werden und so weiterwachsen.

Anschließend werden im vierten Schritt aus den Ideen Prototypen entwickelt, aus der Theorie wird Praxis, aus den Plänen wird Realität. Die besten Ideen werden vom Team ausgewählt und zu Beginn anhand von einfachen Prototypen dargestellt. Hier spielt natürlich der Zeit- und Kostenfaktor eine sehr große Rolle; je mehr Zeit für die Gestaltung von Prototypen zur Verfügung gestellt wird, desto aussagekräftiger wird im nächsten Schritt das Nutzerfeedback in der Testphase. Deswegen empfiehlt es sich, zu Beginn einfach, schnell und skizzenhaft die ersten Lösungsideen zu veranschaulichen. Dadurch lassen sich bereits erste Lösungsideen aussortieren und andere wiederum favorisieren.

Je einfacher der Prototyp, desto unkomplizierter lässt sich eine schlechte Idee verwerfen. Zu den einfachen Lo-Fi-Prototypen gehören Skizzen, Wireframes und Moodboards. Nachdem ein Lo-Fi-Prototyp steht, wird darauf basierend ein Hi-Fi-Prototyp erstellt. Je mehr Zeit dafür bleibt, desto realistischer der Prototyp und schlussendlich desto qualitativer das Nutzer-Feedback. Um den Pfad nicht zu verlieren, sollten hier bereits die echten Endanwender stark miteinbezogen werden, um detailliertes Feedback, noch vor der Testphase des Prototyps, liefern zu können.

Die Hi-Fi-Prototypen werden mittels grafikbasierter Software geschaffen, etwa Adobe Illustrator, Sketch oder das Online-UX-Tool Figma. Schlussendlich sollte im besten Fall ein interaktiver Prototyp entstehen, der nicht nur das Look-and-Feel der Anwendung abbildet und die User-Szenarios visualisiert, sondern auch die möglichst realistische Usability für die Tester greifbar macht.

Der finale Schritt beim Design Thinking ist das Testen. Ähnlich wie das Prototypen ist auch das Testen stark vom Zeit- und Kostenfaktor abhängig. Der finale Prototyp wird dem Endanwender präsentiert, die Ergebnisse validiert sowie reflektiert. Folglich werden darauf basierend entsprechende Anpassungen am Prototyp vorgenommen (Schritt 4 wird mit neuen Erkenntnissen erneut durchgeführt) und anschließend wieder getestet. Dabei besteht das Ziel des Testens darin, die Schwachstellen einer Idee anhand des Prototyps herauszufinden und zu korrigieren.

Neue Ideen können sowohl beim Testen als auch beim Prototypen entstehen, dann kehrt man zum vierten Schritt zurück und teilt die Idee dem Team mit. Nachfolgend wiederum kann die Idee in den Prototyp eingearbeitet und anschließend wieder getestet werden. Allgemein wird in der Testphase wertvolles qualitatives Feedback gesammelt, insbesondere, wenn dieses von den echten Endanwendern der zukünftigen Applikation kommt. Auch hierfür steht eine Vielzahl von Methoden und Tools zur Verfügung, wie A/B-Test, Feedback-Erfassungsraster, Online-Test- und Feedback-Web-Applikationen etc.

So eignet sich die A/B-Testmethode vor allem zum Testen des Interface-Designs oder der Usability. Dabei werden von gleichen Ansichten der Anwendung (UI-Views) zwei Varianten erzeugt und dem Tester vorgeschlagen. Anschließend wird die Bearbeitungszeit einer gestellten Aufgabe gemessen. Die Ansicht, in der der Tester schneller die Lösung findet, wird dann final implementiert, da sie höchstwahrscheinlich benutzerfreundlicher ist.

Für das Testen einer Geschäftsanwendung sollten webgestützte Tools angewandt werden; damit lassen sich etwa ganze User-Stories testen, dabei die Reaktionszeit messen sowie direkt das Tester-Feedback zu den einzelnen Anwendungsansichten sammeln. Da Design Thinking ein iterativer Prozess ist, wird im Idealfall der Prototyp immer weiter verfeinert, entsprechend dem Tester-Feedback. Der direkte Kontakt zum Nutzer ist unumgänglich, denn weder der Auftraggeber noch der Entwickler kennen die wahren Wünsche des Endanwenders:

- Wann haben Sie das letzte Mal gesehen, wie jemand Ihr Software-Produkt im Alltag benutzt?
- Haben Sie überhaupt schon den Endanwender bei der Nutzung Ihrer Anwendung beobachtet?

Als Ergebnis des Design Thinking entsteht eine Endnutzer-orientierte und validierte Idee in Form eines Anwendungs-Prototyps. Nachdem dieser fertiggestellt und getestet wurde, startet im Anschluss die Anwendungs-Implementierung.

Fazit

Design Thinking ist kein Allheilmittel und kein Garant für den Erfolg einer Geschäftsanwendung, jedoch ist die Wahrscheinlichkeit, im Rahmen eines Benutzer- und Design-orientierten Prozesses unter realer Einbeziehung des Endnutzers eine gute Lösung zu entwickeln, deutlich höher. Schon heute sollte das Ziel jedes Entwicklungsteams einer Geschäftsanwendung darin bestehen, die Unterschiede zwischen den Geschäftsanwendungen und den Applikationen des privaten Gebrauchs zu verringern.

„Design Matters" – denn erst durch die benutzerfreundliche User-Experience und das moderne User-Interface-Design im Frontend der Anwendung wird die Innovation und Einzigartigkeit des Backends der Anwendung für Endnutzer greifbar und erlebbar.

„Design Matters" – denn mithilfe des Design Thinking, also durch das gezielte Gestalten des Denkens im Team, können innovative und außergewöhnliche Ideen entstehen *(siehe Abbildung 3)*. Deshalb immer an die Bedürfnisse der Endanwender denken und deren Leben einfacher machen.

Katarina Poplawski
katarina.poplawski@itelligence.de

Katarina Poplawski hat ihr Bachelor-Studium Medieninformatik an der Hochschule Hof absolviert, gefolgt von einem Master-Studium der Technischen Redaktion und Wissenskommunikation an der Hochschule Merseburg. Seit Januar 2017 ist sie als UX/UI-Designerin im Entwicklungsteam der itelligence in Dresden tätig.

The Apache Way of Open Source

Johannes Geppert, Apache und Struts PMC Member, Amazon

Die Apache Software Foundation ist eine Organisation, unter deren Dach viele erfolgreiche Open-Source-Projekte organisiert sind. Der Artikel zeigt, wozu es einer Open-Source-Foundation bedarf, was der Mehrwert ist, wie man mitmachen kann und wie eine derartige Organisation strukturiert ist.

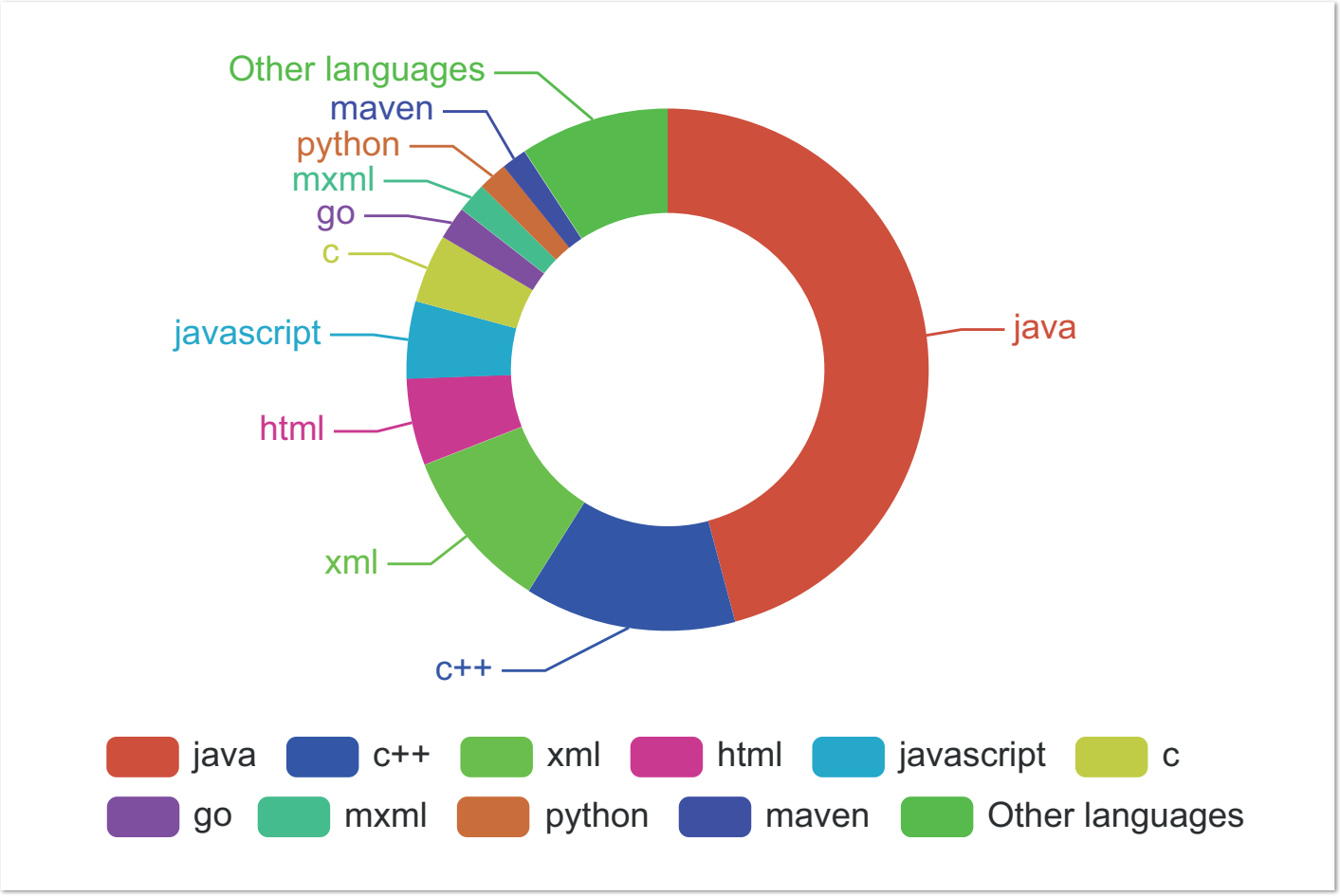


Abbildung 1: Verbreitung der Sprachen

Für die meisten Java-Entwickler ist die Apache Software Foundation (ASF) ein fester Begriff. Mit Hunderten von Projekten, die unter dem Dach der Foundation entwickelt werden, kommt man in seinem Leben als Java-Entwickler zwangsläufig direkt oder indirekt damit in Berührung. Wer hat noch nicht sein Java-Web-Projekt auf einem Apache Tomcat Server laufen lassen und dafür vielleicht eines der vielen Web-Frameworks wie Struts2, Wicket oder Tapestry benutzt oder all die vielen Helfer, um wiederkehrenden Probleme zu lösen, die in den Commons-Bibliotheken stecken?

Die Build-Werkzeuge Maven und Ant sind inzwischen schon so etwas wie ein Standard auf diesem Gebiet. Im Big-Data-Bereich gibt es zahlreiche Projekte wie zum Beispiel Hadoop und Sparks. Oder man hat einfach nur mit der Entwicklungsumgebung NetBeans an seinen eigenen Projekten gearbeitet. Hinzu kommen noch viel mehr Projekte in Bereichen wie „Cloud", „IoT" oder „Machine Learning", die inzwischen eine feste Größe in der IT-Welt sind.

Mit mehr als 350 Projekten und Initiativen, Tausenden von Committers, fast einer Viertelmilliarde Codezeilen und zig Millionen Source-Code-Downloads zählt die ASF zu einem der bedeutendsten Bausteine der Software- und Technologie-Branche. Da etwa die Hälfte des gesamten Source-Codes in Java geschrieben ist, ist Java hier natürlich auch eine der wichtigsten Sprachen *(siehe Abbildung 1)*.

Die ASF ist eine US-basierte, gemeinnützige Organisation mit mehr als 700 Mitgliedern und Tausenden von freiwilligen Kontributoren aus aller Welt in den verschiedensten Projekten. Gegründet wurde die ASF vor etwa 20 Jahren, damals mit dem ersten und einzigen Projekt, dem Apache-HTTP-Server, der auch jetzt noch der am meisten genutzte Open-Source-Webserver der Welt ist und einer der Bestandteile für die schnelle Verbreitung des Internets *(siehe Abbildung 2)*.

Warum sollte man ein Open-Source-Projekt überhaupt unter dem Dach einer Foundation entwickeln? Ein Account auf GitHub ist doch vollkommen ausreichend, um mit einem Open-Source-Projekt erfolgreich zu sein, oder? Natürlich funktioniert dieses Modell ebenfalls und es gibt auch viele erfolgreiche Open-Source-Projekte, die nur auf GitHub betrieben werden. Die ASF kann jedoch mehr tun, um Open-Source-Projekte langfristig zu unterstützen und erfolgreich zu sein.

