Ausgabe 05/2021

Java aktuell

iJUG Verbund
www.ijug.eu

Serverless
Erfahrungsberichte und mehr

Agile Softwareentwicklung
Domain-driven Design, Systemtheorie und Metriken

Cloud Computing
Ethik und Prinzipien

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 05

Liebe Leser/innen der Java aktuell,

sowohl zu unserem Schwerpunkt Serverless als auch zu weiteren spannenden Themen findet ihr in dieser Ausgabe zahlreiche informative Fachbeiträge unserer Autorinnen und Autoren aus der Community. Wie üblich informieren euch Andreas Badelt und Markus Karg zu Beginn im Java-Tagebuch und in der Eclipse Corner über aktuelle Geschehnisse rund um Java und die Eclipse Foundation.

In unserem ersten Beitrag lässt Christian Schmidt mit uns gemeinsam ein Jahr Serverless mit Amazon Web Services Revue passieren. Dabei berichtet er, inwiefern Erwartungen erfüllt oder enttäuscht wurden und an welchen Stellen Serverless nicht die beste Lösung darstellte. Danach zeigt Niko Köbler, dass Java und Serverless keinen Gegensatz darstellen müssen und wie man sie miteinander verbinden kann. Ab Seite 22 erklärt Frank Rosner, wie mithilfe einer Ereignisquellenzuordnung SQS-Ereignisse in einer AWS-Lambda-Funktion verarbeitet werden können.

Low-Code und No-Code sind relativ neue Begriffe in der IT-Welt. Im Jahr 2014 erstmals benannt, gewannen sie erst kürzlich an Aufmerksamkeit. Auch im Java-Ökosystem gibt es viele Tools, die dies umsetzen können. Eines davon stellt René Jahn in seinem Artikel näher vor: VisionX. Auch agile Softwareentwicklung ist heute voll im Trend und bereits fester Bestandteil in vielen Unternehmen. Ab Seite 34 wagt Christian Mennerich den Versuch, mithilfe einer soziologischen Theorie mehr Verständnis bei der Suche nach Domänen und Kontexten zu schaffen. Sinnvoll ist bei agiler Softwareentwicklung auch die Erarbeitung von Metriken. Diese ermöglichen Teams einen besseren Zugang zu Metadaten ihrer Arbeit. Dadurch können die eigenen Tätigkeiten besser verstanden und optimiert werden. Einige Metriken, wie beispielsweise Team-Homogenität, präsentiert Richard Fichtner in seinem Artikel.

Das Zeitalter der Cloud ist da. Sie ist aus unserem Alltag schon seit einer Weile nicht mehr wegzudenken. Corinna Wiedenmann widmet sich ab Seite 52 passend dazu der Ethik des Cloud-Computings. Sie gibt einen Einblick und Überblick in die aktuelle und zukünftige europäische Entwicklung rund um Privatsphäre, Monopolisierung und Nachhaltigkeit.

Zum Abschluss teilt Roland Golla sehr persönliche Einblicke in sein Leben als Programmierer mit uns. Er erklärt, wieso er seinen Job liebt, der allerdings auch seine Schattenseiten mit sich bringt, und wie er diese er- und überlebt hat. Damit möchte er anderen Entwickler/innen Mut machen, Veränderungen zuzulassen.

Wir wünschen euch viel Spaß beim Lesen!

Eure

Lisa Damerow

Redaktionsleitung Java aktuell

Inhalt

Erfahrungen aus einem Jahr Serverless mit AWS

Serverless Java mit Quarkus Funqy

Java Low-Code mit VisionX

Ethik des Cloud Computings

Das Java-Tagebuch

Andreas Badelt, stellv. Leiter der DOAG Java Community

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

2. Juni 2021

Adoptium: TCK-Progress

Ein wichtiger formaler Schritt für das Adoptium-Projekt ist geschafft: Die noch fehlenden „TCK Participant Agreements" mit Oracle für die Mitglieder New Relic, Red Hat und Microsoft liegen jetzt vor. Erst jetzt kann das Team der Eclipse Foundation die TCKs herunterladen und die „vertrauliche" Test-Infrastruktur für das extra für deren Zugang geschaffene Temurin-Compliance-Projekt einrichten. Die Java SE TCKs sind halt immer noch ein Staatsgeheimnis – aber ich glaube, dass sie zumindest nicht ausschließlich in ausgedruckter Form in abgeschirmten Räumen ausliegen werden.

NetBeans 12.4 und Java 16

NetBeans 12.4 unterstützt jetzt Java 16 und Jakarta EE 9 (neue Package-Namen). Außerdem bietet es unter anderem einen Wizard und Code-Unterstützung für Micronaut-Projekte.

8. Juni 2021

Snyk Tools kostenlos für Einzel- und Open-Source-Devs

Das „Static Application Security Testing (SAST)"-Werkzeug Snyk Code, das sich auch in diverse IDEs integriert, wird jetzt von Snyk in einer freien („free beer") Variante angeboten – für Einzelentwickler und Beteiligungen an Open-Source-Projekten. Eine Registrierung ist erforderlich, danach sind 100 Scans im Monat kostenlos *[1]*.

14. Juni 2021

Oracle „JMS"

Oracle bietet jetzt einen neuen JMS an. Nichts mit Messaging, sondern den „Java Monitoring Service". Damit sollen Java-Installationen im eigenen Rechenzentrum, bei Oracle oder bei anderen Cloud-Anbietern überwacht werden können: Wo läuft überall Java (in manchen Unternehmen heißt das auch „Überrasch mich!"), welche Versionen (produktiv oder in der Entwicklung) laufen, sind die neuesten Patches ausgerollt etc. Auch Performance- und Security-Daten (beispielsweise abgelaufene Zertifikate) und Informationen zur „Compliance" können abgerufen werden. Ob Compliance auch einen direkten Draht in die Lizenzabteilung beinhaltet, ist nicht überliefert… Das Tool ist für Oracle-Java-SE-Kunden im Prinzip kostenlos, aber wohl nur mit einem Oracle-Cloud-Infrastructure-Konto nutzbar. Bei großen Datenmengen fallen Kosten an, wenn die Freimengen für das OCI-Monitoring überschritten werden. Details sind im Oracle-Blog zu finden *[2]*.

15. Juni 2021

Spring Native 0.10

Spring Native, die Spring-Variante der Erzeugung nativer Images mithilfe der GraalVM, hat einen weiteren Schritt gemacht, bleibt aber vorerst im Beta-Stadium. Viele der Neuerungen sind wohl in enger Zusammenarbeit mit dem GraalVM-Team entstanden und knüpfen an deren jüngste Arbeit an. Das neue Spring Native Release 0.10.0 bringt zum Beispiel breitere Build-Unterstützung: Das bisherige Plug-in (nur für Maven) wurde durch Graals neue Native-Build-Tools ersetzt, sodass jetzt auch Gradle unterstützt wird. Auch die neue JUnit-5-Unterstützung wird genutzt, sodass ein @SpringBootTest jetzt im nativen Image genauso funktioniert wie in einer normalen JVM. Außerdem sollen mit der neuen @AotProxyHint-Annotation auch Proxys für Klassen während des Builds generiert werden (statt nur der JDK Proxies für Interfaces), etwa für Security und Transaktionskontrolle.

17. Juni 2021

Neue Eclipse-IDE-Version – und neue Working Group

Passend zur Veröffentlichung der Eclipse IDE 2021-06 – jetzt auch mit Java-16-Unterstützung – gibt es eine weitere Neuigkeit: Auch für die IDE wird es jetzt eine eigene Working Group geben, um den über 70 einzelnen Projekten ein gemeinsames Dach zu geben und, so die Ankündigung, „deren Beiträge durch zusätzliche Ressourcen zu unterstützen und zu stärken" *[3]*.

23. Juni 2021

JVM Ecosystem – der Stand 2021

Die von Snyk durchgeführte Umfrage zum „Java Ecosystem 2021" ist ausgewertet. Es gab über 2.000 Beteiligungen, was ganz ordentlich ist, auch wenn wie immer eine gewisse Skepsis bezüglich der Repräsentativität angebracht ist. Sensationelle Ergebnisse konnte ich nicht entdecken, aber ganz interessant ist es trotzdem. 48 % der JDKs zur Entwicklungszeit sind direkte AdoptOpenJDK Builds (Adoptium taucht auch schon auf, aber noch weit hinten mit 1,5 %), vor 29 % Oracle-OpenJDK- und 22 % Oracle-JDK-Varianten – wobei Mehrfachnennungen möglich waren.

Richtung Produktion verschieben sich die Zahlen nur minimal (aber wer tauscht auch auf dem Weg in die Produktion das JDK aus – selbst, wenn die OpenJDK-basierten Builds weitgehend identisch sind?) Java 11 liegt in der Nutzung in Produktion knapp vor Java 8 – zur Entwicklungszeit deutlich. Die Java-Welt bewegt sich doch – obwohl Oracle vor Kurzem noch verkündet hat, dass Java 8 mindestens bis 2030 (kostenpflichtigen) „Extended Support" erfährt – und andere Anbieter wie Azul da mitgehen. Immerhin ein Viertel nutzen das Nicht-Long-Term-Release Java 15 in der Entwicklung, knapp 12 % sogar in Produktion (und migrieren alle sechs Monate, oder?) Also, es geht voran, aber die 8 wird uns noch lange begleiten.

Ein weiteres Detail der Umfrage: Kotlin ist nach Java die zweitbeliebteste Sprache, vor Groovy, Scala und Clojure. Mal sehen, ob Scala 3 hier demnächst für Bewegung sorgt (nach oben durch neues Interesse oder nach unten, falls sich viele wegen der Inkompatibilität mit Scala 2 umorientieren sollten).

25. Juni 2021

Kotlin 1.5.2 und Lombok

Kotlin 1.5.2 unterstützt jetzt Lombok? Da war ich kurz irritiert – ist Kotlin nicht mit dem Anspruch entstanden, den ganzen „Boilerplate Code" von vornherein loszuwerden? Wozu dann noch Lombok? Aber lieber erst die Release Notes durchlesen: Es geht um das Schließen einer Interoperabilitätslücke, nämlich die (noch experimentelle) Unterstützung für den Aufruf von Lombok-erzeugten Methoden in gemischtem Java-/Kotlin-Code. Das macht mehr Sinn.

27. Juni 2021

Jakarta EE 10 wird auf Java 11 basieren

Das nächste Major Jakarta Release wird nach einer Abstimmung unter den Projekt-„Committern" auf Java SE 11 basieren – für API Source und Binary Level. Implementierungen dürfen aber auch Versionen jenseits von 11 unterstützen und diese entsprechend in den TCK-Läufen zum Nachweis der Kompatibilität verwenden.

30. Juni 2021

Mission Wiedervereinigung

AdoptOpenJDK ist inzwischen größtenteils im Eclipse-Adoptium-Projekt aufgegangen. Ein kleiner, aber feiner Rest – JDK Mission Control – soll nun ebenfalls folgen. Dazu wurde jetzt das neue Teilprojekt „Eclipse Mission Control" gestartet.

7. Juli 2021

Jakarta.x Namespace-Migration – wer steht wo?

Um die Umsetzung der Jakarta-EE-Namespace-Änderungen zu tracken, hat Martijn Verburg ein Mitmach-Dokument (neudeutsch „Crowd Source") angelegt, in dem der jeweilige Migrationsstatus nicht nur der betroffenen Application Server, sondern auch IDEs, Frameworks, Test-Werkzeuge etc. festgehalten werden sollen *[4]*.

13. Juli 2021

MicroProfile stimmt CN4J-Plänen zu

Die „Cloud Native for Java"-Initiative von MicroProfile und Jakarta EE wurde in den letzten Monaten weiter vorangetrieben. Beide Eclipse Working Groups haben an einer Kommunikationsstrategie und einem Foliensatz gearbeitet, um die Unterschiede, aber insbesondere die perfekte Ergänzung der beiden hervorzuheben. Das Ergebnis – „A Powerful Collaboration" – ist jetzt von der MicroProfile Working Group einstimmig akzeptiert worden. Keine großen Überraschungen, aber es hat sicher geholfen, das Selbstverständnis der Projekte noch einmal zu schärfen, was in den Folien gut zusammengefasst ist. Zum Beispiel Jakarta EE als Framework für den „Enterprise"-Bereich, mit den gewohnt starken Anforderungen an Abwärtskompatibilität, gegenüber MP als Innovationstreiber mit Microservice-Fokus, bei dem die Kompatibilität schon mal Prio 2 einnimmt. Auch die Lizenz drückt dies aus – bei Jakarta wird die „Compatible Patent License" bei Nachweis vollständiger Kompatibilität (durch das jeweilige TCK) vergeben, bei MicroProfile gibt es die „Implementation Patent License" bereits für begonnene oder Teil-Implementierungen *[5]* (Details zu den Lizenzen sind hier nachzulesen *[6]*).

16. Juli 2021

JDK 17 kommt am 14. September

Das JDK 17 geht nun auch auf die Zielgerade. „Ramp Down Phase 2" ist eingeleitet (nur noch Prio 1 und 2 Bugs werden angegangen), für den 14. September ist die General Availability geplant. 14 Features haben den Weg ins Release gefunden, zwei davon noch als entsprechend instabile „Inkubator"-Versionen (Foreign Function & Memory API und das bereits in 16 enthaltene Vector API), sowie „Pattern Matching for switch" als Preview – nach „Pattern Matching for instanceof", das ja seit 16 produktionsreif ist. Sealed Classes (Preview in 15) sind jetzt ebenfalls produktionsreif. Daneben gibt es unter anderem Verbesserungen für MacOS-Systeme. Ein „Feature" ist aber auch die Entfernung der experimentellen AOT- und JIT-Compiler auf Java-Basis – sie seien kaum genutzt worden und teuer in der Maintenance. Da sie sowieso auf dem Graal-Compiler basieren, wird nun einfach auf die GraalVM als Alternative verwiesen. Die volle Feature-Liste steht wie immer auf den OpenJDK-Projektseiten *[7]*.

20. Juli 2021

MicroProfile 4.1 angenommen

Die MicroProfile 4.1 „Umbrella"-Spezifikation ist von der Working Group ohne lange Diskussion einstimmig angenommen worden – es gibt aber auch nur zwei wesentliche Änderungen: Das Health API ist auf Version 3.1 angehoben worden und Implementierungen müssen nicht mehr mit Java 8 kompatibel sein – sie können auch eine SE-Version größer als 8 voraussetzen.

28. Juli 2021

Jakarta EE: Community Acceptance Testing

Die Jakarta EE Ambassadors haben eine „Jakarta Community Acceptance Testing Initiative" (kurz JCAT) gestartet. Damit sollen möglichst viele Freiwillige gewonnen werden, die Jakarta EE 9 oder 9.1 ausprobieren und Feedback geben. Details sind im Blog der Ambassadors zu finden *[8]*. Die Test-Offensive läuft noch bis 1. Oktober.

29. Juli 2021

Jakarta: Diskussion um optionale Features

Das Jakarta-Projekt sucht nach neuen Wegen, um mit optionalen Features umzugehen. Momentan verlangt der Spezifikationsprozess, dass mindestens eine Implementierung auch alle optionalen Features unterstützt. Das wird aktuell ausschließlich von GlassFish (und dessen Forks) geleistet. Ein Vorschlag lautet, die Regel dahingehend zu ändern, dass lediglich die Gesamtmenge der Implementierungen alle optionalen Features abdecken muss – um weiterhin sicherzustellen, dass diese überhaupt implementierbar sind. Eine weitere Überlegung ist, eine Liste dieser Features und ihrer Implementierungen zu führen – um gegebenenfalls zu zeigen, dass sie keine Verbreitung in den Implementierungen haben und folglich nicht so relevant sein können.

Die Diskussion ist aber schon weiter fortgeschritten und es wird nun darüber gesprochen, optionale Features in einem Minor-Release (vermutlich 9.2) als „deprecated" zu markieren und dann in einem zukünftigen Major Release (10+) zu entfernen. Betroffen sind Entity Beans und Embeddable EJB Containers (für diejenigen, die schon mal den „innovativen" Schritt zu Jakarta hinbekommen haben, wohl eh kein großer Verlust). Zusätzlich sollen ab Release 10 keine neuen optionalen Features mehr erlaubt sein. Bislang hat eine deutliche Mehrheit des Spezifikationskomitees dem zugestimmt, aber es gab mindestens eine Gegenstimme: Ein kategorischer Ausschluss optionaler Features würde eine Ausweitung der Anwendungsbereiche schwieriger machen. Mal sehen, wie es am Ende umgesetzt wird.

31. Juli 2021

Das Beste zum Schluss: JavaLand 2022 wieder vor Ort

Die aktualisierte Website ist schon ein paar Tage online, aber ich habe es mir für den Schluss aufgehoben. So Corona es zulässt, wird die JavaLand 2022 nächstes Jahr „physisch" im Phantasialand stattfinden – vom 15. bis 17. März. Der Call for Papers beginnt am 17. August.

Referenzen

[1] *https://snyk.io/blog/snyk-code-now-available-free-sast/*
[2] *https://blogs.oracle.com/java/post/announcing-java-management-service*
[3] *https://blogs.eclipse.org/post/mike-milinkovich/shaping-future-eclipse-ide*
[4] *https://github.com/jakartaee/collateral/issues/71*
[5] *https://www.eclipse.org/lists/cn4j-alliance/msg00174.html*
[6] *https://eclipse-foundation.blog/2021/05/13/on-patents-and-specifications*
[7] *https://openjdk.java.net/projects/jdk/17/*
[8] *https://jakartaee-ambassadors.io/2021/07/28/jakarta-community-acceptance-testing/*

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 im DOAG e.V. aktiv, zunächst als Co-Leiter der SIG Development und später der SIG Java. Seit 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

Nachdem ich an dieser Stelle seit Jahren lautstark darüber lamentiere, dass das Thema Java bei der Eclipse Foundation einfach nicht schnell genug vorankommt, hat sich inzwischen nun doch einiges getan. Es wird vermutlich niemandem, der dieser Kolumne wirklich aufmerksam folgt, entgangen sein, dass vor einiger Zeit bereits Jakarta EE 9.1 veröffentlicht wurde und mit der 9.1er-Plattform nun portable Unternehmensanwendungen auch in (zumindest halbwegs modernem) Java 11 geschrieben werden können. Ebenso hat sicherlich auch jeder mitbekommen, dass neben der eigenen, von IBM gespendeten, OpenJ9 VM mittlerweile auch eine eigene JDK-Distribution namens Eclipse Adoptium *[1]* (ehemals AdoptOpenJDK) unsere vor Jahren gehegten Lizenzsorgen vom Tisch fegt und seit Kurzem die Downloads nun auch unter dem neuen Namen anbietet. Zu den Unterstützern dieser für Entwicklung und Produktionseinsatz freien Distribution zählen übrigens neben den Anwendervereinigungen LJC und iJUG – man höre und staune – Microsoft(!), IBM und Red Hat – und natürlich eine ganze Reihe einzelner Enthusiasten!

Es geht nunmehr quasi Schlag auf Schlag, denn schon bald soll nach dem Willen der Jakarta Working Group Version 10 der Java-Enterprise-Plattform folgen. Diese wiederum soll nun endlich (im Gegensatz zu Jakarta EE 9.1) Java 11 auch in den APIs selbst nutzen, sowohl in Source- als auch in Binärform, was modernere Schnittstellen ermöglicht. Kompatible Serverprodukte müssen darüber hinaus beweisen, dass sie das TCK auch auf JRE 17 bewältigen. Letzteres macht es Anwendern somit wieder möglich, ihrerseits Java 17 zu verwenden und dadurch schon ziemlich nah an das aktuelle Release von Java SE heranzurücken. Zudem wird es mit Jakarta EE 10 viele neue Features geben. Und nun wird es spannend, denn die einzelnen APIs werden in den kommenden Monaten nach und nach diese neuen Features veröffentlichen, eben in längerem Vorlauf zu Jakarta EE 10. Unter anderem wird JAX-RS 3.1 enthalten sein und mit diesem das für portable Microservices vorhandene Java-SE-Bootstrap-API (also Microservices ohne vollumfängliches Serverprodukt), Unterstützung für Multipart Forms und einiges Langersehntes mehr. Sobald die Feature-Liste endgültig fertiggestellt ist, werden wir natürlich in der Java aktuell darüber berichten! Insofern werden die nächsten Wochen sehr spannend!

Und auch diese Ausgabe geht nicht ohne meinen üblichen Appell zu Ende: Java ist Open Source und lebt von den Contributions jedes Einzelnen. Wir haben lange dafür gekämpft, dass wir an Java mitwirken können, dass es keine Lizenzkosten mehr gibt, dass die TCKs diskriminierungsfrei zur Verfügung stehen. Nun müssen wir im Gegenzug in die Tasten greifen und Entwicklungsleistung beisteuern. Der iJUG, seine Mitglieder, jeder Einzelne und auch deren Arbeitsgeber. Ohne euch geht es nur schleppend voran. Wenn jeder nur eine Stunde pro Woche investieren würde, wären dies Zigtausende Arbeitsstunden pro Java-Release. Bitte beteiligt euch! Falls ihr nicht wisst wie: Ihr könnt Bugs fixen, Handbücher überarbeiten, Tests schreiben oder euch für ein Open-Source-Stipendium bewerben (stipendium@ijug.eu). Außerdem kann euer Arbeitgeber euch einen Tag in der Woche für die Mitarbeit an Adoptium, Jakarta, Microprofile, OpenJ9 etc. freistellen (eine Win-Win-Situation, wenn ihr das Open-Source-Produkt produktiv verwendet!) und eure JUG kann dem neuen „Adopt A Spec"-Programm beitreten *[2]*. Möglichkeiten gibt es also genug und wir sind dankbar für jeden, der mithilft, aktiv die Zukunft der weltbesten Programmierplattform zu gestalten!

Referenzen

[1] Eclipse Adoptium, *https://adoptium.net/*

[2] Eclipse „Adopt a Spec"-Programm, *https://jakarta.ee/community/adopt-a-spec/*

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.

Ein Jahr Serverless: The good, the bad and the ugly

Christian Schmidt, tarent Solutions GmbH

Ein Jahr Serverless mit der Amazon Cloud (AWS). Da erlebt man viel Neues, was Spaß macht, interessant ist und einen in den Wahnsinn treibt. Alte Überzeugungen über Architektur und Testautomatisierung greifen plötzlich nicht mehr und es entsteht Komplexität, wo man sie nicht erwartet hätte. Dieser Artikel zeigt, wo und wie wir Serverless in unserem Projekt eingesetzt haben, wo die Erwartungen sich erfüllt haben, wo es nicht so gut lief und wo Serverless nicht die beste Wahl war. Daraus hervorgegangen sind Schwerpunkte und Best Practises über Staging, Testautomatisierung, Architektur und Komplexitätsbewältigung.

Was ist Serverless?

Serverless beschreibt eine Technologie, bei der die Aufmerksamkeit der Entwicklung vor allem auf die Domainlogik gelenkt werden soll und alles Umgebende allein durch Konfiguration und Environment aufgefüllt wird. Schaut man sich Serverless im Vergleich mit anderen Konzepten an, so fällt auf, dass sich klassische Software oft um Betriebssystembelange kümmern muss. Sie entscheidet, in welche Logfiles geschrieben wird, welche Datenbank auf welchem Port angesprochen wird, wie man auf einen SIGTERM-Command des OS reagiert und wie viele Instanzen auf welchen Servern laufen.

Eine Abstraktion sind Container-basierte Deployments, etwa per Docker, containerd, aber auch klassische Applicationserver. Dabei werden einzelne Positionen aus der Einflusssphäre der Software herausgenommen und von der Laufzeitumgebung übernommen. Zum Beispiel kümmert sich die Container-Runtime um die Betriebssystemabstraktion, aber auch um die Log-Infrastruktur. Als Schnittstelle wird hier oft der Stdout-Channel genommen. In anderen Bereichen werden beispielsweise auch Verbindungen zu Datenbanken über eine Service-Discovery angeboten und eine Skalierung automatisch vorgenommen. Die Software muss sich allerdings weiter um ihre Schnittstellen nach außen kümmern, eine UI oder ein HTTP-API bereitstellen und einen Commit an einen Messagebus senden. Es existiert also eine Korrelation zwischen der Größe des eingesetzten Codes, den genutzten Libraries und der Komplexität der bereitgestellten Infrastruktur.

Eine noch stärkere Abstraktion bildet nun Serverless, das originär auf Cloud-Plattformen zu finden ist. Dabei werden fast alle Elemente, die außerhalb der Domainlogik liegen, von der Cloud übernommen. Einzelne Logikblöcke werden als aufrufbare Funktionen in der Cloud deployt und Elemente wie ein HTTP-API, ein Messagebus, eine Datenbank aber auch Low-Level-Elemente wie Logging oder Skalierung werden gänzlich von der Cloud übernommen. Der Domaincode an sich kommuniziert lediglich über Cloud-SDKs und Glue-Code.

Serverless auf der AWS

In der Amazon Cloud (AWS) basiert Serverless auf den Lambda Functions. Das sind kleine, deploybare Einheiten, die bei Aufruf instanziiert und dynamisch skaliert werden. Unter anderen wird Java als Laufzeitumgebung für Lambdas angeboten. Ein Java-Lambda besitzt keine `main`-Methode. Stattdessen wird ein *RequestHandler* implementiert, der als Einsprungspunkt fungiert. Per Generics werden die Eingabe- und Ausgabetypen definiert, die außerhalb nach JSON serialisiert werden. Ein Lambda kann aus dem Kontext der Cloud aufgerufen werden und seinerseits andere Funktionen, aber auch Cloud-Elemente, ansteuern *(siehe Listing 1)*.

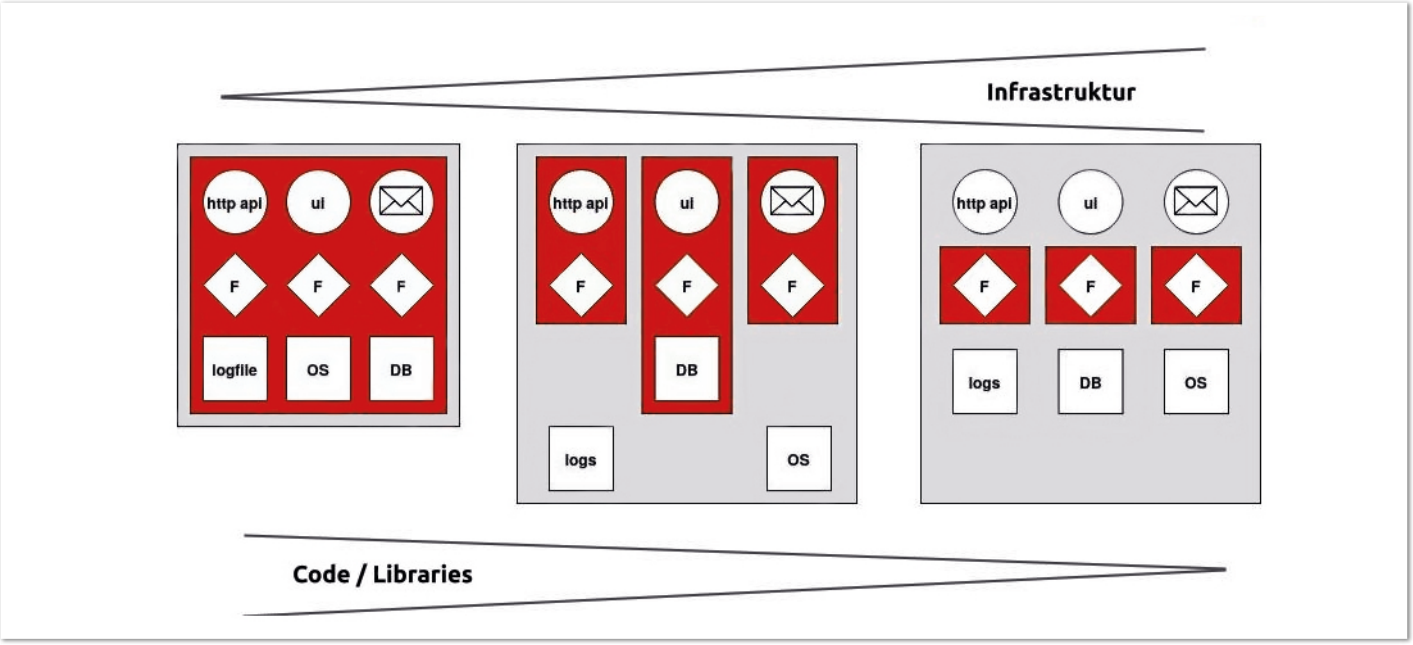


Abbildung 1: Serverless im Vergleich (© Christian Schmidt)

```
public class HelloWorldHandler
    implements RequestHandler<Input, Output> {

    private static final Log log = ...

    @Override
    public Output handleRequest(Input input, Context context) {
        log.info("handleRequest " + context.getAwsRequestId());
        return new Output(
            200, "Hello World " + input.getValue1() + "."
        );
    }
}
```

Listing 1: Ein Beispiellambda in Java

Es existiert also im Grunde nur zum Zeitpunkt seines Aufrufs. Abgerechnet wird dabei in Speicherklassen mal der genutzten Laufzeit. So ist beispielsweise ein Lambda, das 128 MB nutzt und zwei Sekunden läuft, günstiger als ein Lambda mit 256 MB und ebenfalls zwei Sekunden Laufzeit.

Ein Lambda allein ergibt meistens keinen Sinn, da sie nicht dafür ausgelegt sind, zum Beispiel Serverfunktionalitäten bereitzustellen. Stattdessen ist ein Lambda immer ein Teil einer größeren Applikation, die aus weiteren AWS-Elementen besteht. Einige Elemente sind in *Abbildung 2* zu sehen.

API Gateways bieten eine HTTP-Schnittstelle als Eingangstor zur Applikation an und darüber hinaus auch Möglichkeiten eines Internetzugangs zur Applikation.

SNS Topics bieten eine asynchrone Pub-/Sub-Funktionalität, um einen oder mehrere Empfänger per Events zu benachrichtigen.

SQS bildet eine FIFO-Queue ab, wobei eine Nachricht genau einmal verarbeitet wird. Daher bietet es sich an, in Kombination mit SNS Topics und Queues zu arbeiten, wenn man mehrere, fachlich getrennte Empfänger von Events hat.

Der *S3 Bucket* bietet einen BLOB-Store, um zum Beispiel Uploads, Bilder und Assets abzulegen. *DynamoDB* bietet eine NoSQL-Datenbank in der AWS. Beide erzeugen wiederum Events bei Änderung.

CloudWatch bietet eine weite Logging-Funktionalität und ist in der AWS nicht nur Datensenke, sondern kann auch per Filter und SNS Topics Events generieren, auf die wiederum Lambdas reagieren können.

Generell ist es möglich, alle Elemente über die UI der AWS anzulegen und zu konfigurieren, wie in *Abbildung 3* zu sehen ist. Programmatisch bietet AWS aber auch Konfigurationsdeskriptoren an, die über *CloudFormation* eingelesen und als Gesamtapplikation deployt werden können.