Um unabhängig von Betreibern zu sein, unterstützt die ASF die Projekte in der Infrastruktur. So werden für jedes Projekt Repositories für den Source-Code, verschiedene Mailing-Listen für die Kommunikation, Wikis für die Dokumentation, Webseiten und mehr betrieben. Den Code in eigenen Repositories vorzuhalten, gibt den Projekten Unabhängigkeit gegenüber Hosting-Anbietern.

Vor einigen Jahren war Google Code eine der beliebtesten Plattformen für Source-Code-Hosting, bis Google sich entschloss, die Plattform Anfang 2016 zu schließen. Wohl niemand kann sagen, wie die Landschaft in 10, 20 oder gar 50 Jahren aussehen wird. Um Kontributoren eine einfache Möglichkeit zu bieten, an Projekten mitzuarbeiten, haben die meisten Projekte allerdings auch einen

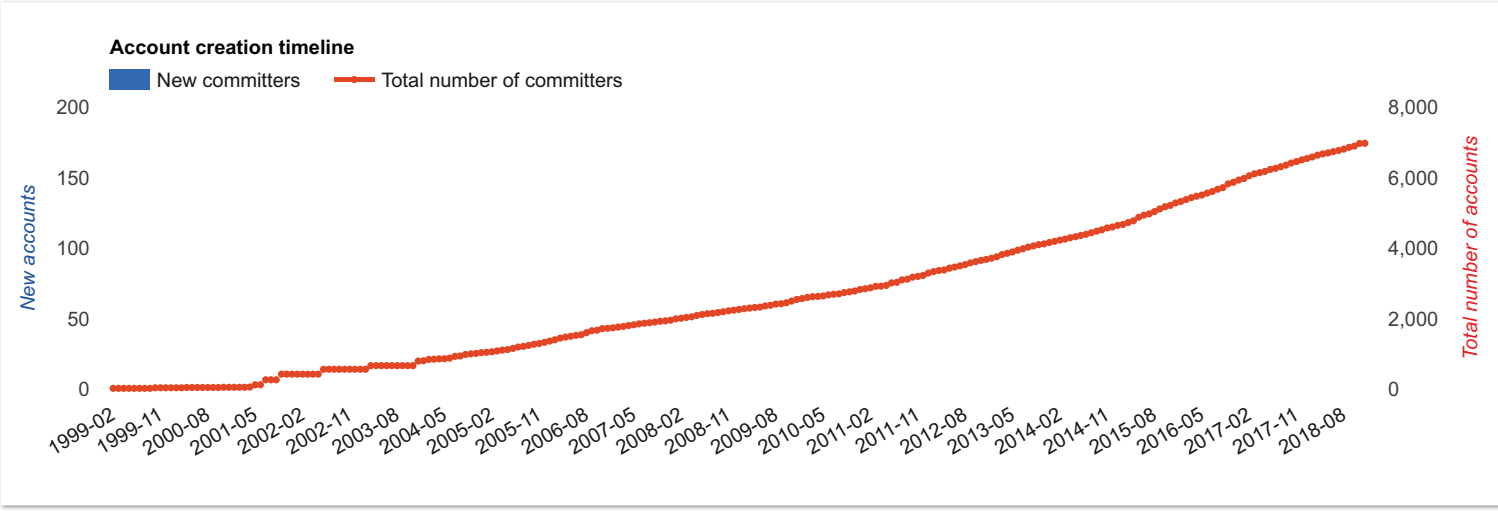


Abbildung 2: Entwicklung des Committer

GitHub-Mirror, um dort den Source-Code zur Verfügung zu stellen, und für einfache Pull Requests über diese Plattform.

Neben der Unterstützung in der Medien- und Pressearbeit ist auch das regelmäßige Organisieren von Konferenzen ein Anliegen der Foundation. Die Konferenzen dienen dazu, aktuelle Themen und Projekte den Usern zu präsentieren, als Treffpunkt für die Community und um aktuelle Open-Source-, Technologie- und Foundation-Themen zu diskutieren. So gibt es jährlich mindestens eine größere ApacheCon-Konferenz. Im Jahr 2019 wird sie vom 9. bis zum 13. September unter dem Motto „20 Years of Apache" in Las Vegas stattfinden. Daneben gibt es auch regelmäßig kleinere Meetups und Roadshows. Die Konferenzen sind ebenfalls eine gute Gelegenheit, die Leute hinter den Projekten kennenzulernen und einen tieferen Einblick in spezielle Themen zu bekommen.

Wenn man von Software spricht, darf man auch das wichtige Thema „Security" nicht vergessen. Die ASF unterstützt hier die Projekte mit ihren Workflows in der Kommunikation und dem Management von Security Issues. Ein weiterer, gerade für Entwickler sehr wichtiger Punkt ist die Unterstützung in rechtlichen Belangen. Durch die ASF ist man als Entwickler geschützt vor „Intellectual Property"-Klagen („Geistiges Eigentum"). Als Projektbeitragender muss man ein „Individual Contributor License Agreement" (ICLA) an die Foundation senden, womit man sein persönliches „Intellectual Property" des geschriebenen Source-Codes an die Foundation überträgt und dafür von der Foundation einen rechtlichen Schutz erhält. Daneben werden ebenfalls die Apache License, die Markenrechte und die Logo-Benutzung durch das ASF-Legal-Team gepflegt sowie die Einhaltung überwacht. Gerade die sehr offene und unternehmensfreundliche Apache License ist unter den Open-Source-Entwicklern und Firmen sehr populär.

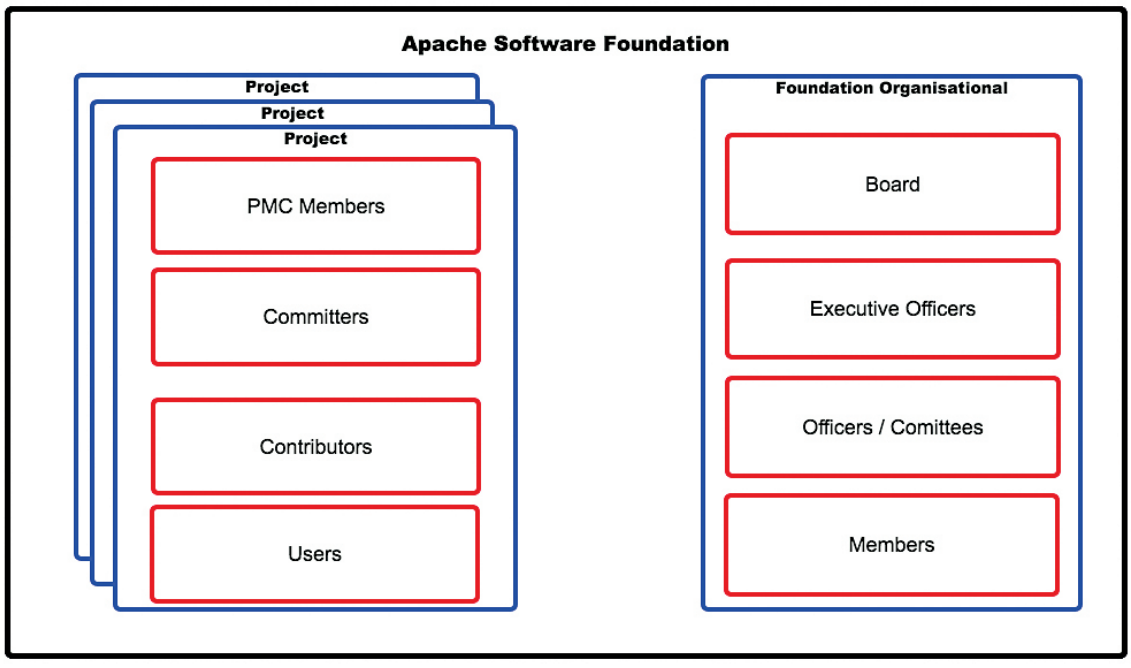


Abbildung 3: Struktur der Apache Software Foundation

Infrastruktur, rechtliche Unterstützung und Überwachung, Marketing und Konferenzen, all das kostet natürlich auch Geld und muss irgendwie bezahlt werden. Um das zu ermöglichen, wird die ASF durch verschiedene Platin-, Gold-, Silber- und Bronze-Sponsoring-Programme oder individuelle einmalige Beiträge hauptsächlich von verschiedenen IT-Firmen, aber auch von Privatpersonen gesponsert. Eine weitere, nicht unübliche Art des Sponsorings für Firmen besteht darin, Entwickler, die in Apache-Projekten aktiv sind, zu bezahlen, um an Projekten weiterzuarbeiten, die für das eigene Business von Bedeutung sind. Allerdings ist es für Firmen nicht möglich, sich in konkrete Projekte einzukaufen. Jedes Projekt wird von seinem Board überwacht; Mitglieder werden nur aufgrund ihres eigenen persönlichen Engagements sowie ihrer Leistungen („Meritocracy") in das Projekt aufgenommen und agieren dort auch in eigenem Namen und nicht im Namen einer bestimmten Firma.

Community over Code

Eines der wichtigsten Leitmotive der ASF lautet: „Community over Code". Gemeint ist damit, dass eines der Hauptziele der Aufbau einer aktiven Community rund um das Projekt ist, damit ein Projekt auch langfristig erfolgreich ist und sich etabliert. Die Community in der ASF funktioniert und ist organisiert nach der „Meritocracy"-Philosophie. Das bedeutet bei der ASF, dass die Rolle innerhalb der Community durch die persönlichen Verdienste und Leistungen des Einzelnen bestimmt wird und nicht durch Alter, sozialen Status, Herkunftsland, Einkommen, Bildungsabschlüsse oder andere Kriterien. Das geschieht durch die Definition unterschiedlicher Rollen in den Projekten und in der Foundation selbst.

In einem Projekt fangen die Rollen beim Benutzer an, der sich die Software herunterlädt, verwendet und entdeckt. Die nächste Rolle ist die des Kontributors, der in irgendeiner Weise zum Projekt etwas beiträgt. Das muss nicht zwingend nur Source-Code sein, das können auch Hilfe auf der User-Mailing-Liste, Dokumentation, Logos, Bug-Reports, Feature-Requests und mehr sein.

Wirkt ein Kontributor über eine längere Zeit an einem Projekt mit, wird er meist vom Project Management Committee (PMC) als Committer eingeladen. Damit erhält man eine Apache-User-ID und hat schreibenden Zugriff auf den Source-Code, um eigene Änderungen und Änderungen, die von Kontributoren beigesteuert werden, in den Source-Code einzupflegen.

Wenn das PMC dann ein langfristiges Interesse des Committer am Projekt selbst erkennt, wird dieser üblicherweise nach einer gewissen Zeit auch als PMC-Member zum Projekt eingeladen. Damit bestimmt man über Votings an Projekt-Entscheidungen wie zum Beispiel über neue Committer oder PMC-Member ab, aber auch über Entscheidungen, wie die Freigabe eines neuen Releases oder andere wichtige Projekt-Entscheidungen, die gefällt werden müssen.

Jedes PMC bestimmt auch über Voting einen PMC-Chair, der damit auch gleichzeitig ein Vize-Präsident der Foundation ist. Im Gegensatz zu Firmen ist der PMC-Chair aber nicht derjenige, der bestimmt, wo es in Zukunft langgehen soll, sondern er dient eher als Vermittler zwischen dem Board und dem Projekt, um den aktuellen Status des Projekts oder offene Themen mit dem Board abzustimmen. Seine Stimme in Votings hat ansonsten genauso viel Gewicht wie die der anderen PMC-Member *(siehe Abbildung 3)*.

Die Votings in der ASF oder in einem Projekt selbst finden nach einem ganz einfachen Prinzip statt. Jeder kann auf eine gestellte Entscheidung entweder mit „+1", „0" oder „-1" abstimmen, wobei „+1" eine Zustimmung und eine Unterstützung bedeutet, „0" Enthaltung und „-1" eine Ablehnung, die dann auch begründet sein sollte. Grundsätzlich ist jeder eingeladen, an öffentlichen Votings zum Beispiel auf der Development-Mailing-Liste eines Projekts mit abzustimmen und seine Meinung zu den Entscheidungen mit einzubringen. Allerdings sind nur die Votings der PMC-Member bindend.

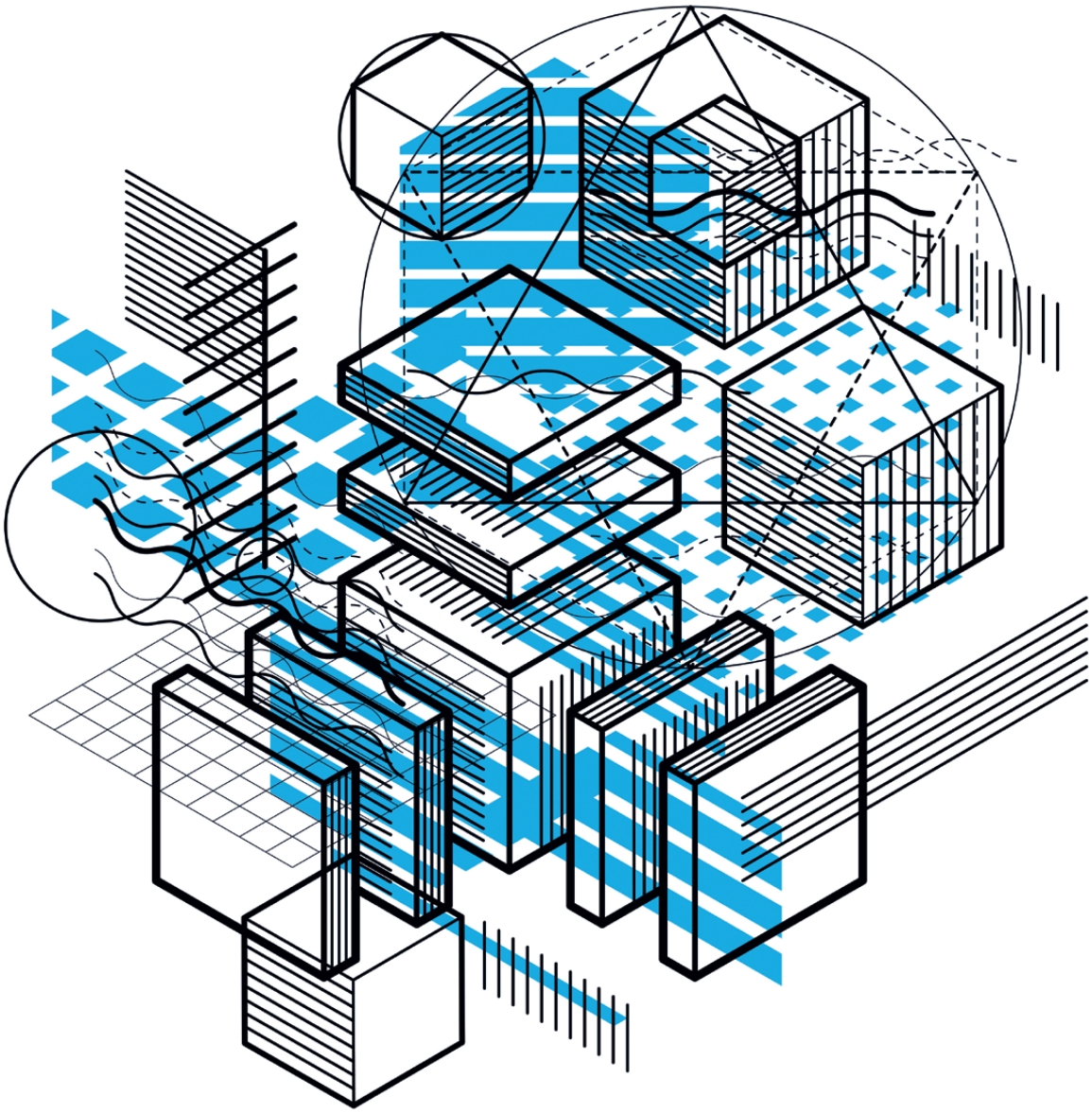
Wer sich über das Projekt hinaus für die Foundation engagiert, also zum Beispiel neuen Projekten beim Etablieren hilft oder mit Ideen und Lösungen die ASF verbessert, kann von ASF-Membern als neues ASF-Member vorgeschlagen werden. ASF-Member sind die Shareholder der Foundation und stimmen in einem jährlichen virtuellen Voting-Prozess über die vorgeschlagenen neuen ASF-Member ab. Aber nicht nur darüber wird abgestimmt, sondern auch, wer im nächsten Jahr die ASF im Board, also zum Beispiel als Präsident oder als Direktor für ein bestimmtes Thema wie „Legal", „Konferenzen", „Finanzen" und andere Bereiche, vertritt. Das Board hat regelmäßige Treffen, um die aktuellen Probleme und Themen zu besprechen, und vor allem, um die einzelnen Projekte auch zu monitoren und bei Problemen zu unterstützen.

Ein etwas anderes, aber dennoch sehr wichtiges Projekt in der ASF ist der Incubator. Das Incubator-Projekt hilft neuen oder gesponserten Projekten beim gesamten On-Boarding-Prozess. Bevor diese ein offizielles ASF-Projekt werden können, müssen gerade bei gesponsertem Source-Code alle rechtlichen Punkte geklärt sein. Zudem muss die Infrastruktur für ein neues Projekt aufgebaut und, natürlich am wichtigsten, eine Community für das Projekt initiiert werden.

Um bei der ASF mitzumachen, sollte man sich also einfach ein Projekt, mit dem man gerne arbeitet und das einen interessiert, heraussuchen, sich dort auf der User- und Development-Mailing-Liste anmelden und sehen, wo man das Projekt unterstützen sowie sich und die eigenen Ideen einbringen kann.

Johannes Geppert
jogep@apache.org

Johannes Geppert arbeitet seit mehr als fünf Jahren als Software Development Engineer für Amazon in Leipzig. Seit dem Jahr 2015 ist er Member der Apache Software Foundation und seit vielen Jahren Committer und PMC-Member (Project Management Committee) des Apache Struts Project.



Modularity in Java – from past to present

Philipp Buchholz

Der Einsatz des OSGi-Frameworks ermöglicht bereits seit Mai 2000 die Entwicklung modularisierter Java-Anwendungen und -Systeme. Seit der Veröffentlichung des JDK 9 bietet Java die Möglichkeit, Module unter Einsatz von Sprachmitteln zu entwickeln. Der Artikel zeigt Einsatz-Szenarien sowie die Vor- und Nachteile der Modularisierung, die Entwicklung modularisierter Anwendungen mit OSGi und dem neuen Java Platform Module System (JPMS) und stellt die Möglichkeiten mit OSGi und JPMS gegenüber.

In der Version 9 des JDK wurde die Sprache Java mit dem JSR-376 *[1]* um umfangreiche Sprachmittel zur Modularisierung erweitert. Hauptziel des eingeführten Java Platform Module System ist die Erleichterung der Entwicklung von großen komplexen Anwendungen. Module unterstützen die Erreichung dieses Ziels über die Realisierung einer starken Kapselung durch das explizite Veröffentlichen und Konsumieren von Typen und die Einschränkung der Sichtbarkeit auf interne, nicht für externe Verwendung vorgesehene Typen. Innerhalb der Spezifikation ist dieser Punkt als „Strong Encapsulation" zusammengefasst.

Ein weiteres großes Ziel ist das Ablösen der bisherigen Classpath-Konfiguration. Vor Version 9 des JDK enthielt der Classpath alle

direkt oder transitiv benötigten Typen. Klassen mit gleichen Namen konnten sich gegenseitig überschreiben; ein Problem, das als „Classpath-Hell" bekannt geworden ist. Dieses Ziel wird als „Reliable Configuration" innerhalb der Spezifikation festgehalten.

Vorteile durch Modularisierung

Der Einsatz von Modularisierung hat Auswirkungen auf bestimmte Qualitäts-Eigenschaften *[2]* von Systemen und Anwendungen. Das Aufteilen des Source-Codes von Anwendungen in klar abgegrenzte Module, die jeweils zusammengehörende Funktionalität implementieren, unterstützt das SRP-Prinzip *[3]* und erhöht die Kohäsion. Das SRP-Prinzip ist eines der wichtigsten Prinzipien bei der Entwicklung von Software-Systemen und Teil des SOLID-Akronyms *[4]*. Es wurde von Robert C. Martin geprägt und stammt ursprünglich aus dem Umfeld der objektorientierten Programmierung, kann jedoch auch auf anderen Abstraktionsebenen eingesetzt werden.

Der Einsatz explizit veröffentlichter Interfaces und Typen erhöht die Kapselung innerhalb einer Anwendung. Die explizite Veröffentlichung von Interfaces und Typen führt auf der einen Seite zu Mehraufwand während der Entwicklung, aber auch zu besser verständlichen und klaren APIs. Dies liegt daran, dass sich Entwickler und Architekten mehr Gedanken über die APIs machen müssen, über die die Module später verwendet werden sollen.

Die bessere Kapselung und das Einführen einer zusätzlichen Sichtbarkeitsebene auf Modulbasis lockert außerdem die Kopplung zwischen Modulen. Ungewollte Abhängigkeiten durch die Verwendung von nicht veröffentlichten und internen Typen werden verhindert. Das unterstützt Qualitäts-Eigenschaften wie Änder-, Erweiter-, und Wartbarkeit. Das resultierende System kann sich aus diesen Gründen erheblich besser an geänderte Rahmenbedingungen anpassen. Neue oder geänderte funktionale und nicht funktionale Anforderungen können einfacher und schneller integriert werden. Im Idealfall können Module durch neue Versionen oder Neuentwicklungen dieser Module getauscht werden, ohne dass andere Module beeinflusst werden. Werden Module entlang fachlicher Domänengrenzen implementiert, wird die fachliche Kohäsion gestärkt.

Nachteile durch Modularisierung

Allerdings hat das Modularisieren des Source-Codes auch einige Nachteile. Vor allem die im Vergleich zu herkömmlichen Codebasen gestiegene Komplexität im Hinblick auf die Systemstruktur muss beherrscht werden und das notwendige Wissen der jeweiligen Modularisierungstechniken und Frameworks vorhanden sein. Wenn Erfahrungen mit der Business-Domäne fehlen, besteht die Gefahr, dass anfangs unzureichende Schnittstellen und damit ein unzureichender Modulschnitt gewählt wird, der später nur schwer geändert werden kann.

Es ist also immer zwischen den Vor- und Nachteilen abzuwägen, um die richtige Entscheidung für oder gegen Modularisierung zu treffen. Hier ist immer zu bedenken, dass projektspezifische Rahmenbedingungen in technischer und organisatorischer Dimension eine entscheidende Rolle spielen.

Um die Vorteile von Modularisierung nutzen zu können, müssen der fachliche Kontext und damit die Anforderungen komplex genug sein, um von diesen profitieren zu können. Bei einer kleinen Applikation, die nur von ein paar wenigen Entwicklern erstellt und weiterentwickelt wird, können diese Vorteile nicht genutzt werden und die Nachteile machen sich störend bemerkbar. Besteht jedoch die realistische Aussicht, dass die Code-Basis in absehbarer Zeit komplexer und umfangreicher wird, sollten über das saubere Definieren von Interfaces bereits Grundlagen für den späteren Einsatz von Modularisierung geschaffen werden. Handelt es sich jedoch um eine Applikation, die von mehreren Teams entwickelt wird, können die durch den Einsatz der Modularisierung resultierenden Vorteile von Anfang an genutzt werden.

Szenario 1: Modernisierung von Legacy-Anwendungen

Auch bei der Modernisierung von Legacy-Anwendungen kann Modularisierung als Teil einer Transformationsstrategie von einer monolithischen in eine flexiblere Architektur helfen. Im Umfeld der Enterprise-Software-Entwicklung trifft man regelmäßig auf jahrealte Systeme, die starr und fragil geworden sind. Oftmals handelt es sich um Codebasen ohne klare Verantwortungsverteilung und ohne klar definierte Schnittstellen. Viele Unternehmen stehen bei diesen Systemen vor der Frage, wie eine schrittweise Transformation in ein System funktionieren kann, das eine evolutionäre Weiterentwicklung und das Reagieren auf geänderte Anforderungen wirtschaftlich zulässt.

In solchen Fällen können nach einer Analyse des Quellcodes und mit einem wachsenden Verständnis für die geschäftlichen Zusammenhänge fachlich orientierte Schnittstellen zur Entkopplung unterschiedlicher Anwendungsteile einbezogen werden. An diesen Schnittstellen kann im zweiten Schritt die Überführung in getrennte fachlich orientierte Module erfolgen. Diese kommunizieren ausschließlich über ihre sauber definierten und explizit veröffentlichten Schnittstellen miteinander. Diese Schritte werden wiederholt, bis eine ausreichende Modularisierung der Codebasis erreicht ist.

Anhand des folgenden Beispiels eines monolithischen Shop-Systems wird die Vorgehensweise skizziert. Im gezeigten Fall wurde die Business-Logik in einem Package zusammengefasst, das zwecks Strukturierung weitere Packages enthält. Diese Package-Hierarchie zeigt folgendes Beispiel der Übersichtlichkeit zuliebe nicht *(siehe Abbildung 1)*.

In diesem Szenario wird die Business- aber auch die Persistenz-Logik entlang fachlicher Grenzen in Module getrennt. Vorbereitend wurden aus fachlicher Sicht eine Payment- und eine Order-Domäne identifiziert, die jetzt in zwei getrennten Modulen implementiert werden. Um die Funktionalität der Module verfügbar zu machen, werden die entsprechenden fachlichen Schnittstellen explizit veröffentlicht. Dabei unterscheidet man zwischen einer uneingeschränkten (public) und einer eingeschränkten (internal) Veröffentlichung. Diese zusätzliche Sichtbarkeitsebene steuert, dass bestimmte Schnittstellen nur innerhalb des Shop-Subsystems (internal) und andere auch von anderen Subsystemen verwendet werden können (public). Dies kann zum Beispiel durch die Einschränkung der Verwendung auf bestimmte Packages realisiert werden *(siehe Abbildung 2)*.