Listing 2 zeigt exemplarisch ein CloudFormation-Manifest, das das Lambda aus *Listing 1* und eine DynamoDB konfiguriert und in der AWS deployt. Das Lambda wird dabei über Environment-Variablen

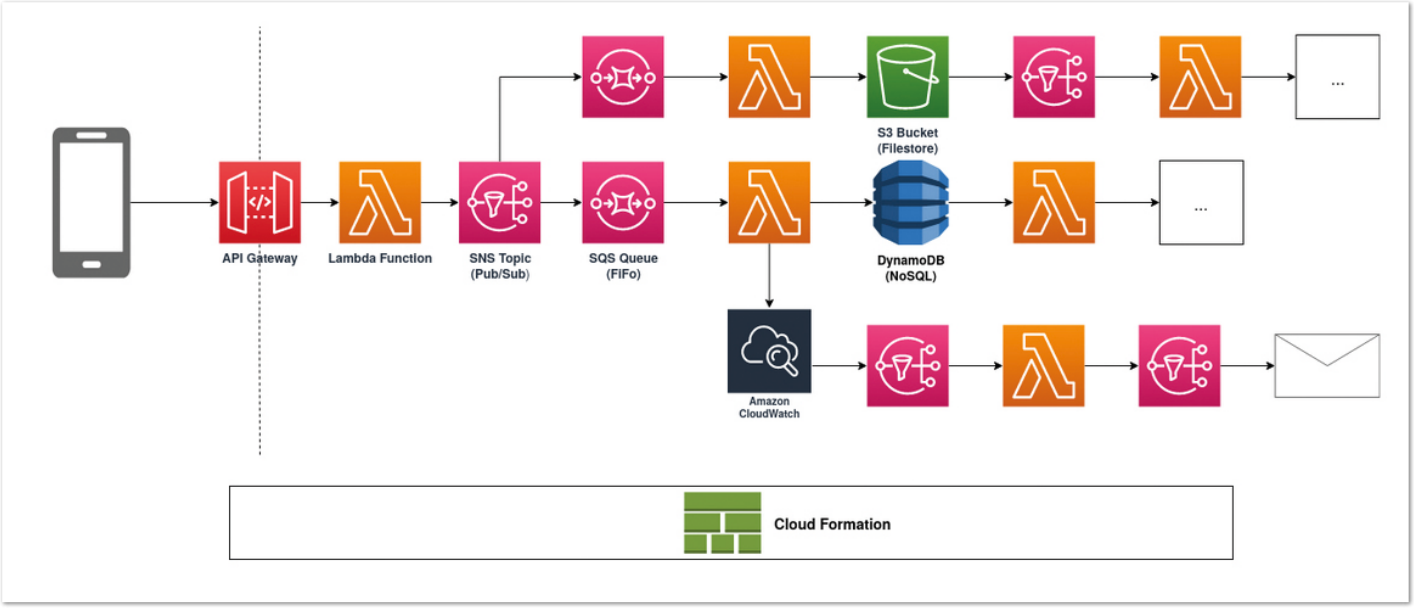


Abbildung 2: Lambdas als Teil einer größeren Applikation (© Christian Schmidt)

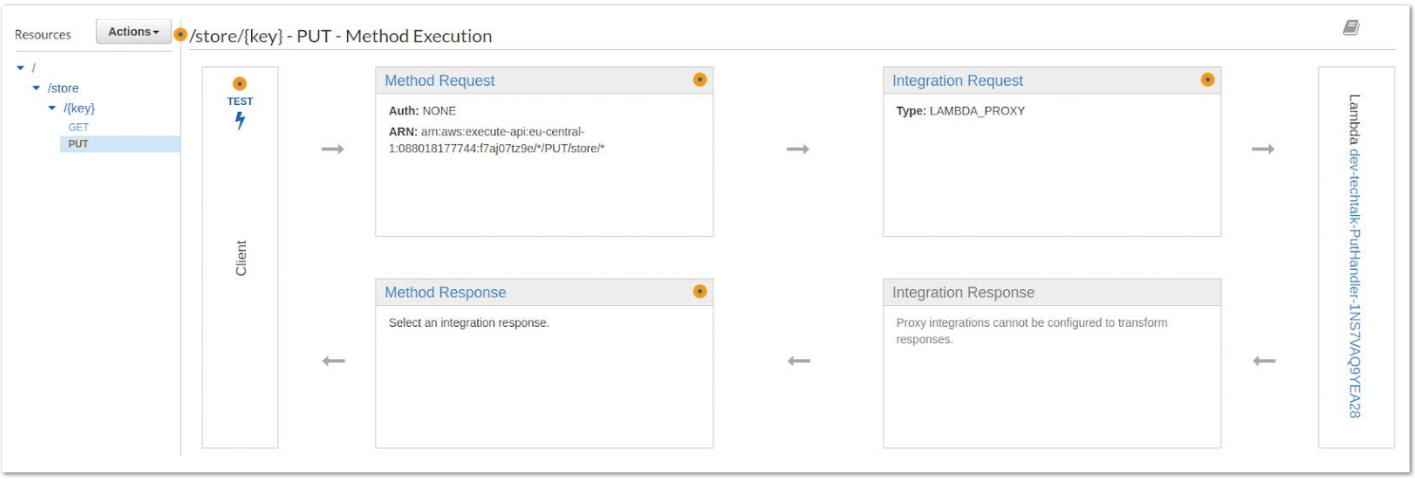


Abbildung 3: Ein per UI konfiguriertes API Gateway (© Christian Schmidt)

konfiguriert und bekommt die Referenz auf die DynamoDB per Variable `KEY_STORE_REF`, um damit die Verbindung aufzubauen und Schreiboperationen durchführen zu können. Die Referenz auf die DynamoDB wird dabei als „Amazon Resource Name" (ARN) abgebildet, die eine ID für jedes instanziierte Element in der AWS ist.

Der Vollständigkeit halber erweitert *Listing 3* das Lambda um einen Zugriff auf die DynamoDB. Dabei wird auf das SDK der AWS zurückgegriffen, das sämtliche Cloudelemente unterstützt. Mit dem ARN der Datenbank als Referenz kann eine Verbindung aufgebaut und ein neues Element in der NoSQL-Datenbank abgelegt werden.

Stacks

Eine Serverless-Applikation besitzt nach einer Zeit eine Vielzahl von Elementen. Um nicht den Überblick zu verlieren, bedarf es einer Architektur, die dem Rechnung trägt – und hierbei hilft CloudFormation.

```
Parameters:
  Stage:
    Type: String

Resources:
  HelloWorld:
    Type: 'AWS::Serverless::Function'
    Properties:
      Runtime: java8
      Handler: de.tarent.HelloWorldHandler::handleRequest
      CodeUri: ./build/libs/aws-lambda-tryout-1.0-all.jar
      Environment:
        Variables:
          KEY_STORE_REF: !Ref TestDynamoDb
      Tags:
        Environment: !Ref Stage
        Name: HelloWorld

  TestDynamoDb:
    Type: 'AWS::Serverless::SimpleTable'
    TableName: 'dynamoe'
    PrimaryKey:
      Name: id
      Type: String
    ProvisionedThroughput:
      ReadCapacityUnits: 5
      WriteCapacityUnits: 5
    Tags:
      - Key: "Environment"
        Value: !Ref Stage
      - Key: "Name"
        Value: "dynamoe"
```

Listing 2: Ein CloudFormation-Manifest mit einem Lambda und einer DynamoDB

Ein CloudFormation-Deployment umfasst immer eine zusammenhängende Gruppe von Elementen und bildet innerhalb der AWS einen Stack. Jeder Stack besitzt einen eindeutigen Namen und wird als Ganzes deployt. Die Überlegung ist nun, dass ein Stack einem klassischen Microservice entspricht. Also zum Beispiel lesende und schreibende HTTP-APIs, entsprechende Logik und eine Persistenzschicht. Auf Codeebene hat sich dabei angeboten, pro Stack ein Git-Repository anzulegen.

Abbildung 4 zeigt drei verschiedene Stacks, die zusammen kommunizieren. Dabei hat sich im Projekt für die Inter-Stack-Kommunikation das SNS Topic als asynchrone und das API Gateway als synchrone Schnittstelle etabliert.

Da sich der ARN eines Elements bei jedem Deployment eines Stacks ändert, um alle Elemente eineindeutig auch über die Zeit identifizieren zu können, kommt es zu Problemen, wenn nicht mehr gültige ARNs von anderen Stacks referenziert werden. CloudFormation hat hier das Element der Output-Referenzen.

Listing 4 zeigt ein Beispiel. Das SNS Topic besitzt einen ARN, der sich bei Deployments über die Zeit ändern kann. Um das Topic von anderen Stacks unabhängig referenzieren zu können, wird der Output *UpdateTopicRef* definiert. Dieser Output hat ebenfalls einen eigenen ARN, ändert sich aber im Verlauf der Deployments nicht mehr und kann von anderen Stacks jederzeit referenziert werden.

Testautomatisierung

Da Lambdas einen minimalen Umfang besitzen und auf einen einzelnen Domainfall begrenzt sind, ist die Unterscheidung zwischen Unit- und Komponententests auf Dauer zu fließend. Daher sind die Tests eines Lambdas so gebaut worden, dass sie das Lambda aufrufen und die angesteuerten Elemente per AWS SDK gemockt werden. Dies entspricht eher einem Komponententest als einem Unittest. Der Testumfang ist dabei überschaubar.

Ein Stack umfängt einen abgegrenzten Domainkontext und sollte somit auch als Ganzes getestet werden. Allein schon, um die verschiedenen Kommunikationswege und die Infrastruktur zu testen. Da eine Serverless-Applikation nur innerhalb der Cloud vollständig deployt werden kann, war hier ein Systemtest nötig, der auf einem lauffähigen Deployment aufbaut. Per AWS CLI ist es möglich, direkt auf Lambdas zuzugreifen, aber auch ein API Gateway auf die Testmaschine oder das CI/CD-System zu tunneln. Die Tests werden dann in ein eigenes Repository ausgelagert und zum Beispiel nach

```
public class HelloWorldHandler implements RequestHandler<Input, Output> {
    private static final String dynamoDbRef = System.getenv("DYNAMO_DB_REF");

    public Output handleRequest(Input input, Context context) {
        DynamoDbClient dbClient = DynamoDbClient.create();
        PutItemRequest putItemRequest =
            PutItemRequest.builder().tableName(dynamoDbRef).item(
                new ItemBuilder()
                    .with("id", context.getAwsRequestId())
                    .with("value1", input.getValue1())
                    .build()
            ).build();
        PutItemResponse putItemResponse = dbClient.putItem(putItemRequest);
…
```

Listing 3: Zugriff des Lambas auf die DynamoDB

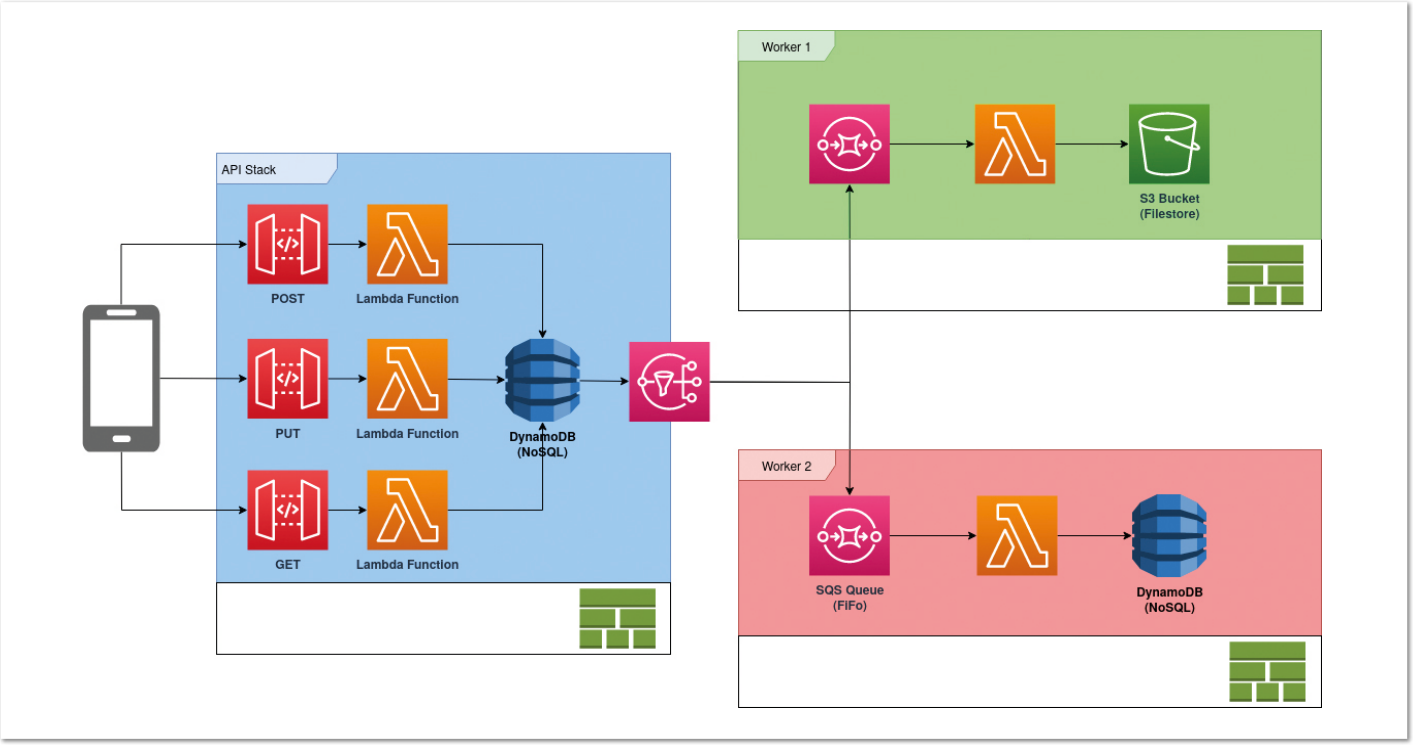


Abbildung 4: Elemente werden in Stacks zusammengefasst (© Christian Schmidt)

```
Resources:
 UpdateTopic:
   Type: AWS::SNS::Topic
...
Outputs:
 UpdateTopicRef:
   Description: update topic
   Export:
     Name: "UpdateTopicRef"
   Value: !Ref UpdateTopic
```

Listing 4: Stack mit einem Output

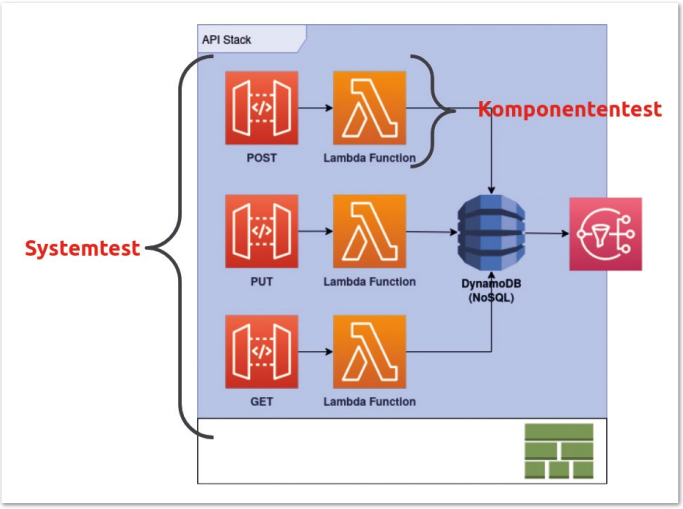


Abbildung 5: Übersicht über die verschiedenen Testarten (© Christian Schmidt)

einem Deployment ausgeführt. Ein Integrationstest bildet zum Schluss einen Stack-übergreifenden Test der gesamten Applikation. Dabei werden lediglich die Happy-Path-Fälle abgedeckt *(siehe Abbildung 5)*.

Staging

Natürlich sollen Änderungen in der Applikation im Vorfeld auf einer kontrollierten Umgebung ohne Kundeneinfluss getestet werden können. AWS bietet dabei kein originäres Staging-Konzept an, um zum Beispiel eine Produktiv- von einer Testumgebung zu unterscheiden.

Als Alternative bieten sich aber wieder die Stacks an, da sich hiermit eine Abgrenzung bewerkstelligen lässt. So wird den Stacknamen ein Stagename vorangestellt. So existieren zum Beispiel ein „Pre-Prod-Stack" und ein „Prod-Stack". Den einzelnen Elementen wird im Namen der Stackname ebenfalls vorangestellt, sodass sich diese in Übersichtsseiten ebenfalls unterscheiden lassen. Als Abschluss werden einzelnen Elementen Tags hinzugefügt, die ihre Zugehörigkeiten zu Stacks und Stages widerspiegeln. In *Listing 2* wird den Elementen ein „Environment"-Tag hinzugefügt, das die entsprechende Stage enthält.

Eine produktive Stage sollte aber im Sinne einer härteren Abgrenzung gegenüber den Teststages immer auf einem eigenen Cloud-Account laufen, da sich eine versehentliche Vermischung von Stages hier wesentlich schwerer gestaltet.

CI/CD

In der Kombination lässt sich nun mit einem beliebigen CI/CD-Tool eine Build-Kette aufbauen, um einzelne Stacks zu deployen und zu testen. Dabei kommt das AWS-CLI zum Tragen, das die Kommunikation in die AWS-Cloud stemmt.

Im ersten Schritt wird das Repository ausgecheckt, das einen Stack umfasst. Einzelne Lambdas, die im Stack liegen, können lokal mit Mocks getestet werden. Im nächsten Schritt wird der Dev-Stack deployt. Dieser wird dann per Systemtest einzeln, und im Zusammenspiel mit anderen Stacks mittels Integrationstest getestet.

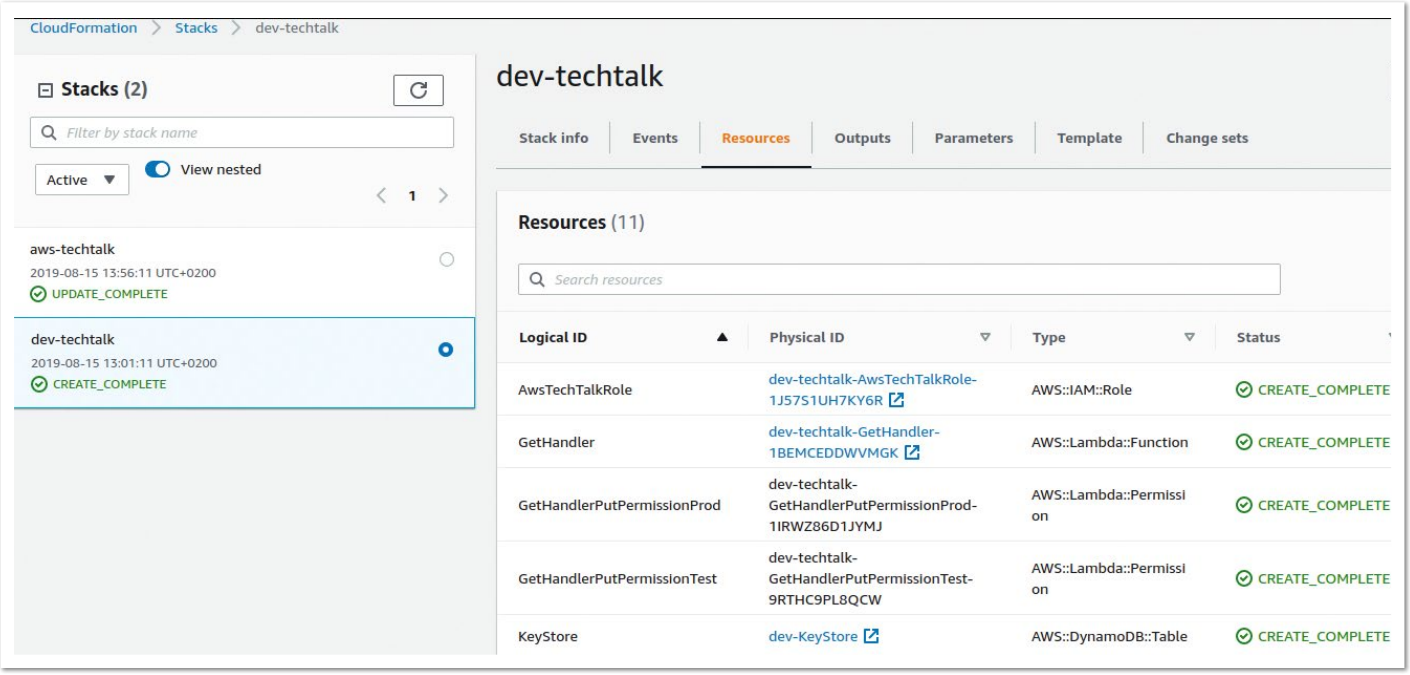


Abbildung 6: Zwei verschiedene Stacks mit entsprechend benannten Elementen (© Christian Schmidt)

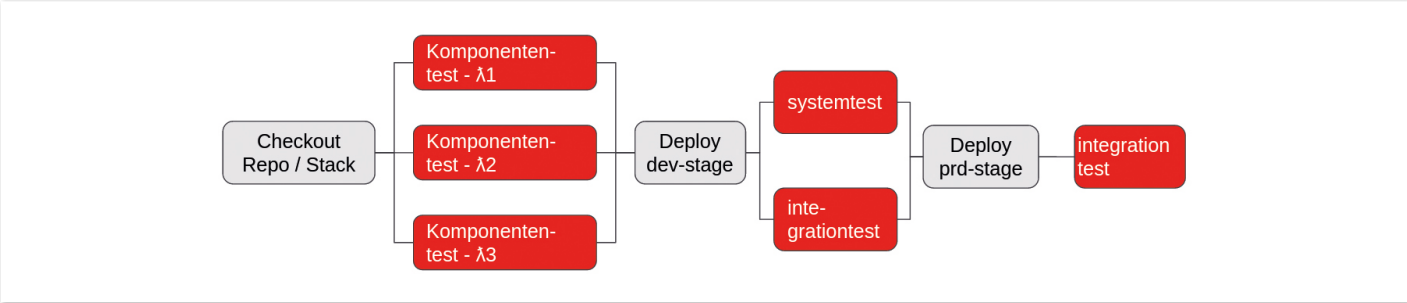


Abbildung 7: Eine mögliche CI/CD-Kette (© Christian Schmidt)

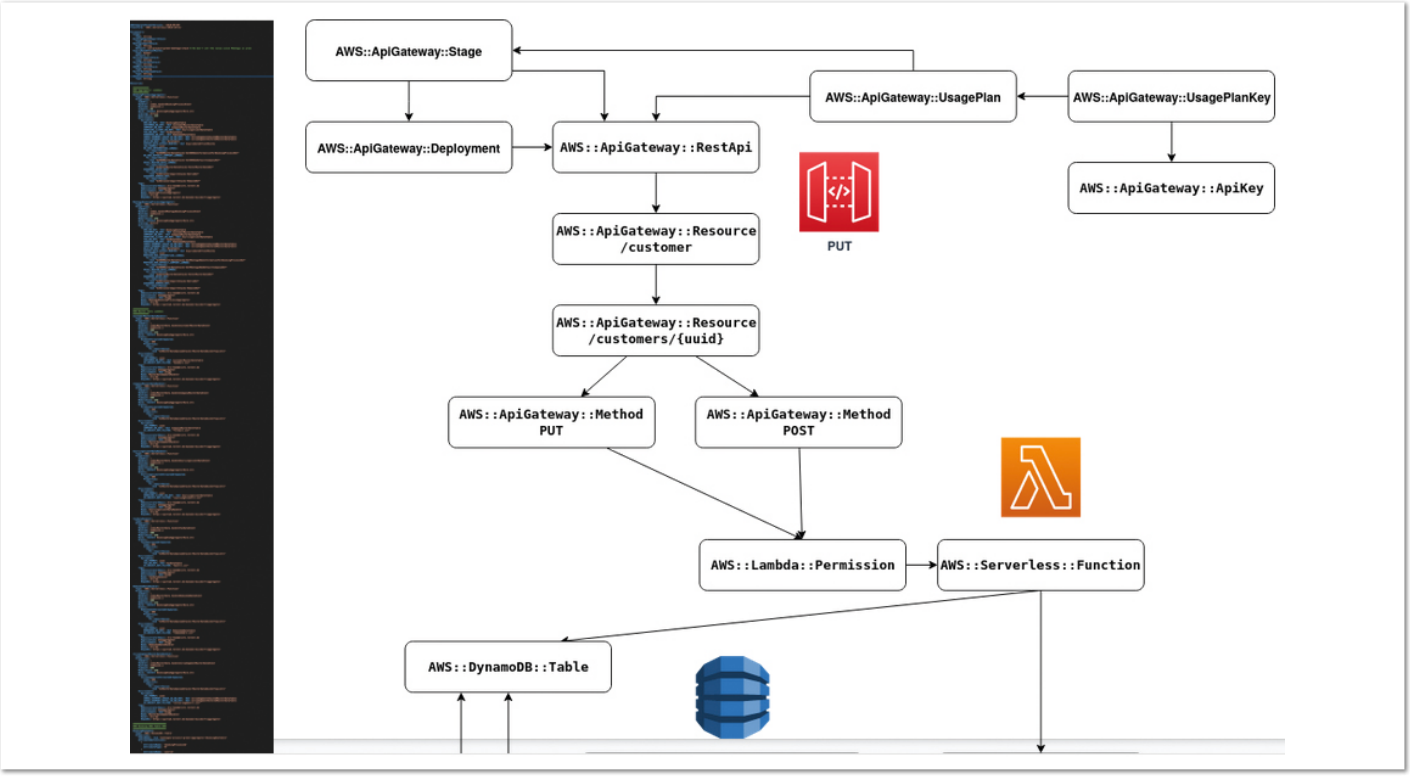


Abbildung 8: Auswahl an verschiedenen Subelementen (© Christian Schmidt)

Bei erfolgreichen Tests wird dann der Prod-Stack deployt und ebenfalls per Integrationstest auf Funktionsfähigkeit getestet.

The Good

Mit Serverless lassen sich vor allem sehr schnell Prototypen bauen aber auch Applikationen mit stark schwankender Auslastung. Die Skalierung geht dabei im Standard bis zu 1.000 parallelen Funktionsaufrufen pro Lambda. Abgerechnet wird einzig über die Anzahl der Aufrufe beziehungsweise die verbrauchte Laufzeit. Wird ein Lambda nicht aufgerufen, so entstehen auch keine Kosten.

Durch eine Vielzahl an Cloudelementen, die „out of the box" existieren, konzentriert sich die Entwicklung stark auf die Domainlogik. Für Cloud-Elemente wie dem API Gateway, der DynamoDB einschließlich Lambdas existieren UI-Elemente und eine umfassende Kontrolle über CloudFormation. Nutzt man Lambdas mit Python oder JavaScript, so bietet sich sogar ein Live-Editor im UI an.

Selbstverständlich bedeutet die Konfiguration der entsprechenden Elemente ebenfalls Aufwand, jedoch lässt sie sich von der eigentlichen Entwicklung innerhalb der Domain sehr gut trennen. Ein transparenter Technologiefluss stellt sich darüber hinaus ein, da es ohne Weiteres möglich ist, ein Lambda mit einem Container oder einer VM-Instanz zu verbinden und umgekehrt.

The Bad

Eine Nutzung von Serverless bedeutet auch immer eine Abhängigkeit gegenüber dem Cloudanbieter. Auch bei anderen Anbietern ist die Nutzung des entsprechenden SDK essenziell. Eine native Entwicklung ist unumgänglich, will man den vollen Funktionsumfang nutzen, da Abstraktionsframeworks immer nur den kleinsten gemeinsamen Nenner abbilden.

Ein effektives Testen über ein Staging beziehungsweise eines einzelnen Stacks ist nur mit einem Deployment in der Cloud möglich, wodurch auch zusätzliche Kosten pro Umgebung entstehen. Natürlich werden Lambdas nur pro Aufruf abgerechnet, was den finanziellen Aufwand überschaubar gestaltet. Vorsicht aber vor Elementen, die per Provisionierungsintervall abgerechnet werden, wie relationale Datenbanken und etwaige VM-Instanzen.

Ein weiterer Kostenfaktor sind kaskadierende Lambdaaufrufe. Zu beachten ist immer, dass vom Start bis zum Ende des Aufrufs bezahlt wird, also auch für zwischenzeitliche IOwaits. Bei drei aufeinanderfolgenden Lambdas können sich somit die Kosten schnell verdreifachen. Auch externe APIs, die relativ langsam sind, können bei Lambdas schnell Kosten verursachen. Ein Legacysystem mit einer Roundtripzeit von über 20 Sekunden war im Projekt ein schnell übersehener Kostentreiber.

The Ugly

Jedes Element in der AWS besitzt eine Vielzahl von Eigenschaften und Konfigurationsmöglichkeiten. Gerade beim Deployment mit CloudFormation stößt man sehr schnell auf eine Fülle von verschiedensten Sub-Elementen, um beispielsweise ein API-Gateway zu konfigurieren oder eine DynamoDB zu skalieren. *Abbildung 8* veranschaulicht ein simples HTTP PUT, gesichert mittels API Key, auf ein Lambda mit Schreiboperationen in eine automatisch skalierte DynamoDB. Das Listing links daneben kann man bewusst nicht lesen, soll aber einen Eindruck von Umfang und Komplexität geben.

Eine weitere Unschönheit liegt in der Natur der Element-Referenzen. In *Listing 4* wird in der letzten Zeile der ARN von *UpdateTopic* per `!Ref` ermittelt. Leider ist die Referenz eines Elements mitnichten immer ein ARN! Beispielsweise verbirgt sich bei einer SQS-Queue dahinter die Queue-URL statt des ARN. Möchte man den ARN, so lässt er sich per `!GetAtt MyQueueRef.Arn` ermitteln. Unter welchen Umständen man nun einen ARN oder etwas anderes bekommt, ist Sache des entsprechenden Elements. Eine hilfreiche Übersicht bietet die Quelle *[1]*, sie eignet sich auch generell als Hilfe.

Fazit

Serverless-Applikationen bieten eine faszinierende Art der Entwicklung, die sehr schnell brauchbare Ergebnisse liefert und in einem sehr hohen Maß skaliert. Die eigentliche Programmierung konzentriert sich dabei stark auf die Domain und verlagert nichtfachliche Aspekte in die Konfiguration. Die Cloud als Environment bietet viel out of the box und ein einheitliches Look-and-Feel. Aber so wie sich die Applikation leicht in die horizontale Dimension skalieren lässt, so stößt sie bei der Konfiguration recht schnell an die Grenze.

Denn Serverless ist beileibe kein Garant für eine Reduktion der Komplexität. Sie verschiebt sich lediglich in die Konfiguration, die schnell umfangreich wird und ein weites Netz von Elementen und Subelementen umfasst. Mittels Stacks und Konventionen lässt sich dies aber gut managen.

Quellen

[1] *https://theburningmonk.com/cloudformation-ref-and-getatt-cheatsheet*

[2] *https://github.com/DarkToast/aws-techtalk*

Christian Schmidt

tarent solutions GmbH

c.schmidt@tarent.de

Christian Schmidt, 35 Jahre aus Siegburg, ist nun seit 2010 bei der tarent Solutions GmbH in Bonn tätig. Hauptsächlich auf der JVM, aber auch mit Golang und NodeJS, managt er moderne Microservice-Architekturen und Cloud-native Applications in der Entwicklung oder Transformation. Dabei liegt das Augenmerk auf der ganzheitlichen Betrachtung in einer domainzentrierten und anforderungsgetriebenen Entwicklung von der Planung bis zum Betrieb mit agilen Entwicklungsmethoden und Domain-driven Design.

Mit Java Serverless to the Future!

Niko Köbler

Werden die Begriffe Java und Serverless in einem Atemzug genannt, zucken (leider) immer noch viele zusammen. Zu groß ist die Angst, durch die ominöse Cold-Startup-Latency, das Starten der JVM und Initialisieren der benötigten Java-Objekte zu viel Zeit zu verlieren und damit Wasser auf die Mühlen derer zu gießen, die seit über 25 Jahren nicht müde werden zu behaupten, dass Java langsam und schwerfällig sei und überhaupt die Sprache keine Zukunft habe. Aber nicht nur im Serverless-Umfeld ist die Startgeschwindigkeit von Ressourcen (Services, Containern etc.) ein Thema, auch in komplexen Microservice- und Self-Contained-Systems(SCS)- basierten Systemen, in denen mithilfe einer Container-Orchestrierung (zum Beispiel Kubernetes, OpenShift etc.) schnell neue Instanzen eines Containers gestartet werden müssen, zählt heute jede Millisekunde.

Verschiedene Framework-Anbieter haben sich darum gekümmert und die verschiedensten Optimierungen an ihren Frameworks vorgenommen. Red Hat hat mit Quarkus *[1]* 2019 gleich ein ganz neues Application Framework entwickelt und musste sich nicht mit der Anpassung von bestehenden Ressourcen auseinandersetzen. So ist die Verlagerung von vielen Bytecode-Optimierungen, die die JVM eigentlich zur Laufzeit durchführt, hin zum Kompilierungszeitpunkt eine der wichtigsten und auffälligsten Änderungen. Ein sehr viel schnellerer Start einer Java-Anwendung ist somit möglich. Quarkus verspricht auf der Website große Verbesserungen nicht nur hinsichtlich der Startup-Zeit, sondern auch des Speicherverbrauchs: So soll deutlich weniger Speicher benötigt werden, als bislang gewohnt.

Eine weitere Optimierung, ebenfalls für Startup und Speicherallokation, bietet Quarkus durch die sehr einfache Integration und Nutzung der GraalVM *[2]*. Sie ermöglicht es, Java-Bytecode in nativen Bytecode der jeweils genutzten Plattform (Linux, OS X, Windows etc.) zu übersetzen. Somit wird zur Laufzeit keine JVM mehr benötigt, die ihrerseits eben auch eine Startup-Zeit hat und weiteren Speicher braucht.

Die Nutzung der GraalVM ist nicht gerade trivial und hat in ihren Anfangstagen viele Entwickler/innen mit ihrer Komplexität etwas abgeschreckt. Diese Hürde nimmt Quarkus und bietet eine einfache Nutzung und Konfiguration der GraalVM an, um den eigenen Java-basierten Service in ein natives Artefakt übersetzen zu können,

ohne viel Detailwissen rund um Graal haben zu müssen. Möchte man von der Default-Konfiguration weg und eigene Optimierungen vornehmen, so ist dies dennoch möglich. Quarkus nutzt überall viele Standard-Einstellungen und Konventionen, bietet aber auch jeweils die Möglichkeit der eigenen Anpassung.

Diese Optimierungen machen Quarkus zu einem Container-First Framework, wie Red Hat es auf der Quarkus Website nennt. Doch Quarkus verfolgt nicht nur eine Container-Strategie, sondern versucht auch, anderen Cloud-Anforderungen gerecht zu werden. So bietet Quarkus beispielsweise viele Extensions für einzelne Services von verschiedenen Cloud-Anbietern an und vereinfacht das Arbeiten mit diesen APIs um ein Vielfaches.

Quarkus Funqy

Ein weiterer Schwerpunkt von Quarkus ist eine Serverless-Strategie mit dem Namen Funqy *[3]*, der eine Anspielung auf die Functions ist, die vorrangig in einer Serverless-Umgebung betrieben werden. Mit Funqy stellt Quarkus ein portables Java-API bereit, um zielumgebungsunabhängig Funktionen schreiben zu können, die dann in einer von vielen FaaS-Umgebungen (Function-as-a-Service), wie etwa AWS Lambda, Azure Functions und Knative/Knative Events, betrieben werden können.

Möchte man die Funqy Functions nicht in einer dieser Cloud-Umgebungen betreiben, können diese auch als eigenständiger, HTTP-basierter Service deployt werden (und sogar nicht-serverless in einem Container!) Die umgebungsspezifischen Wrapper und Deployment-Artefakte werden zum Build-Zeitpunkt automatisch durch Quarkus und die einzelnen Extensions erzeugt, sodass man sich nicht (oder nur wenig) darum kümmern muss, solange man keine umfangreichen Anpassungen vornehmen oder auf umgebungsspezifische APIs zugreifen möchte.

Ein portables und umgebungsunabhängiges API setzt natürlich immer eine gewisse Abstraktion voraus und kann sich bei den Features und Protokollen nur auf den kleinsten gemeinsamen Nenner fokussieren. Dies schränkt die Funktionalität und Flexibilität des API gegebenenfalls etwas ein, führt bei Funqy jedoch dazu, dass es ein sehr stark optimiertes (auf die Anforderungen von FaaS) und gleichzeitig leichtgewichtiges Interface ohne großen Overhead geworden ist.

Bill Burke, einer der Red-Hat-Entwickler im Quarkus-Team, schreibt auf seiner Website *[4]* zur Idee hinter Funqy: Man muss lediglich eine Java-Methode schreiben, die einen optionalen Eingabeparameter und einen optionalen Rückgabewert hat *(siehe Listing 1)*. Die Ein- und Ausgabeparameter können primitive Datentypen sein, oder aber auch POJOs (Plain Old Java Object) *(siehe Listing 2)*. An diese Methode kommt die neue Annotation `@Funq` – und fertig ist die Funktion. Funqy-Klassen sind normale Quarkus-Komponenten, die in andere Komponenten via CDI oder auch Spring DI injiziert werden können. Außerdem ist es (natürlich) auch möglich, eine asynchrone Funktion mittels Smallrye Mutiny, der reaktiven Implementierung in Quarkus, zu schreiben *(siehe Listing 3)*.

HTTP, aber kein REST

Wie bereits zuvor beiläufig erwähnt, spricht Funqy HTTP. Hierbei muss allerdings betont werden, dass es wirklich HTTP und kein REST ist. Funqy will auch gar kein „Ersatz" oder Ähnliches für

```
import io.quarkus.funqy.Funq;

public class SimpleFunction {
  @Funq
  public String hello(String name) {
    return "Hello, " + name;
  }
}
```

Listing 1: Eine einfache Funktion mit simplen Datentypen – das Minimum für Funqy

```
import io.quarkus.funqy.Funq;

import javax.inject.Inject;
import javax.enterprise.context.ApplicationScoped;

public class GreetingFunction {
  @Inject
  GreetingService service;

  @Funq
  public Greeting greet(Friend friend) {
    return service.greet(friend.getName());
  }
}

@ApplicationScoped
public class GreetingService {
  public Greeting greet(String name) {
    return new Greeting(name);
  }
}

public class Greeting {
  private String message;
  public Greeting(String name) {
    this.message = "Hello, " + name;
  }
  // getter...
}

public class Friend {
  public String name;
  // getter/setter ....
}
```

Listing 2: Komplexere Funktion mit POJOs als Ein-/Ausgabeparameter und CDI-injiziertem Service

```
import io.quarkus.funqy.Funq;
import io.smallrye.mutiny.Uni;

public class SimpleFunction {
  @Funq
  public Uni<Greeting> reactiveGreeting(String name) {
    ...
  }
}
```

Listing 3: Eine asynchrone, reaktive Funktion mit Smallrye Mutiny

etablierte Konzepte wie REST sein, sondern eher die Einfachheit und Flexibilität von HTTP nutzen, um in möglichst vielen Umgebungen betrieben werden zu können.

So sprechen die Cloud-Services in AWS, Azure & Co. intern auch alle über HTTP miteinander und beschränken sich hierbei meist auf die Methoden `GET` und `POST`. Und eben dies macht auch Funqy – andere Methoden außer GET und POST sind schlicht nicht möglich

und jede mit `@Funq` annotierte Methode stellt einen Endpunkt dar, der gleichzeitig mit GET und POST aufgerufen werden kann. Der Methodenname ist hierbei der Pfad des HTTP-Request. Soll ein anderer Pfadname verwendet werden, akzeptiert die `@Funq`-Annotation noch ein String-Value als alternativen Namen.

Der Content-Type ist, sowohl eingehend als auch ausgehend, auf JSON festgelegt. Damit zielt Funqy ebenfalls wieder auf Einfachheit ab. Bestmögliche Kompromisse, wo nur irgendwie möglich. Wird eine Funqy-Methode über GET aufgerufen, stellen die Eingabeparameter, etwa eine `Map<String, String>` oder ein POJO, die Query-Parameter dar. Die Feld- beziehungsweise Schlüsselnamen sind die Parameter-Keys, deren Werte die Parameter-Values. Wird die gleiche Methode über POST aufgerufen, ist der Eingabeparameter der Methode der Request-Body als JSON-Repräsentation.

Wer in seinen Services und Funktionen auf HTTP-Header, Cache-Control, Conditional GETs etc. zurückgreifen möchte oder muss, wird mit Funqy nicht glücklich werden, da diese Features mit Funqy nicht möglich sind. Hierfür ist der Einsatz eines REST-basierten Ansatzes, wie JAX-RS/RESTeasy mit Quarkus, die bessere Alternative.

Auch wer Zugriff auf umgebungsspezifische APIs benötigt, wie zum Beispiel das AWS `Context`-Objekt in Lambda-Funktionen, wird mit Funqy schnell an die Grenzen stoßen. Der Zugriff auf Context-Objekte ist eigentlich nicht unterstützt. Es gibt zwar Ausnahmen davon, auf die sollte man sich jedoch nicht verlassen. Die eigentliche Absicht von Funqy ist ja, wie bereits mehrfach erwähnt, die Unabhängigkeit der Umgebung für die Implementierung zu gewährleisten und erst durch die jeweiligen Extensions die umgebungsspezifischen Artefakte zu generieren.

Für den Einsatz in Cloud-spezifischen Diensten ist Funqy hingegen ideal. Die meiste Inter-Service-Kommunikation in einer Cloudumgebung geschieht vornehmlich über HTTP-POST-Requests und die benötigte Information ist im Body des Request zu finden. Servicespezifische Wrapper werden dann von den Funqy Extensions dazu generiert und als Entwickler/in kann man sich auf die Implementierung der Fachlichkeit und der benötigten Funktionalität konzentrieren. Events aus Streams, Queues, Topics und anderen Services zu verarbeiten wird so einfach wie das Schreiben eines einfachen POJOs.

Deployment – up, up to the clouds

Quarkus unterstützt auch in bekannt komfortabler Manier die Funqy-Deployments. Als Beispiel soll hier das Funqy-Demo-Projekt `dasniko/funqy-demo` des Autors mit dem AWS-Branch *[5]* dienen. Es kann mit Maven über den Befehl `mvn package` gebaut und paketiert werden. Dieser Build ist noch Java-basiert. Möchte man einen nativen Build mit der GraalVM durchführen, kann dies mit dem Profil `-Pnative` geschehen. Eine lokale GraalVM ist dafür aber notwendig. Kann oder möchte man nicht auf eine lokale GraalVM zurückgreifen, stellt Quarkus bekanntermaßen einen Container-basierten Build mit den Schaltern beziehungsweise Properties zur Verfügung *(siehe Listing 4)*. Damit wird der Build in einem vollständig mit der GraalVM vorkonfigurierten Container durchgeführt und man erhält ein Linux-natives Binary als Ergebnis. Nach dem Build existieren mehrere Artefakte im target-Verzeichnis:

- function.zip
- funqy-demo.jar
- funqy-demo-runner.jar (wenn Java-basiert, oder)
- funqy-demo-runner (wenn nativ)
- sam.jvm.yaml
- sam.native.yaml

Die JAR-Dateien enthalten erwartungsgemäß die kompilierten Projektklassen, die Runner-JAR zusätzlich noch die generierten Quarkus-Klassen mit der benötigten Quarkus-„Infrastruktur", die für einen erfolgreichen Betrieb des Service notwendig sind. Im lib-Ordner befinden sich die nötigen Drittbibliotheken, die zur Laufzeit ebenfalls benötigt werden. Alles zusammen packt Quarkus in die function.zip-Datei, die so direkt zu AWS Lambda hochgeladen und dort in einer Lambda-Java-Laufzeitumgebung deployt werden kann.

Im Falle eines nativen Builds wird die *...runner*-Datei (ohne Dateiendung) erzeugt, die das native Artefakt darstellt und bereits alle benötigten Abhängigkeiten beinhaltet. Dieses Runner-Binary wird ebenfalls mit einer für AWS bestimmten Konvention zur *function.zip* gepackt, um dort als sogenannte „provided Laufzeitumgebung" *[6]* ausgeführt werden zu können.

Es fehlt allerdings noch etwas an Cloud-Infrastruktur, denn der Service stellt HTTP-Endpunkte bereit, AWS Lambda ist aber bekanntermaßen nicht direkt von außen aufrufbar. Hierfür wird noch ein HTTP-API-Gateway im AWS-Universum benötigt. Damit auch die erste Hürde der Konfiguration der benötigten Infrastruktur einfacher genommen werden kann, generiert die Quarkus Amazon Lambda HTTP-Extension *[7]* gleich zwei SAM-Templates mit, die *sam.*.yaml*-Dateien. SAM steht für „Serverless Application Model" und ist ein Deployment-Tool von AWS für Serverless-Komponenten *[8]*. SAM ist ein Superset der AWS-CloudFormation-Spezifikation, dem Infrastructure-as-Code-Werkzeug in AWS, und vereinfacht in vielen Teilen das Deployment von Serverless-Artefakten.

SAM bietet nicht nur das Deployment zu AWS an, sondern mit *SAM local* auch eine Möglichkeit, die zu deployende Funktion vorab lokal zu testen. Hierfür wird lokal ein Docker-Container mit der AWS-Lambda-Umgebung gestartet und die *function.zip* deployt und initialisiert. Das ist hilfreich für lokale Tests und das Debugging des eigenen Codes. Die entsprechende SAM-YAML-Datei (jvm oder native) dient hierfür als Basis. Der Befehl *(siehe Listing 5)* startet die lokale Lambda-Umgebung und die Funktion ist unter http://localhost:3000 aufrufbar. Für das Deployment hin in die AWS Cloud sind zwei Schritte nötig.

Der in *Listing 6* gezeigte Befehl paketiert die Funktion für AWS-Zwecke und lädt sie anschließend in den angegebenen S3-Bucket hoch.

```
quarkus.native.container-build=true
quarkus.native.container-runtime=docker   (alternativ: 'podman')
```

Listing 4

Das hochgeladene Artefakt bekommt einen zufälligen Dateinamen, damit mehrere Deployments keine anderen Versionen oder Deployments im S3-Bucket überschreiben. Eine neue YAML-Definition mit dem generierten Dateinamen wird als *package.yaml* erzeugt. Dieser Schritt ist notwendig, da ein direktes Deployment von einem lokalen Ordner hin zu Lambda nicht möglich ist und immer der Weg über einen S3-Bucket vorgenommen werden muss. Anschließend kann das eigentliche Deployment starten, wie auch die Ausgabe des package-Befehls in *Listing 7* zeigt.

Der Stack-Name ist frei wählbar und dient der Identifizierung des Deployment in CloudFormation. Das Deployment erzeugt dann die eigentliche Lambda-Funktion, das HTTP-API-Gateway (aufrufbar über https!) und gegebenenfalls noch weitere notwendige Infrastrukturkomponenten (Policies, Log-Groups etc.) Wurde das Deployment erfolgreich durchgeführt, gibt SAM die URL aus, unter der der Service in AWS erreichbar ist.

Die generierte YAML-Datei ist für einen ersten, schnellen Erfolg ein sehr guter Ansatz. Spätestens jedoch, wenn eine komplexere, für Quarkus nicht-vorhersehbare Konfiguration notwendig wird, reicht das generierte Artefakt nicht mehr aus. Solche Konfigurationen können beispielsweise erweiterte Rechte zum Zugriff auf andere AWS-Ressourcen wie S3-Buckets, SQS Queues, eine DynamoDB oder Ähnliches sein. Aber auch die Deployment-Beschreibung einer weiteren Ressource selbst könnte Bestandteil der YAML-Datei werden, wenn diese gemeinsam mit der Lambda-Funktion verwaltet werden soll. Dies ist möglich, da SAM, wie bereits erwähnt, ein Superset der CloudFormation-Spezifikation ist und damit auch alle CloudFormation-Definitionen enthalten kann und nicht nur auf Serverless-Artefakte beschränkt ist.

Besteht dieser Bedarf, empfiehlt es sich, eine generierte YAML-Definition als Vorlage in ein Projektverzeichnis zu kopieren, an die Projektbedürfnisse anzupassen und gemeinsam mit den anderen Projektressourcen in die Versionsverwaltung einzuchecken. Somit steht die Deployment-Definition jederzeit allen Projektbeteiligten zur Verfügung und kann sich zusammen mit dem Lebenszyklus des Projektes verändern und verwaltet werden.

Analog zum hier vorgestellten Deployment in die AWS Cloud zum Lambda-Dienst stehen auch für andere Clouds und FaaS-Umgebungen Quarkus Extensions bereit. Je nach Umgebung werden auch hier benötigte und leicht verwend- und anpassbare Vorlagen erzeugt. Teilweise, wie bei AWS, während des Deployment, teilweise auch schon während des Erzeugens der Projekt-Vorlage, wenn der Quarkus-Code-Generator *[9]* verwendet wird. Eine gut verständliche Dokumentation und Guides helfen hier bei den verschiedenen Extensions weiter, für einen schnellen und nachhaltigen Erfolg.

Fazit

Manchmal liegt das Einfache so nah. Und oftmals ist das Einfache, das sich auf eine Kernkompetenz konzentriert, das, was wirklich zählt und wichtig ist. Es muss nicht immer mit Kanonen auf Spatzen geschossen werden, wenn man den Overhead an Funktionalität eh nicht benötigt. Und gerade im Serverless-Umfeld, wo less doch more ist, tun es oft die einfachen, kleinen, schmalen Services. Diesem Bedarf trägt das Quarkus Funqy Serverless API Rechnung. Nicht mehr, aber auch nicht weniger. Und das ist gut so.

```
sam local start-api \
  --template target/sam.jvm.yaml
```

Listing 5

```
sam package \
  --template target/sam.jvm.yaml \
  --output-template-file packaged.yaml \
  --s3-bucket quarkus-deployment-bucket
```

Listing 6

```
sam deploy \
  --template-file packaged.yaml \
  --stack-name funqy-demo
```

Listing 7

Java für Serverless wird mit Quarkus nicht nur Funqy, sondern vor allem performant und einfach wartbar. Nutzt man dann noch die per GraalVM erzeugten nativen Artefakte, wird man schnell über Cold Startup Latencies und Speicherverbrauch nur noch müde lächeln können. Mit Java Serverless to the Future!

Referenzen

[1] *https://quarkus.io*
[2] *https://www.graalvm.org*
[3] *https://quarkus.io/guides/funqy*
[4] *https://bill.burkecentral.com/2020/06/09/quarkus-funqy-portable-function-api/*
[5] *https://github.com/dasniko/funqy-demo/tree/aws*
[6] *https://docs.aws.amazon.com/lambda/latest/dg/runtimes-custom.html*
[7] *https://quarkus.io/guides/amazon-lambda-http*
[8] *https://aws.amazon.com/serverless/sam/*
[9] *https://code.quarkus.io/*

Niko Köbler

Niko Köbler macht irgendetwas mit Computern, On-Premises, im Web und in der Cloud. Er schreibt und deployt Software und hilft anderen dabei, dies zu tun. Serverless und Identity & Access Management mit Keycloak sind dabei seine Steckenpferde. Seit Kurzem veröffentlicht er auch regelmäßig YouTube-Videos zu diesen Themen. Daneben ist er Co-Lead der JUG Darmstadt, Sprecher auf internationalen Konferenzen und Autor des Buchs „Serverless Computing in der AWS Cloud".

Die AWS-Lambda-SQS-Integration unter der Lupe

Frank Rosner, Datastax

AWS bietet einen großen Baukasten, um skalierbare, hochverfügbare und sichere Cloudanwendungen zu erstellen. AWS Lambda ist ein Dienst, um Code auf Anfrage auszuführen. Er wird üblicherweise verwendet, um zustandslose Programme, die auf bestimmte Ereignisse reagieren müssen, kostengünstig und flexibel zu implementieren.

Lambda-Funktionen können auf verschiedene Arten aufgerufen werden, beispielsweise durch ein API-Gateway als Teil eines Serverless-Backends. Im Bereich der Ereignisverarbeitung (engl. event processing) wird Lambda häufig in Kombination mit Ereignisquellen wie Amazon SQS (Simple Queue Service = Einfacher Warteschlangendienst), Amazon SNS (Simple Notification Service = Einfacher Benachrichtigungsdienst) oder Amazon Kinesis Data Streams (Datenströme) verwendet. Dieser Artikel wird sich näher mit der Integration von Lambda und SQS auseinandersetzen.

Ereignisverarbeitung mit Lambda und SQS

SQS ist ein verwalteter, verteilter Warteschlangendienst. Konsumenten müssen die Warteschlange aktiv abfragen (engl. polling), um neue Nachrichten lesen zu können. Praktischerweise kümmert sich AWS automatisch um das regelmäßige Abfragen, wenn man einer Lambda-Funktion eine SQS-Ereignisquelle zuordnet.

Aber wie funktioniert die Integration genau? Wie beeinflussen die verschiedenen Konfigurationsparameter wie das Abrufverhalten, Sichtbarkeitszeitbeschränkungen und Nebenläufigkeitsbeschränkungen das Verhalten der Integration? Genau diese Fragen werden wir im Folgenden beantworten. Der verbleibende Teil des Artikels ist wie folgt strukturiert: Der nächste Abschnitt beleuchtet die relevanten Konfigurationsparameter von Lambda, SQS und der Lambda-SQS-Ereignisquellenzuordnung. Anschließend nehmen wir die Integration selbst genauer unter die Lupe und schauen uns dabei an, wie sie implementiert ist und worauf bei der Verwendung zu achten ist. Abschließend fassen wir die wichtigsten Punkte noch einmal zusammen.

Konfiguration

Lambda

Eine Lambda-Funktion hat viele unterschiedliche Konfigurationsparameter *[1]*. Im Rahmen dieses Artikels sind die folgenden drei Parameter von besonderer Bedeutung, da sie einen erheblichen Einfluss darauf haben, wie sich die Lambda-Funktion als Ereignisverarbeitungskomponente verhält:

1. Reservierte gleichzeitige Ausführungen (`ReservedConcurrentExecutions`)
2. Maximale Laufzeit (`Timeout`)
3. Konfiguration der Warteschlange für unzustellbare Nachrichten (`DeadLetterConfig`)

`ReservedConcurrentExecutions` dient der Limitierung der gleichzeitigen Ausführungen. Lambda skaliert automatisch basierend auf den gleichzeitigen Aufrufen, bis zu maximal 1.000 gleichzeitigen Ausführungen pro Region. Setzt man `ReservedConcurrentExecutions` bei einer Funktion auf 10, so werden das regionale Limit auf 990 heruntergesetzt und für die Funktion 10 gleichzeitige Ausführungen reserviert. Diese können dann jedoch auch nicht mehr überschritten werden.

Der zweite wichtige Parameter, die maximale Funktionslaufzeit, wird von AWS herangezogen, um Ausführungen nach einer bestimmten Zeit zu terminieren. Die maximale Laufzeit sollte entsprechend hoch gewählt sein, sodass Funktionsausführungen im normalen Betrieb nicht aufgrund einer Zeitüberschreitung beendet werden.

Der dritte Parameter, das Angeben einer Warteschlange für unzustellbare Nachrichten, erlaubt es dem Anwender, eine SQS-Warteschlange oder ein SNS-Thema anzugeben, in die unzustellbare Nachrichten weitergeleitet werden. Sollte die Lambda-Funktion also nicht in der Lage sein, Nachrichten erfolgreich zu verarbeiten, können diese so beispielsweise für manuelle Nachbearbeitung weitergereicht werden.

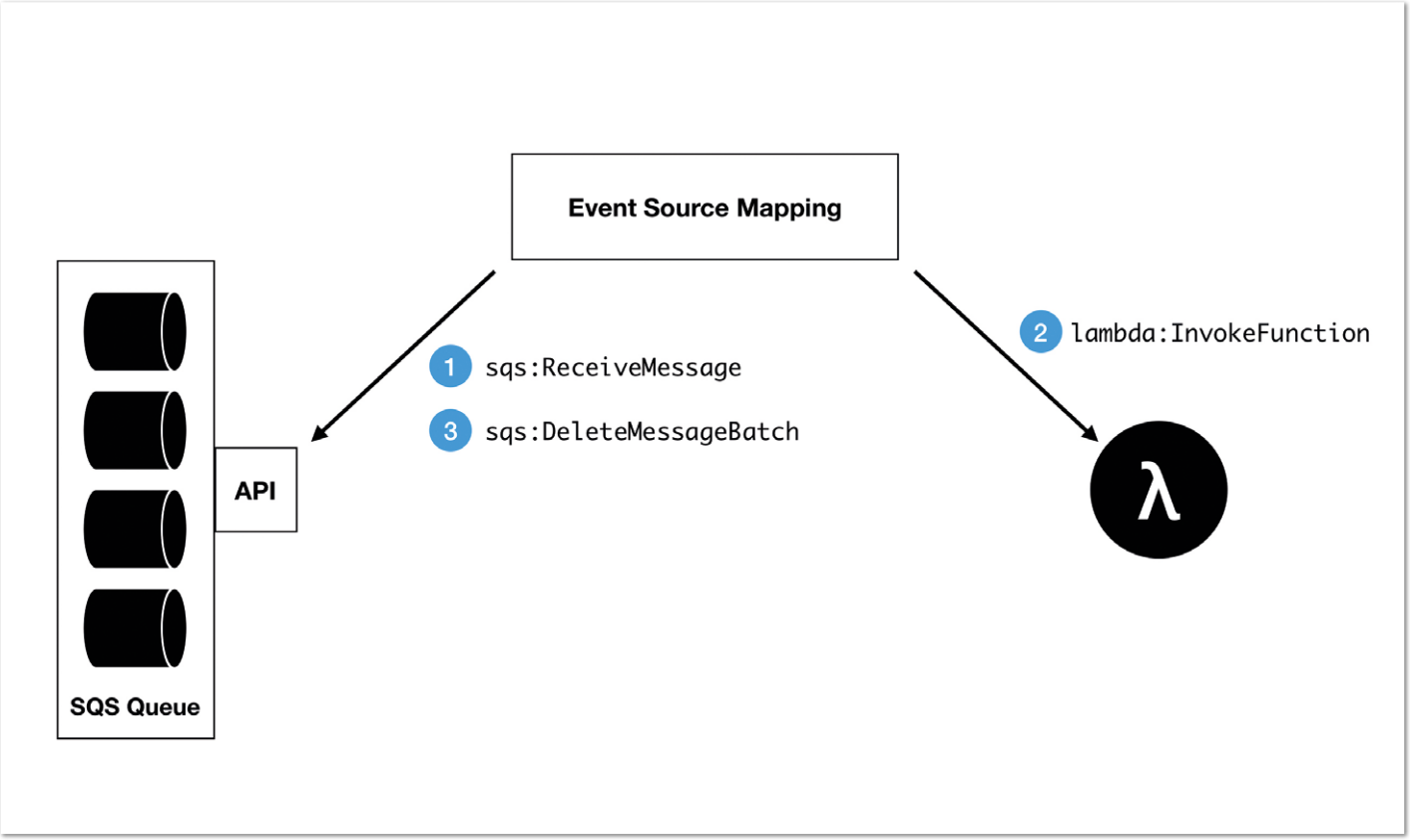


Abbildung 1: Zusammenspiel von SQS, Lambda und der Ereignisquellenzuordnungskomponente im Falle einer erfolgreichen Stapelverarbeitung von Warteschlangen-Nachrichten (Quelle: Frank Rosner)

```
{
    "Records": [
        {
            "messageId": "059f36b4-87a3-44ab-83d2-661975830a7d",
            "receiptHandle": "AQEBwJnKyrHigUMZj6rYigCgxlaS3SLy0a...",
            "body": "Test message.",
            "attributes": {
                "ApproximateReceiveCount": "1",
                "SentTimestamp": "1545082649183",
                "SenderId": "AIDAIENQZJOLO23YVJ4VO",
                "ApproximateFirstReceiveTimestamp": "1545082649185"
            },
            "messageAttributes": {},
            "md5OfBody": "e4e68fb7bd0e697a0ae8f1bb342846b3",
            "eventSource": "aws:sqs",
            "eventSourceARN": "arn:aws:sqs:us-east-2:123456789012:my-queue",
            "awsRegion": "us-east-2"
        },
        {
            "messageId": "2e1424d4-f796-459a-8184-9c92662be6da",
            "receiptHandle": "AQEBzWwaftRI0KuVm4tP+/7q1rGgNqicHq...",
            "body": "Test message.",
            "attributes": {
                "ApproximateReceiveCount": "1",
                "SentTimestamp": "1545082650636",
                "SenderId": "AIDAIENQZJOLO23YVJ4VO",
                "ApproximateFirstReceiveTimestamp": "1545082650649"
            },
            "messageAttributes": {},
            "md5OfBody": "e4e68fb7bd0e697a0ae8f1bb342846b3",
            "eventSource": "aws:sqs",
            "eventSourceARN": "arn:aws:sqs:us-east-2:123456789012:my-queue",
            "awsRegion": "us-east-2"
        }
    ]
}
```

Listing 1 - Beispiel eines SQS-Ereignisses [6], mit dem eine Lambda-Funktion aufgerufen wird. Ein solches Ereignis kann je nach Konfiguration bis zu 10 Nachrichten im Records-Feld beinhalten

SQS

Für diesen Artikel sind die folgenden SQS-Konfigurationsparameter *[2]* von Interesse: Abrufverhalten (engl. polling strategy) und Sichtbarkeitszeitbeschränkung (engl. visibility timeout). Zusätzlich lässt sich eine Warteschlange auch als FIFO (First In – First Out) konfigurieren, sodass gewisse Ordnungsgarantien gegeben werden.

Das Abrufverhalten kann über den Parameter `ReceiveMessageWaitTimeSeconds` gesteuert werden. Setzt man ihn auf 0, sendet man eine Kurzabfrage (engl. short polling). Ein Wert > 0 verschickt eine Langabfrage (engl. long polling).

Wenn ein Client eine Kurzabfrage stellt, antwortet SQS, ohne die Antwort aller Partitionen abzuwarten. Dies kann gegebenenfalls zu einer leeren Antwort führen, obwohl noch Elemente in der Warteschlange sind. Da jedoch die Liste an angefragten Partitionen mit jeder Anfrage verschieden ist, werden mehrere Anfragen letztendlich alle Partitionen einmal getroffen haben und es geht keine Nachricht verloren.

Im Falle einer Langabfrage fragt SQS alle Partitionen an und wartet auf deren Antwort bis zum maximalen Wert von `ReceiveMessageWaitTimeSeconds`. Kurzabfragen erlauben kürzere Verarbeitungslatenzen, kosten jedoch potenziell mehr Geld, da mehrere/häufiger Abfragen gestellt werden müssen. Solange keine Latenzen im Millisekunden-Bereich erforderlich sind, ist es aus Kostengründen zu empfehlen, Langanfragen mit wenigen Sekunden Wartezeit zu stellen.

Der zweite wichtige Parameter erlaubt, die Sichtbarkeitszeit zu beschränken. Da SQS nicht wissen kann, wann ein Konsument ein Element erfolgreich verarbeitet hat, löscht es Nachrichten niemals automatisch. Stattdessen obliegt es dem Konsumenten, erfolgreich konsumierte Nachrichten aus der Warteschlange zu entfernen. Um zu verhindern, dass eine sich in der Verarbeitung befindende Nachricht bei einer nebenläufigen Anfrage an einen anderen Konsumenten erneut ausgeliefert wird, werden abgefragte Nachrichten für eine bestimmte Zeit unsichtbar gemacht.

Der Parametername `VisibilityTimeout` ist hierbei eventuell etwas irreführend, denn er gibt eigentlich an, wie lange eine abgefragte Nachricht unsichtbar bleibt, bevor sie wieder Teil einer Abfrageantwort werden kann, das bedeutet, ab wann sie wieder sichtbar wird. Die Unsichtbarkeitszeitspanne sollte so gewählt sein, dass Konsumenten ausreichend Zeit haben, die erfolgreich verarbeitete Nachricht zu löschen, jedoch Nachrichten im Falle einer fehlgeschlagenen Verarbeitung zeitnah erneut abgefragt werden können.

SQS-Lambda-Ereignisquellenzuordnung

Das Konfigurieren der Ereignisquellenzuordnung *[3]* ist kein Hexenwerk. Ein nennenswerter Konfigurationsparameter ist die Stapelgröße (engl. batch size). Die SQS-Schnittstelle erlaubt es, mit einer Abfrage bis zu zehn Elemente aus der Warteschleife zu laden. Je nach konfigurierter Stapelgröße ruft die SQS-Lambda-Integration die Lambda-Funktion mit entsprechend vielen Elementen auf.

Nun haben wir die Grundlagen für die Konfiguration der einzelnen Bestandteile kennengelernt. Im folgenden Abschnitt schauen wir uns die Integration einmal im Detail an.

Die Lambda-SQS-Integration im Detail

Architektur und Datenfluss

Damit SQS und Lambda erfolgreich zusammenspielen, erzeugt AWS beim Zuordnen einer SQS-Lambda-Ereignisquelle eine Komponente (Event Source Mapping), die die SQS-Schnittstelle (API) abfragt und die Lambda-Funktion aufruft. *Abbildung 1* illustriert das Zusammenspiel der drei Akteure im Falle einer erfolgreichen Nachrichtenverarbeitung.

Zunächst wird im ersten Schritt die Warteschlange abgefragt, indem die `sqs:ReceiveMessage`-Aktion *[4]* mit den in der Ereignisquellenzuordnung spezifizierten Parametern aufgerufen wird. SQS markiert die zurückgegebenen Nachrichten sofort als in Verarbeitung befindlich, sodass sie für Folgeabfragen unsichtbar sind. Dieser Zustand wird entweder durch Ablaufen der Unsichtbarkeitszeitspanne beendet oder indem die Nachrichten endgültig gelöscht werden.

Im zweiten Schritt wird die Lambda-Funktion synchron über die `lambda:InvokeFunction`-Aktion *[5]* aufgerufen, wobei die Nachrichten aus der Warteschlange in ein SQS-Ereignis *[6]* eingebettet werden. *Listing 1* zeigt ein Beispiel eines solchen Ereignisses.

Nachdem der synchrone Aufruf der Lambda-Funktion erfolgreich ist, wird in Schritt 3 die `sqs:DeleteMessageBatch`-Aktion *[7]* aufgerufen, um die verarbeiteten Nachrichten zu löschen. Damit endet der Prozess. Schlägt die Ausführung der Lambda-Funktion aus irgendeinem Grund fehl, wird das Unsichtbarkeitszeitfenster abgewartet, wodurch die Nachrichten erneut zur Verarbeitung freigegeben sind.

Idempotenz ist wichtig

Der im vorigen Abschnitt beschriebene Ablauf ist komfortabel, da AWS das Abfragen der Warteschlange und das Löschen erfolgreich verarbeiteter Nachrichten übernimmt. Es gibt jedoch ein paar Tücken, die es zu beachten gibt. Man betrachte einmal das folgende Szenario:

1. Eine neue Nachricht trifft in der Warteschlange ein
2. AWS ruft die Lambda-Funktion mit der neuen Nachricht auf
3. Die Lambda-Funktion benötigt sehr lange, um die Nachricht zu verarbeiten, weil beispielsweise ein anderer Dienst langsam ist. Die Verarbeitungsdauer überschreitet das Unsichtbarkeitszeitfenster
4. AWS ruft erneut die Lambda-Funktion mit der gleichen Nachricht auf
5. Der erste Lambda-Aufruf ist erfolgreich beendet
6. Der zweite Lambda-Aufruf ist ebenfalls erfolgreich beendet und die Nachricht wurde doppelt verarbeitet

Leider lässt sich dieses Szenario nicht einfach ausschließen, indem man die Lambda-Laufzeitbeschränkung und die Unsichtbarkeitszeitbeschränkung auf denselben Wert setzt, da es eine Verzögerung zwischen dem Abruf der Nachricht aus der Warteschlange und dem Aufruf der Lambda-Funktion gibt. Diese Verzögerung kann größer werden, falls Lambda-Funktionsaufrufe gedrosselt werden oder die Lambda-Schnittstelle gerade nicht verfügbar ist. Eine Drosselung findet beispielsweise statt, wenn das Kontingent gleichzeitiger Ausführungen ausgereizt ist.

Um die Wahrscheinlichkeit einer Mehrfachverarbeitung möglichst gering zu halten, empfiehlt AWS, `VisibilityTimeout` auf das Sechsfache der Funktionslaufzeitbeschränkung zu setzen. Das bedeutet jedoch auch, dass eine Funktion mit maximal fünfminütiger Laufzeit zur Folge hat, dass nicht erfolgreich verarbeitete Nachrichten erst nach einer halben Stunde wieder sichtbar werden.