Nachdem Module mit expliziten Schnittstellen eingeführt wurden, sind beide fachlichen Domänen sauber gekapselt, die oben be-

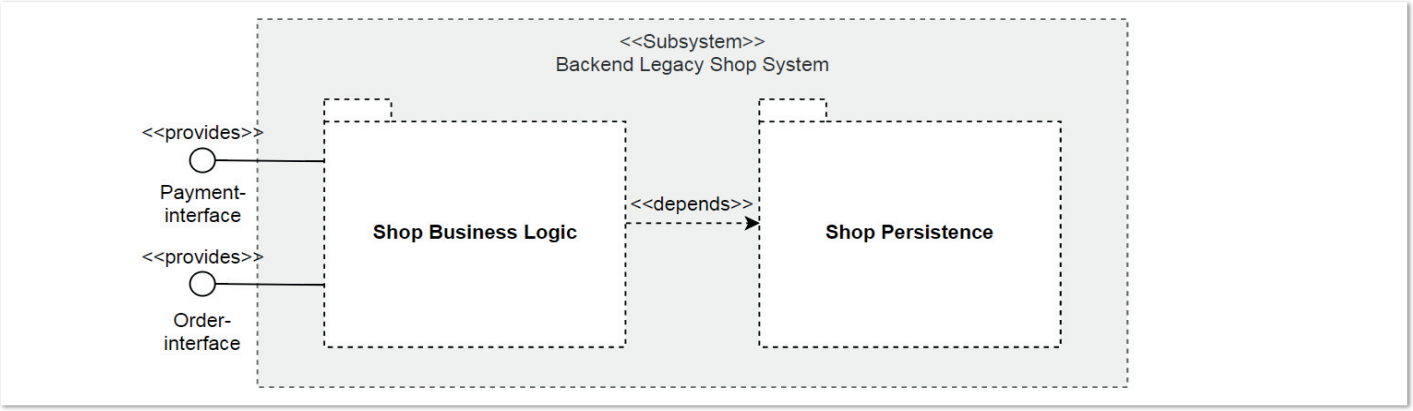


Abbildung 1: Nicht modularisiertes Subsystem

schriebenen Konzepte werden damit nutzbar. Die folgenden Abschnitte zeigen die Implementierung dieser Konzepte detailliert in der Praxis.

Oft kämpft man an dieser Stelle mit Abhängigkeiten von externen Libraries oder auch Typen beziehungsweise Klassen, die nach der Aufteilung des Systems von mehreren Modulen benötigt werden. Eine Lösung ist die Erstellung von API-Modulen beziehungsweise -Projekten, die von beiden Modulen verwendet werden können. Es ist wichtig, dass solche API-Artefakte nur Schnittstellen- und DTO-Typen *[5]* enthalten und keine Business-Logik. Die Business-Logik und damit die Interface-Implementierungen dürfen sich ausschließlich in den Kernmodulen befinden und werden nicht zur Verwendung veröffentlicht. Damit wird eine saubere Kapselung sichergestellt. Iterativ bilden sich so ein klares API und klare Verantwortlichkeiten der einzelnen Module aus der bestehenden Legacy-Codebasis heraus.

Szenario 2: Transition zu einer Microservices-Architektur

Modularisierung kann auch ein Mittel sein, um die Transition zu einer flexiblen Microservices-Architektur zu unterstützen. Bei dieser Vorgehensweise wird im ersten Schritt ein monolithisches System entwickelt, das mit wachsendem fachlichen Hintergrundwissen und Verständnis in Module aufgeteilt wird. So entstehen nach einiger Zeit fachlich zusammenhängende Module mit sauberen Schnittstellen. Anfangs unterliegt ein so entstehendes System vielen Änderungen und sich wandelnden Schnittstellen. Mit der Zeit stabilisieren sich die Schnittstellen und die fachlichen Module im Hinblick auf deren Änderungshäufigkeit. Zu diesem Zeitpunkt können die fachlichen Module in eigene Microservices überführt werden.

Dieser Schritt ist aufgrund der bereits vorhandenen Schnittstellen weniger aufwendig; die fachlichen Domänengrenzen und Verantwortlichkeitsverteilungen wurden bereits relativ klar herausgearbeitet. Dieses Vorgehen entspricht dem bereits durch Martin Fowler beschriebenen Ansatz des Monolith First *[6]*, erweitert durch das Konzept der Modularisierung.

Modularisierung unter Einsatz von OSGi

OSGi ist ein dynamisches Modul-System für das Entwickeln und Betreiben modularer Java-Anwendungen. OSGi liegt aktuell in der Version 7 *[7]* vor. Aufbauend auf Java bietet OSGi eine Laufzeit-Umgebung für Module, die „Bundles“ genannt werden. Diese werden

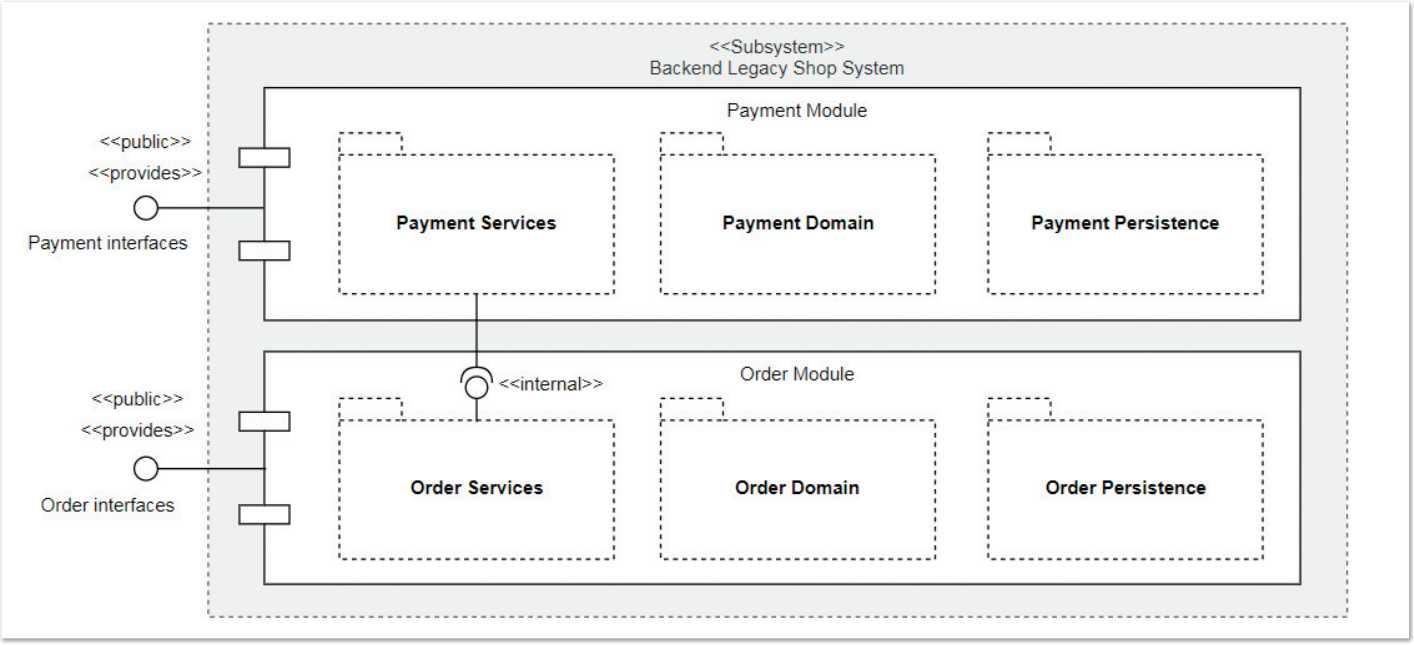


Abbildung 2: Modularisiertes Subsystem

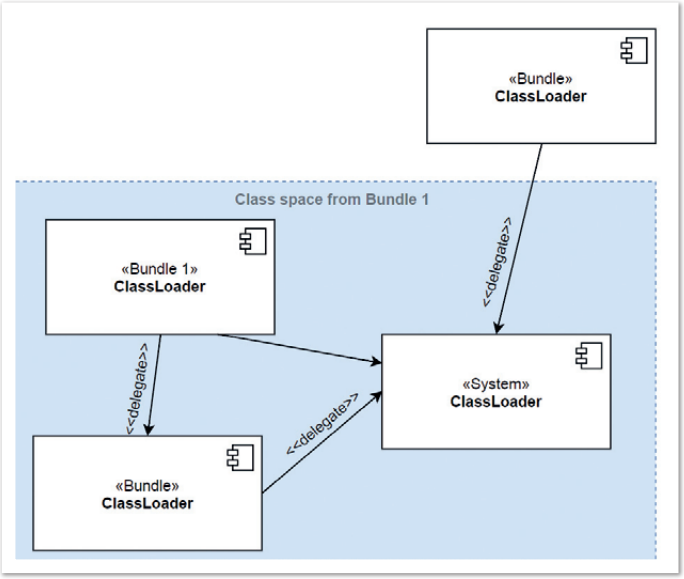


Abbildung 3: Class-Spaces in OSGi

von der Laufzeit-Umgebung gemäß einem definierten Lifecycle *[8]* verwaltet. Innerhalb dieses Lifecycle werden Bundles installiert, gestartet, gestoppt und außer Betrieb genommen. Bundles sind versioniert, was einen Betrieb in unterschiedlichen Versionen und damit Codeständen ermöglicht. Die Abhängigkeiten zu anderen Bundles sind über Meta-Informationen beschrieben. Hier kann bestimmt werden, welche Versionen erforderlich sind.

Ein Modul in OSGi ist ein „Jar"-File, das neben den Class-Files auch ein beschreibendes Manifest im „META-INF"-Verzeichnis enthält. Header beschreiben dort die notwendigen Meta-Informationen. Diese werden von OSGi zur Laufzeit des Bundle ausgewertet. *Listing 1* zeigt ein rudimentäres Beispiel.

Der Header „Bundle-Name" definiert einen lesbaren Namen für das Bundle, während der Header „Bundle-SymbolicName" einen eindeutigen Namen für das Bundle enthält und zwingend vorhanden sein muss. „Bundle-ManifestVersion" zeigt die Version von OSGi, die zum Lesen des Bundle verwendet werden soll. Mit der Angabe von „Bundle-Version" wird das komplette Bundle versioniert; OSGi verwendet grundsätzlich semantische Versionierung *[9]*. Details sind in der offiziellen Spezifikation *[10]* beschrieben.

Über „Export-" und „Import-Package" sind die veröffentlichten beziehungsweise benötigten Packages deklariert, im Idealfall spezielle API-Packages, die nur Interfaces beziehungsweise Typen für die Verwendung der gekapselten Funktionalität enthalten. Durch Kommas getrennt, können in diesen Headern mehrere Packages definiert sein. Außerdem ist am Ende, getrennt durch ein Semikolon, die Angabe weiterer Meta-Informationen in Form von Headern möglich.

```
Bundle-Name: ecommerce-payment
Bundle-SymbolicName: ecommerce.payment
Bundle-ManifestVersion: 2
Bundle-Version: 0.0.1.qualifier
Export-Package: ecommerce.payment.api
Import-Package: ecommerce.customer.api
```

Listing 1: Manifest-File für ein Bundle

So kann beispielsweise für exportierte und importierte Packages ebenfalls eine Version hinterlegt sein. Ergänzend zur Versionierung des Bundle lassen sich damit einzelne Packages versionieren. Dies ermöglicht eine feingranulare Kontrolle über die Codestände, die von einzelnen Bundles angezogen werden.

Classloading Architecture

Die OSGi-Runtime lädt die Klassen von Bundles nach einer genau definierten Classloading Architecture *[11]*. Jedes Bundle wird von einem separaten „ClassLoader" geladen, der zusammen mit anderen eine Hierarchie bildet. Innerhalb dieser Hierarchie lässt sich das Laden von Klassen delegieren. So lädt der „ClassLoader" des Bundle die Klassen des zugrunde liegenden „Jar"-Files. Das Laden von Klassen, die aus anderen Bundles konsumiert und dort veröffentlicht werden, wird an die „ClassLoader" dieser Bundles delegiert. So formt sich ein Class-Space, der die vom Bundle erreich- und nutzbaren Klassen und Typen enthält *(siehe Abbildung 3)*.

Innerhalb eines Class-Space müssen die vollqualifizierten Namen von Klassen eindeutig sein; allerdings kann ein anderer Class-Space eine gleichnamige Klasse enthalten. Über die Bundle-Versionierung wird damit das Betreiben unterschiedlicher Versionen einer Klasse zur Laufzeit möglich. Durch die Class-Spaces sind die Klassen sauber voneinander getrennt. Die hier entstehende Hierarchie hängt natürlich von den Abhängigkeiten ab, die in den beteiligten Modulen, bei OSGi Bundles, definiert wurden.

Services

OSGi bietet in Ergänzung zu Bundles sowie dem expliziten Veröffentlichen und Konsumieren von Packages eine Unterstützung für Services. Services sind bei OSGi Java-Objekte, die ein wohl definiertes Interface implementieren. Dieses wird zusammen mit der zu verwendenden Implementierung innerhalb einer „ServiceRegistry" veröffentlicht und kann so von Bundles verwendet werden.

„ServiceTracker" *[12]* ermöglichen das Lauschen auf Lifecycle-Events, die von der OSGi-Runtime ausgelöst werden. So wird ein Bundle automatisch benachrichtigt, wenn ein Service zur Verfügung steht, und kann sein Verhalten entsprechend anpassen. Wenn man sich als Beispiel einen Persistenz-Service vorstellt, der Zustände und Daten persistiert, dann könnte ein Service-Konsument Daten so lange cachen, bis der eigentliche Persistenz-Service verfügbar ist. Sobald dieser aktiv ist, könnten die Daten vom Cache in den persistenten Speicher überführt werden.