Eine andere Möglichkeit, mit diesem Problem umzugehen, ist, die Nachrichtenverarbeitung idempotent zu gestalten. Idempotenz bedeutet, dass erneutes Ausführen der Funktion mit den gleichen Parametern keine Zustandsveränderung zur Folge hat. Mehrfachverarbeitungen stellen somit kein Problem dar. Das Versenden einer E-Mail bei jedem Funktionsaufruf wäre nicht idempotent, da die Empfänger so mehrere identische E-Mails erhalten würden.

Idempotenz hat noch einen weiteren Vorteil. Selbst, wenn das obige Szenario nicht eintritt, können Nachrichten mehrfach auftauchen. Man stelle sich beispielsweise vor, dass die Nachricht erfolgreich verarbeitet wurde, die `sqs:DeleteMessageBatch`-Aktion jedoch fehlschlägt, weil die betreffende Partition offline ist. Sobald diese wieder online kommt, kann die Nachricht erneut sichtbar werden.

Es gilt ebenfalls zu beachten, dass immer nur komplette Nachrichtenstapel gelöscht werden können. Sollte also in einem SQS-Event mit mehreren Nachrichten eine davon zu einem Fehler in der Lambda-Funktion führen, so werden alle Nachrichten wieder sichtbar, da AWS nicht wissen kann, welche Nachrichten aus dem Stapel erfolgreich verarbeitet wurden. Auch hier kann Idempotenz helfen, Probleme mit der Mehrfachverarbeitung einzelner Nachrichten zu vermeiden, sollte eine Stapelgröße > 1 gewählt sein.

Skalierung und Drosselungsverhalten

SQS unterstützt praktisch unbegrenzten Durchsatz. Lambda skaliert automatisch bis zu maximal 1.000 gleichzeitigen Ausführungen pro Region für Standardkonten. Die Ereignisquellenzuordnungskomponente kann jedoch weder das SQS- noch das Lambdaskalierungsverhalten beeinflussen. Sie beginnt zunächst mit fünf Threads, die unabhängig voneinander die Lambda-Schnittstelle mit neuen Nachrichten aufrufen. Sobald Lambda an die Grenze der maximal gleichzeitigen Ausführungen stößt, werden Anfragen gedrosselt.

Um dieses Verhalten zu illustrieren, haben wir ein kleines Experiment durchgeführt. Hierfür legten wir 100 Nachrichten in eine Warteschlange, wobei jede Nachricht 20 Sekunden benötigt, um verarbeitet zu werden. Die Lambda-Funktion haben wir auf 5 gleichzeitige Ausführungen begrenzt. Bis alle Nachrichten verarbeitet wurden, wurden ungefähr 200 Drosselungen der Lambda-Schnittstelle aufgezeichnet.

Der Grund für dieses Verhalten liegt in der simplen Architektur der Integration. Die Ereignisquellenzuordnungskomponente hat keinerlei internes Wissen über SQS oder Lambda. Das Zusammenspiel funktioniert über eine Kombination von öffentlichen Schnittstellen, Zeitüberschreitungen und wiederholten Aufrufen.

Zusammenfassung

In diesem Artikel haben wir gesehen, wie man mithilfe einer Ereignisquellenzuordnung SQS-Ereignisse in einer Lambda-Funktion verarbeiten kann. Wichtige Konfigurationsparameter sind:

- die SQS-Konfiguration zum Abrufverhalten und zur Unsichtbarkeitszeitspanne
- die Lambda-Konfiguration der Warteschlange für unzustellbare Nachrichten, die reservierten gleichzeitigen Ausführungen sowie der maximalen Laufzeit
- die Ereignisquellenzuordnungskonfiguration der maximalen Stapelgröße

Es gilt zu beachten, dass SQS selbst bei optimaler Konfiguration Mehrfachzustellungen von Nachrichten nicht ausschließen kann. Wir als Entwickler/innen sind in der Verantwortung, unsere Lambda-Funktionen idempotent zu entwerfen.

Referenzen

[1] Dokumentation der AWS::Lambda::Function-Ressource, *https://docs.aws.amazon.com/de_de/AWSCloudFormation/latest/UserGuide/aws-resource-lambda-function.html*

[2] Dokumentation der AWS::SQS::Queue-Ressource, *https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-properties-sqs-queues.html*

[3] Dokumentation der AWS::Lambda::EventSourceMapping-Ressource, *https://docs.aws.amazon.com/en_us/AWSCloudFormation/latest/UserGuide/aws-resource-lambda-eventsourcemapping.html*

[4] Dokumentation der SQS-ReceiveMessage-Aktion, *https://docs.aws.amazon.com/AWSSimpleQueueService/latest/APIReference/API_ReceiveMessage.html*

[5] Dokumentation der Lambda-Invoke-Aktion, *https://docs.aws.amazon.com/de_de/lambda/latest/dg/API_Invoke.html*

[6] Verwenden von AWS Lambda mit Amazon SQS, *https://docs.aws.amazon.com/lambda/latest/dg/with-sqs.html*

[7] Dokumentation der SQS-DeleteMessageBatch-Aktion *https://docs.aws.amazon.com/AWSSimpleQueueService/latest/APIReference/API_DeleteMessageBatch.html*

Frank Rosner
Datastax
frank.rosner@datastax.com

Franks Interessen liegen im Bereich von Big Data, Machine Learning, Cloud-Native-Anwendungen und Softwareentwicklung. Er liest gerne Paper und Quelltexte, um zu verstehen, wie die Dinge funktionieren, die er verwendet. Auf der JVM entwickelt er in Java, Kotlin und Scala.

Java-Low-Code-Plattform

René Jahn, SIB Visions GmbH

Die Begriffe Low-Code und No-Code sind zwei neue Sterne am Himmel. Im Jahre 2014 wurde Low-Code von Forrester Research benannt und erst in den letzten Monaten wurde richtig Fahrt aufgenommen. Fast wöchentlich tauchen neue Tools auf. Eine Empfehlung auszusprechen, ist nahezu unmöglich. Auch im Java-Umfeld gibt es viele Tools, die den Anforderungen von Low- und No-Code gerecht werden. Eines dieser Tools wird in diesem Artikel vorgestellt: VisionX [1].

Es gibt ausführliche Definitionen für Low-Code [2] und No-Code [3]. Der Autor versteht darunter Folgendes:

Eine Person, die nicht programmieren kann, soll mit einem No-Code-Tool eine lauffähige Applikation erstellen können, die den eigenen Anforderungen entspricht. Bei Low-Code-Tools sind die Applikationen mit eigenem Code erweiterbar. Dazu sollte der Anwender jedoch ein paar wenige Programmierkenntnisse mitbringen.

Als Applikation zählt sowohl eine mobile App für das Scannen eines Barcodes, als auch eine komplexe Verwaltungsanwendung, die im Browser läuft. Weder von Low-Code noch von No-Code wird vorgegeben, welche Art von Applikation erstellt werden kann. Das ist einzig und allein abhängig vom eingesetzten Tool.

Aus Sicht der Software-Entwickler/innen wären sowohl Low-Code als auch No-Code unnötig, denn genau das ist *ihre* Aufgabe. Doch das sehen die Fachabteilungen und IT-Entscheider ganz anders. Sie sind bemüht, Prozesse und Anforderungen so schnell wie möglich in Software abzubilden, um der Konkurrenz einen Schritt voraus zu sein. Nach Möglichkeit auch so günstig wie möglich.

Das ist aber häufig gar nicht realisierbar, weil die Entwicklungsabteilung keine oder zu wenige Ressourcen frei hat oder bereits mit anderen Aufgaben beschäftigt ist. Es ist auch nicht einfach möglich, das Entwicklungsteam aufzustocken. Zum einen ist es schwierig, eine geeignete Person zu finden, und zum anderen bedarf es einer gewissen Einarbeitungszeit. Ganz abgesehen von den Zusatzkosten.

Diese Probleme sollen mit Low-Code- und No-Code-Tools gelöst werden. Warum denn eigentlich nicht?

Aus Sicht des Autors spricht auch nichts dagegen, bestimmte Aufgaben von Laien oder technisch versierten Personen umsetzen zu lassen. Mit Excel funktioniert das bisher doch auch ganz gut und die Anwender erstellen komplexe Formeln und Eingabemöglichkeiten relativ schnell und problemlos. Erst wenn es um verstrickte Abläufe oder die Anbindung von Fremdsystemen geht, könnte es schnell vorbei sein mit Low-Code. Doch auch das ist zum Teil lösbar, wenn vorgefertigte Bausteine oder Vorlagen vom Tool angeboten werden.

Wenn der Laie wirklich nicht mehr weiterkommt und Hilfe benötigt, dann sollten Softwareentwickler/innen einspringen. Doch hier beginnt es wieder knifflig zu werden, denn ein Laie hat keine technischen Ansprüche. Die Lösung muss lediglich das tun, was gefordert wird, egal mit welcher Technologie oder mit welchem Framework das umgesetzt ist.

Ein/e Softwareentwickler/in hat hingegen konkrete Vorstellungen und arbeitet lieber mit der gewohnten IDE und bekannten Frameworks. Wenn das nicht möglich ist, kommt es häufig zu Konflikten und einer Abwehrhaltung gegenüber den Tools.

Um den/die Softwareentwickler/in anzusprechen, braucht es also ein Low-Code-Tool, das sich mit einer oder allen IDEs im Einklang befindet. Es sollte keine Probleme mit beliebigen Frameworks haben und auch einfach zu bedienen sein. Der Laie soll nicht abgeschreckt werden und die Anforderungen auch ohne Programmierkenntnisse lösen können. Doch auch im Idealfall braucht es eine gewisse Offenheit für Neues, denn Low-Code-Tools vereinfachen die Arbeitsweise und bringen bereits Lösungen für Alltagsprobleme mit. Diese können sich von den Lösungsansätzen der Entwickler/innen unterscheiden. Das bedeutet eine Anpassung der Arbeitsweise.

VisionX

Ein Tool, das den Ansprüchen von Laien und Entwickler/innen gerecht wird, ist VisionX. Dabei handelt es sich um ein mit Java entwickeltes Low-Code- und auch No-Code-Tool. Es verwendet ausschließlich Open-Source-Frameworks und setzt auf etablierte Standards. Die damit erstellten Applikationen verwenden ebenfalls ausschließlich Open-Source-Frameworks und können, sowohl im Tool selbst als auch mit beliebigen Java IDEs (Eclipse, IntelliJ, NetBeans), weiterentwickelt werden. Es funktioniert auch beides gleichzeitig. Als Entwickler/in wäre es somit möglich, eine Eingabemaske zusammenzuklicken, anschließend die Logik in der IDE umzusetzen und zugleich im Tool zu testen. Der Laie kann dann direkt damit arbeiten und gegebenenfalls das Layout an seine Wünsche anpassen und beispielsweise um Berichte erweitern.

Durch die Kombination von Low-Code und IDE können sowohl Software-Entwickler/in als auch Laie an einer Applikation arbeiten, ohne auf die jeweiligen Vorteile verzichten zu müssen. Da Java als Programmiersprache zum Einsatz kommt, gibt es keinerlei Einschränkungen hinsichtlich Frameworks. Die fertige Lösung steht also im Vordergrund und nicht die Technologie.

Was macht VisionX aber nun so besonders? Das ist eine berechtigte Frage, denn schließlich gibt es unzählige Plattformen und Low-Code-Tools.

Wie bereits erwähnt, setzt VisionX auf Open-Source-Frameworks und bietet die Möglichkeit, beliebige Java IDEs und Frameworks einzusetzen. Dadurch ist auch eine Unabhängigkeit zum Hersteller gegeben, denn die Entwicklung der Applikation ist auch ohne VisionX möglich. Mit VisionX können sowohl (native) mobile als auch Browser-basierte und selbst klassische Desktop-Applikationen erstellt werden. Jede Applikation läuft auf allen Plattformen. Es ist nicht nötig, eine Applikation für mobile Geräte und eine eigene Applikation für Browser zu erstellen. Es handelt sich immer um ein und dieselbe Applikation. Durch den Einsatz des Java-Applikations-Frameworks JVx [4] wird dies ermöglicht.

Mit VisionX werden datengetriebene Verwaltungsapplikationen erstellt, die immer auf relationale Datenbanken zugreifen. Von VisionX werden gängige Anbieter wie Oracle, MySQL, MSSQL, PostgreSQL, DB2 und mehr unterstützt. Auch darin liegt ein großer Vorteil, denn die Datenbank ist frei wählbar. Es entsteht auch hier keine Abhängigkeit zum Tool. Doch noch viel wichtiger ist, dass die Installation von Applikationen auf beliebigen Java-Applikationsservern möglich ist und keine Kosten für den Betrieb anfallen. Man zahlt für die Verwendung des Tools und nicht für die Anzahl der Benutzer oder verwendete Ressourcen.

Neben der Flexibilität und Unabhängigkeit gibt es auch auf der Feature-Seite einige relevante Punkte, die ein Alleinstellungsmerkmal darstellen können. So ist beispielsweise ein Applikationsrahmen vorhanden, der eine Authentifizierung bietet, ein Benutzer- und Rollen-

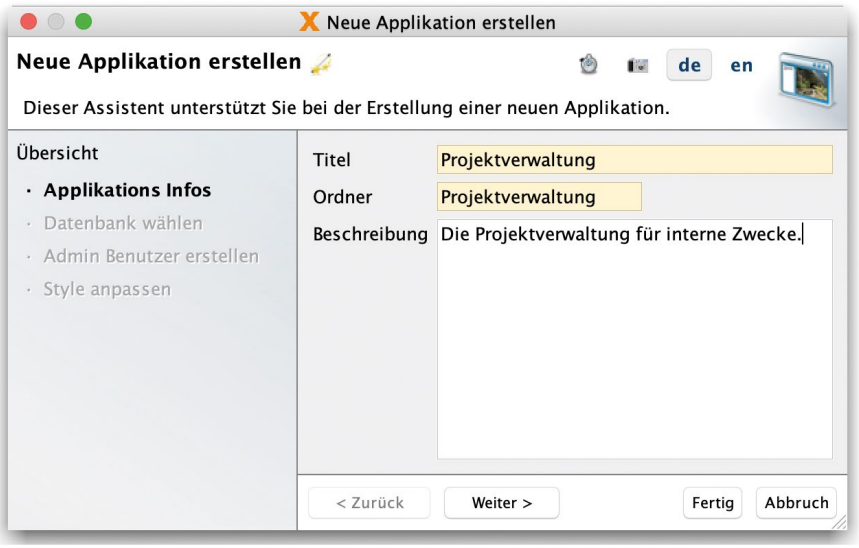


Abbildung 1: Neue Applikation erstellen (Quelle: René Jahn)

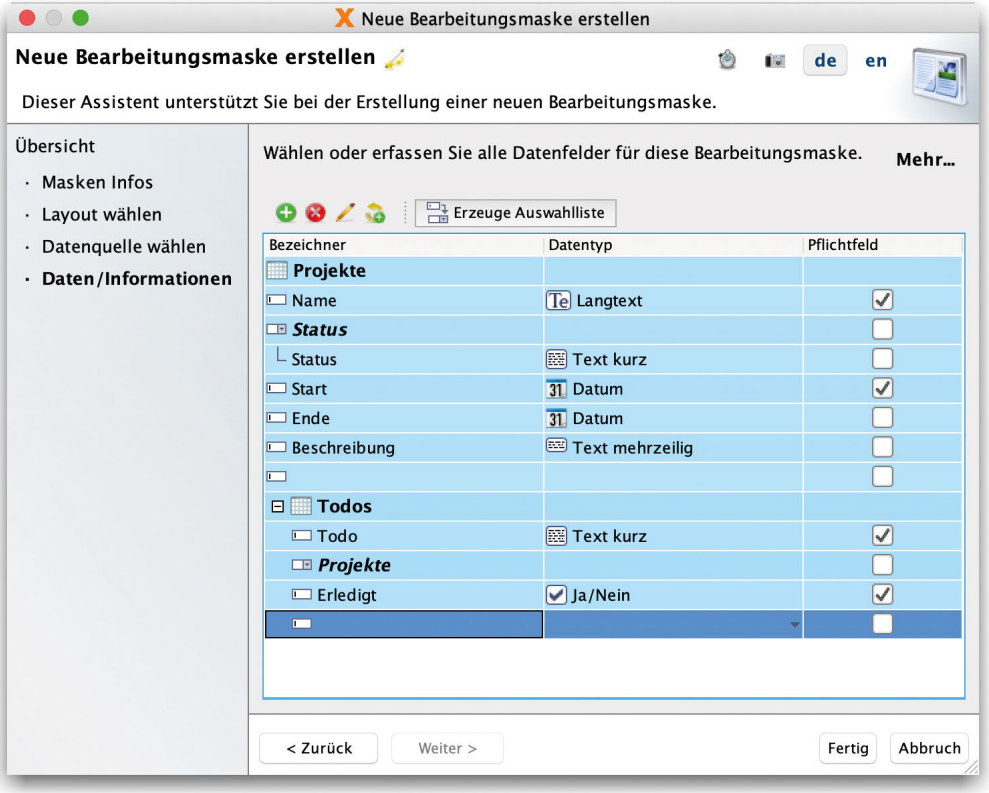


Abbildung 2: Neue Bearbeitungsmaske erstellen (Quelle: René Jahn)

konzept vollständig umsetzt und Mehrsprachigkeit ermöglicht. Der Rahmen ist komplett responsive und passt sich automatisch an die Ausgabegröße an. Ein richtig praktisches Feature ist die Live-Voransicht einer Applikation. Ob im Browser, mobil oder am Desktop. Das spart wertvolle Zeit und der Anwender sieht sofort das Endergebnis. Auf die Live-Preview auf mobilen Geräten werden wir etwas später noch zurückkommen.

Um einen Eindruck von der Arbeit mit VisionX zu bekommen, werden wir eine triviale Projekteverwaltung erstellen. Dazu wird lediglich eine einzige Bearbeitungsmaske benötigt, in der eine Liste von Projekten verwaltet werden kann. Jedes Projekt hat einen Namen, eine Beschreibung, einen Status, Start-/End-Datum und eine To-do-Liste mit To-do-Name und Erledigt-Option.

Zu Beginn erstellen wir eine neue Applikation mit dem Namen „Projektverwaltung", wie in *Abbildung 1* zu sehen ist. Die Vergabe eines Titels ist vollkommen ausreichend.

Nachdem die Applikation erstellt wurde, erscheint ein Assistent, mit dem eine neue Bearbeitungsmaske erstellt werden kann *(siehe Abbildung 2)*.

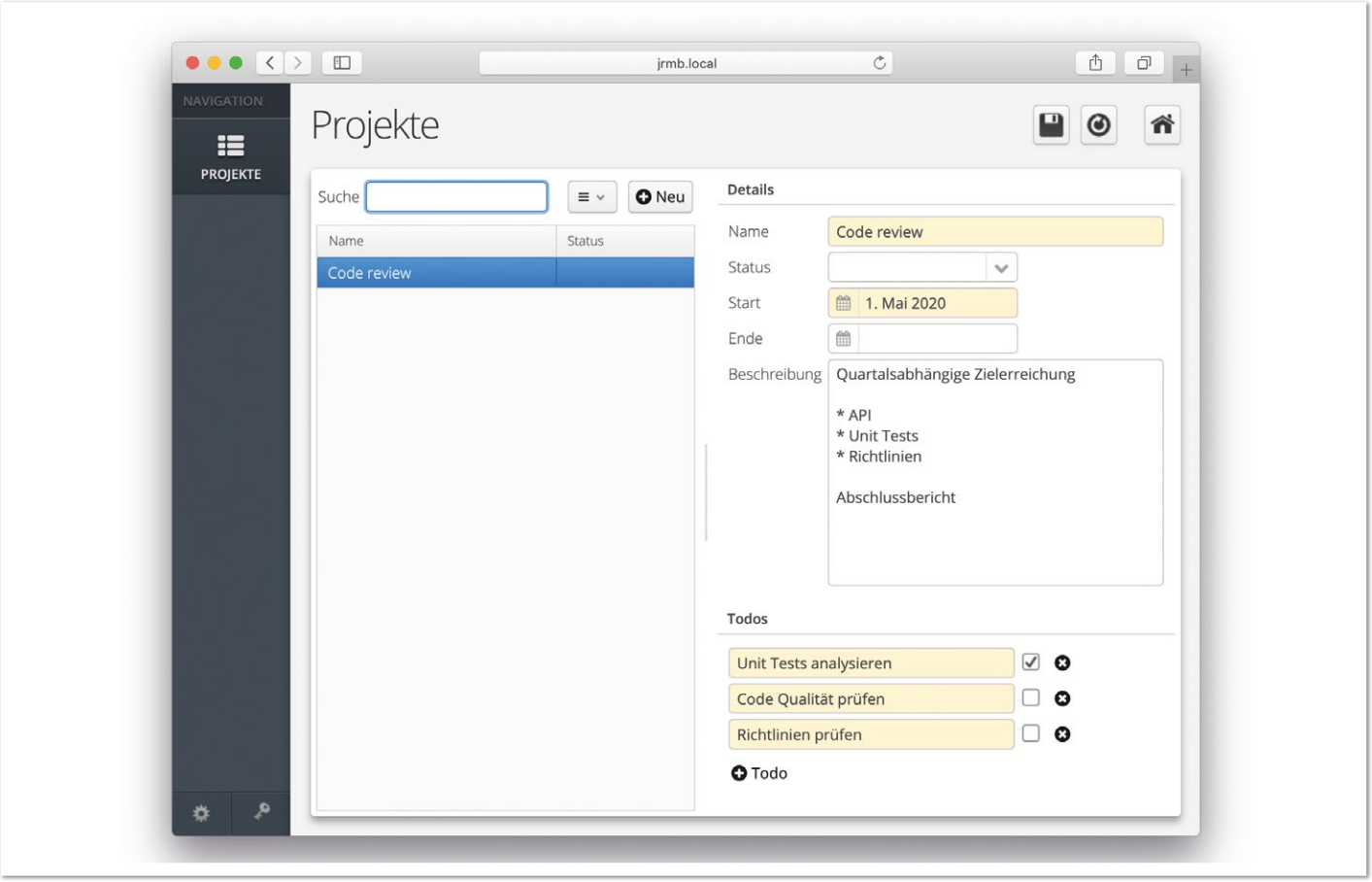


Abbildung 3: Live-Voransicht (Quelle: René Jahn)

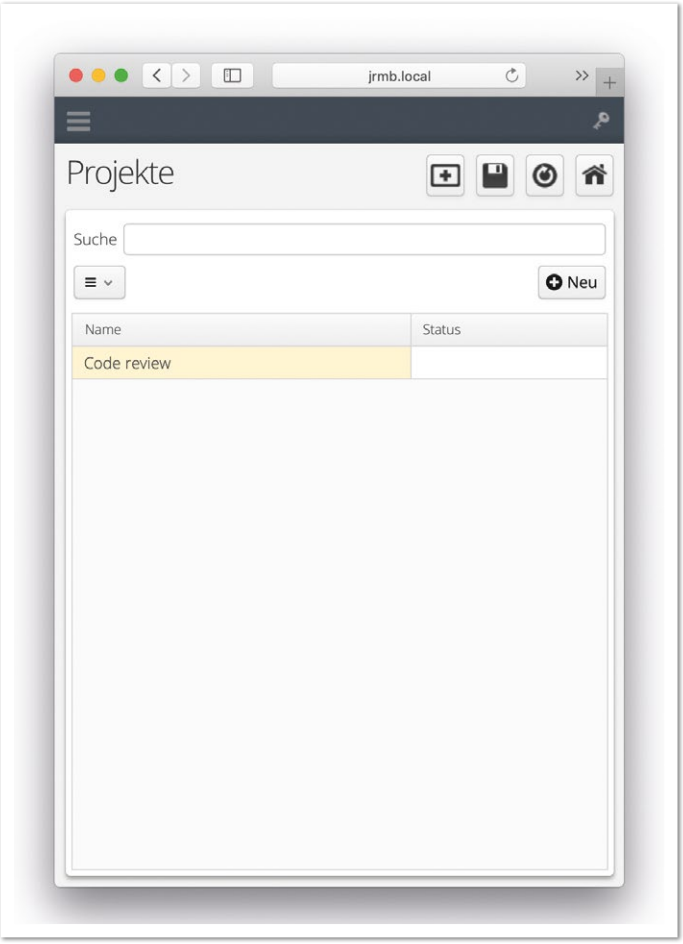


Abbildung 4: Minimale Anzeige, Projektliste (Quelle: René Jahn)

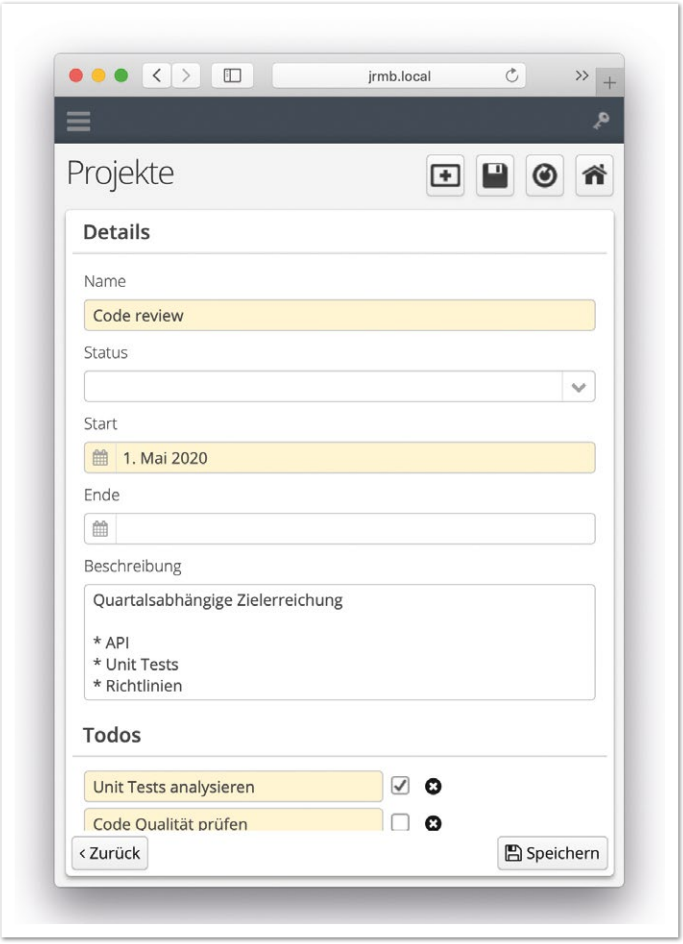


Abbildung 5: Minimale Anzeige, Detail (Quelle: René Jahn)

In diesem Assistenten werden ganz einfach die benötigten Felder eingegeben. Das Status-Feld wird als Auswahlliste und die To-dos werden als Sub-Tabelle definiert. Gleich im Anschluss wird die Maske erstellt; in der Voransicht sieht diese dann wie in *Abbildung 3* gezeigt aus.

In knapp einer Minute wurde in wenigen Schritten eine vollständige Applikation mit Authentifizierung und Bearbeitungsmaske erstellt. Wenn das Ausgabegerät etwas kleiner wäre, beispielsweise ein Smartphone, dann würde die Applikation wie in *Abbildung 4* angezeigt werden. Bei Auswahl eines Datensatzes wird dann das Detail angezeigt, siehe *Abbildung 5*.

Eigener Code

Für Entwickler/innen wäre die Erstellung dieser Applikation inklusive Bearbeitungsmaske ohne einen Code-Generator – in dieser Zeit – nicht möglich gewesen. Auch nicht mit Copy & Paste von vorhandenem Code. Hier spielt ein Low-Code-Tool natürlich seine ganze Stärke aus.

Doch wie ist das nun mit eigenem Code? Die Maske enthält aktuell keinerlei Logik. Nehmen wir mal an, dass es im Meeting-Raum unseres Büros einen großen Monitor gibt, der die aktuellen Projekte des Unternehmens, grafisch aufbereitet, anzeigt. Dieser Monitor soll bei Button-Klick in unserer Maske all unsere Projekte anzeigen. Die Schnittstelle zum Monitor ist nicht standardisiert und somit wird eine spezielle Anbindung benötigt.

Diese Aufgabe kann nur ganz schwer von einem Laien gelöst werden. Er kann aber einen Button integrieren und die Aufgabe von dem/der Softwareentwickler/in lösen lassen. Die Bearbeitungsmaske wird also um einen Button erweitert, wie in *Abbildung 6* unterhalb der Projektliste zu sehen ist.

Der/die Softwareentwickler/in bevorzugt eine IDE, deshalb wird das Projekt ganz einfach importiert. In unserem Fall kommt Eclipse zum Einsatz *(siehe Abbildung 7)*.

Für die IDE ist das Projekt bereits vorkonfiguriert; einzig und allein die Funktion beim Ausführen des Buttons *(siehe Listing 1)* muss implementiert werden.

Als Entwickler/in kann man nun seiner Fantasie freien Lauf lassen. Jeglicher Code, der eingefügt wird, kann von VisionX ausgelesen werden. Der Laie sieht dann lediglich, dass benutzerdefinierter Code verwendet wurde *(siehe Abbildung 8)*. Es kann aber ohne Einschränkungen gearbeitet werden. Es ist auch möglich, strukturelle Änderungen oder Namensänderungen am Code durchzuführen. Es gibt keine speziellen Bereiche, die nicht geändert werden dürfen. Von VisionX wird immer der Sourcecode gelesen und interpretiert.

Somit ist die Applikation auch schon fertig. Durch die Zusammenarbeit von Laie und Entwickler/in kann ein perfektes Ergebnis erzielt werden – ohne technologische Einschränkungen. Die aufgewendete Zeit ist mit manueller Codierung nicht vergleichbar und um ein Vielfaches effizienter. Der Testaufwand kann ebenfalls auf ein Minimum reduziert werden, denn die Basisfunktionalität wird bereits durch die Plattform gewährleistet. Im konkreten Beispiel wäre nur die Funktion zum Übertragen der Daten an den Monitor zu testen.