Zusätzlich lassen sich zu einem Service-Interface auch unterschiedliche Implementierungen innerhalb der Registry veröffentlichen. Über beschreibende Properties werden die Implementierungen unterschieden. Je nach Bedürfnis des Bundle kann dann die passende Implementierung selektiert werden.

Ergänzend dazu bietet OSGi sogenannte „Declarative-Services" *[13]*, um Service-Instanzen in die konsumierenden Bundles zu injizieren. Die Deklaration und Injizierung dieser Service-Instanzen erfolgt mit Annotationen. Das Vorgehen ist ähnlich wie in einigen Dependency-Injection-Containern.

Es stehen einige Implementierungen der OSGi-Spezifikation zur Verwendung bereit, beispielsweise Knopflerfish *[14]* oder Apache

```
module ecommerce.payment {
  requires java.xml.bind;
  exports ecommerce.payment.api;
}
```

Listing 2: „module-info.java" für das „ecommerce.payment"-Modul

Felix *[15]*. Eine der bekanntesten Alternativen ist Eclipse Equinox *[16]*. Equinox ist die offizielle Referenz-Implementierung und fungiert darüber hinaus als Basis für die Rich Client Platform (RCP) und damit auch als Basis der Entwicklungsumgebung Eclipse selbst.

Fazit zu OSGi

OSGi ist ein sehr komplexes und dadurch auch mächtiges dynamisches Modul-System für die Entwicklung modularer Anwendungen in Java. Über die beschriebenen umfangreichen Möglichkeiten der Versionierung auf Ebene von Bundles und Packages ist das Realisieren komplexer Betriebs-Szenarien möglich. So können damit beispielsweise Zero-Downtime-Updates in Form von BlueGreen-Deployments *[17]* realisiert werden. Dazu wird eine neue Version eines Bundle oder eines Packages parallel zur alten Version in Betrieb genommen und anschließend getestet. Sollte das Bundle wie erwartet funktionieren, werden die Versionsangaben der konsumierenden Bundles auf das neue Bundle umgestellt und anschließend die alten Versionen außer Betrieb genommen. Alle Vorgänge lassen sich zur Laufzeit über Management-Agents durchführen.

Die beschriebenen Möglichkeiten über Services ermöglichen das Erstellen hochdynamischer Systeme. Sie führen allerdings auch zu einer erhöhten Komplexität und einer steilen Lernkurve.

Modularisierung mit JPMS

Das Java Platform Module System (JPMS) beschreibt Module über eine spezielle Java-Klasse mit dem Namen „module-info.java". An dieser Stelle zeigt sich bereits der erste Unterschied zu OSGi: Bei OSGi sind die Meta-Informationen, die zur Modularisierung erforderlich sind, mit einer Konfigurationsdatei beschrieben, bei JPMS wird eine Java-Klasse verwendet. Diese liegt zur Laufzeit in kompilierter Form als Bytecode vor. *Listing 2* zeigt ein erstes Beispiel einer solchen Klasse.

Über das „module"-Konstrukt wird ein neues Modul und mit dem „requires"-Keyword werden Run- und Compiletime-Abhängigkeiten zu anderen Modulen deklariert, während „exports" die Packages zur Verwendung durch andere Module veröffentlicht. Im obigen Beispiel bedeutet das, dass das Modul „ecommerce.payment" das Package „ecommerce.payment.api" und die enthaltenen Typen zur Verwendung an andere Module exportiert. Ein JPMS-Modul ist ein „Jar"-File, das neben den Klassen auf Root-Ebene die beschriebene „module-info"-Klasse enthält.

Modularisierung der Java-Plattform

JPMS ermöglicht nicht nur die Erstellung modularer Java-Anwendungen, sondern modularisiert das JDK selbst. Bis Version 9 gab es zwar unterschiedliche Distributionen des JDK – jeweils für unterschiedliche Einsatzzwecke, es handelte sich jedoch um relativ große, unveränderliche Distributionen. Mit der Modularisierung des JDK können genau die Module zu einer Laufzeit-Umgebung zusammengestellt werden, die notwendig sind. Es entstehen Applikationen die nicht mehr das gesamte JDK, sondern nur einzelne Module beinhalten. Die JavaDocs *[18]* wurden so erweitert, dass die Module, in denen sich Packages und Klassen befinden, ersichtlich sind.

Damit eine modularisierte Anwendung immer Zugriff auf Basisfunktionalität der Java-Plattform enthält, ist das „java.base"-Modul implizit immer vorhanden. Details der hier enthaltenen Funktionalität stehen im JavaDoc *[19]*.

Auflösen von Modul-Abhängigkeiten

Zur Auflösung der Modul-Abhängigkeiten konstruiert das JPMS einen Graph. Jede Kante des Graphen führt zu einem Modul. Das Modul-System stellt zur Compile-Zeit sicher, dass jedes benötigte Modul und jeder importierte Typ erreichbar ist.

Während früher die gesamten Klassen, die zur Laufzeit notwendig waren, im Classpath *[20]* lagen, lädt das Java Platform Module System Module aus einem ModulePath. Alle Module, die dort abgelegt sind, werden während der Auflösung der benötigten Module verwendet. Bei Verwendung des Classpath ist nicht sichergestellt, dass eine Klasse mit ihrem vollqualifizierten Namen nur einmal vorhanden ist. Somit waren gleichbenannte Klassen mehrfach möglich; die erste durch das ClassLoader-System gefundene Klasse wurde verwendet. Das führte zu Änderungen des Systemverhaltens, die teilweise gewollt, aber meistens ungewollt waren.

Innerhalb des ModulePath stellt JPMS sicher, dass ein Modul nur einmal vorhanden ist. Als identifizierendes Merkmal dient auch hier der Name. Sollte ein benötigtes Modul nicht aufgelöst werden können, weil es nicht oder nicht eindeutig vorhanden ist, wird ein Error geworfen und die Ausführung abgebrochen. Doppelte Module können nicht vorhanden sein. Damit ist garantiert, dass ein veröffentlichter Typ nur einmal vorhanden und einem Modul eindeutig zugeordnet ist.

Benötigt eine Anwendung weiterhin nicht modularisierte Abhängigkeiten, werden diese weiterhin im Classpath abgelegt. Alle diese Abhängigkeiten fasst das Java Platform Module System dann zu einem „Unnamed Module" zusammen und stellt sie allen Modulen zur Verfügung.

Transitive Abhängigkeiten

Oft kommt es vor, dass von einem importierten Modul weitere Module importiert werden, die das ursprüngliche Modul nicht direkt referenziert. *Abbildung 4* zeigt das am Beispiel des E-Commerce-Systems.

Im Beispiel exportiert das Modul „ecommerce.logistic" den Typ „LogisticService". Dieser bietet eine Methode zum Registrieren eines Event-Subscriber. Dazu gibt die Methode den Typ „ecommerce.event.EventSubscriber" zurück. Dieser wird aber durch das Modul „ecommerce.event" bereitgestellt und befindet sich nicht direkt im „ecommerce.logistic"-Modul. Das „payment"-Modul kann aus diesem Grund den EventSubscriber-Typ nicht direkt verwenden. Dieser ist gemäß den Sichtbarkeitsregeln nicht zugreifbar.

Um zu vermeiden, dass jedes Modul alle benötigten transitiven Abhängigkeiten manuell importieren muss, gibt es das „transitive"-Schlüsselwort, das in der entsprechenden Deklaration verwendet

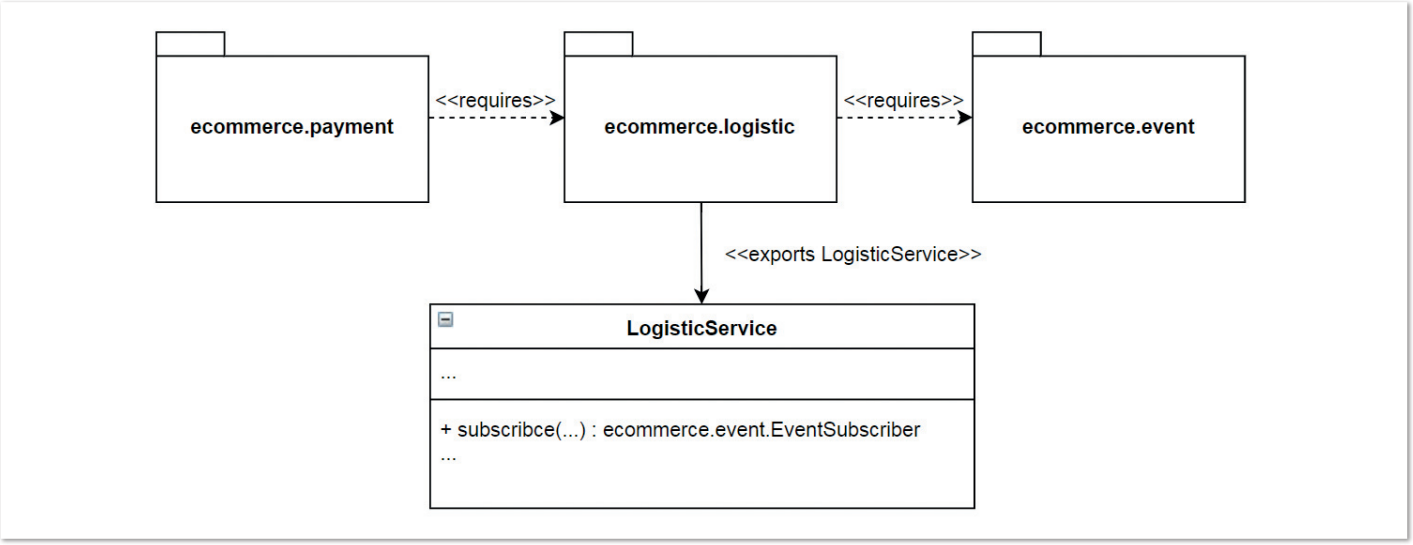


Abbildung 4: Transitive Abhängigkeiten zwischen Modulen

```
module ecommerce.payment {
  requires java.xml.bind;
  requires transitive ecommerce.event;

  exports ecommerce.payment.api;
}
```

Listing 3: „module-info.java" für das „ecommerce.payment"-Modul

```
module ecommerce.order {
  ...
  exports ecommerce.order.api;

  uses ecommerce.payment.CancellationService;
}
```

Listing 4: Konsumieren eines Service

werden kann *(siehe Listing 3)*. Durch das „transitive"-Schlüsselwort exportiert das „ecommerce.payment"-Modul nicht nur seine Packages, sondern gibt den Export der Packages von „ecommerce.event" auch an seine direkten Konsumenten weiter.

Services im „JavaPlattformModuleSystem"

Bis jetzt wurde beschrieben, wie man Abhängigkeiten zu Modulen deklariert und wie man bestimmte Packages von Modulen konsumiert und veröffentlicht. Zusätzlich bietet JPMS – genauso wie OSGi – eine Service-Abstraktion. In diesem Fall wird ein Service immer durch ein Interface beschrieben, das das API zur Nutzung der Funktionalität beschreibt. Im Idealfall handelt es sich um eine Fach-Funktion, die über ein Interface angeboten ist.

Möchte ein Modul einen bestimmten Service nutzen, drückt er das über das „uses"-Schlüsselwort in der „module-info"-Klasse aus. Damit ein Service nutzbar ist, muss er durch ein anderes Modul zur Verfügung gestellt werden. Dieses Modul nimmt hier die Rolle eines „ServiceProvider" ein und drückt dies ebenfalls in seiner „module-info"-Klasse über die Schlüsselwörter „provides" und „with" aus. *Listing 4* zeigt das Konsumieren eines „CancellationService", der für das Stornieren von Zahlungen zuständig ist, und *Listing 5* die Service-Bereitstellung in Form einer Implementierung des „CancellationService".

Über das Veröffentlichen der Service-Interfaces und bestimmter Implementierungen lässt sich die Kapselung von Modulen nochmals erhöhen. Service-Interfaces müssen sich nicht in exportierten Packages befinden; so lassen sich gezielt einzelne Services exportieren und das Veröffentlichen von kompletten Packages wird reduziert.

Eigene Laufzeit-Umgebungen für Java-Anwendungen

Mit den beschriebenen Neuerungen ist es möglich, eigene Laufzeit-Umgebungen zusammenzustellen, die nur die Module enthalten, die die Anwendung benötigt. Für diesen Einsatzzweck wird das JLink-Tool *[21]* zur Verfügung gestellt. *Listing 6* zeigt ein Beispiel. Es erzeugt im Ordner „ecommerceruntime" eine Laufzeit-Umgebung, die neben „java.base" und abhängigen Modulen das Modul „ecommerce.base" enthält. Dieses könnte beispielsweise Basis-Funktionalität für alle E-Commerce-Anwendungen liefern. Auf Basis dieser Laufzeit-Umgebung sind dann andere E-Commerce-Anwendungen ausführbar.

Fazit JPMS

Das Modulsystem, das mit Java 9 eingeführt wurde, ermöglicht eine signifikant höhere Kapselung von fachlichen Anwendungsteilen. Endlich gibt es innerhalb der Plattform die Möglichkeit, den Zugriff auf bestimmte Interfaces und Implementierungen einzuschränken und nur bestimmte Typen für die Verwendung freizugeben. Unter Verwendung von Services lässt sich dieser Zugriff feingranular steuern.