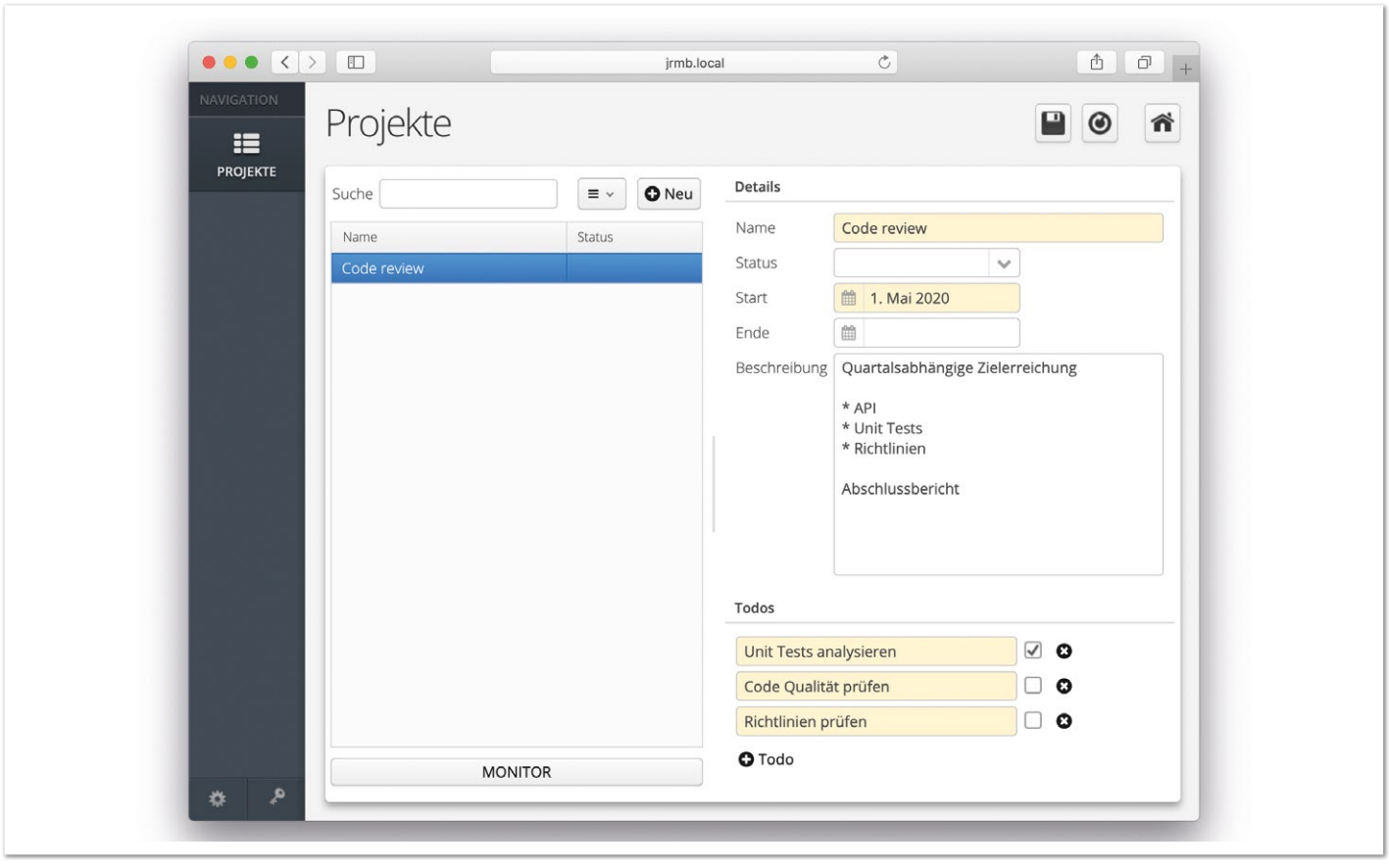


Abbildung 6: Button (Quelle: René Jahn)

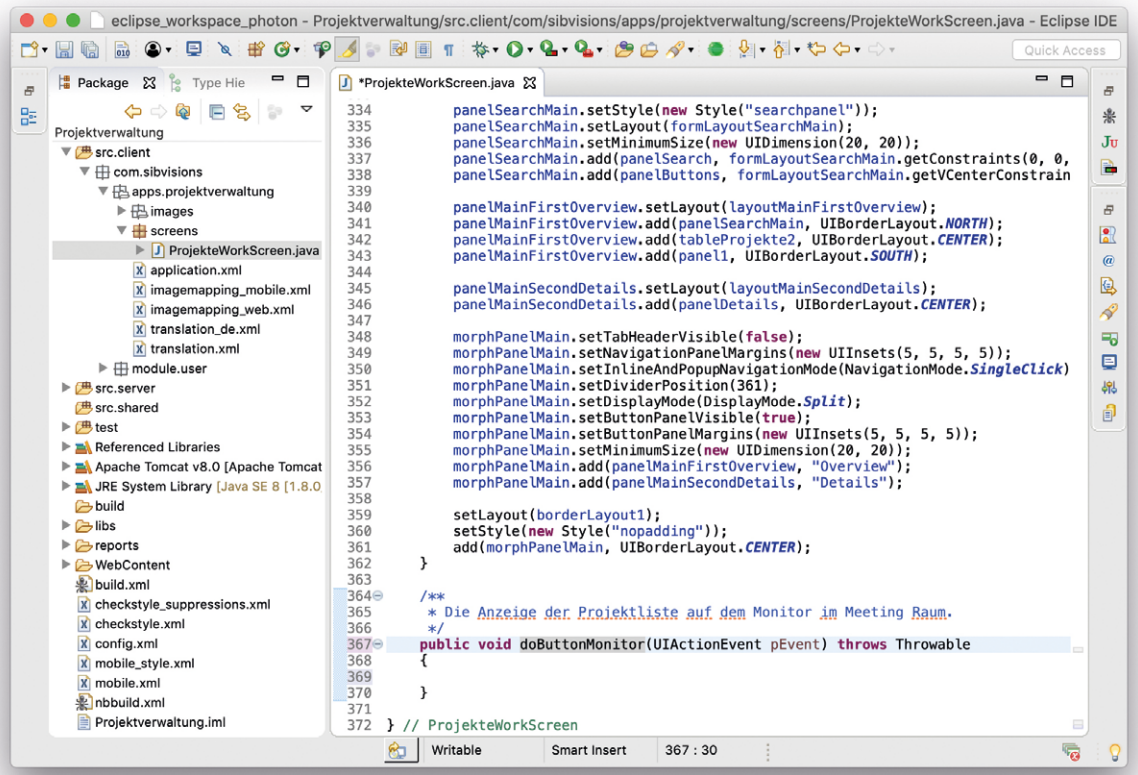


Abbildung 7: Projektverwaltung in Eclipse (Quelle: René Jahn)

```
/**
 * Die Anzeige der Projektliste auf dem Monitor im Meeting Raum.
 */
public void doButtonMonitor(UIActionEvent pEvent) throws Throwable
{
    System.out.println("Mein Code");
}
```

Listing 1: Button Action

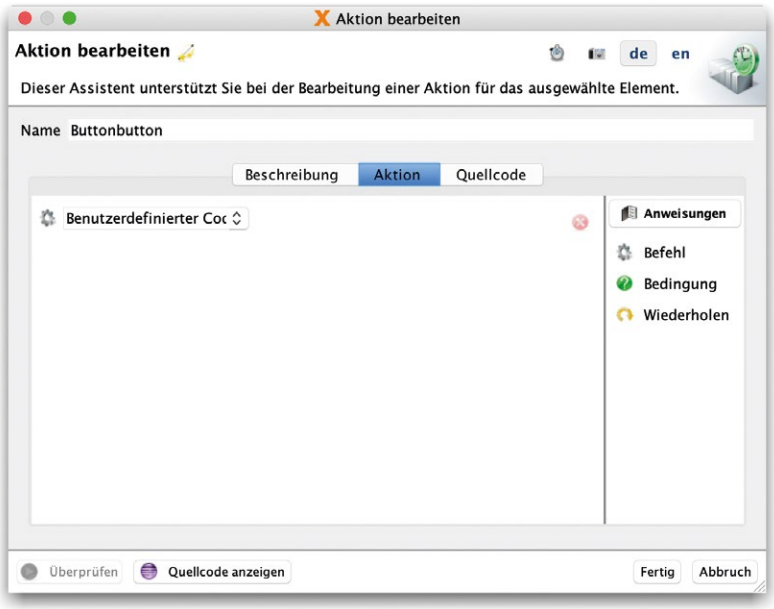


Abbildung 8: Benutzerdefinierter Code (Quelle: René Jahn)

Mobile Live Preview

Wie bereits erwähnt, kann eine Applikation auch als (native) mobile Applikation gestartet werden. Das kann mit einem echten Smartphone erfolgen, wie in *Abbildung 9* zu sehen ist.

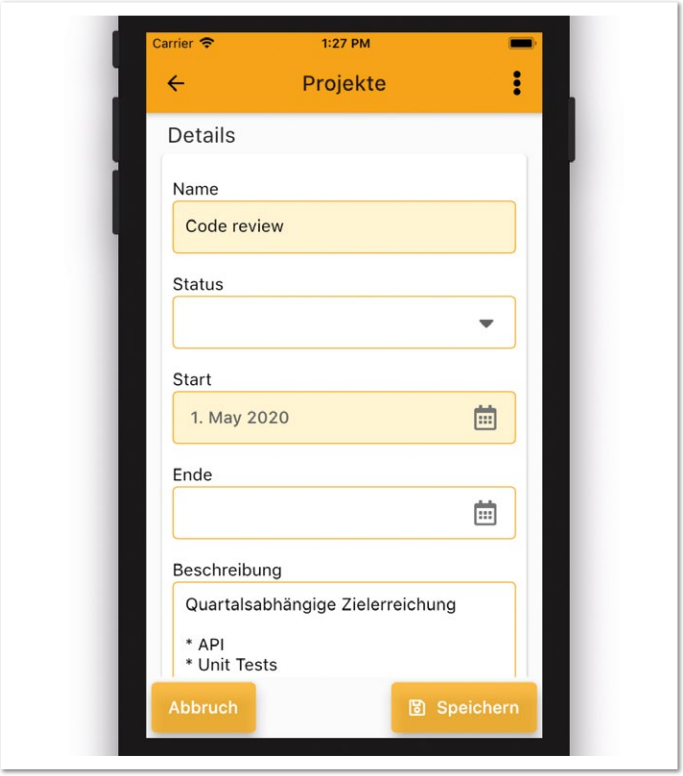


Abbildung 9: Live-Voransicht, Smartphone (Quelle: René Jahn)

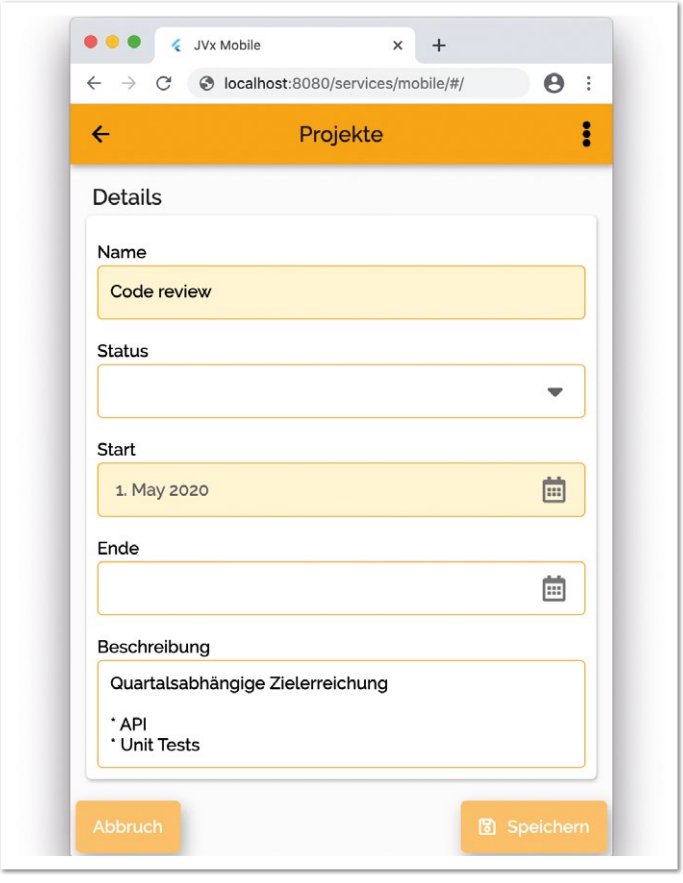


Abbildung 10: Live-Voransicht, Browser (Quelle: René Jahn)

Wer den Artikel „Es fluttert gewaltig" in der Java aktuell 3/20 *[5]* gelesen hat, wird feststellen, dass hier eine Flutter-Applikation *[6]* zum Einsatz kommt. Für diejenigen, die den Artikel nicht gelesen haben, sei kurz erwähnt, dass die in *Abbildung 9* dargestellte Applikation keine Java-Anwendung ist, sondern eine Flutter-Applikation, die direkt am mobilen Gerät läuft. In Ausgabe 3/20 wurde darauf hingewiesen, dass für Flutter auch die Möglichkeit besteht, eine HTML5-Applikation zu erstellen, mit derselben Source-Basis. Doch zum damaligen Zeitpunkt war der Entwicklungsstand noch nicht weit genug.

Darum möchte der Autor in dieser Ausgabe die HTML5-Applikation *(siehe Abbildung 10)* zeigen. Diese wird in VisionX für die Live-Preview verwendet, falls kein Smartphone zur Verfügung steht, oder einfach nur, weil es bequemer ist. Die Darstellung unterscheidet sich im Browser nicht von der Darstellung am Smartphone. Die Lösung ist einzigartig und kommt nur in VisionX zum Einsatz.

Fazit

Der Einsatz von Low-Code-Plattformen wäre für viele Bereiche der Softwareentwicklung ein enormer Boost. Wenn Fachabteilung und Entwicklung Hand in Hand arbeiten, dann entstehen Lösungen, die genauso sind, wie sie in der Praxis benötigt werden. Die Entwicklungszeit reduziert sich und Anpassungen können zeitnah umgesetzt werde, auch ohne Entwickler/in. Aber es ist auch Vorsicht geboten, denn wenn Entwickler/innen nicht mit den Fachabteilungen zusammenarbeiten, entstehen Insellösungen, die aus der Reihe tanzen. Auf die Offenheit von Low-Code-Plattformen sollte geachtet werden, denn Unabhängigkeit ist ein wichtiger Faktor für die Zukunft.

Quellen

[1] *https://visionx.sibvisions.com/*
[2] *https://de.wikipedia.org/wiki/Low-Code-Plattform*
[3] *https://en.wikipedia.org/wiki/No-code_development_platform*
[4] *https://de.wikipedia.org/wiki/JVx*
[5] *https://mydoag.doag.org/formes/pubfiles/12236053/docs/Publikationen/Java-Aktuell/2020/03-2020/03-2020-Java_aktuell-Magazin-WEB[1].pdf*
[6] *https://flutter.dev/*

René Jahn
SIB Visions GmbH
rene.jahn@sibvisions.com

René Jahn ist Mitbegründer der SIB Visions GmbH und Head of Research & Development. Er verfügt über langjährige Erfahrung im Bereich der Framework- und API-Entwicklung. Neben der Open-Source-Sparte bringt er seine Expertise auch in der Produktentwicklung ein. Seine Leidenschaft ist die Evaluierung neuer Technologien und Integration in klassische Business-Applikationen.

Agile Softwareentwicklung systemisch gedacht

Christian Mennerich, synyx GmbH & Co. KG

Software wird heute agil entwickelt, der Wert agiler Methoden ist auch weitgehend anerkannt. Ebenso ist akzeptiert, dass die Geschäftsprozesse im Zentrum der Wertschöpfung stehen sollen, Domain-driven Design ist immer häufiger (wieder) das Mittel der Wahl. Dazu kommen technische Entwicklungen wie nachrichtenbasierte und stark serviceorientierte Softwarearchitekturen. Doch welche Bedeutung haben diese Entwicklungen für die Gemeinschaft der Nutzer? Wir wollen einen Vorschlag wagen, eine soziologische Theorie zu bemühen, um bei der Suche nach (Sub-) Domänen und (abgeschlossenen) Kontexten etwas mehr Verständnis zu schaffen.

Softwareentwicklung heute

Das zurzeit vorherrschende Modell für moderne Softwareentwicklung bedient sich agiler Methoden. Diese kommen in verschiedenen Ausprägungen daher, wie etwa Scrum, Kanban oder Extreme Programming. Iteratives Entwicklungsvorgehen, kleine Inkremente und crossfunktionale Teamaufstellungen verstehen sich meist auch von selbst, werden oft aber unterschiedlich verstanden. Häufig wird nach Prinzipien des Domain-driven Designs (DDD) entwickelt, wie es Evans *[1]* und Vernon *[2]* geprägt haben. Dies soll gewährleisten, dass Entwicklungsteam und Nutzer des Softwareproduktes durch die Businessdomäne zusammengeschweißt werden.

Darum herum sind viele richtungsweisende Techniken entstanden, wie etwa Event Storming *[3]*, Teamzusammensetzungen nach Ideen der Team Topologies *[4]* und im weitesten Sinne auch die Ideen der DevOps-Kultur *[5]*. Wissenschaftliche Studien unterstützen die agilen Ansätze auf verschiedenen Ebenen (vgl. zum Beispiel *[6]*).

Ein Kernpunkt im DDD ist das Finden der sogenannten abgeschlossenen Kontexte (bounded contexts), der abgeschlossenen Einheiten im Lösungsraum, in denen eine universelle Sprache über ein gemeinsames Modell Geschäft und Software eint. Auch auf der technischen Seite ist in den letzten Jahren viel passiert. Von nachrichtengetriebenen, asynchronen Architekturen, von Services und Microservices-Architekturen sowie von Serverless- und Lambda-Architekturen ist die Rede. Software wird oft modern in der Cloud betrieben, nicht nur (horizontale) Skalierung und hohe Service Level Agreements (SLAs) sind hier treibende Faktoren.

Auch ein Verschieben der Betriebsverantwortung in die Teams zur Entlastung der Betriebsmannschaft ist ein Anreiz. Dies verändert das Denken über und das Arbeiten mit Software. Nicht zuletzt kehrt mit NoSQL und polyglotten Persistenz-Ansätzen die Verantwortung für die Datenhaltung in die Anwendungen zurück *[7]*. Mit diesen Entwicklungsprinzipien und getrieben durch den DevOps-Gedanken „You build it, you run it!" rücken Entwicklung und Betrieb von Software weiter zusammen. Dem DDD-Gedanken folgend schließt dies auch Requirements Engineering und Systemdesign mit ein.

Soziologie und Softwareentwicklung

Was aber bedeutet das alles für die Gesellschaft und die Gemeinschaft der Nutzer sowie die (Weiter-)Entwicklung des Softwareprodukts? Unbestreitbar ist die Wirkung zwischen Gesellschaft und Softwareprodukt wechselseitig: Die Strukturen – ob Organigramm, Teamzusammenstellung oder Nutzergemeinschaft – beeinflussen das Softwareprodukt. Dieses wiederum aber, wenn es genutzt wird, verändert die Gesellschaft, zumindest die der Nutzer. Bestenfalls sind beide Welten – zumindest zeitweise – harmonisch im Einklang: Die Nutzer sind zufrieden.

Soziologische Aspekte und Erkenntnisse finden in der modernen Softwareentwicklung, unserem Empfinden nach, noch nicht die Aufmerksamkeit, die ihnen gebühren. Das Zurückfließen anerkannter und auch neuerer soziologischer, psychologischer und neurologischer Ergebnisse der wissenschaftlichen Forschung scheint gerade zu beginnen (vgl. auch *[10]*). Von einem Ansatz wollen wir uns hier inspirieren lassen, nämlich dem der Systemtheorie nach Niklas Luhmann *[11]*. Mit diesem wollen wir uns der Aufgabe nähern, abgeschlossene Kontexte im DDD zu finden. Die Luhmann'sche Systemtheorie ist in ihrem Kern zwar sehr abstrakt, bei näherem Hinsehen entpuppt sie sich aber als ausgesprochen verträglich mit dem DDD und den Techniken, die darum entstanden sind.

Für die nachfolgende Diskussion setzen wir einen Softwareentwicklungsprozess nach den oben diskutierten agilen Prinzipien voraus. Wir nähern uns dann den Fragen, wie abgeschlossene Kontexte im DDD gefunden werden können und was diese im Kontext der Organisation, die die Software einsetzt, bedeuten können. Und vor allem, wie ihre Wechselwirkungen verstanden werden können.

Die Systemtheorie nach Niklas Luhmann

In der Systemtheorie nach Niklas Luhmann werden verschiedene sogenannte Systeme unterschieden: die organischen, die psychischen und die sozialen Systeme. Organische und psychische Systeme sind für die Existenz des Menschen zwar unabdingbar, können von Soziologen allerdings nicht untersucht werden. Luhmanns Interesse gilt den sozialen Systemen, diese sind in Interaktion, Organisation und Gesellschaft unterteilt. Grundlegend ist hier die Kommunikation, der wechselseitige sinnhafte Bezug aufeinander. Wir konzentrieren uns hier ausschließlich auf eine einfache Sicht auf die sozialen Systeme.

Ein soziales System kennt ausschließlich Kommunikation. Kommunikation ist das einzige Element, das kommuniziert, weitere Operationen gibt es nicht. Die Idee ist ebenso einfach wie weitreichend: Auf Kommunikation folgt anschlussfähige Kommunikation. Dennoch entstehen auf diese Weise komplexe Regeln, Regelkreise, Muster und Prozesse (dies ist beispielhaft vergleichbar mit der aus der Mathematik bekannten Theorie abstrakter Gruppen).

Eine wichtige Eigenschaft der sozialen Systeme ist, dass kein anderes soziales System direkten Zugriff auf die Kommunikation eines anderen hat! Auf natürliche Weise entsteht so eine Grenze zwischen

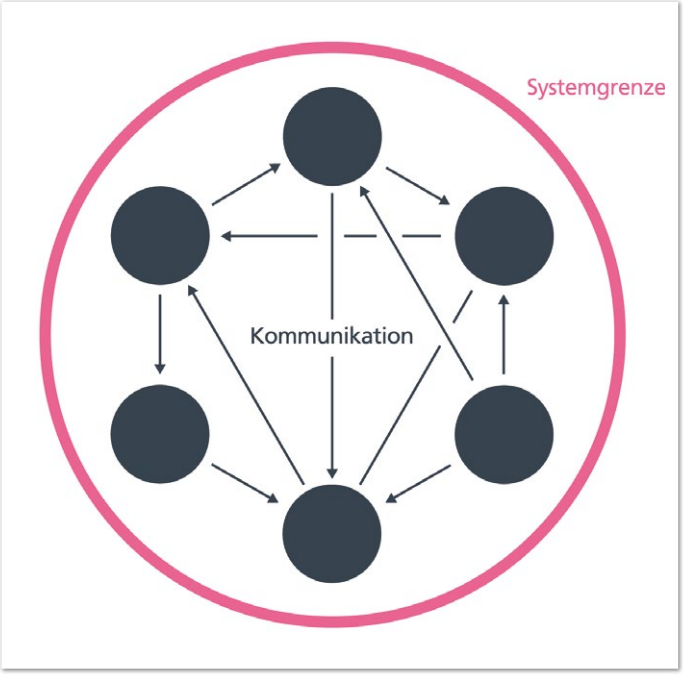


Abbildung 1: Skizziert ist die Grundidee sozialer Systeme. Ausschließlich Kommunikation ist als Operation erlaubt, dadurch grenzt sich ein soziales System von einem Außen ab (Quelle: Christian Mennerich)

einem Außen und dem sozialen System. Formal sind soziale Systeme als autonom, selbsterzeugend und -erhaltend (im Fachjargon autopoietisch) sowie operativ geschlossen bezüglich der Operation der Kommunikation definiert *(siehe Abbildung 1)*.

Wir lassen in unserem vereinfachten Modell auch die Interaktion sozialer Systeme miteinander zu: Auf die Grenze eines sozialen Systems treffen Reize, die es aufnimmt und interpretiert *(siehe Abbildung 2)*. Die Interpretation geschieht nach den Regeln und Mustern, die es definieren. Damit ist die Reizverarbeitung, das heißt die Art und Weise, ob und wie auf einen Reiz reagiert wird, allein durch das den Reiz aufnehmende System bestimmt! Kommunikation, die häufig auftritt, nennen wir relevante Kommunikation: Dieser folgend können gegebenenfalls auch bisher unentdeckte soziale Systeme gefunden werden. Ideen zu Fragetechniken, die das Auffinden unterstützen, werden später im Artikel beschrieben.

Weiterhin ist ein konstruktivistischer Grundgedanke tief in der Systemtheorie nach Luhmann verankert: Realität ist nicht objektiv erfassbar. Wahrgenommen wird immer eine individuelle Momentaufnahme, und diese unterliegt naturgemäß Voreingenommenheiten (bias) und Rauschen (noise) *[12]*. Was wir bestenfalls erreichen können, ist ein Realitätskonsens – ein möglichst einheitliches Verständnis über eine gemeinsame Sache.

Wie aber hilft uns die soziologische Systemtheorie nun bei der agilen, iterativen Softwareentwicklung? Dieser Frage nähern wir uns, indem wir den Zusammenhang zur DDD herausstellen.

Domain-driven Design und Systemtheorie

Domain-driven Design ist eine Arbeitsweise, die die Geschäftsdomäne in das Zentrum der Softwareentwicklung stellt. Es gibt ein Modell, das immer weiterentwickelt wird und über das mit einer universellen Sprache (ubiquitous language) gesprochen wird. Die Domänensprache ist allgegenwärtig, in Diskussionen, Tickets und im Quellcode wird sie verwendet (im Quellcode heißt dies gegebenenfalls auch, dass hier nicht-englische Begriffe verwendet werden, was manch einem Entwickler seltsam erscheinen mag).

Ein abgeschlossener Kontext ist die sinngebende Einheit darum herum. Abgeschlossene Kontexte grenzen sich also gegenüber dem ab, was sie nicht sind. Die Einheit, in der ein (agiles) Team Software entwickelt, ist in der Regel dann so ein abgeschlossener Kontext. Dadurch sind Teams entkoppelbar und die Prinzipien guter Softwareentwicklung wie DRY, YAGNI oder KISS sind kontextlokal zu verstehen. In der Regel interagieren die Kontexte und ihre Softwarelösungen nun wiederum miteinander. Das kann über REST genauso gut geschehen wie asynchron über einen Nachrichtenbus.

Sowohl Systemtheorie als auch DDD definieren abgeschlossene Einheiten über Sprache. Daneben gibt es Regelsätze, innerhalb derer in der Einheit agiert wird, sowie ein Interaktionsmodell zwischen verschiedenen Einheiten, das entkoppelt funktionieren kann. Soziale Systeme können als Subdomänen im Problemraum gewonnen werden, die auf abgeschlossene Kontexte im Lösungsraum projiziert werden *(siehe Abbildung 3)*. Den Part von Reizen können Nachrichten auf einem Nachrichtenbus übernehmen: Wie in den sozialen Systemen gibt es, per se, keine Möglichkeit, eine Interpretation oder

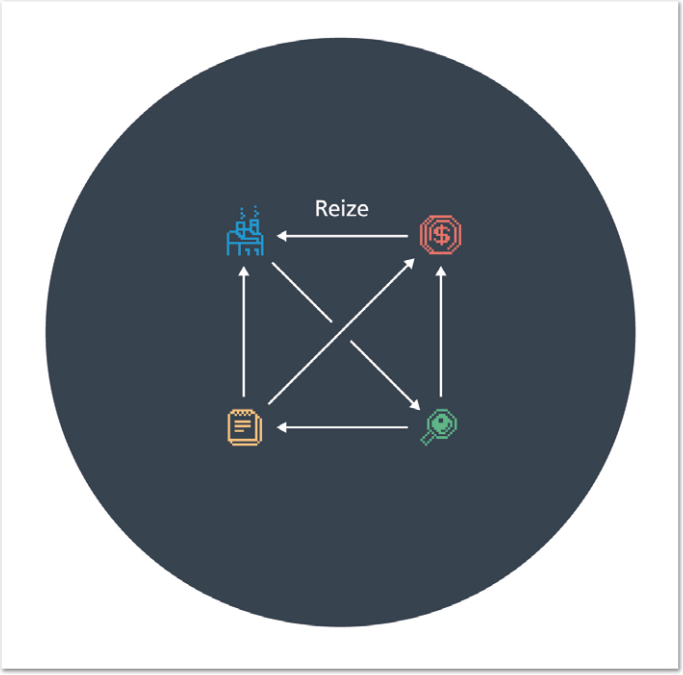


Abbildung 2: Verschiedene Abteilungen einer fiktiven Organisation, die als soziale Systeme über Reize miteinander interagieren (Quelle: Christian Mennerich)

Abbildung 3: Die Projektion der fiktiven Organisation auf abgeschlossene Kontexte und betriebliche Aspekte der Software. Angedeutet sind hier Redundanz- und Resilience-Ansätze sowie die Kommunikation per Nachrichten über einen Nachrichtenbus (Quelle: Christian Mennerich)

Reaktion auf eine Nachricht in einem anderen Kontext zu erzwingen. Ob und wie eine Nachricht verarbeitet wird, ist alleinig eine Konsensleistung des gesamten Teams, das an der Lösungsentwicklung in dem Kontext beteiligt ist. In der Regel sind es verschiedene Teams, die in verschiedenen Kontexten entwickeln. Das Finden von Realitätskonsensen ist also eine Leistung in den Kontexten und über die Kontexte hinaus. Für eine beispielhafte Diskussion im Rahmen eines Requirements Engineering siehe *[13]*.

Das Finden neuer Kontexte

Fragetechniken aus dem systemischen Coaching oder der systemischen Therapie können Wege aufzeigen, bisher „verborgene" soziale Systeme „zu entdecken" und in den Lösungsraum zu projizieren. Als abgeschlossene Kontexte verändern sie den Systemschnitt und verleihen ihm einen neuen Sinn. Das wirkt zurück in die Organisation und führt auch hier zu veränderter Kommunikation. Damit bleibt das Gesamtsystem evolutionär in Bewegung. Etwas detaillierter schauen wir uns das im Kontext des Gesetzes von Conway *[8]* an, hier wollen wir beispielhaft ein paar Fragetechniken beleuchten.

Im systemischen Coaching gibt es die Ideen des zirkulären oder hypothetischen Fragens, der Skalierungs- oder auch Wunderfragen *(siehe Abbildung 4)*. Diese sollen helfen, auf eine andere Art und Weise über sich und seine Umwelt nachzudenken. Die Annahme hypothetischer Zustände oder freie Wunderwünsche dienen dazu, soziale Systeme und Kommunikation von anderer Warte aus zu beleuchten, und können zu neuen Erkenntnissen, Ideen oder dem Aufdecken bisher unerkannter relevanter Kommunikation führen.

Beispielhafte Fragen

- Angenommen, Ihre Kollegin aus der Buchhaltung trifft sich mit Ihrem gemeinsamen Vorgesetzten: Wie würde sie Ihre Arbeitsabläufe beschreiben?
- Woran erkennen Sie, dass Sie einen gelungenen Arbeitstag hatten? Woran macht dies Ihr Kollege aus dem Marketing fest?
- Auf einer Skala von 1 bis 10: Wie gut können Sie Ihre Arbeit als Krankenschwester momentan ausführen? Wofür stehen aus Ihrer Sicht die Werte 1 und 10? Was in Ihrem Arbeitsalltag müsste sich ändern, um sich einen Punkt weiter in Richtung 1 oder 10 zu bewegen?
- Angenommen, viele Herausforderungen Ihrer Arbeit würden wundersam verschwinden: Was würde sich für Sie und Ihre Kollegen in der Verkaufsabteilung ändern? Was konkret würden Sie anders machen als jetzt?

Wechselwirkung: Software und Nutzergemeinschaft

Das Gesetz von Conway besagt, dass jede Organisation, die ein Design entwirft, im Wesentlichen eine Kopie seiner Kommunikationsstruktur erzeugt *[8]*. Da in der Softwareentwicklung das ausgerollte Design auch unmittelbar zurückwirkt auf die Nutzerschaft, ist diese Beziehung aber in beide Richtungen zu verstehen (vgl. die Diskussionen in *[9]*).

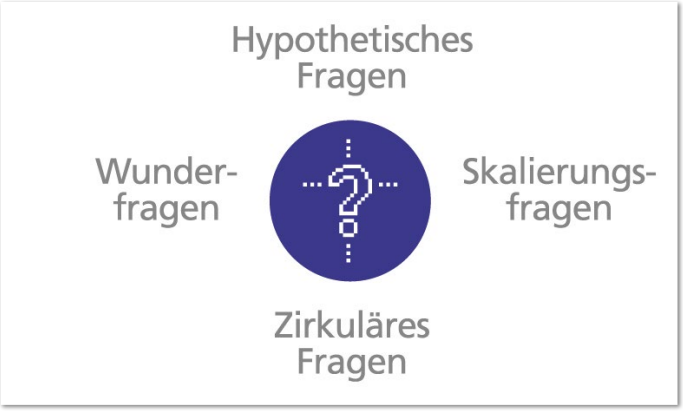


Abbildung 4: Fragetechniken im systemischen Coaching (Quelle: Christian Mennerich)

Optimalerweise ist die Wechselwirkung isomorph (das heißt im mathematischen Sinne ein-eindeutig und strukturerhaltend), die Software bildet genau das ab, was die Nutzer sich gewünscht haben: Die Kundenzufriedenheit, als Maß der Güte eines Softwareproduktes verstanden, ist dann hoch. Im Gebrauch der Software entscheidet sich, ob die gewünschte Funktionalität auch die ist, die der Benutzer braucht. Änderungswünsche und deren Implementierung setzen nun den agilen, iterativen Entwicklungsgedanken in Gang. Begreifen wir diesen Softwareentwicklungsprozess ganzheitlich, dann schließt er Anforderungsfindung sowie -implementierung in crossfunktionalen Teams ein, und die Systemtheorie bietet uns wertvolle Ansätze, das Vorgehen besser zu verstehen.

Systemisch-agil

Zum einen besagt der konstruktivistische Grundgedanke, dass Realität immer eine Konstruktion eines Beobachters ist. Es ist illusorisch anzunehmen, dass verschiedene Beobachter die Realität exakt gleich wahrnehmen und konstruieren (vgl. *[14]*, und für unterhaltsam geschriebene, illustrative klinische Beispiele auch *[15]*). Für die Praxis heißt dies, dass das Erreichen eines Konsenses wichtig ist, dass ein gleiches Verständnis über eine gemeinsame Sache wichtiger ist als das Erfassen „der Realität". Da aber viele verschiedene Individuen am Prozess beteiligt sind, vom Endnutzer über Stakeholder, Product Owner und Entwickler, wird dieser nicht sofort gefunden werden können: Ausprobieren heißt die Devise! Und sich so einem Realitätskonsens schrittweise und iterativ nähern. Fehler machen gehört ebenso natürlich dazu wie kontinuierliches Lernen. Dies benötigt ein Umdenken in der Fehlerkultur, weg vom „Blaming", hin zu einer offenen, fehlertoleranten Kultur *[16]*.

Die iterative und agile Arbeitsweise unterstützt hier auf verschiedenen Ebenen: Zum einen sind kleine Auslieferungsinkremente geeignet, die Kundenzufriedenheit unmittelbar erfahr- und messbar zu machen. Schnelles Feedback und kurze Lernzyklen schaffen Vertrauen und den Rahmen, im Sinne der Kundenzufriedenheit agieren zu können. Kleine Fortschritte (marginal gains) können zusammen einen erheblichen positiven Einfluss haben *[16]*, schwarze Schwäne (black swans) instantan alle bisherigen Planungen zunichtemachen *[17]* und allgegenwärtige Unsicherheiten und Rauschen müssen bewusst gemacht werden *[14, 19]*. Fehler sind Möglichkeiten zu lernen – und in einer komplexen Welt wie der unseren sind Fehler keine Fehlleistung, sondern unvermeidbar! Diese Ideen liegen auch dem Event Storming zugrunde, das einen Ansatz liefert, auf verschiedenen Ebenen ereignisgetrieben Anforderungen zu finden, evolutionär festzuhalten und zu pflegen *[3]*.

Kontinuierliches Lernen

Technisch unterstützen testgetriebene Build Pipelines, Continuous Integration und Delivery den Prozess des kontinuierlichen Lernens: Wenn Features und Patches nur noch Sekunden brauchen, um den Weg in die Produktion zu finden, und im Vorfeld Sicherheitsnetze gespannt sind, die fehlerhafte Prozesse früh abfangen (fail fast), schafft das Vertrauen. Und das sowohl bei Entwicklung und Betrieb als auch in der Nutzerschaft.

Sollte dennoch etwas schiefgehen, so sind kleine Features oft ebenso schnell zurück- wie vorher ausgerollt. Wissenschaftliche Studien und Erfahrungen stützen diese Ansätze *[6, 5]*. Es gibt weithin Vorgehensweisen, auch Softwarearchitekturen von vornherein evolutionierbar zu gestalten (vgl. zum Beispiel *[18]*).

Was wir daraus lernen können

Wir hoffen, motivieren zu können, dass soziologische Ansätze wichtig und notwendig sind, um Gebrauch und Entwicklung von Software besser verstehen zu lernen. Gerade in unserer heutigen Welt, die sich zu komplex gestaltet, um sie vollständig zu verstehen, braucht es Bewusstsein für die Verschiedenheit der Wahrnehmungen, Arbeits- und Wirkungsweisen sozialer Prozesse. Nur so können wir uns auch im Sinne der Kundenzufriedenheit einem guten Softwareprodukt nähern. Die diskutierte Sicht auf die Systemtheorie nach Luhmann ist eine Möglichkeit, sich hier anzunähern. Insbesondere der konstruktivistische Gedanke liefert Ansatzpunkte, agil-iteratives Vorgehen anders zu motivieren und die technischen Möglichkeiten bewusst einzusetzen, um deren Potenzial auszuschöpfen.

Literatur

[1] E. J. Evans. Domain-Driven Design: Tackling Complexity in the Heart of Software, Addison Wesley 2003

[2] V. Vernon. Implementing Domain-Driven Design, Addison-Wesley Professional 2013

[3] A. Brandolini. Introducing Event Storming. Leanpub 2021

[4] M. Skelton, M. Pals, Team Topologies, IT Revolution Press 2019

[5] D. Farley, j: Humble. Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation, Addison-Wesley Professional 2010

[6] N. Forsgren, J. Huble, G. Kim, Accelerate: The Science of Lean Software and DevOps: Building and Scaling High Performing Technology Organizations, IT Revolution Press 2018.

[7] C. Mennerich, J. Arrasz. Eine Reise durch die Welt von NoSQL (shortcuts 149), entwickler.press 2015

[8] M. Conway: How Do Committees Invent? In: F. D. Thompson Publications 1968

[9] What do I think about Conway's Law now? Conclusions of a European Conf. on Pattern Languages of Programs 2005 Focus Group 2005

[10] G. Beine: Conway's Law und Soziologie in der Softwarearchitektur, *https://www.youtube.com/watch?v=mVgZ6whwtsA*

[11] N. Luhmann: Die Realität der Massenmedien, VS Verlag für Sozialwissenschaften 2004

[12] D. Kahneman, O. Sibony, C.R. Sunstein. Noise: A Flaw in Human Judgment, Little, Brown Spark 2021

[13] C. Mennerich, F. Meseck. Systemtheorie und Softwaredesign. OBJEKTspektrum 04/2020

[14] D. Kahneman. Thinking, Fast and Slow: D. Kahneman, Penguin 2012

[15] O. Sacks. Der Mann, der seine Frau mit einem Hut verwechselte. Rowohlt Taschenbuch 1990

[16] M. Seyd. Black Box Thinking: Marginal Gains and the Secrets of High Performance. John Murray 2016

[17] N.N. Taleb. The Black Swan: The Impact of the Highly Improbable: The Impact of the Highly Improbable. Random House Publishing Group 2010

[18] N. Ford, R. Parsons, P. Kua. Building Evolutionary Architectures: Support Constant Change. O'Reilly UK Ltd. 2017

Christian Mennerich

Synyx GmH & Co. KG

mennerich@synyx.de

Christian Mennerich hat bereits in seinem Studium der Diplom-Informatik einen Fokus auf die Theorie und Implementation von Datenbanksystemen gelegt. Seit 2013 arbeitet er als Entwickler bei der synyx GmbH & Co. KG in Karlsruhe. Dort beschäftigt er sich unter anderem mit NoSQL und nachrichtenorientieren, verteilten Systemen. Christian ist immer wieder als Sprecher auf Tagungen und Konferenzen unterwegs, schreibt Artikel und lektoriert gelegentlich.

Metriken für agile Softwareentwicklungsteams

Richard Fichtner, XDEV Software GmbH

Bei agiler Softwareentwicklung nach dem Vorgehensmodell Scrum stehen die Anpassung an veränderte Bedingungen beziehungsweise Anforderungen und die Optimierung des Prozesses im Mittelpunkt. Dabei gehen Anpassungen und Optimierungen von dem Entwicklungsteam selbst aus. Dieses reflektiert regelmäßig und passt seine Vorgehensweise und Planungen entsprechend an.

Um den Verlauf einer Iteration begründet beurteilen zu können, kommen in der Regel Metriken zum Einsatz. Als Grundlage für diese Metriken benötigt das Scrum Team Daten zu Arbeitszeiten und Aufgaben der Teammitglieder im Betrachtungszeitraum. Die Metriken ermöglichen den Teammitgliedern einen besseren Zugang zu den Metadaten ihrer eigenen Arbeitsrealität. Damit soll das Team in die Lage versetzt werden, die eigene Arbeit besser zu verstehen und auf Basis der gewonnenen Informationen seine Prozesse sinnvoll anzupassen.


25%

Softwareentwicklung ist ein anspruchsvolles Aufgabenfeld auf vielschichtigen Ebenen. Neben technischen und fachlichen Aspekten sind auch die Gestaltungsmöglichkeiten beim Prozess der Softwareentwicklung vielfältig. In den vergangenen Jahren haben sich agile Vorgehensmodelle etabliert und damit die geradlinigen Wasserfälle aufgewirbelt und in Wildwasserbahnen verwandelt. Damit werden der bekannte Entwicklungsprozess und alle damit verbundenen Gewohnheiten infrage gestellt. Entwicklungsteams erhalten neue Frei- und Spielräume, um ihre Arbeitsabläufe zu gestalten. Durch eine optimale Anpassung an die jeweilige Realität des konkreten Projektes lassen sich Wettbewerbsvorteile schaffen. Kurze Iterationen und enge Zusammenarbeit mit Anwendern ermöglichen einen hohen Wirkungsgrad der knappen Ressource „Softwareentwickler".

Gleichzeitig bedeutet dies, dass Softwareentwicklungsteams in agilen Projekten stärker mit Entscheidungen rund um das eigene Vorgehensmodell konfrontiert werden. Die sprichwörtliche „Qual der Wahl" kommt als Preis der neuen Freiheiten. Welche Veränderungen sind sinnvoll? Muss überhaupt etwas geändert werden? Wo haben wir Verbesserungspotenzial? Woher wissen wir, ob Änderungen den gewünschten Effekt erzielen? Dies sind grundlegenden Fragen, die auch meine Kolleginnen und Kollegen und mich in den letzten Jahren intensiv beschäftigt haben. Daher haben wir Metriken konzipiert, um die Metadaten zur eigenen Arbeit allen Teammitgliedern in visualisierter Form zugänglich zu machen. Im Spannungsfeld zwischen Datenschutz und Transparenz wollen wir die eigene Arbeit besser verstehen und auf Basis der gewonnenen Informationen unsere Prozesse sinnvoll anzupassen.

Welchen Zweck erfüllen Metriken?

Metriken können bei einer Vielzahl von Anwendungsfällen hilfreich sein. In der Regel werden Metriken in der Vorbereitung von Entscheidungen eingesetzt. Dabei lassen sich Metriken drei grundlegende Aufgaben zuordnen (vgl. Fenton, et al., 2014 p. 16f):

1. Metriken können helfen, besser zu **verstehen**, was tatsächlich passiert. Durch die Untersuchung der aktuellen Situation lässt sich eine Baseline herstellen, die es ermöglicht, Ziele für zukünftiges Verhalten zu definieren. In diesem Sinn werden Veränderungen an Eigenschaften von untersuchten Prozessen beziehungsweise Produkten im Zusammenhang mit Maßnahmen besser sichtbar und verständlich.
2. Metriken erlauben es, zu **kontrollieren**, was passiert. Durch das gewonnene Verständnis für Beziehungen zwischen Maßnahmen und deren Auswirkungen können gezielt Handlungen durchgeführt werden, die mit hoher Wahrscheinlichkeit zur Erreichung der angestrebten Ziele führen. In diesem Bereich spielen Metriken eine wichtige Rolle beim Risikomanagement.
3. Metriken können zur **Verbesserung** von Prozessen und Produkten ermutigen. Metriken können Menschen motivieren, ihr Verhalten zu verändern. So kann mit Gamification auf Basis von Metriken eine Motivationssteigerung für Arbeiten erreicht werden, die ansonsten als wenig herausfordernd oder zu komplex empfunden werden (vgl. Burke, 2014 pp. 5-8). Im Gegenzug können Metriken auch dazu führen, dass Demotivation einsetzt. Aus Angst vor negativen Auswirkungen, als Reaktion auf „schlechte Werte", werden die Basisdaten systematisch geschönt. Die Metrik wird somit unbrauchbar für den angestrebten Zweck (vgl. Nicolette, 2015 S. 161f).

Herleitung von Metriken

Metriken sind oft auch ein iterativer Prozess. Dies zeigt beispielsweise die Geschichte der Messung der Temperatur. In *Abbildung 1* werden einige Meilensteine der Entwicklung der Temperaturmessung dargestellt.

Jahr	Messverfahren
2000 v. Chr.	Einordnung: „heißer als"
1600 n. Chr.	Das erste Thermometer misst „heißer als"
1720 n. Chr.	Fahrenheit-Skala
1742 n. Chr.	Celsius-Skala
1854 n. Chr.	Absolut Null, Kelvin-Skala

Abbildung 1: Entwicklung der Temperaturmessung (vgl. Fenton, et al., 2014 p. 30)

Wir können anfangen, die Welt zu verstehen, indem wir intuitiv Beziehungen beschreiben, ohne dass wir diese reproduzierbar und absolut messen können. Nachdem einige Erfahrungen gesammelt wurden und ein Grundverständnis für den Gegenstand der Betrachtung entstanden ist, wird eine Möglichkeit benötigt, um eine genauere Messung durchzuführen. Dafür wiederum müssen meist spezielle Methoden oder Hilfsmittel herangezogen werden. Die anschließende Analyse der Ergebnisse führt oft zu neuen Erkenntnissen, höherer Messgenauigkeit und einem tieferen Verständnis (vgl. Fenton, et al., 2014 p. 30). Metriken müssen also erarbeitet werden.

Im Zusammenhang mit Scrum gibt es bereits einige Metriken und Visualisierungen, die weit verbreitet sind und in der Community diskutiert werden. Dazu zählen Metriken wie Velocity, die angibt, wie viele Story Points ein Team in einem Sprint leistet, und Visualisierung wie der Burndown-Chart. Über die gängigen Metriken lassen sich grundlegende Fragen zu agiler Softwareentwicklung beantworten. Wir brauchen für unsere Fragen jedoch andere Metriken.

Rahmenbedingungen und Annahmen

Für alle Metriken gelten folgende Rahmenbedingungen und Annahmen:

- Basis für die Metriken sollen Daten aus bestehenden Systemen sein
- Die Metriken sollen für Scrum-Teams unabhängig vom Einsatz spezifischer Software-Tools anwendbar sein
- Als Betrachtungsrahmen sollen ein Entwicklungsteam und dessen Aufgaben in einem bestimmten Zeitraum herangezogen werden. Der Zeitraum ist standardmäßig eine Iteration
- Die Visualisierungen der Metriken sollen sich durch Standard-Diagramm-Bibliotheken in Webanwendungen implementieren lassen, um eine einfache Integration in bestehende Webanwendungen zu ermöglichen
- Es wird angenommen, dass bezüglich der Verarbeitung persönlicher Daten eine der im BDSG festgelegten Optionen erfüllt und die Verarbeitung daher zulässig ist

Datenquellen – Woher kommen die Daten?

Auf die manuelle Erhebung von Daten verzichtet man in der Regel aus zwei Gründen. Zum einen ist die manuelle Erhebung von Daten mit Formularen, egal ob analog oder digital, der Gefahr von absichtlicher und unabsichtlicher Verfälschung, Verweigerung und Verspätung ausgesetzt.

Zum anderen wirken sich zusätzliche bürokratische Maßnahmen negativ auf die Netto-Arbeitszeit des Entwicklungsteams aus (Fenton, et al., 2014 p. 204f). Daher soll ausschließlich auf bereits vorhandene beziehungsweise bereits durch den Prozess oder die Verwendung von Tools entstehende Daten zurückgegriffen werden.

Softwareentwicklungsteams generieren über den gesamten Software-Lebenszyklus hinweg eine Menge Daten in verschiedenen Systemen:

- Projekt-Tracking-System/Issue Tracker: Informationen zu Aufgaben und Aufwänden
- Version-/Source-Control-System führt Buch über Änderungen an Dateien
- Continuous Integration (CI) Server prüfen stetig die Qualität der Software

Durch den Einsatz der genannten Systeme existiert für die meisten Teams bereits eine breite Datenbasis als Grundlage für eine Vielzahl von Metriken, die mit Daten aus einem System hergeleitet werden können. Diese Metriken beschränken sich auf eine spezielle Sicht auf das Team. Interessant ist eine ganzheitlichere Sicht auf das Team, um Zusammenhänge über Grenzen der Dateninseln hinaus zu verstehen. Dafür müssen die Daten aus den einzelnen Systemen zusammengeführt werden. Die Herausforderung dabei ist, die jeweiligen Datenmengen sinnvoll zu integrieren. Es müssen Verbindungspunkte zwischen den Datenmengen gefunden oder geschaffen werden, um die Daten aus mehreren Quellen sinnvoll zu verbinden. Als Verbindungspunkte können Daten dienen, die in jeweils zwei zu verbindenden Datenmengen vorliegen.

Ein universelles Konzept, das in vielen Datenmengen existiert, ist Zeit. Genauer: ein Zeitpunkt, der einem Wert in einer Datenmenge zugeordnet ist. Über den Zeitpunkt können Datenmengen integriert werden. Anspruchsvoller wird die Integration von Datenpunkten, die sich nicht eindeutig zuordnen lassen. Unterschiedliche Datenstrukturen, die dieselben Entitäten abbilden, können normalisiert werden.

Ein prominentes Beispiel dafür sind Benutzer. In der Regel halten die Systeme eigene Schlüssel für ihre Benutzerdatensätze. Daher sind diese Schlüssel allein in der Regel nicht ausreichend, um die Datensätze zusammenzuführen. Über ein eindeutiges Merkmal, etwa die E-Mail-Adresse, können Benutzer auf eine einheitliche Basis konsolidiert werden. Kann kein durchgängig verwendetes Merkmal ausgemacht werden, kann auf eine externe Zuordnung der Datensätze (Mapping) ausgewichen werden. Diese Lösung ist nicht wartungsfrei (neue Benutzer müssen in der Zuordnungsregel eingetragen werden) und sollte daher nur als Workaround zum Einsatz kommen.

Metriken

In den folgenden Absätzen werden einige Metriken vorgestellt, die uns in der Vergangenheit geholfen haben.

Team Homogeneity (TH)

Beantwortete Frage(n)

- Wie homogen ist das betrachtete Team? (Fachliche Segmentierung)
- Handelt es sich um ein Team oder mehrere Teams?
- Wer hat welche Aufgabentypen bearbeitet? (Technische Segmentierung)

Datenquellen

- Projekt-Tracking-System
- Version-Control-System

Homogenität

Das Team soll bezüglich seiner Arbeits- beziehungsweise Aufgabenhomogenität untersucht werden. Dies lässt sich in zwei Dimensionen betrachten: technisch (horizontal) und fachlich (vertikal).

Bei der technischen Dimension wird zwischen den horizontalen Elementen eines Produkts unterschieden, beispielsweise Frontend, Backend, Tests, Dokumentation, Framework A, Framework B etc.

Bei der fachlichen Dimension wird zwischen den vertikalen Elementen eines Produkts unterschieden, etwa Feature A, Feature B, Module C, Module D etc. Für beide Dimensionen gilt, je gleichmäßiger die Aufgaben über das Team verteilt sind, desto homogener ist das Team. Wird jedes Element nur von einer Entwicklerin bearbeitet, ist das Team vollständig heterogen. Beide Extreme sind nicht erstrebenswert.

Vollständige Heterogenität bedeutet, dass spezifisches Wissen nur an einer Stelle im Team existiert (Single Point of Failure). Ist ein Teammitglied nicht verfügbar, fehlt dem Team ersatzlos das Wissen über einen Teil des Projekts. Die Gründe für Engpässe in der Verfügbarkeit von Teammitgliedern sind vielfältig: Urlaub, Elternzeit, Krankheit, Team- oder Arbeitgeberwechsel sind mögliche Faktoren. Um weiterarbeiten zu können, muss das Team das Wissen ohne den designierten Spezialisten erarbeiten oder warten, bis dieser wieder zur Verfügung steht. Somit bringt vollständige Heterogenität Risiken für die Planbarkeit neuer Features sowie für die Durchführung von Wartungsarbeiten.

Vollständige Homogenität bedeutet, dass jedes Teammitglied allumfassendes Wissen über die technischen beziehungswese fachlichen Gegebenheiten des Projekts besitzt (Total Knowledge Sharing). Ist ein Teammitglied nicht verfügbar, können dessen Aufgaben durch ein beliebiges anderes Teammitglied übernommen werden. Der hohe Grad der Redundanz beziehungsweise die Kosten in Form von Zeit, um diese Redundanz herzustellen, stehen in Konkurrenz zum Ziel „Business Value" in Form von „Features liefern". Somit birgt vollständige Homogenität das Risiko, dass dem Team wenig Zeit bleibt, um neue Features zu entwickeln.

Projekte haben unterschiedliche Anforderungen daran, wie viel Redundanz angestrebt wird, beziehungsweise wo sich das Team zwischen den Extremen positionieren möchte. Wir haben festgestellt, dass die Selbsteinschätzung von Teams über ihre aktuelle Position weit von dem abweicht, was wir gemessen haben. Es lohnt sich also, einen Blick auf die Zahlen zu werfen.

Anwendungsgebiet

In der Regel liegt die Teamhomogenität zwischen den Extremen. Folgende Motivationen können bestehen, um die Teamhomogenität zu messen, zu beobachten und in eine Richtung zu entwickeln.

Bei Scrum gibt das Product Backlog beziehungsweise das Sprint Backlog die Priorität und damit die Reihenfolge der Aufgaben vor (vgl. Sutherland, et al., 2016 p. 5). Im Idealfall kann ein Team im Sprint

genau nach dieser Reihenfolge die Aufgaben bearbeiten. Je höher die Heterogenität im Team ist, desto höher ist die Wahrscheinlichkeit, dass die Aufgaben nicht in der vorgegebenen Reihenfolge erledigt werden können. Es entstehen Knowledge-Bottlenecks, die dazu führen, dass das Team Aufgaben überspringen oder warten muss. Somit wird das Team nicht dem Anspruch gerecht, die Features zu liefern, die am dringendsten benötigt werden.

Eine weitere Motivation ist der „Bus Factor" oder „Truck Number", eine Kennzahl, die der Autor James Coplien zur Abschätzung von Risiken in Software-Projekten vorschlägt. Der Wert gibt die Wahrscheinlichkeit des Scheiterns eines Projekts bei Ausfall eines Mitarbeiters an (Coplien, et al., 2005 p. 157).

> ***„How many or few would have to be hit by a truck (or quit) before the project is incapacitated?"***
>
> (Williams, et al., 1990)

Die Ermittlung dieses Faktors wird in der Regel aus dem Kopf durchgeführt. Frei nach der Überlegung: Wer müsste ausfallen, damit wir ein Problem haben? Dabei besteht das Risiko, dass man sich verschätzt und eine falsche Annahme trifft. Sicherer ist eine Herleitung des Faktors über die Teamhomogenität. Je nach Qualität der Datenbasis kann man auf diesem Weg sehr genaue Aussagen erhalten.

Weiterhin kann Teamhomogenität helfen, Sub-Teams – Teams im Team – zu identifizieren. Möglichweise arbeiten zwei Untergruppen im Team stark disjunkt an fachlichen oder technischen Themen. Mit dieser Information kann man entweder erkennen, dass es sich tatsächlich um zwei Teams handelt, und die Teams trennen oder man arbeitet daran, dass aus zwei Teams eins wird.

Berechnung

Variable	Beschreibung
k	Komponente: Was als Komponente angesehen wird, kann für das jeweilige Projekt definiert werden. Die erste Instanz ist eine Unterscheidung in technische beziehungsweise fachliche Ebene. Anschließend können projektspezifische Komponentengrenzen definiert werden. Eine fachliche Komponente in einer E-Commerce-Anwendung könnte beispielsweise der Warenkorb sein.
$\mathbf{t_k}$	Zeit, die für die Arbeit an der Komponente k vom Team aufgebracht wurde. Im Idealfall fließen hier die Zeiten aller Aufgaben und derer Prozessschritte ein, das heißt auch Spezifikation und Tests.
$\mathbf{m_k}$	Anzahl der Teammitglieder, die an der Komponente k gearbeitet haben
$\mathbf{m_{all}}$	Anzahl der Teammitglieder (die dem Projektteam angehören)
$\mathbf{h_k}$	Teamhomogenität für die Komponente k

Erster Ansatz für die Berechnung der Teamhomogenität:

$$h1_k := \frac{m_k}{m_{all}}$$

Diese Berechnung ist nicht nach Arbeitszeit beziehungsweise Aufwand gewichtet. Das bedeutet, dass ein Teammitglied nur eine Codezeile ändern muss, um mit dem gleichen Gewicht in die Berechnung einzugehen wie ein anderes Teammitglied, das die Komponente seit Jahren betreut. Der Anteil der Teammitglieder wird im Folgenden mit der aufgewendeten Arbeitszeit gewichtet, um eine Abbildung näher an der Realität zu bekommen. Dabei steht die Annahme im Raum: „Arbeitszeit an einer Komponente und Kenntnis der Komponente sind direkt proportional zueinander."

$$TH_k := \prod_m \left(1 + \frac{t_{m,k}}{t_k} \right)$$

Diese Formel hat den Vorteil, dass die Arbeitszeit als Gewichtung einfließt und gleichzeitig die Werte beim Vergrößern des Teams stabil bleiben.

Der Definitionsbereich und der Wertebereich der Metrik stellen sich wie folgt dar:

$D_{TH}(m) = [2;\infty[$
$W_{TH} = [u;o]$
$u = 2$ Das Team ist vollständig heterogen
$o = \left(1+ \frac{1}{m_{all}}\right)^{m_{all}}$ Das Team ist vollständig homogen

Zur Vereinfachung der Darstellung kann der Wertebereich wie folgt auf $W_{TH} = [0;1]$ korrigiert werden:

$$TH_{k,\ korrigiert} := \frac{TH_k - 2}{o - 2}$$

Die Metrik ergibt somit einen Wert zwischen 0 und 1. Es wird eine Darstellung in Prozent empfohlen.

Datenquellen

Version-Control-System

Über das Version-Control-System lässt sich nachvollziehen, welches Teammitglied wann Änderungen an welchen Dateien vorgenommen hat. Dateien lassen sich auf Klassen abbilden. Klassen wiederum können Komponenten zugeordnet werden (vgl. Mahler, 2017). *Abbildung 2* zeigt den Datenpfad von einem User zu einer Komponente.

Diese Variante der Datengewinnung funktioniert gut in Projekten, deren Implementierung eine Zuordnung von Klassen zu Komponenten aus der Codebasis zulassen. Beispiele hierfür sind Java oder C#, die über eine Paket- beziehungsweise Namespace-Struktur eine Hierarchie erzeugen, über die üblicherweise Komponenten abgebildet werden.

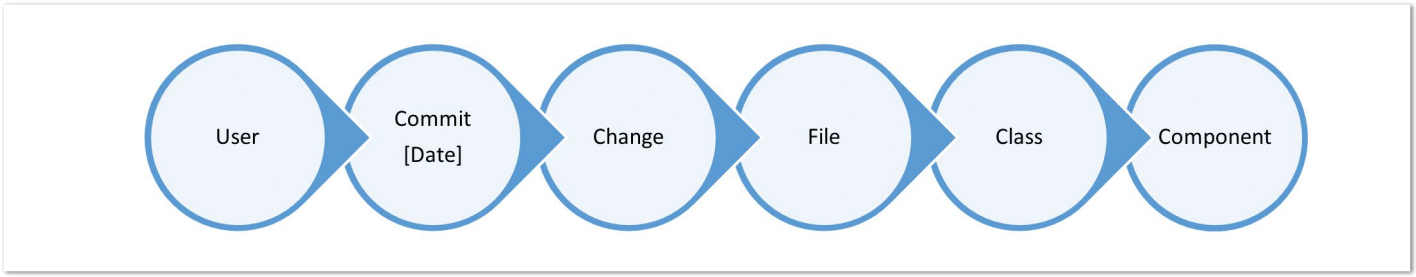


Abbildung 2: Zusammenhang von Änderung der Codebasis und Komponenten

Projekt-Tracking-System

Wenn eine Herleitung des Komponentenbezugs aus dem Version-Control-System nicht oder nicht zufriedenstellend möglich ist, können diese Informationen gegebenenfalls auch alternativ oder ergänzend aus dem Projekt-Tracking-System gewonnen werden. Viele Projekt-Tracking-Systeme sehen in der Standardkonfiguration das Datenfeld „Komponente" vor. Dies ist die offensichtlichste Anlaufstelle. Über Metadaten wie Labels oder Tags kann auch ein Bezug zu Komponenten hergestellt werden. Manche Teams arbeiten auch mit Hashtags in Kommentaren. Durch die Analyse der Hashtags in Kommentaren lässt sich ebenfalls ein Komponentenbezug herstellen.

Die Qualität der Daten, die aus dem Projekt-Tracking-System gewonnen werden, ist dabei stark abhängig von der Disziplin des Teams, Labels und (Hash-)Tags überhaupt und zusätzlich korrekt zu setzen. Die Daten aus dem Projekt-Tracking-System können mit den Daten aus dem Version-Control-System kombiniert werden, um bessere Ergebnisse zu erzielen und gegebenenfalls Inkonsistenzen zu erkennen.

Visualisierung

Als Visualisierung für Teamhomogenität (TH) wird ein kombiniertes Diagramm bestehend aus einem gestapelten Säulendiagramm (100 %) und einem Liniendiagramm vorgeschlagen. Über das Liniendiagramm wird der Verlauf von TH dargestellt.

Element	Beschreibung
X-Achse	Zeitlicher Verlauf
Y-Achse (links): Säulen	Arbeitsanteile der Teammitglieder in Prozent, die zum jeweiligen Zeitpunkt (gestapelt) als Grundlage für die Berechnung von TH stehen
Y-Achse (rechts): Linie	Teamhomogenität in Prozent

In *Abbildung 3* wird beispielhaft die TH für ein Team bestehend aus Alice, Bob und Claire gezeigt. Über den Verlauf der Messpunkte wird die Veränderung der TH des Teams von vollständiger Heterogenität zur vollständigen Homogenität sichtbar. Im Beispiel wird die betrachtete Komponente zum Zeitpunkt der ersten Messung (t1) allein von Alice betreut. Im Verlauf der Zeit beteiligen sich Bob und Claire zunehmend an den Arbeiten an der betrachteten Komponente. Zum Zeitpunkt der letzten Messung (t6) haben alle Teammitglieder schließlich gleiche Zeitanteile. Das Team ist damit für die betrachtete Komponente vollständig homogen.

Hinweise

Verlässt ein Mitarbeiter das Team, sollten dessen Arbeitsanteile nicht mehr in die Berechnung einfließen, da dessen Know-how dem Team nicht mehr zur Verfügung steht. Zugänge zum Team, die nicht an der betrachteten Komponente arbeiten, haben durch die Art der Berechnung keinen Einfluss auf das Ergebnis.

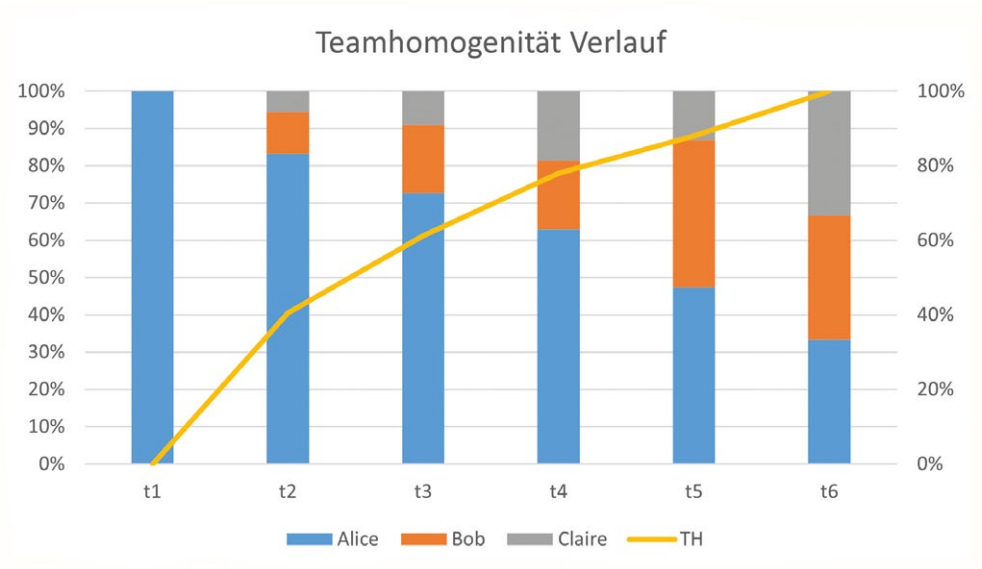


Abbildung 3: Teamhomogenität Verlauf

Sprint-Profile (SP)

Beantwortete Frage(n)

- In welche Typen von Aufgaben (Features, Bugs, Hotfixes, Meetings) hat das Team im Sprint wie viel Zeit investiert?

Beschreibung

- Empirische Messung der eingebrachten Zeit des Entwicklungsteams nach Aufgabentyp

Wert

- Rückblickend: Sprints werden in der Dimension „Aufgabentyp" vergleichbar
- Vorrausschauend: Als Basis für Voraussagen zu Kapazitäten je „Aufgabentyp"

Datenquellen

- Projekt-Tracking-System
- Kostenstellenzuordnung

Anwendungsgebiet

Agile Softwareentwicklung möchte Kundenzufriedenheit durch die schnelle Lieferung von funktionierender Software erreichen. Im Idealfall wird möglichst viel Arbeitszeit in die Entwicklung von Features gesteckt, da diese den Wert der Software steigern. Arbeitsaufwand, der in die Behebung von Bugs oder die Bearbeitung von Hotfixes fließt, vermindert die Zeit, die für Features zur Verfügung steht. Hinzu kommt die Zeit, die das Team in Meetings verbringt, hierzu zählen etwa Daily Scrum, Sprint Retrospektive und Sprint Review (vgl. Roock, et al., 2015, S. 22-24). Das Team soll die Möglichkeit erhalten, Sprints in Bezug auf den Aufgabentyp zu vergleichen, und Schlussfolgerungen daraus zu ziehen.

Mit diesen Informationen kann das Team zukünftige Sprints besser planen, da Erfahrungswerte für die Zeitanteile je Aufgabentyp aus den letzten Sprints vorliegen (vgl. Cohn, 2014 S. 161). Auch Überlegungen zum Thema Qualität können angestellt werden, zumal viele Bugfixes ein Indikator für mangelnde Qualität beziehungsweise Qualitätssicherung sein können.

Berechnung

Variable	Beschreibung
$t_{m,b,p,a}$	Eingebrachte Arbeitszeit in Stunden des Teams (m) im Betrachtungszeitraum (b) für das zu untersuchende Projekt (p) für den Aufgabentyp (a)

$$\text{Sprint-Profil} := \sum_{m} t_{m,b,p,a}$$

Der Wertebereich der Metrik stellt sich wie folgt dar:

W_{SP} = [u;o]
u = 0 Das Team hat keine Zeit für den Aufgabentyp eingebracht.
o = |m| * |b| Alle Teammitglieder haben ununterbrochen am Aufgabentyp gearbeitet.

Die Metrik ergibt einen Zeitwert. Als Einheit wird Stunden vorgeschlagen.

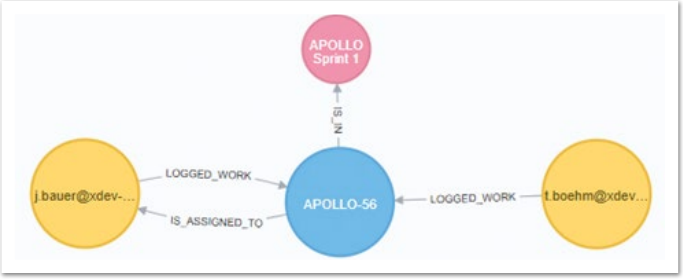


Abbildung 4: Zuordnung von Zeiterfassung und Aufgabentyp

Datenquellen

Über die Kostenstellenzuordnung kann die investierte Zeit des Teams für einen Sprint ermittelt werden. Diese Zeiten müssen mit den Daten aus dem Projekt-Tracking-System angereichert werden, um eine Zuordnung zum Aufgabentyp zu erhalten.

In *Abbildung 4* ist beispielhaft die Aufgabe „APOLLO-56" (blauer Knoten) aus dem „Sprint 1" (rosa Knoten) dargestellt. Zugeordnet sind die Zeiteinträge zweier Kollegen, die an dieser Aufgabe gearbeitet haben (gelbe Knoten). Der Aufgabentyp ist wiederum mit der Aufgabe verknüpft. Somit lassen sich alle notwendigen Daten für SP ermitteln.

Visualisierung

Als Visualisierung wird ein Netzdiagramm vorgeschlagen, da sich damit mehrere Profile gut vergleichen lassen.