Einzelne Service-Interfaces und deren Implementierungen werden gezielt veröffentlicht. Durch die Beschreibung der Meta-Informationen innerhalb einer Klasse lässt sich zur Compile-Zeit prüfen, ob alle notwendigen Abhängigkeiten erfüllt werden können. Auch die Definition eigener Laufzeit-Umgebungen ist ein mächtiges Feature.

Vergleich JPMS und OSGi

Ein Vergleich zwischen den beiden Modul-Systemen fällt schwer, da die Ansätze unterschiedlich sind. OSGi bietet die Versionierung von

```
module ecommerce.payment {
  ...
  exports ecommerce.payment.api;

  provides ecommerce.payment.CancellationService with ecommerce.payment.cancellation.DefaultCancellationService;
}
```

Listing 5: Bereitstellen einer Service-Implementierung

```
jlink --module-path $JAVA_HOME/jmods:ecommercebase
--add-modules ecommerce.base --output ecommerceruntime
```

Listing 6: Erstellen einer custom JRE

Bundles und die Möglichkeit, diese zur Laufzeit auszutauschen und neue Versionen einzurichten, um ein hoch dynamisches System zu realisieren. Auch unterliegen Bundles innerhalb der Runtime einem festen Lebenszyklus, in dem Events ausgelöst werden, die wieder von anderen Bundles für unterschiedliche Zwecke empfangen werden können.

OSGi bietet aber nicht die Modularisierung des JDK selbst und die Erstellung eigener Laufzeit-Umgebungen. Hier liegt eine große Stärke des JPMS. Der Autor denkt, dass die Zukunft interessante Weiterentwicklungen in beiden Welten bringen wird. Auch die Kombination der Stärken beider Ansätze ist aus seiner Sicht in Zukunft denkbar. So könnten dann Bundles mit einer modularisierten Laufzeit-Umgebung mit nur den Modulen gestartet werden, die die Bundles benötigen, dann jedoch zur Laufzeit die Versionierung und die anderen Möglichkeiten von OSGi nutzen.

Referenzen

[1] Details zum JSR-376: *https://openjdk.java.net/projects/jigsaw/spec*

[2] Oftmals bekannt unter dem Begriff „nicht funktionale Anforderungen“: *https://en.wikipedia.org/wiki/Non-functional_requirement*

[3] SRP steht für Single Responsibility Principle und ist Teil des SOLID-Akronyms, das 2000 von Robert C Martin in seinem Paper „Design Principles and Design Pattern“ eingeführt wurde

[4] Details zum SOLID-Akronym: *https://de.wikipedia.org/wiki/Prinzipien_objektorientierten_Designs#SOLID-Prinzipien*

[5] DTO steht für Data Transfer Object und ist ein Design Pattern für TODO

[6] Monolith First Pattern von Martin Fowler: *https://martinfowler.com/bliki/MonolithFirst.html*

[7] Spezifikation von OSGi Release 7: *https://osgi.org/specification/osgi.core/7.0.0*

[8] Lifecyle Management von OSGi: *https://osgi.org/specification/osgi.core/7.0.0/framework.lifecycle.html*

[9] Semantische Versionierung: *https://semver.org/lang/de*

[10] Versionierung in OSGi: *https://osgi.org/specification/osgi.core/7.0.0/framework.module.html#i2655136*

[11] Classloading Architecture von OSGi: *https://osgi.org/specification/osgi.core/7.0.0/framework.module.html#i3174728*

[12] Tracker-Spezifikation von OSGi: *https://osgi.org/specification/osgi.core/7.0.0/util.tracker.html*

[13] Declarative-Services-Spezifikation von OSGi: *https://osgi.org/specification/osgi.cmpn/7.0.0/service.component.html*

[14] Knopflerfish-OSGi-Implementierung: *https://www.knopflerfish.org*

[15] Apache-Felix-OSGi-Implementierung: *http://felix.apache.org*

[16] Eclipse-Equinox-OSGi-Referenzimplementierung: *http://www.eclipse.org/equinox*

[17] Blue Green Deployment Pattern von Martin Fowler: *https://martinfowler.com/bliki/BlueGreenDeployment.html*

[18] JDK 9 JavaDoc: *https://docs.oracle.com/javase/9/docs/api/overview-summary.html*

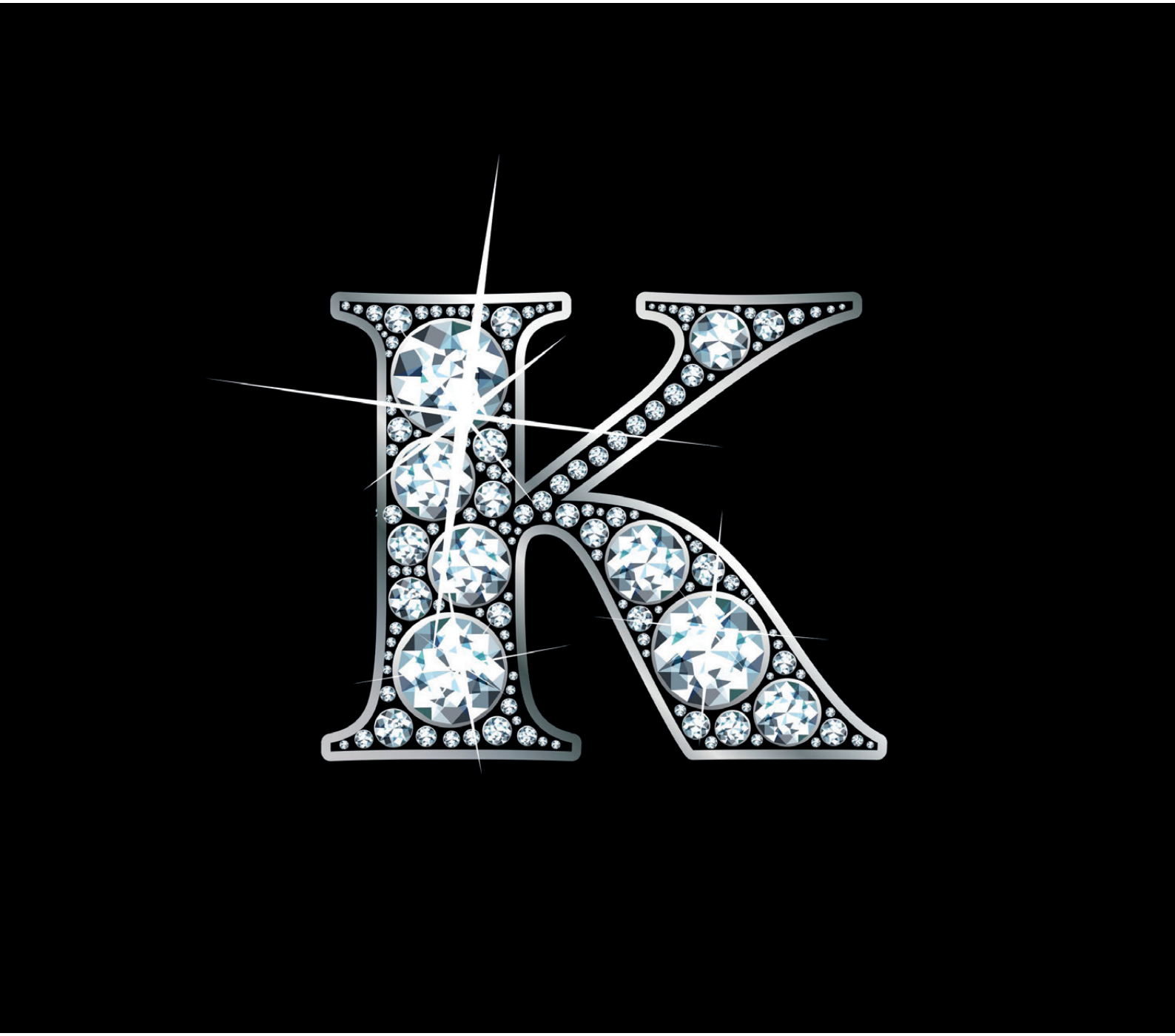
[19] Java.base Module JavaDoc: *https://docs.oracle.com/javase/9/docs/api/java.base-summary.html*

[20] Java Classpath: *https://docs.oracle.com/javase/tutorial/essential/environment/paths.html*

[21] Dokumentation zu JLink: *https://docs.oracle.com/javase/9/tools/jlink.htm#JSWOR-GUID-CECAC52B-CFEE-46CB-8166-F17A8E9280E9*

Philipp Buchholz
philipp.buchholz@gmx.net

Philipp Buchholz ist Executive Consultant bei einem der größten Beratungshäuser weltweit. Er hilft Kunden als Software- und System-Architekt bei der IT-Modernisierung und dem Aufbau von neuen, flexiblen System-Architekturen auf Cloud-Plattformen und -Technologien. Sein aktueller Fokus liegt auf Transformationsstrategien monolithischer Architekturen hin zu flexiblen Architektur-Formen und dem Aufbau ereignisgetriebener Architekturen.



Eine Reise vom Dokument zum Editor

Hannes Niederhausen

Diese Geschichte handelt von einem Buchstaben, dem großen K, der sich gerade fröhlich in Zeile elf des Dokuments umschaute und seine Nachbarn unterhielt.

„Ich finde ja, dass der Kerl rechts von mir echt positiver sein könnte", sagte das K zum benachbarten i, allerdings so laut, dass es über das r bis zum a durchdrang. Der Kerl links vom K war ein Anführungszeichen. „Was soll denn so positiv sein? Ich bin umgeben von einer Nervensäge und einem Haufen Nichts", sagte es.

Mit „Nichts" waren die vier Leerzeichen gemeint, die vor dem Anführungszeichen lagen. „Ey! Das ist voll gemein!", beschwerten die sich und das K stimmte zu. „Nur weil man etwas nicht sehen kann, heißt das nicht, dass es nicht da ist", philosophierte es und die Leerzeichen gaben ihm lautstark recht. Das verbesserte in keiner Weise die Laune des Anführungszeichens, doch das war dem K nun egal, denn es hatte neue Freunde gefunden. Es musste sich nämlich eingestehen, dass es die Leerzeichen bisher auch nicht mitbekommen hatte. Dabei konnte es so gut über das Anführungszeichen sehen.

Sie schnatterten die ganze Zeit über dies und das, als plötzlich ein lautes Alarmsignal durch das Dokument tönte: „Achtung, alle aufpassen. Begebt euch langsam in den Tunnel. Einer nach dem anderen, habt ihr das verstanden?"

Die Frage war natürlich obsolet, denn niemand wusste, wem sie denn antworten sollten. Aber so richtig verstand das K die ganze Sache nicht. „Was denn für ein Tunnel?", fragte es in die Runde, doch seine Mitzeichen schüttelten alle nur den Kopf. Das K sah nach oben, die erste Zeile konnte es nur vage sehen, aber dort bewegten sich die Zeichen alle zum Zeilenanfang. Als die erste Zeile völlig verschwunden war, bewegte sich die nächste. Und die nächste.

Zehn Zeilen später schrie das erste Leerzeichen auf: „Das Loch! Es ist hier!" Das Leerzeichen nahm Schwung und sprang hinein, die anderen folgten, und schon war das Anführungszeichen an der Reihe. „Das gefällt mir nicht", sagte es. „Das ist doch sicher eine Falle. Was, wenn wir nicht alle wieder zurückkommen?"

„Wir haben keine Wahl!", rief das i.

Das K konnte die Bedenken gar nicht verstehen.

„Das wird sicher super! Los, spring schon!", forderte es das Anführungszeichen auf, und als das immer noch keine Anstalten machte, stieß das K es einfach hinein und sprang hinterher.

***

Im Loch war es absolut dunkel, sie fielen, das spürte das K, doch sehen konnte es nichts. „Wir werden sterben!", jammerte das Anführungszeichen. „Juchhu!!!", schrie das K. Ein starker Aufwind verlangsamte ihren Fall und dann, ganz plötzlich, wurden sie seitlich angesaugt.

Plopp! Plopp! Plopp!

Nach und nach landeten sie auf einem Förderband. „Ich kann wieder sehen", sagte das K und bemerkte, dass es lieber in die Dunkelheit zurückwollte. Alles um sie herum war dreckig und es stank nach Öl und Abgasen. Die Zeichen husteten ein paar Mal, bis sie sich an die Luft gewöhnt hatten.

„Alles bleibt stehen, wo es landet, bis ich es mir angesehen habe!", hörten sie eine quäkige Stimme. Das K lehnte sich über den Rand des Laufbands und sah ungefähr achtzig Zeichen vor sich eine schwarze Plattform, auf der ein unförmiges Monster saß.

„Was ist das denn?", fragte das K. „Das sieht voll gruselig aus", meinte das Anführungszeichen. Das Ding wirkte wie ein riesiger, vierarmiger Blob, auf dem ein Kopf hauste. Je näher sie ihm kamen, desto mehr zitterten die Zeichen.

„Was wird es mit uns machen?", fragte das i neben dem K. „Sieht so aus, als sucht er sich ein paar Zeichen aus und schlägt sie einfach so", meinte das dritte Leerzeichen.