Element	Beschreibung
Achsen	Je Aufgabentyp wird eine Achse eingeführt. Auf der Achse werden die Stunden pro Aufgabentyp von innen nach außen angetragen.

Abbildung 5 zeigt zwei Sprint-Profile im Vergleich. Sprint 1 hat ein wünschenswertes Profil, es wurde viel Zeit in Features investiert und wenig Zeit in andere Aufgaben. Im Gegensatz dazu steht das Profil von Sprint 2, es wurde mehr Zeit in Bugs investiert als in Features. Dies deutet darauf hin, dass in Sprint 1 auf Kosten der Qualität das Sprintziel erreicht wurde. Je nachdem, wie das Team mit Bugs umgeht beziehungsweise wie diese in Sprints eingeplant werden, können diese Effekte auch erst in einem späteren Sprint sichtbar werden. Es lohnt sich, die Augen nach solchen Mustern offen zu halten und als Gesprächseinstieg zu nutzen.

Team Time Commitment (TTC)

Beantwortete Frage(n)

- Wie viel Zeit wurde vom Entwicklungsteam eingebracht?

Wert

- Rückblickend: Zeigt auf, in welchem Maß das Team am Sprint arbeiten beziehungsweise nicht arbeiten konnte. Es kann helfen, Störungen des Teams zu erkennen
- Vorausschauend: Über die erhaltenen Datenpunkte lässt sich der Zeiteinsatz für folgende Sprints genauer abschätzen und somit die Sprintkapazität besser planen

Datenquellen

- Projekt-Tracking-System
- gegebenenfalls Zeiterfassungssystem

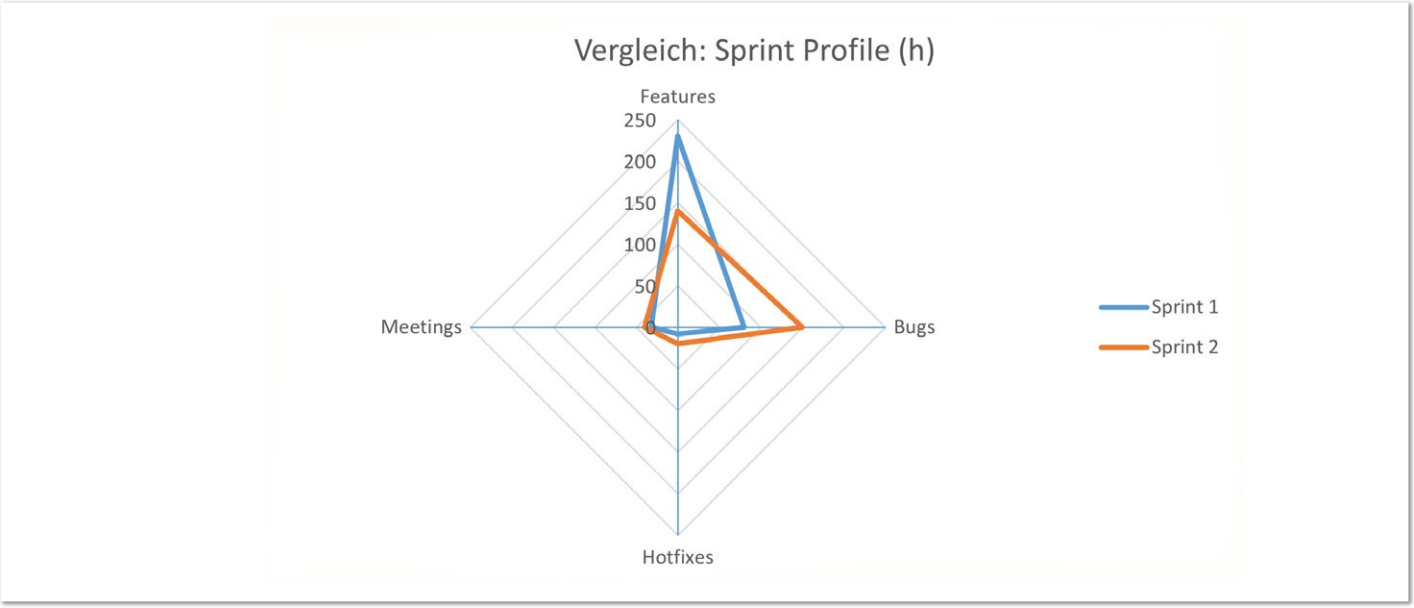


Abbildung 5 Vergleich Spint-Profile (h)

Anwendungsgebiet

Das Team verpflichtet sich bei Scrum auf ein Sprintziel. Wie genau diese Verpflichtung zu interpretieren ist, geht aus dem Scrum-Guide nicht hervor, es ist lediglich die Rede von „Commitment". In der Scrum Community gibt es verschiedene Interpretationen, die an dieser Stelle jedoch keine Rolle spielen.

Interessant ist in der Regel, ob das Sprintziel erreicht werden konnte oder nicht. Das Sprintziel zu erreichen ist desto einfacher, je:

- genauer die Schätzungen sind,
- geschützter der Sprint ist,
- genauer die Sprintkapazität bekannt ist.

Die Genauigkeit von Schätzungen ist ein breites Thema, das nicht in den Fokus dieses Artikels passt und daher besser separat behandelt werden sollte.

TTC adressiert die letzten beiden Punkte. Der Schutz des Sprints wird oft nur aus Sicht der Aufgaben im Sprint betrachtet, es gibt jedoch einen zweiten Angriffspunkt, die Mitglieder im Team (vgl. Röpstorff, et al., 2015, S. 210-212). Würden Teammitglieder dauerhaft entfernt, dann würde das Team explizit darauf reagieren und sich möglicherweise neu organisieren. Unmerklicher sind kleine Veränderungen, etwa, wenn ein Entwickler einige Stunden bei einem anderen Projekt aushelfen soll, weil er dort vor Jahren die Grundlagen geschaffen oder einfach nur Expertise zu einer Technologie hat, die im anfordernden Projekt fehlt.

Genauso ist aber auch ein Teamzuwachs möglich. Beispielweise könnte für eine Story stundenweise eine Kollegin aus dem Infrastrukturteam das Entwicklungsteam unterstützen. Hier grenzen sich Sprint-Schutz und Sprintkapazität ab. Waren die Änderungen am Team geplant, ist der Sprint geschützt und die Kapazität korrekt geplant. Bei der Planung der Sprintkapazität werden gut planbare Termine wie Feiertage und Urlaube berücksichtig. Weniger gut planbare Termine wie Elternzeitbeginn oder Krankheitstage stellen eine Herausforderung dar.

Rückblickend können das geplante TTC mit dem tatsächlichen TTC verglichen werden. Abweichungen in beide Richtungen sind zu untersuchen. Wurde signifikant mehr Zeit eingebracht als geplant, könnte dies auf verdeckte technische oder fachliche Probleme hindeuten. Die Ursache kann auf Aufgabenebene gesucht werden. Wurde signifikant weniger eingebracht als geplant, deutet dies auf Störungen des Teams hin.

Vorausschauend kann TTC verwendet werden, um eine bessere Grundlage für die Planung der Sprintkapazität zu erhalten.

Abgrenzung von Velocity

Die Metrik Velocity beschäftig sich mit der Leistung des Teams bezogen auf fertiggestellte Aufgaben im Sprintzeitraum (Nicolette, 2015 S. 39f). Wir raten davon ab, die Produktivität eines Teams allein anhand von Velocity zu messen.

$$Leistung = \frac{\Delta\, Arbeit}{\Delta\, Zeit} \rightarrow Velocity = \frac{Tasks\ Completed}{Sprint\ Timeframe}$$

TTC hingegen bezieht sich auf die vom Team eingebrachte Zeit. Diese Größe kommt bei der Berechnung der Velocity nicht zum Tragen.

Berechnung

$t_{m,b,p}$ = Eingebrachte Arbeitszeit in Stunden eines Teammitglieds (m) im Betrachtungszeitraum (b) für das zu untersuchende Projekt (p).

$$\text{Team Time Commitment} := \sum_{m} t_{m,b,p}$$

Der Wertebereich der Metrik stellt sich wie folgt dar:

W_{TTC} = [u;o]
u = 0 Das Team hat keine Zeit eingebracht.
o = |m| * |b| Alle Teammitglieder haben ununterbrochen am Projekt gearbeitet. Dieser Wert wird mit steigender Dauer von b zunehmend unwahrscheinlich, da wir Menschen und nicht Maschinen betrachten. Die Metrik ergibt einen Zeitwert. Als Einheit wird Stunden vorgeschlagen.

Datenquellen

Über die Kostenstellenzuordnung kann die investierte Zeit des Teams für den Betrachtungszeitraum ermittelt werden. Durch das Zusammenführen der Aufgaben aus dem Projekt-Tracking-System und den dazugehörigen Zeiten aus dem Zeiterfassungssystem lässt sich TTC berechnen.

Visualisierung

TTC lässt sich in einer Abwandlung des Burndown-Chart darstellen, sodass das Team während des Sprints ein anschauliches Feedback bekommt. Der Burndown-Chart ist eine weitverbreitete Visualisierung für agile Softwareentwicklung. Herkömmlich wird der verbleibende Aufwand in einem Projekt in Relation zur verbleibenden Zeit dargestellt. Ein Beispiel für einen Burndown-Chart zeigt *Abbildung 6* (Nicolette, 2015, S. 52-56).

Element	Beschreibung
X-Achse	Zeitlicher Verlauf
Y-Achse	Verbleibender Aufwand im Betrachtungszeitraum. Der geschätzte Restaufwand zu jedem Zeitpunkt wird an dieser Achse gemessen.
Startpunkt	Am weitesten links liegender Punkt im Diagramm, der Beginn des Betrachtungszeitraums.
Endpunkt	Am weitesten rechts liegender Punkt im Diagramm, der das Ende des Betrachtungszeitraums markiert. Sein Abstand zum Startpunkt ergibt sich aus dem gesamten Aufwand im Betrachtungszeitraum geteilt durch die täglich leistbare Menge an Arbeit.
Ideal Tasks Remaining	Eine gerade Linie von Start- zu Endpunkt, die die idealisierte Annahme, der Aufwand im Projekt würde über die Zeit gleichmäßig geleistet werden, repräsentiert. Am Startpunkt zeigt die Ideallinie den geschätzten gesamten Aufwand im Betrachtungszeitraum. Am Endpunkt trifft sie die X-Achse, denn dann verbleibt kein Aufwand.
Actual Tasks Remaining	Diese Linie zeigt den tatsächlich verbleibenden Restaufwand. Zu Anfang und, bei Einhaltung des Endtermins auch am Ende, liegt diese Linie gleichauf mit der Ideallinie.

Wenn „Actual Tasks Remaining" oberhalb des „Ideal Tasks Remaining" liegt, ist noch mehr zu erledigen, als geplant war, sodass das Projekt hinter dem Zeitplan liegt. Ist das Gegenteil der Fall, dann ist weniger Aufwand notwendig, als geplant war. Damit liegt das Projekt vor dem Zeitplan.

Im Gegensatz zu dem herkömmlichen Burndown-Chart mit der Darstellung der „Completed Tasks" wird beim Burndown-Chart mit der Abbildung von TTC die geleistete Zeit des Teams in Stunden beschrieben. Damit werden zwei wesentliche Effekte erzielt:

1. Es wird sichtbar, was beziehungsweise wie viel Zeit das Team geleistet hat.
2. Es wird sofort sichtbar, wo das Team steht. Die Darstellung von „Completed Tasks" ist jedoch träge, da erst nach Abschluss eines Tasks eine Veränderung sichtbar wird.

Abbildung 7 zeigt ein Beispiel dafür.

Der Burndown-Chart für TTC ist analog zu dem oben beschrieben Burndown- Chart zu interpretieren, mit folgenden Ausnahmen:

Element	Beschreibung
Ideal Invested Time	Eine gerade Linie von Start- zu Endpunkt, die die idealisierte Annahme, die investierte Zeit im Projekt würde über die Zeit gleichmäßig geleistet werden, repräsentiert. Am Startpunkt zeigt die Ideallinie die geplante Gesamtzeit im Betrachtungszeitraum.
Actual Invested Time	Diese Linie zeigt die tatsächlich investierte Zeit.

Obwohl die Linie „ideal" genannt wird, muss es im konkreten Projekt nicht richtig sein, ihr im gesamten zeitlichen Verlauf möglichst genau zu folgen. Dennoch hilft die Ideallinie, den Projektfortschritt einzuschätzen. Im Beispiel liegt „Actual Invested Time" zum Ende des Betrachtungszeitraums unter „Ideal Invested Time". Das bedeutet, das Team hat mehr Zeit investiert als geplant.

In *Abbildung 8* werden die beiden Serien „Completed Tasks" und „Invested Time" in einem Burndown-Chart gezeichnet. Im Vergleich lässt sich gut erkennen, wie unterschiedlich die Werte sind. „Completed Tasks" zeigt zwischen Tag 4 und Tag 7 keine Veränderung und vermittelt den Eindruck, dass das Team keine merklichen Fortschritte macht. „Invested Time" hingegen verändert sich täglich. Das Team scheint in diesem Beispiel am Tag 5 zu erkennen, dass es mehr Zeit einbringen muss, um das Sprintziel zu erreichen.

Schwankungen während des Sprints sind normal: Bei schönem Wetter passiert in der Regel weniger, auch Sportereignisse wie der Super Bowl oder ein Champions-League-Spiel hinterlassen ihre Spuren im Diagramm.

Zum Ende des Sprints hat das Team das Sprintziel erreicht, jedoch mit 30 Stunden mehr Einsatz als geplant. Wenn das Team regelmäßig mehr Zeit als geplant einbringt, um das Sprintziel zu erreichen, sollte das thematisiert werden.

Zusammenfassung und Ausblick

Mit der Idee, durch Introspektion an sich selbst zu arbeiten und neue Erkenntnisse zu erlangen, setzten sich bereits um 400 v. Chr. griechische Philosophen wie Sokrates und Platon auseinander.

Gnothi seauton
(griechisch „Erkenne dich selbst!")

Als Grundlage dafür wurde die Erkenntnis um das eigene Unwissen vorausgesetzt. Erst anschließend könne man sich daran machen, sich selbst zu verstehen (Ferber, et al., 2011, S. 25). Durch die Arbeit an diesem Thema hat sich uns ein neuer Blickwinkel erschlossen. Die richtige Metrik zur richtigen Zeit mit einer sinnvollen Visualisierung kann ein guter Einstiegspunkt für nachhaltige Veränderungen für ein Entwicklungsteam sein. Metriken können Softwareentwicklungsteams helfen, die Metaebene ihrer täglichen Arbeit besser zu verstehen.

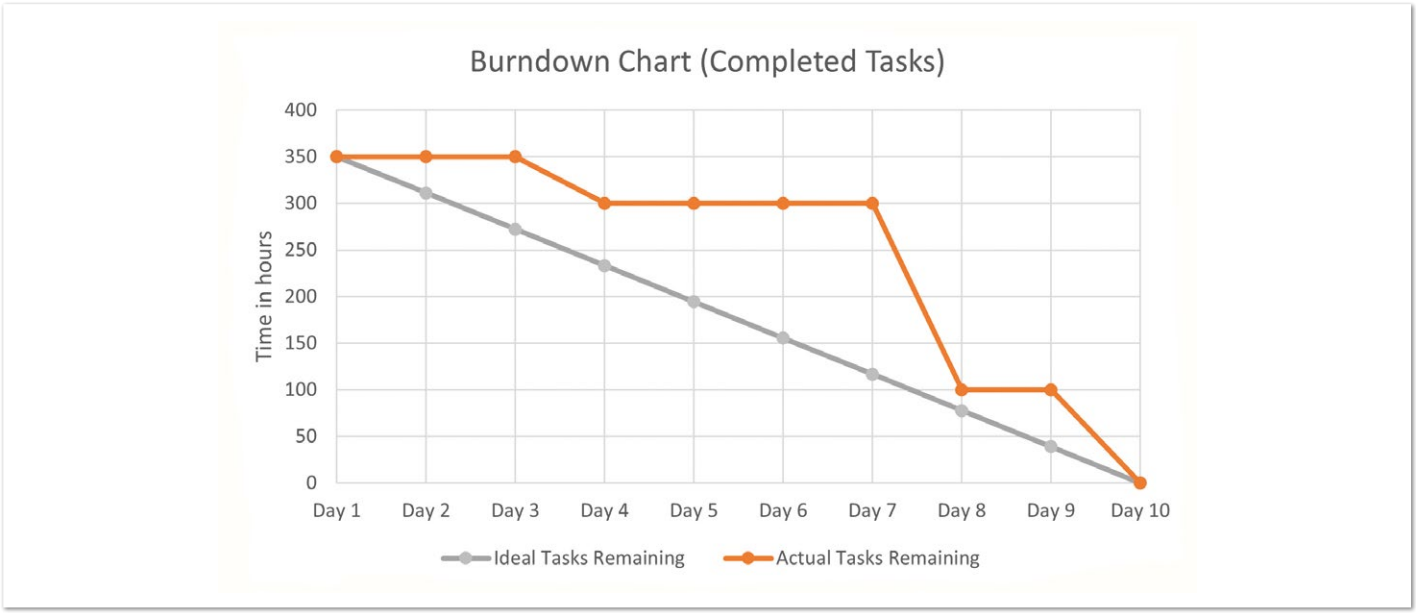


Abbildung 6: Burndown-Chart (Completed Tasks)

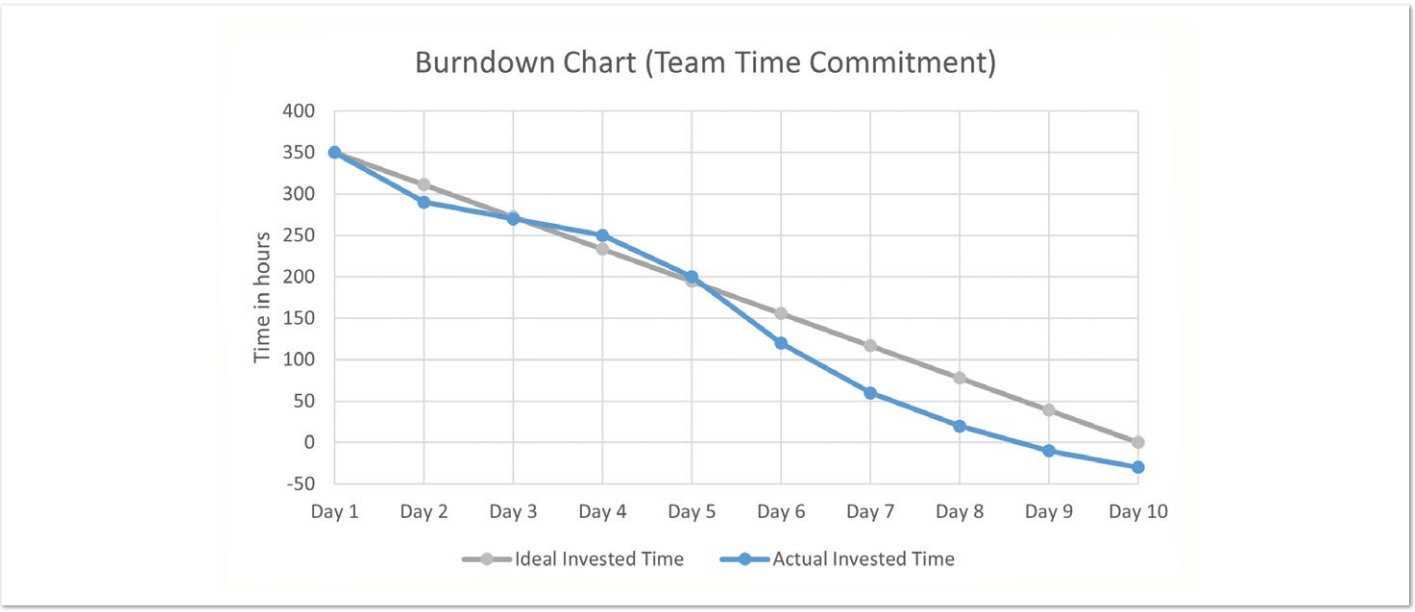


Abbildung 7: Burndown-Chart (Team Time Commitment)

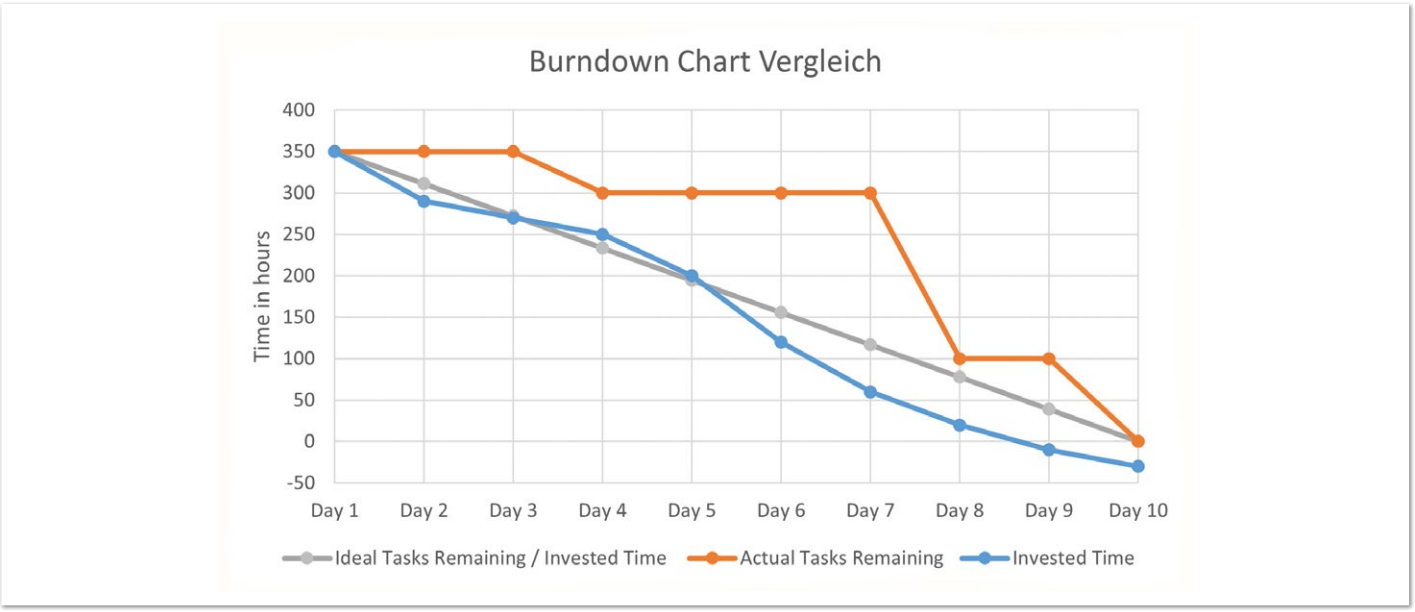


Abbildung 8: Burndown-Chart-Vergleich „Completed Tasks" und „Team Time Commitment"

Die Vorstellung der Metriken war oft der Anstoß für eine angeregte Diskussion, die tiefe Einblicke in die Arbeitsrealität des Teams erlaubte. Die persönlichen Meinungen zu verschiedenen Metriken gehen weit auseinander, es zeigte sich jedoch, dass allein schon das Gespräch über Metriken sehr fruchtbar ist. Hier sehen wir den wahren Wert. Die Zahlen sind immer mit Annahmen versehen und bilden die Realität unzureichend ab. Die Gespräche, die durch die Zahlen angeregt wurden, haben jedoch viele Teams trotzdem weitergebracht. Daher kann ich nur ermutigen, über Metriken zu sprechen und getreu dem agilen Motto „inspect and adapt" zu erforschen, was daraus erwächst.

Literaturverzeichnis

- Burke, Brian. 2014. *Gamify: How Gamification Motivates People to Do Extraordinary Things.* New York : Bibliomotion Inc, Gartner Inc., 2014. 978-1937134853.
- Cohn, Mike. 2014. *Agile Estimating and Planning.* s.l.: Prentice Hall, 2014. 978-0131479418.
- Coplien, James O. and Harrison, Neil B. 2005. *Organizational Patterns of Agile Software Development.* Upper Saddle River: Prentice Hall, 2005. 978-0131467408.
- Fenton, Norman and Bieman, James. 2014. *Software Metrics: A Rigorous and Practical Approach.* 3 Rev. Boca Raton : Chapman & Hall/Crc Innovations in Software Engineering and Software Development Series, 2014. 978-1439838228.
- Ferber, Rafael und Platon. 2011. *Apologie des Sokrates.* München : C.H.Beck, 2011. 978-3406622212.
- Mahler, Dirk. 2017. Shadows Of The Past: Analysis Of Git Repositories - jQAssistant. *jQAssistant.* [Online] 17. 05 2017. [Zitat vom: 21. 05 2017.] *https://jqassistant.org/shadows-of-the-past-analysis-of-git-repositories/*
- Nicolette, David. 2015. *Software Development Metrics.* Shelter Island: Manning, 2015. 978-1617291357.
- Roock, Stefan und Wolf, Henning. 2015. *Scrum - verstehen und erfolgreich einsetzen.* Heidelberg: dpunkt.verlag GmbH, 2015. 978-3864902611.
- Röpstorff , Sven und Wiechmann, Robert. 2015. *Scrum in der Praxis: Erfahrungen, Problemfelder und Erfolgsfaktoren.* Heidelberg: dpunkt.verlag GmbH, 2015. 978-3864902581.
- Sutherland, Jeff and Schwaber, Ken. 2016. The Scrum Guide. *The Scrum Guide.* [Online] 07 2016. [Cited: 05 04, 2017.] *http://www.scrumguides.org/scrum-guide.html*
- Williams, Laurie and Kessler, Robert. 1990. *Pair Programming Illuminated.* Boston : Addison Wesley Pub Co Inc, 1990. 978-0201745764.

Richard Fichtner
XDEV Software GmbH
r.fichtner@xdev-software.de

Richard Fichtner ist passionierter Java-Entwickler mit mehr als 15 Jahren Erfahrung in der Softwarebranche. Er engagiert sich in der Open-Source-Community, um das Wissen über Java-Technologien zu verbreiten. Er spricht auf Konferenzen und leistet einen Beitrag zu verschiedenen Open-Source-Projekten. Als Oracle Groundbreaker Ambassador und AWS Certified Solution Architect unterstützt er Teams beim Einsatz von Cloud-Lösungen. Seine Interessenschwerpunkte sind Clean-Code, Cloud, neue Technologien und alles, was agil ist. In seiner Freizeit organisiert er die Java User Group Oberpfalz.

Mitmachen und Autor werden!

Sie kennen sich in einem bestimmten Gebiet aus dem Java-Themenbereich bestens aus und möchten als Autor Ihr Wissen mit der Community teilen?

Nehmen Sie Kontakt zu uns auf und senden Sie Ihren Artikelvorschlag zur Abstimmung an **redaktion@ijug.eu.**

Wir freuen uns, von Ihnen zu hören!

iJUG Verbund

Ethik des Cloud-Computings – ein Überblick und Ausblick auf aktuelle europäische Entwicklungen

Corinna Wiedenmann, Freshfields Bruckhaus Deringer

Mit der wachsenden Menge an Daten nimmt auch die strategische Bedeutung von Cloud- und Edge-Technologien zu. Für Gesellschaft und bestimmte Wirtschaftsbereiche haben sie das Potenzial, zu kritischen Infrastrukturen zu werden. In der Europäischen Union wird dies sowohl als Chance als auch als Risiko gesehen. Insbesondere die Risiken werfen ethische Fragestellungen rund um Sicherheit und Privatsphäre, Monopolisierung und Nachhaltigkeit auf, die ich in diesem Artikel näher betrachten möchte.

Dafür ist zunächst die Betrachtung des Zusammenspiels der verschiedenen Stakeholder erforderlich, um ein Verständnis für die notwendigen Ethik-Dimensionen zu gewinnen: Datenethik für Cloud-Computing, Ethik für Cloud-Service-Provider und Cloud-Nutzer. Abschließend werde ich einen Ausblick auf politische Entwicklungen in Europa geben, um die aufgezeigte Theorie mit aktuellen Initiativen zu verknüpfen.

Cloud-Computing-Angebote haben in den vergangenen Jahren immer mehr Beliebtheit erlangt. Sowohl für private Anwendungsfälle als auch für Unternehmen bietet die Technologie zahlreiche Vorteile. Daher nimmt die Einführung von Cloud-Technologien bei Unternehmen im Jahr 2021 weiter Fahrt auf. Es zählt zu den am schnellsten wachsenden Bereichen der Computing-Industrie. Als Teilbereich der IT-Service-Industrie generierten Unternehmen in diesem Segment laut Statista im Jahr 2020 weltweit über 300 Milliarden US-Dollar Jahresumsatz [1]. Das rasante Wachstum und die durch Cloud-Computing zur Verfügung gestellten Funktionalitäten bringen jedoch auch Risiken mit sich, die wiederum zum Teil in ethischen Fragestellungen verwurzelt sind. Sie zu entschlüsseln, Aufmerksamkeit der unterschiedlichen Stakeholder zu gewinnen, die mit Cloud-Computing-Technologien interagieren, und ein harmonisiertes ethisches Rahmenwerk zu schaffen, wird Aufgabe der nächsten Jahre sein.

Stakeholder des Cloud-Computings

Um die ethischen Herausforderungen zu verstehen, die mit Cloud-Computing einhergehen, ist es wichtig, sich zunächst eine Übersicht darüber zu verschaffen, wer mit wem bei der Nutzung dieser Technologie interagiert und von möglichen Risiken betroffen ist. Diese Stakeholder lassen sich mithilfe der Betrachtung der Cloud-Wertschöpfungskette identifizieren [2].

Das operative System der Cloud muss zunächst als Ausgangsbasis von den Cloud-Service-Providern zur Verfügung gestellt werden. Somit sind Cloud-Service-Provider die erste zu nennende Stakeholdergruppe. Sie wird häufig in vier Segmente unterteilt, die unterschiedliche Komponenten für das System zur Verfügung stellen. Diese sind: Infrastruktur, Plattform, Ausführungsmodelle und Softwaredienstleistungen.

Bei der ersten Komponente handelt es sich unter anderem um die Bereitstellung von Servern, Netzwerk, Speicherkapazitäten und Virtualisierung der verschiedenen Ebenen. Die zweite Komponente umfasst Funktionalitäten, die für die Entwicklung von Anwendungen erforderlich sind, wie beispielsweise Webserver, Datenbanken und der Zugang zu einer oder mehreren Programmiersprachenumgebungen. Darüber hinaus bieten Ausführungsmodelle als dritte Komponente die Möglichkeit, Ressourcen nach Nutzungsbedarf zuzuweisen.

Die Komponentengruppe mit der größten Vielseitigkeit umfasst die Bereitstellung verschiedener Softwares, die von Kundenmanagementsystemen, Ressourcenplanung über Datenmanagement noch viele weitere Gesichter haben können. Gelegentlich werden diese Komponentenangebote der sogenannten „Infrastructure as a Service" (IaaS), „Platform as a Service" (PaaS), „Function as a Service" (FaaS) und „Software as a Service" (SaaS) auch als Subtypen des Cloud-Computings bezeichnet.

Cloud-Carrier sind die zweite Gruppe an Stakeholdern. Sie ermöglichen die Verknüpfung von Diensten zwischen den Cloud-Service-Providern und anderen Providern wie auch zwischen Providern und Kunden. Netzwerk- und Telekommunikationsunternehmen bieten durch die von ihnen gewährte Struktur an Internet-Netzwerken sowie die erforderliche Telekommunikationsinfrastruktur für Geräte den Zugang zu Cloud-Diensten.

Mit der kontinuierlichen Weiterentwicklung des Cloud-Computings wird die dritte Stakeholdergruppe bereits in manchen Diskussionen infrage gestellt. Die sogenannten Cloud-Broker wurden in der vom National Institute of Standards and Technology (NIST) publizierten Referenz-Architektur für das Cloud-Computing als Akteure benannt, um die Nutzung, Leistung und Bereitstellung von Cloud-Diensten zu verwalten [3].

Zur Überwachung der Einhaltung von Sicherheit, Leistungsfähigkeit, Privatsphäre und anderem spielen Auditoren und Supervisor ebenfalls eine Rolle entlang der Wertschöpfungskette.

Der (End-)Nutzer, der die über die Cloud zur Verfügung gestellten Funktionen und Dienstleistungen privat oder unternehmerisch nutzt, zählt ebenfalls zu der Vielzahl von Stakeholdern. Er profitiert von der Skalierbarkeit und Elastizität der Technologie, die in der Regel geräte- und ortsunabhängig zugreifbar ist und nach Nutzung abgerechnet wird. Dies wiederum erlaubt dem Nutzer eine möglichst kosteneffiziente Nutzung der Dienste. Insbesondere durch die Auslagerung von Wartung und Sicherheit an Service-Provider genießt er vor allem die Vorzüge der Technologie.

Ein letzter Stakeholder, der in Aufzählungen immer wieder vergessen wird, ist die Umwelt. Sie ist jedoch von der Wertschöpfungskette ebenfalls betroffen. Durch die Nutzung von Cloud-Diensten, beispielsweise virtualisierten Datenzentren, kann die Energieeffizienz von Unternehmen gesteigert werden. Die Cloud-Dienste ermöglichen nämlich eine verbrauchsorientierte Nutzung. Dadurch, dass nicht voll ausgelastete lokale Datenzentren so abgestellt werden können, profitiert die Umwelt von den Vorzügen der Technologie. Zugleich ist die Natur jedoch von den Risiken, die sich aus der enormen weltweiten Nutzungszunahme ergeben, ebenfalls direkt betroffen.