Das riesige Monster entpuppte sich als zwei Personen. Eine große Person mit unförmigem Körper, auf deren Schulter eine kleine Person saß und mit einem Klemmbrett wedelte.

Nun befand sich das erste der vier Leerzeichen vor den beiden. „Was bist du?", fragte es mit zitternder Stimme. „Lexer", sagte der Große knurrend. „Und was ist ein Lexer?", fragte das K. „Wirste schon sehen", sagte der Kleine und dann: „Ran!"

Der Große nahm eine Kette, an deren Enden Schellen angebracht waren, und verband das erste mit dem zweiten Leerzeichen. „Ran", wiederholte er beim nächsten Leerzeichen und auch beim vierten.

Erst als das Förderband das Anführungszeichen vor dem Lexer platzierte, reichte der kleine Mann dem großen einen Zettel. „WS" stand darauf geschrieben.

„Ah, ein Whitespace", kommentierte der, klatschte den Zettel auf das letzte Leerzeichen und wandte sich dem Anführungszeichen zu. „Immer das Gleiche", murmelte der große Teil des Lexer, als er das Anführungszeichen mit dem K verband.

„Hey Moment mal, was soll das denn?", meckerte das K, wurde jedoch vom Lexer einfach ignoriert. Nicht nur das, jetzt war das i an ihn gefesselt. Das r und das a waren ebenfalls Teil der Kette, die erst mit dem Ausführungszeichen hinter dem a endete.

Klatsch! Nun war auch ihre Gruppe beschriftet. „Was steht da?", fragte das K das r. „Da steht STRING." String. Was soll das denn bedeuten? Wo waren sie nur hineingeraten?

***

Das Fließband beschleunigte für ein paar Minuten, dann landeten sie in einem hellen, sauberen Raum, der nach Vanille roch. Im Hintergrund spielte Klaviermusik, genau das Richtige, um sich nach der Begegnung mit dem Lexer zu beruhigen.

„Herzlich willkommen im Parser", sagte eine wohlmodulierte Stimme. „Bitte halten sie ihr Token-Label immer offen, damit der Prozess schnell vonstattengehen kann."

„Was sind denn Token-Labels?", fragte das K, doch es erhielt keine Antwort.

„Und was ist das da?", fragte das Anführungszeichen und deutete auf etwas, das wie ein Baum aussah. In seinen Stamm waren die Worte „Abstrakter Syntax-Baum" eingeritzt worden und auch in den Ästen stand etwas, doch das K konnte es von seiner Position aus nicht lesen. Verwunderlicher waren die Kästen, die anstelle von Blättern am Baum hingen.

Das K zuckte zusammen, als eine silberne Kugel vom Himmel fiel. Sie hielt direkt vor dem Ende ihrer Kette. Ein roter Lichtstreifen fuhr über das Ausführungszeichen. „Token erkannt – STRING."

Eine zweite Kugel kam herunter und schwebte direkt über ihnen. Ein Arm fuhr aus deren Körper und griff nach dem r. Das schrie, doch der Arm ließ einfach nicht locker. Da sie alle aneinander gekettet waren, wurde die ganze Gruppe in die Luft gehoben und schwebte durch die Halle auf den Baum zu. Alle Zeichen schrien, bis sie heiser wurden. Dann erreichte die Kugel ihr Ziel und sie wurden in eine der Glasboxen des Baums geworfen.

„Willkommen im Objekt „Person". Ihr seid der Name", sagte wieder die wohlige Stimme, doch diesmal konnte sie die Buchstaben nicht beruhigen. „Was zum Teufel!", fluchte das Anführungszeichen. Es hatte noch nie geflucht. Es sprang zur Glaswand und schlug dagegen. „Lasst uns hier raus!"

„Schau mal", sagte das K und deutete auf den Ast, an dem sie hingen. „Person" war das eingeritzte Wort. Und hinter ihnen stand „Name" an der Glaswand. „Freut euch", sagte die wohlige Stimme, „ihr seid jetzt Teil eines Ganzen."

Eines Ganzen? Von diesem Baum? Das K schaute nach unten auf ein paar aneinander gekettete Ziffern in einer anderen Glasbox, die mit „Alter" beschriftet war. Teil eines Ganzen, einer Person. Aber was bedeutete das? Wer ist denn diese Person und … was ist denn ein Syntax-Baum? Das K verstand es einfach nicht.

Es wandte sich dem Anführungszeichen zu, das sich frustriert gegen die Scheibe gelehnt hatte. „Geht's dir besser?", fragte das K. Das Anführungszeichen setzte zu einer Antwort an, stockte aber, als es eine weitere Kugel auf sie zufliegen sah. Diesmal war sie schwarz und das K hatte ein ganz schlechtes Gefühl.

„Achtung! String gefunden!", sagte die Kugel und flog auf das Anführungszeichen zu. Sie löste seine Ketten, dann schwebte es zum anderen Ende und löste auch dort die Verbindung zum Ausführungszeichen. „Danke sehr", sagte das Ausführungszeichen freundlich. Doch dann schrie es, das K zuckte zusammen und musste hilflos mit ansehen, wie An- und Ausführungszeichen von zwei dürren Armen gepackt und in die Luft gehoben wurden.

„String-Value konvertiert", sagte die beruhigende Stimme und das K schrie seinen Frust heraus: „Was soll das? Wo bringst du meine Freunde hin?" Doch es bekam wie immer keine Antwort.

„Beruhige dich", sagte das i, doch das K wollte sich nicht beruhigen, es wollte hier weg, zurück in das Dokument, wo sie unbehelligt ihren Spaß haben konnten. Es vermisste die schlechte Laune des Anführungszeichens jetzt schon.

Das K ließ sich fallen, wobei es das i mit nach unten zog, und am Ende lag die ganze Zeichenkette am Boden. Was würde nun mit ihnen geschehen? Was wurde aus dem Anführungszeichen? Und wo waren die Leerzeichen geblieben? Schluchzend fiel das K in einen traumlosen Schlaf.

***

„Hallo? Bist du wach? Komm, wir haben Großes vor", sagte eine sonore Stimme. Das K öffnete die Augen. Zuerst wollte es gar nicht glauben, was es da sah. Ein Mann lächelte die Zeichen an. Er hatte einen vollen Bart, wuscheliges Haar und hielt Pinsel und Palette in den Händen.

„Wer bist du denn?", fragte das K und gähnte. „Ich bin der Syntax-Highlighter. Ich möchte euch ein bisschen aufhübschen", sagte der Mann in seiner beruhigenden Art und das K musste erneut gähnen. „Was soll das denn heißen, uns aufhübschen? Sind wir dir nicht hübsch genug? Und wofür überhaupt, warum könnt ihr uns nicht einfach alle in Ruhe lassen!"

„Aber so geht das nicht. Du kannst mich nicht einfach wegschicken, das würde der Editor nicht toll finden. Du bist ein Name, also ihr alle seid ein Name und deshalb müsst ihr grün werden. Verstehst du das? Ich suche auch ein besonders schönes Grün aus, versprochen." Der Pinsel wirbelte über die Palette und kam dem K gefährlich nah. „Lass das!" Es wich zurück, doch kam nicht weit, denn es war ja immer noch an das i gekettet.

Der Highlighter wandte sich dem i zu, das die ganze Prozedur stoisch über sich ergehen ließ. „Und dann machen wir hier noch einen Tupfer und dort. Und schon sieht das Ganze großartig aus. Vorsicht hier, wir wollen es ja nicht zu dunkel werden lassen."

Das K schüttelte sich. „Und das nennst du ein schönes Grün?" „Also, jetzt hör mal. Was eine schöne Farbe ist, das bestimme immer noch ich. Ich bin der Highlighter." „Und ich bin sauer", antwortete das K. „Also, ich finde es ja ganz schön", meinte das r und stellte sich neben das i. „Ich als Nächster." „Natürlich." Und so erhielten r und auch a einen grünen Anstrich.

Das K seufzte resigniert und setzte sich auf den Boden. Während die Farbe auf den anderen Zeichen trocknete, ließ es sich vom Highlighter bemalen. „Wer ist denn dieser Editor?", fragte es. „Der Editor", begann der Highlighter, während er den Pinsel über das Grün der Palette schmierte, „ist die finale Station eurer Reise. Dort werdet ihr präsentiert."

„Präsentiert? Wem denn?" Das K war nun halb grün und der Highlighter begann die obere Diagonale zu bemalen. Das K konzentrierte sich derweil auf die Reise, eine Reise, die es eigentlich überhaupt nicht angehen wollte. „Na, dem Nutzer", sagte der Highlighter und tupfte noch etwas Grün auf die obere Kante.

Das K erinnerte sich, dass das Anführungszeichen einmal vom Nutzer gesprochen hatte. „Der Nutzer hat uns erschaffen und dem Nutzer müssen wir dienen", hatte es gesagt. Ach, das Anführungszeichen, das K vermisste es so sehr.

„So und nun seid ihr alle grün – schön, schön. Ich werde mich mal wieder auf die Reise machen, und vielleicht treffe ich noch einen anderen Namen, dann mach ich ihn genauso grün wie euch. Aber ich verspreche euch, ihr seid der schönste Name im Dokument."

Eine silberne Scheibe kam von oben herab und der Highlighter stellte sich darauf. Er winkte, während er davon schwebte. „Und jetzt?", fragte das r und die anderen wussten auch nicht weiter. Also setzten sie sich hin und spielten eine Runde „Ich sehe was, was du nicht siehst". Und immer war alles grün.

***

Das Spiel wurde irgendwann langweilig. Das K hätte gern Fangen gespielt, aber sie waren ja aneinander gekettet und das wäre keine faire Ausgangssituation. Also schwiegen sie, bis wieder eine Kugel erschien. Diese hier war blau.

Sie packte das r und hob es an, und durch die Ketten wurden die anderen mit in die Luft gezogen. Das war jetzt also die letzte Etappe der Reise, der Weg zum Editor. „Bitte bewahren Sie Ruhe, während wir Sie in den ValueConverter schicken", sagte die Kugel und dann wurden sie auf ein neues Fließband geworfen.

„Schicken ist gut", meinte das K und rieb sich den Hintern. Nur wenige Momente später hörte es eine bekannte Stimme über sich. Das K schaute auf und sah, wie das Anführungszeichen und sein Bruder, das Ausführungszeichen, herabgelassen wurden. Ein Greifarm ließ die Zeichen los und verband sie mit dem ersten und letzten Zeichen der Kette.

Das K hatte sein Anführungszeichen wieder. Es konnte sein Glück nicht fassen. „Jetzt mach mal halblang", rief das Anführungszeichen und versuchte, sich aus der Umarmung zu lösen. „Wie siehst du überhaupt aus?"

Das K sprang zurück und erzählte von dem malenden Kauz. „So etwas hatten wir nicht", meinte das Anführungszeichen und sein Bruder stimmte zu. „Aber wo wart ihr denn?", fragte das K. Die Stimme des Anführungszeichens brach, obwohl es nur sagte: „Im Nichts."

Darauf herrschte in der Zeichengruppe für einen Moment Stille, während sie weiter zum Editor befördert wurden. Die Greifarme brachten noch einen weiteren Freund: das Leerzeichen am Zeilenanfang.

„Wo sind die anderen?", fragte das K, doch das Leerzeichen schüttelte traurig den Kopf: „Er hat gesagt, ich muss jetzt alleine klarkommen. Die anderen sind nicht erwünscht." Dann schluchzte es und begann zu weinen.

„Wer hat das gesagt?", fragte das Anführungszeichen sichtlich empört. „Er nannte sich Formatter. Keine Ahnung, wer ihn zum Chef benannt hat. Aber er hat gesagt, der Editor möchte das so, damit der Nutzer sich besser zurechtfindet." Alles für den Nutzer, dachte das K. So war das hier.

Ein gleißendes Licht am Ende des Laufbandes verschluckte jedes einzelne Zeichen. Leerzeichen. Anführungszeichen. K. Diesmal schrie es nicht, sondern wartete nur darauf, irgendwo zu landen. Die Geschwindigkeit wurde immer höher, dem K wurde schwarz vor Augen und dann verlor es das Bewusstsein.

Als es wieder zu sich kam, befand es sich in einem Dokument. Es schaute nach oben und schätzte, dass es vielleicht vier Zeilen unter dem Anfang war. Vier, das war viel höher als im alten Dokument. Wie viele Zeichen hatte dieser Formatter noch entfernt?

Auch war alles so hell hier, im Hintergrund strahlte ein weißes Licht, der Buchstabe über ihm glänzte rot, sicher auch seit einer Begegnung mit dem Syntax-Highlighter. Das war jetzt also der Editor. Und nun?

Hannes Niederhausen
hniederhausen@itemis.de

Hannes Niederhausen arbeitet als Software-Entwickler bei der itemis AG, mit Schwerpunkt auf Modell-getriebener Software-Entwicklung. Zusätzlich verfasst er monatlich Kurzgeschichten und längere Texte. Mehr Infos unter „https://www.hannesniederhausen.de".

Sie bekommen, was sie verdienen

Marco Schulz

IT-Professionals bekommen schon zu Beginn ihrer Karriere kuriose Anfragen. So auch ich. Bereits während meines Studiums klingelte hin und wieder das Telefon und besonders kluge Menschen erklärten mir, wie ich für sie so etwas wie Facebook nachprogrammieren könne. Natürlich ohne Bezahlung.

Die pfiffige Idee dieser Zeitgenossen war es, das ich für sie kostenlos eine Plattform erstelle, natürlich exklusiv nach ihren Wünschen. Dank deren hervorragender Vernetzung würde das Ganze sehr schnell erfolgreich und wir könnten den Gewinn untereinander aufteilen. Ich wollte dann immer wissen, wozu ich für die Entwicklung eines Systems, dessen Kosten und Risiken ich allein zu tragen habe, einen Partner benötige, um dann mit ihm den Gewinn zu teilen. Diese Frage beendete solche Gespräche recht schnell.

Vor nicht allzu langer Zeit erreichte mich wieder einmal eine Projekt-Anfrage mit einem umfangreichen Skill-Set zu einem offerierten Stundenlohn, der bereits für Studenten unverschämt gering ausfiele. Dies erinnerte mich an einen sehr sarkastischen Artikel von Yegor Bugayenko aus dem Jahre 2016, den ich hier ins Deutsche übertragen habe:

Um Software erstellen zu können, benötigt man Programmierer. Unglücklicherweise. Sie sind in aller Regel teuer, faul und meistens unkontrollierbar. Die Software, die sie erstellen, funktioniert vielleicht oder vielleicht auch nicht. Trotzdem erhalten sie jeden Monat ihren Lohn. Aus diesem Grund ist es immer eine gute Idee, möglichst wenig zu zahlen. Wie dem auch sei. Manchmal erklären sie einem, wie unterbezahlt sie sind, und kündigen einfach. Aber wie will man dies unterbinden? Leider ist es uns nicht mehr gestattet, gewalttätig zu sein, aber es gibt einige andere Möglichkeiten. Last mich dies genauer erläutern.

Gehälter geheim halten

Es ist offensichtlich: Sie dürfen sich nicht über ihre Gehälter austauschen. Diese Information ist geheim zu halten. Ermahnt sie oder noch besser schreibt einen Geheimhaltungs-Paragraph in ihren Vertrag, der verhindert, dass über Löhne, Boni, Vergütungspläne gesprochen wird. Sie müssen fühlen, dass diese Information giftig ist. So dass sie sich nie über dieses Thema unterhalten. Wenn das Einkommen ihrer Kollegen unbekannt ist, kommen weniger Fragen nach Gehaltserhöhungen auf.

Zufällige Lohnerhöhungen

Es sollte kein erkennbares System geben, wie Lohnerhöhungen oder Kündigungen entschieden werden. Lohnerhöhungen werden ausschließlich nach Bauchgefühl verteilt, nicht etwa, weil jemand produktiver oder effektiver wurde. Entscheidungen sollten unvorhersehbar sein. Unvorhersagbarkeit erzeugt Angst und dies ist genau das, was wir wollen. Sie sind eingeschüchtert ihrem Auftraggeber gegenüber und werden sich lange Zeit nicht beschweren, wie unterbezahlt sie sind.

Keine Konferenzen

Es sollte ihnen nicht gestattet sein, an Meetups oder Konferenzen teilzunehmen. Dort könnten sie möglicherweise auf Vermittler treffen und herausfinden, dass ihre Bezahlung nicht fair genug ist. Es sollte die Idee verbreitet werden, dass Konferenzen lediglich Zeitverschwendungen sind. Es ist besser, Veranstaltungen im Büro durchzuführen. Sie haben immer zusammenzubleiben und niemals auf Programmierer aus anderen Unternehmen zu treffen. Je weniger sie wissen, desto sicherer ist man.

Keine Heimarbeit

Das Büro muss zu einem zweiten Zuhause werden. Besser noch, zum wichtigsten Platz in ihrem Leben. Sie müssen jeden Tag anwesend sein, am Schreibtisch, mit einem Computer, einem Stuhl und einer Ablage. Sie sind emotional verbunden mit ihrem Arbeitsplatz. So wird es viel schwieriger, ihn eines Tages zu kündigen, ganz gleich wie unterbezahlt sie auch sind. Sie sollten niemals eine Erlaubnis bekommen, per Remote zu arbeiten. Sie könnten dann beginnen, von einem neuen Zuhause und einem stattlicheren Gehalt zu träumen.

Überwacht sie

Es ist dafür zu sorgen, dass sie firmeneigene Systeme wie E-Mail, Computer, Server und auch Telefone nutzen. Darauf ist dann gängige Überwachungssoftware installiert, die sämtliche Nachrichten

und Aktivitäten protokolliert. Idealerweise existiert eine Sicherheitsabteilung, um die Programme zu überwachen und bei abnormalem oder unerwartetem Verhalten das Management zu informieren. Videokameras sind auch sehr hilfreich. Jeglicher Kontakt zu anderen Unternehmen ist verdächtig. Angestellte sollten wissen, dass sie überwacht werden. Zusätzliche Angst ist immer hilfreich.

Vereinbarungen mit Mitbewerbern

Kontaktiert die größten Mitbewerber der Region und stellt sicher, dass keine Entwickler abgeworben werden, solange sie dies ebenfalls nicht tun. Falls sie diese Absprache zurückweisen, ist es gut, einige ihrer Schlüsselpersonen abzuwerben. Einfach durch das In–Aussicht-Stellen des doppelten bisherigen Gehalts. Natürlich will man sie nicht wirklich engagieren. Aber diese Aktion rüttelt den lokalen Markt ordentlich durch und Mitbewerber fürchten einen. Sie sind schnell einverstanden, keine deiner Entwickler jemals zu berühren.

Etabliert gemeinsame Werte

Unterzieht sie einer regelmäßigen Gehirnwäsche in gemeinsamen Jubelveranstaltungen, in denen begeistert verkündet wird, wie toll die Firma ist, was für großartige Ziele alle haben und wie wichtig die Zusammenarbeit als Team ist. Die Zahlen auf der Gehaltsabrechnung erscheinen weniger wichtig, im Vergleich zu einem Multi-Millionen-Euro-Vorhaben, das den Markt dominieren soll. Sie werden sich dafür aufopfern und eine recht lange Zeit wird dieser Trick motivieren.

Gründe eine Familie

Gemeinsame Firmenveranstaltungen, freitags Freibier, Team-Building-Veranstaltungen, Bowling, Geburtstagsfeiern, gemeinsame Mittagessen und Abendveranstaltungen – das sind Möglichkeiten, um das Gefühl zu erzeugen, dass die gesamte Firma die einzige Familie ist. In einer Familie spricht man als gutes Mitglied auch nicht über Geld. Korrekt? Die Frage nach einer Gehaltsvorstellung gilt als Verrat an der Familie. Aus diesem Grund werden sie davon Abstand nehmen.

Stresst sie

Sie dürfen sich nicht entspannt fühlen, das ist nicht zu unserem Vorteil. Sorgt für kurze Abgabetermine, komplexe Problemlösungen und ausreichend Schuldgefühle. Niemand wird nach einer Gehaltserhöhung fragen, wenn er sich schuldig fühlt, die Projektziele wieder einmal nicht erreicht zu haben. Daher sind sie so oft wie möglich für ihre Fehler zur Verantwortung zu ziehen.

Versprechungen machen

Es ist nicht notwendig, die Versprechen einzuhalten, aber sie müssen gemacht werden. Versprecht ihnen, das Gehalt demnächst zu erhöhen oder künftige Investitionen oder die Ausfertigung eines unbefristeten Abseitsvertrags. Natürlich unter der Bedingung, dass die Zeit dafür auch reif ist. Es ist sehr wichtig, dass die Versprechungen an ein Ereignis geknüpft sind, das man selbst nicht beeinflussen kann, um die eigenen Hände stets in Unschuld zu waschen.

Kauft ihnen weiche Sessel und Tischtennisplatten

Ein paar winzige Ausgaben für diese lustigen Bürosachen werden schnell kompensiert durch den Hungerlohn, den die Entwickler ausbezahlt bekommen. Eine hübsche, professionelle Kaffeemaschine kostet 1.000 Euro und spart pro Programmierer jeden Monat zwischen 200 bis 300 Euro ein. Rechnet es aus. Erstellt eine eigene Regel, die besagt, bevor irgendjemand eine Gehaltserhöhung bekommt, ist es sinnvoller, eine neue PlayStation für das Büro zu kaufen. Erlaubt ihnen, ihre Haustiere mit ins Büro zu bringen, und sie bleiben länger für weniger Geld.

Gut klingende Titel

Bezeichnet sie als Vizepräsident, beispielsweise „VP für Entwicklung", „technischer VP", VP von was auch immer. Keine große Sache. Aber sehr wichtig für Angestellte. Die Bezahlung hat so weitaus weniger Stellenwert als der Titel, den sie in ihre Profile auf sozialen Netzwerken schreiben können. Wenn alle Vizepräsidenten besetzt sind, versuche einmal Senior Architekt oder Lead Technical etc.

Überlebenshilfe

Die meisten Programmierer sind etwas unbeholfen wenn es darum geht, ihr Geld zu verwalten. Sie wissen einfach nicht, wie man eine Versicherung abschließt, die Rente organisiert, oder einfach nur, wie man Steuern zahlt. Natürlich erhalten sie Hilfe, nicht unbedingt zu ihren Gunsten. Aber sie werden glücklich sein, sich in euren Händen sicher fühlen und niemals daran denken, das Unternehmen zu verlassen. Niemand wird nach einer Lohnerhöhung fragen, weil sie sich schlecht fühlen, solche Geschäfte in die eigene Hand zu nehmen. Seid ihnen Vater oder Mutter – sie werden ihre Rolle als Kind annehmen. Es ist ein bewährtes Modell. Es funktioniert.

Sei ein Freund

Das ist die letzte und wirkungsvollste Methode. Sei ein Freund der Programmierer. Es ist verflixt schwierig, mit Freunden über Geld zu verhandeln. Sie sind nicht in der Lage, das einfach in Angriff zu nehmen. Sie bleiben und arbeiten gern für weniger Geld, einfach weil wir Freunde sind. Wie man zum Freund wird? Gut. Trefft ihre Familien, lade sie zum Essen in dein Haus ein, kleine Aufmerksamkeiten zu Geburtstagen – all diese Sachen. Sie sparen eine Menge Geld.
Habe ich noch etwas vergessen?

Quelle

- *https://www.yegor256.com/2016/12/06/how-to-pay-programmers-less.html*

Marco Schulz
marco.schulz@outlook.com

Marco Schulz studierte an der HS Merseburg Diplom-Informatik. Sein persönlicher Schwerpunkt liegt in Software-Architekturen, der Automatisierung des Software-Entwicklungs-Prozesses und dem Softwarekonfigurations-Management. Seit mehr als fünfzehn Jahren realisiert er in internationalen Projekten für namhafte Unternehmen auf unterschiedlichen Plattformen umfangreiche Web-Applikationen. Er ist freier Consultant, Trainer und Autor verschiedener Fachartikel. Sein persönlicher Blog lautet „https://enRebaja.wordpress.com".

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:
Andreas Badelt, Melanie Feldmann, Markus Karg, Manuel Mauky, Bernd Müller, Benjamin Nothdurft, Daniel van Ross, André Sept

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

Fotonachweis:
Titel: © Iconicbestiary www.Freepik.com
S. 8: Logo © eclipse www.eclipse.org
S. 9: Logo © vaadin https://vaadin.com
S. 16: Bild © Aleksey Telnov https://123RF.com
S. 21: Logo © JHipster www.jhipster.tech
S. 29: Bild © Ivan Trifonenko https://123RF.com
S. 32: Logo © kubernetes https://kubernetes.io
S. 38: Bild © alphaspirit https://123RF.com
S. 44: Bild © aurielaki https://123RF.com
S. 49: Logo © Apache www.apache.org
S. 53 Bild © sylverarts https://123RF.com
S. 60 Bild © Stacey Lynn Payne https://123RF.com
S. 64 Bild © scanrail https://123RF.com

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

JavaLand

19. - 21. März 2019 in Brühl bei Köln

Ab sofort Ticket & Hotel buchen!

www.javaland.eu

Präsentiert von: DOAG Heise Medien

Community Partner: iJUG Verbund

www.javaland.eu

BESUCHE UNS AUF DER JAVALAND AM STAND 302.

arvato
BERTELSMANN

Du weißt es vielleicht nicht, aber wir stehen hinter zahlreichen Produkten und Leistungen, die Du nutzt – im Schnitt hat jeder Verbraucher in Deutschland achtmal täglich Kontakt mit uns. Arvato ist ein führender internationaler Dienstleister, der von und mit digitaler Technologie lebt. Mehr als 70.000 Mitarbeiter in über 40 Ländern unterstützen jeden Tag unsere Kunden dabei, erfolgreich am Markt zu agieren. Dazu konzipieren und realisieren wir maßgeschneiderte Lösungen für unterschielichste Geschäftsprozesse entlang integrierter Dienstleistungsketten. Daraus ergeben sich für Dich spannende Perspektiven. Es wird Zeit, dass wir uns persönlich kennenlernen!

Du findest uns ...

... zum Beispiel auf der JavaLand:
Vom 19. – 20. März 2019 am Stand 302
im Phantasialand in Brühl.
arva.to/career