Ethische Problemstellungen

Aus ethischer Perspektive ergeben sich aus dieser Multi-Stakeholder-Landschaft Risiken, die nicht nur technischer Lösungen, sondern auch ethisch-rechtlicher Vorgaben bedürfen. Auf einige der am stärksten diskutierten Probleme möchte ich im Folgenden eingehen: Sicherheit, Privatsphäre, Monopolisierung und Nachhaltigkeitsfragen.

Sicherheit

Der Vorzug der Auslagerung bestimmter Dienste bringt zugleich den Nachteil mit sich, dass der Cloud-Nutzer, ebenso wie der Cloud-Service-Provider, der zum Beispiel Ausführungsmodelle auslagert, seine Daten und deren Verarbeitung nicht mehr allein kontrollieren kann. Daraus resultieren ethische Fragen für die Stakeholder. Bei unautorisierten Zugriffen, Datenkorruption, Infrastrukturfehlern oder Zugriffsunverfügbarkeiten können Dilemmas auftreten [4]. Denn wenn bei einem Missbrauch nicht festgestellt werden kann, wer die Verantwortung zu tragen hat, weil diese zwischen verschiedenen Ebenen aufgeteilt ist und auch die Zuständigkeit für sicherheitsrechtliche Fragen sich überlappen, ist die Klärung der Schuldfrage kaum möglich.

Privatsphäre

Die geteilten Daten, insbesondere wenn es sich dabei um sensible oder sogar hochsensible Daten handelt, sind bei der Nutzung von

Cloud-Computing-Diensten mit dem Problem konfrontiert, dass die Datenzentren häufig weltweit verteilt sind. Aus ethischer Sicht ergibt sich daraus das Dilemma, dass unterschiedliche rechtliche und kulturelle Handhabungen bezüglich der Privatsphäre existieren können. Aufgrund der Vielfalt und Komplexität der verschiedenen Elemente kann dies unter Umständen dem Nutzer auch nicht in dem erforderlichen Maß transparent gemacht werden. Zur Sicherstellung, dass Daten nicht zweckentfremdet werden oder Zugriff ohne Kenntnis erteilt wird, ist die Klärung von Privatsphäre-Standards daher unerlässlich.

Monopolisierung

Sowohl Cloud-Nutzer als auch IaaS-, PaaS-, FaaS- und SaaS-Provider sind häufig dem Problem gegenübergestellt, auf bestimmte Serviceanbieter festgelegt zu sein, sobald bestimmte Entscheidungen zum Setup getroffen wurden. Interoperabilität und Multi-Vendor-Clouds werden zwar bereits als Erfordernis für die nächste Generation der Cloud-Computing-Technologie diskutiert, die aktuellen Migrationsherausforderungen bringen jedoch hohe Wechselkosten mit sich. Die Entwicklungsförderung von Interoperabilität erhält deshalb von staatlichen, nicht-staatlichen und unternehmerischen Institutionen große Aufmerksamkeit. Aktuell führt die zur Verfügung stehende Technologie jedoch häufig zu dem Risiko der ungewollten Abhängigkeit und so auch zu einer proprietären Bindung an einen Anbieter oder ein Produkt, die nur mit erheblichen Kosten verändert werden kann *[4]*.

Nachhaltigkeit

Aktuelle Studien gehen davon aus, dass 1,8 bis 2,8 % der globalen Treibhausgasemission auf die Informations- und Kommunikationstechnologie zurückzuführen ist *[5]*. Viele Unternehmen haben die Herausforderung, die mit dem hohen Energiebedarf einhergeht, bereits erkannt. Obgleich viele Cloud-Rechenzentren doppelt oder um ein Mehrfaches so energieeffizient wie ein typisches Rechenzentrum sind, ist der Stromverbrauch enorm. Es bleibt daher ein ethisches Dilemma, bis eine stärkere Adaption von grünem Strom und der Einsatz von intelligenten Stromsparkreislaufmodellen Lösungen anbieten.

Ethischer Dreiklang als Lösungsansatz

Bereits die beschriebene Auswahl an Risiken erfordert Antworten aus der Ethik. Um dem Multi-Stakeholder-Ansatz gerecht zu werden, ist für die Erarbeitung eines ethischen Rahmenwerks ein Dreiklang erforderlich, der Datenethik für Cloud-Computing sowie Ethik für Cloud-Provider und Cloud-Nutzer gleichermaßen berücksichtigt *[6]*.

Datenethik für Cloud-Computing

Wem gehören die Daten, die mit der Cloud-Computing-Technologie verarbeitet werden? Diese Frage (und andere Fragen ähnlicher Natur) leitet unmittelbar zu einer Forderung nach ethischen Standards über. Da Daten als Eigentum und die Achtung des Rechts auf Eigentum direkt mit den grundlegenden Menschenrechten verknüpft sind, ist deren Berücksichtigung auch in diesem Modell enorm wichtig.

„Die Würde des Menschen ist nicht nur ein Grundrecht an sich, sondern bildet auch die Grundlage für weitere Freiheiten und Rechte, einschließlich des Rechts auf Privatsphäre und den Schutz personenbezogener Daten." *[7]*

Die Erfüllung dieses Anspruches bedarf eines ethischen Rahmenwerks, das mit zukunftsorientierten Regelungen die Einhaltung bestimmter Verhaltenskodizes im Umgang mit Daten von allen

Stakeholdern einfordert und diese auch regelmäßig überprüft. Die Verantwortlichkeit jedes einzelnen Stakeholders dabei herauszuarbeiten, besitzt große Bedeutung. Denn gerade wenn die Bereitschaft, über rechtliche Vorgaben sogar hinauszugehen, erzielt werden soll, braucht es Ethikkodizes. Das grenzüberschreitende und interkulturelle Setup vieler Cloud-Computing-Technologien zeigt den Bedarf auf.

Ethik für Cloud-Service-Provider

Wer ist zuständig und kann man diesem Stakeholder vertrauen? Dadurch, dass Cloud-Computing-Angebote häufig auf die Verknüpfung mehrerer Dienstleistungen unterschiedlicher Service-Provider aufbauen, ist diese Frage wichtig. Vertrauen ist in Multi-Stakeholder-Umgebungen zu klären, um Haftungs- und Verantwortungsfragen zu beantworten. Weil die Zuständigkeiten häufig ineinandergreifen oder aufeinander aufbauen, gestaltet sich dies nicht immer einfach.

Durch die Einführung eines Ethikkodex können gemeinsame Standards etabliert werden, die über die kulturellen Unterschiede der Stakeholder hinweg gültig sind – sodass auch in globalem Kontext, in dem unterschiedliche rechtliche, kulturelle und unternehmerische Normen gelten, Vertrauen für Interaktionen unter den Service-Providern gestärkt werden kann. Dadurch können für alle Stakeholder Umgebungen geschaffen werden, in denen Innovationen verprobt werden können.

Mit Blick auf das Problem im Zusammenhang mit Nachhaltigkeit befinden sich Cloud-Service-Provider in einer zwiegespaltenen Situation. Auf der einen Seite bietet Cloud-Computing, wie aufgezeigt, großes Potenzial für positive Veränderungen, insbesondere durch die Ablösung anderer Technologie-Setups. Jedoch führt die zunehmende Nutzung und bisher geringe Regulierung im Gegensatz zu anderen Industrien dazu, dass die Cloud-Computing-Branche möglicherweise zu höheren Kohlenstoffemissionserzeugungswerten anwachsen könnte als die Automobil-, Luftfahrt-, und Energiewirtschaft zusammen *[6]*. Daher ist die Betrachtung und Erarbeitung ethischer Standards für diese Fragen auch für die technische Weiterentwicklung von fundamentaler Relevanz.

Auch wenn Cloud-Computing-Services zumeist Endnutzern weltweit (oft in bestimmtem Umfang auch kostenfrei) zur Verfügung gestellt werden, steht auf der anderen Seite eine Industrie, deren Subtypen oft in der Hand einer kleinen Gruppe von Service-Providern liegt. Dies kann für diejenigen, die neu in den Markt als Anbieter einsteigen wollen, zum Problem werden. Diesem Festgelegtsein und damit auch dem Verlieren der Wahl- und Entscheidungsfreiheit sollte sowohl mit ethischen als auch rechtlichen Antworten entgegengetreten werden.

Die rechtlichen Forderungen nach kartellrechtlicher Überprüfung von großen Technologieanbietern erhielten in jüngster Zeit große Aufmerksamkeit; Maßnahmen zur Verhinderung von Monopolisierung stehen bei vielen Regulatoren weit oben auf der Prioritätenliste. Darüber hinaus kommen in der Diskussion auch immer wieder Rufe nach Interoperabilität als grundlegender Standard oder als Norm auf, die auch in einem ethischen Rahmenwerk festgeschrieben werden könnten, um die Gefahr der Monopolisierung einzudämmen *[6]*.

Ethik für Cloud-Nutzer

Welche Verantwortung liegt bei Nutzern, die selbst zu Angebotsproduzenten werden können? Die Funktionalitäten der Technologie können von Nutzern nicht nur zum persönlichen Konsum genutzt werden, sondern sie ermöglichen auch, aus der Interaktion mit den Services selbst zu Wertschaffenden zu werden. Als die größte Gruppe der Stakeholder sollten die Nutzer deshalb und weil sie direkt Betroffene sind, nicht rein passiv an Ethikkodizes teilhaben, sondern selbst als Verantwortungsträger an diese gebunden werden.

In der ethischen Forschung gibt es erste Diskussion, ob (neben weiteren) Nüchternheit und Kontinuität Elemente eines Ethikkodex für Nutzer sein sollten, um Risiken zu minimieren, die vom Nutzer ausgehen *[8]*. Nüchternheit verstanden als die Fähigkeit, sich gegenüber einer ungeprüften Übernahme von versprochenen Vor- und Nachteilen von Cloud-Computing kritisch zu zeigen. Was wiederum einhergehen sollte mit der Informationseinholung von Experten zu Fragen wie Sicherheit und Privatsphäre.

Kontinuität im Aufbau von Wissen rund um Funktionalitäten und Standards könnte eine zweite Ethikkodexforderung sein, die Nutzer erinnert, dass Informationsdefizite selbstständig überwunden werden müssen.

Europäische Lösungsansätze

So viel zur Theorie. Auf europäischer Ebene gab es in den vergangenen Jahren wichtige Entwicklungen, um den genannten Risiken und ethischen Problemstellungen zu begegnen.

Erst kürzlich, am 19. Juli 2021, hat die Europäische Kommission eine Allianz zum Thema Cloud ins Leben gerufen: die Allianz für industrielle Daten, Edge und Cloud *[9]*.

Diese Allianz soll sowohl Investmentsynergien heben, indem sie die gemeinsame Entwicklung und Einführung von Cloud-Projekten unterstützt. Darüber hinaus hat sie das Ziel, eine gemeinsame Plattform zu schaffen, die Synergien mit dem Projekt des gemeinsamen europäischen Datenraums hebt. Auch als Koordinationshilfe zwischen der Europäischen Kommission und den zuständigen Institutionen der Mitgliedstaaten soll sie unterstützen. Und schließlich – und das ist für unsere ethischen Problemstellungen besonders interessant – eine Beratungsplattform gründen, die Vorbereitungen für ein EU-Cloud-Rulebook erarbeiten soll.

Mit dieser Allianz soll die Entwicklung eines unabhängigen europäischen Cloud-Stacks mit intelligenter Middleware vorangetrieben werden, um in Zukunft reibungslose Datenverarbeitung zwischen verschiedenen Computing-Technologien zu ermöglichen. Dies beinhaltet auch die Erarbeitung von Interoperabilitätsstandards für europäische Infrastrukturlandschaften.

Auch die Entwicklung von Open-Source-Multi-Cloud-Management-Services hat die Allianz sich auf die Fahnen geschrieben, um die Integration heterogener Cloud-Umgebungen von verschiedenen Service-Providern in einer anbieterungebundenen Art und Weise möglich zu machen. Auch State-of-the-Art-Cyberkontrollen und weitere Instrumente zur Unterstützung einer stabilen Sicherheitsarchitektur sollen entwickelt werden.

Schließlich steht auch die Förderung klimaneutraler, energieeffizienter und nachhaltiger Technologien als hohe Priorität in der Roadmap für die nächste Generation von Cloud-Angeboten auf der Liste. Dies steht im Einklang mit dem Ziel der Europäischen Union, als Vorreiter bei der Etablierung klimaneutraler Datencenter bis 2030 tätig zu werden *[10]*.

Ausblick

Initiativen, wie sie von Europa vorangetrieben werden, zeigen, dass der Diskurs über Ethik, Normen und Rechtsetzung rund um die Cloud-Technologie in vollem Gange ist. Es ist gut zu sehen, dass die Europäische Kommission für die Erarbeitung eines Rulebook die Zusammenarbeit und den Austausch mit Industrievertretern im Rahmen einer Allianz sucht. Denn die Mitgliedschaft an der Allianz und die Möglichkeit, an den gesetzten Zielen mitzuarbeiten, stehen öffentlichen wie privaten Einrichtungen offen, die bereits im Bereich Sicherheit, Interoperabilität, Ressourceneffizienz für Cloudkapazitäten oder in ähnlichen Gebieten tätig sind. Die genauen Kriterien sowie der Prozess für eine Bewerbung sind in der gemeinsamen Deklaration einsehbar *[11]*.

Man darf also erwarten und gespannt sein, welche Fortschritte in den kommenden Jahren hinsichtlich weiterer Klärung und der Harmonisierung von bestehenden Regelwerken auf die unterschiedlichen Stakeholder in Europa zukommen.

Quellen:

[1] Statista (2021): Cloud Computing – Statistics & Facts. Technology & Telecommunications, *https://www.statista.com/topics/1695/cloud-computing/*

[2] David Kolevski, Roba Abbas, Katina Michael, Mark Freeman (2020): Stakeholders in the cloud computing value-chain. A socio-technical review of data breach literature. IEEE International Symposium on Technology and Society, *https://ieeexplore.ieee.org/document/9462169*

[3] National Institute of Standards and Technology (2011): NIST Cloud Computing Reference Architecture. NIST Publications. *https://www.nist.gov/publications/nist-cloud-computing-reference-architecture*

[4] Job Timmermans, Veikko Ikonen, Engin Bozdag, Bernd Carsten Stahl (2021): The Ethics of Cloud Computing. A Conceptual Review.

[5] Charlotte Freitag, Bran Knowles, Gordon Blair, Adrian Friday (2020): The climate impact of ICT: A review of estimates, trends and regulations. *https://arxiv.org/abs/2102.02622*

[6] Brid Murphy, Marta Rocchi (2021): Ethics and Cloud Computing: Data Privacy and Trust in Cloud Computing, Palgrave Studies in Digital Business & Enabling Technologies. Palgrave Macmillan, Cham *https://link.springer.com/chapter/10.1007/978-3-030-54660-1_6*

[7] European Data Protection Supervisor (2015): Towards a new digital ethics. Data, dignity and technology. Übersetzung gemäß Seite 12, *https://edps.europa.eu/sites/edp/files/publication/15-09-11_data_ethics_en.pdf*

[8] Boudewijn De Bruin, Luciano Floridi (2017): The Ethics of Cloud Computing. Science and Engineering Ethics, *https://philpapers.org/rec/DEBTEO-11*

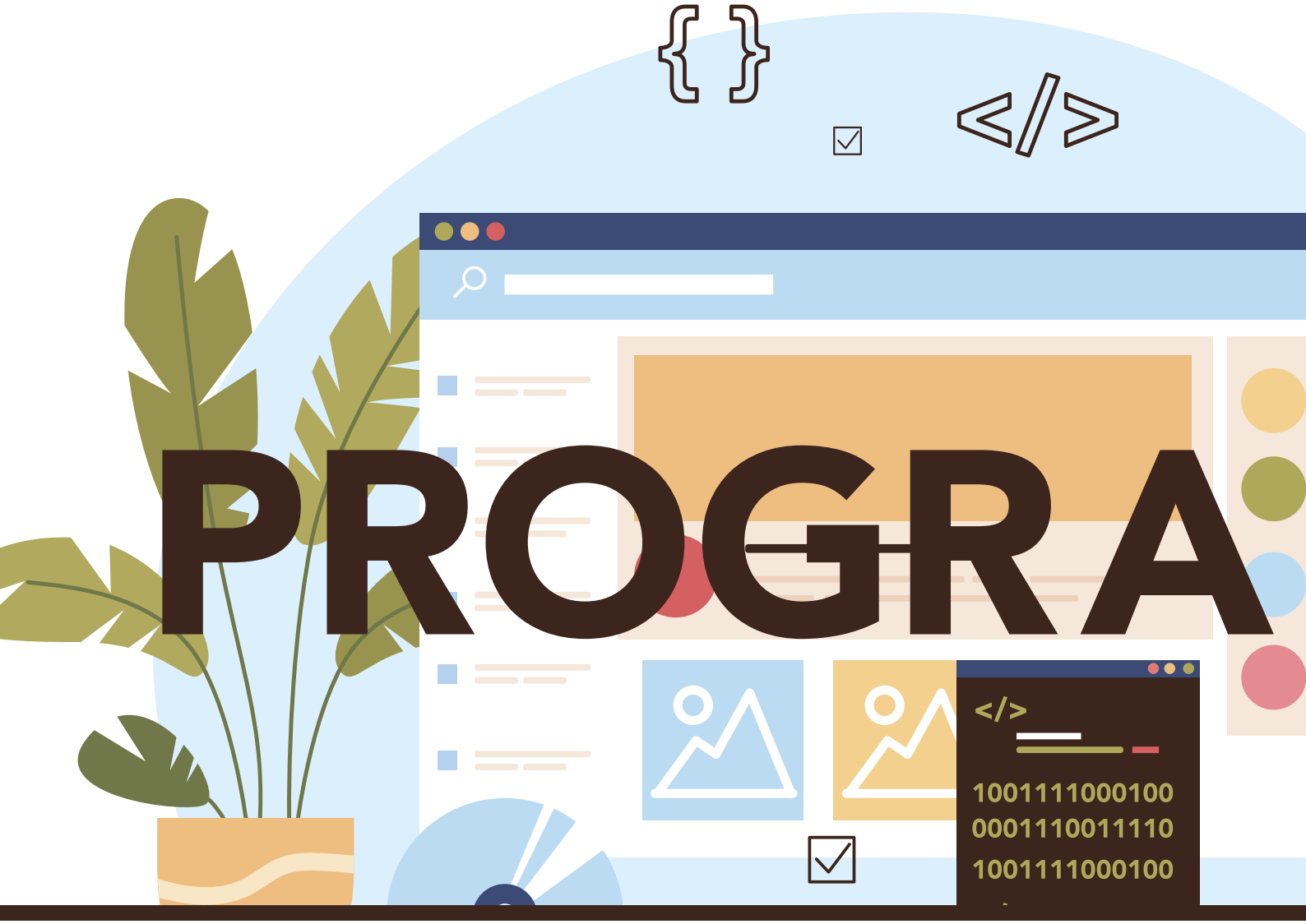
[9] Europäische Kommission, Pressemitteilung (19.07.2021): Commission launches alliances for microelectronics and cloud technologies, *https://digital-strategy.ec.europa.eu/en/news/commission-launches-alliances-microelectronics-and-cloud-technologies*

[10] Europäische Kommission (2021): Shaping Europe´s digital future. Cloud Computing, *https://digital-strategy.ec.europa.eu/en/policies/cloud-computing*

[11] Declaration of the European Alliance for Industrial Data, Edge and Cloud (2021), *https://digital-strategy.ec.europa.eu/en/policies/cloud-alliance#:~:text=The%20European%20Alliance%20for%20Industrial,generation%20edge%20and%20cloud%20technologies.&text=Cloud%20and%20edge%20technologies%20are,Internet%20of%20Things%2C%20and%205G.*

Corinna Wiedenmann
Freshfields Bruckhaus Deringer
corinna.wiedenmann@gmail.com

Corinna Wiedenmann studierte Philosophie und Wirtschaft, arbeitete mehrere Jahre im Innovationsmanagement und beschäftigt sich daher stark mit Themen rund um Ethik und Digitalisierung. Heute arbeitet sie als Business Development Managerin für die Kanzlei Freshfields Bruckhaus Deringer und ist dort für den europäischen Financial-Institution-Sektor verantwortlich.



Developer: Ein Job, der rockt – kein Grund unglücklich zu sein

Roland Golla

Wenn Programmierer/innen bessere Arbeitsbedingen fordern, wird das leider oft belächelt. Nirgendwo bekommt man sonst so schnell einen neuen Job in einem schicken Büro. Da reicht oft sogar nur ein Anruf. Mich persönlich haben der mentale Druck der Eigenverantwortung und mangelndes Selbstbewusstsein vor einigen Jahren in den buchstäblichen Wahnsinn getrieben. Dabei habe ich einfach mit ein paar falschen Arbeitgebern in Folge Pech gehabt. Das hätte fast mein gesamtes Leben zerstört. Mit einer Therapie über vier Jahre, viel Disziplin, einer tollen Familie und viel Glück bin ich heute zurück – und habe sogar meine alte Liebe zu meinem nerdigen Job wiedergefunden. Mit diesem Artikel möchte ich andere vor dem gleichen Schicksal bewahren, Auswege aufzeigen und vor allem Mut machen, eine fantastische Reise in einem Traumjob zu beginnen. Es gibt einfach keinen Grund, in unserem Job traurig zu sein!

Wir haben Respekt verdient – keiner darf uns drücken oder stressen

Im Jahr 2000 – und das ist über 20 Jahre her – gab es eine Zeit, in der man froh sein musste, Aufträge zu bekommen. Der Dotcom-Einbruch. Das ist heute völlig anders. Und doch reichen die Schatten dieser Zeit bis in unsere heutigen Köpfe hinein. Wir müssen stets dankbar für unsere Arbeit sein. Deshalb fehlt uns leider immer noch der Mut, laut Nein zu sagen. Wer von uns keine gute Arbeit abliefern darf, sich Sorgen um aktuelle Projekte in seiner Freizeit macht und keine Slack-Time bekommt, um sich frei zu entfalten, muss auch einfach mal kündigen. Nur Mut! Wahrscheinlich hat jeder schon am selben Abend einen neuen Job für mehr Geld. Aber das darf nicht das Ziel sein. Einfach mal zwei Monate nichts zu machen, ist auch toll – und den finanziellen Verlust holt man schnell wieder auf. Versprochen!

Persönliche Ziele setzen und erreichen: Die Karriere bist du

Viele unserer Kunden und die Leute dazwischen haben jede Relation zum tatsächlichen IT-Zeitalter verloren. Künstlich und mit enorm großer Anstrengung werden schlechte IT-Projekte am Leben gehalten. Da niemand mehr darin arbeiten möchte, wurde der Jobtitel des „Full-stack Developer" erfunden – dass dieser von vielen Entwickler/innen nicht besonders ernst genommen wird, scheint nicht aufzufallen.

Meine Interpretation für solche Stellen steht in erster Linie für Einsamkeit und eine komplette Verantwortung für alles. Als guter Entwickler weiß ich aber nur zu genau, dass es wichtig ist, tiefes Wissen in einer Domäne zu haben und das mit anderen ergänzend als Team anzuwenden. Halbwissen wird eben immer nur für durchschnittliche Ergebnisse reichen – und die füllen mich auf jeden Fall nicht mehr aus. Die Projekte rocken einfach nicht. Zudem war Nine to Five noch nie mein Ding.

Aufgrund dessen habe ich die freie Zeit nach meiner letzten Jobkündigung genutzt, um mich intensiv mit dem Sulu CMS *[1]* zu beschäftigen. Da bin ich mittlerweile richtig fit und sicher – ein großartiges Gefühl. In dem System bin ich längst kein User mehr, sondern kann hier schnell, präzise und sehr gut arbeiten, ohne googeln zu müssen. Ich bin sogar mittlerweile auch öfter als Open Source Contributor dabei.

Open-Source-Training – echtes Wissen von Profis für Ambitionierte

Wer bekommt für seine eigene berufliche Weiterbildung in der Freizeit schon professionell organisierte Events mit Verpflegung und Top-Speakern? Das ist ein enormes Privileg und echter Luxus. War es vor Corona bald zu viel und zu gut, wird es während der Lockdowns schmerzlich vermisst. Hier passiert aber auch schon viel remote.

Der persönliche Austausch auf Augenhöhe innerhalb der Community ist einfach wichtig. Das darf man nicht aus Gründen der Bequemlichkeit einfach links liegen lassen. Manchmal kommt es mir so vor, als würden manche von uns nicht hingehen, weil sie resignieren.

Warum sollten sie einen Einblick in eine schöne neue Welt wollen, die sie nie betreten dürfen? Es würde den bereits großen Frust nur verstärken. Aber genau dann muss man sich aufraffen und hingehen! Denn hier gibt es auch Hoffnung, Lösungen, gleichgesinnte Kontakte und Wege raus aus dem Legacy-Sumpf, die hier schon viele gegangen sind. Natürlich gibt es hier auch Kontakte zu guten Arbeitgebern, denn die schicken glückliche Entwickler auch gerne zu Community-Events. Also win, win, win!

„Don't stop until you are proud" – wir sind die Experten und treffen technologische Entscheidungen

Der renommierte Karrierecoach und früherer Konzernchef Martin Wehrle hat einmal das Buch „Ich arbeite in einem Irrenhaus" *[2]* herausgegeben. Im Wesentlichen geht es darin um katastrophale Management-Fehlentscheidungen. Ich habe daraus die Strategie „Sparen, koste es, was es wolle!" erlebt. Übertragen auf meine Welt bedeutet das, dass alle meine Verschläge in Bezug auf Softwarequalität und nachhaltige Entwicklung abgelehnt wurden. Das sei unnötiger Luxus und Ballast. Solche Projekte werden dann im Unterhalt extrem teuer.

Tatsächlich sind die persönlichen Folgen für mich unter anderem Kopfschmerzen, Existenzängste und Frust. Abgesehen davon kann ich „Legacy Fighter" nicht als USP- und Karriere-Silver-Bullet verkaufen. Das will ich auch überhaupt nicht. Es ist für mich einfach der blanke Horror und das Gegenteil von dem, was ich möchte. Jetzt habe ich nur einen Ausweg in der Selbständigkeit gesehen. Und ich möchte auch ein guter Arbeitgeber werden. Tatsächlich hätte ich mich einfach nur bei guten Arbeitgebern bewerben, diese besser recherchieren und befreundete Entwickler fragen müssen, statt auf falsche Versprechen in Vorstellungsgesprächen reinzufallen. Noch besser wäre es, direkt nach Arbeitgebern zu suchen, die sich mit Open Source engagieren und Entwickler für Contributions freistellen.

Leidenschaft und Spaß – dann ist es tatsächlich keine Arbeit mehr

Lieber 12 Stunden für mich als 8 Stunden für den anderen. Das ist definitiv eine wichtige und wahre Erkenntnis in meinem Leben. Auch, wenn ich mittlerweile sehr viel weniger Stunden arbeite. In seiner Biografie „Total Recall" *[3]* spricht auch Arnold Schwarzenegger davon, 16-stündige Arbeitstage aufgrund spannender Inhalte nicht als Arbeit empfunden zu haben. Zeit ist eben doch relativ.

Voraussetzung dafür ist ein leicht zugänglicher Quellcode, den man versteht und in dem man sich auskennt. Kein Chaos, Zauberwald, Monolith oder wie man das auch immer benennen möchte. Alles andere ist eine extrem hohe Belastung für den Kopf. Und man verliert auch bei komplexen Projekten das Vertrauen in den Code. Side-Effekte in Form von Bugs bestimmen hier die Tasks. Arbeit wird nicht mehr fertig, sondern muss ständig neu deployt und gefixt werden. Das reibt uns nervlich auf. Zeitlicher Druck entsteht. Das ist eine allgemeine Gefahr für das Projekt und vor allem unsere persönliche Gesundheit.

Daher müssen wir gemeinsam alle Dinge, die keinen Spaß machen, ablehnen und auch als Teams Nein sagen. Wenn wir das geschafft haben und nachhaltig Software entwickeln dürfen und können, ist alles deutlich besser. Darum dürfen wir Dinge nicht akzeptieren und müssen, wie die Jedi-Ritter, gegen die dunkle Bedrohung kämpfen. Gut gelaunt zur Arbeit und zufrieden und sorgenfrei raus. Denkt dran: Das Leben ist zu kurz für später!

Software Craftsmanship – wer glaubt, was zu sein, hat aufgehört, was zu werden

Wissen geben und Wissen nehmen. Open Source ist ein Commitment. Jedes Mal, wenn ich etwas für Open Source tue, lerne ich sehr viel und mein Gegenüber auch. Hier wird nur mit vereintem Wissen ein Projekt weitergebracht. Dinge werden nicht ins Projekt gebracht, weil sie laufen, sondern wenn alle Beteiligten, also Reviewer und Mitstreiter, damit einverstanden sind, die Dinge verstehen und für richtig und gut befinden. Das ist etwas wirklich Tolles.

Ich darf auch als gestandener Senior-Entwickler immer noch jede Frage wie ein neugieriges Kind stellen. Und ich bekomme auch gern neues Wissen vermittelt. Denn das ist alles im Sinne des Projekts. Es gibt hier keine Konkurrenz – nur Teamwork. Wenn ein Teammitglied besser wird, dann ist das gut fürs Team, und wenn ich besser werde, dann wird das auch anerkannt. Gamification ist hier auch ein wichtiges Stichwort. Es geht eben darum, offen und freundlich in die Welt und auf die Leute zuzugehen. Dann kann man einfach nur gewinnen!

Transparente Gehälter sind wichtig, damit wir nicht gedrückt werden können

In der Nahrungskette stehen Entwickler und gerade Entwicklerinnen oft leider noch ganz unten. Gerade die viel niedrigeren Gehälter in der IT für Frauen sind unfair und weit überholt. Mein letztes Vollzeit-Gehalt als PHP-Entwickler lag bei 65.000 Euro im Jahr ohne Personalverantwortung. Rückwirkend kann ich sagen, dass diese Zahl absolut nichts mit meiner Leistung oder meinem Lebenslauf zu tun hatte. Ganz im Gegenteil. Selbstbewusstsein und Jobwechsel haben mich dahin gebracht.

Die Folge war, dass Entwickler, obwohl sie besser im Thema und auch besser drauf waren, bei diesem Arbeitgeber weniger Geld bekommen haben als ich. Mein Gehalt war für die IT und im Vergleich zu anderen Branchen niedrig. Dahinter steckt ein Markt, in dem die Kunden die Preise bestimmen, weil unsere Chefs sich gegenseitig drücken. Das muss sich grundlegend ändern. Dafür müssen wir zusammenstehen und als Community besser werden. Uns sollte es erlaubt sein, Projekte und ihre Zustände oder Rahmenbedingungen abzulehnen. Keine Projekte mehr ohne Tests. Keine komplette technische Projektverantwortung auf einer oder wenigen Personen. Und natürlich müssen technische Meinungen und Rahmenbedingungen erfüllt und dürfen nicht vom Kunden bestimmt werden.

Work, Travel und willkommen in der ganzen Welt

Es ist schwer, Corona etwas Gutes abzugewinnen. Bei uns in der IT und auch in meinem beruflichen Umfeld hat es eine Menge ausgelöst. Sind wir doch einmal ehrlich zu uns selbst: Vorher war das Home-Office nicht gern gesehen und echte Remote-Work, also nahezu undenkbar, dass jemand aus einem entfernten Urlaubsland arbeitet. Denn was wäre bloß gewesen, wenn das auf einmal jeder eingefordert hätte? Und einige können das ja auch gar nicht machen. Vorgeschobene Gründe aufgrund fehlender Kontrolle.

Vertrauen und Wertschätzung haben dann auf einmal einen echten Boom erlebt. Die Welle ist so stark, dass sie große Veränderungen und spektakuläre Chancen bietet. Auf einmal ist es wirklich nicht mehr wichtig, ob man jetzt von Duisburg oder Portugal aus arbeitet. Richtig große Firmen, wie Mozilla, beschäftigen sich gerade mit den Problemen und Chancen der globalen Zeitzonen ihrer Mitarbeiter.

Kurz gesagt, man braucht nicht mehr eigene Kunden und exotische Arbeitgeber für seinen Lebensunterhalt zu finden. Durchaus konservative Arbeitgeber, sowohl Corporates als auch Agenturen, bieten Fulltime-Remote-Jobs an. Das ist etwas ganz Besonderes. Nicht jeder kann das und vielen bleibt es auch auf ewig verwehrt. Sie können diesen Traum nur träumen – wir können ihn tatsächlich leben.

Wir können viel bessere Arbeitsbedingungen fordern und fördern

Kurz zu meinem Hintergrund, den ich auf einem sehr emotionellen Talk auf den Code.Talks vor über 1.000 Entwickler/innen vorgestellt habe *[4]*. Das Video ist auf YouTube verfügbar. Ich hatte einen starken Nervenzusammenbruch. Dank einer guten Therapeutin, einer sehr guten Ehefrau, einer sofortigen Kündigung bei einer schlimmen Agentur und viel Glück ist mein Leben jetzt wieder normal.

Ansonsten möchte ich über die Konsequenzen nicht nachdenken. Mein Leben wäre auf jeden Fall komplett zerstört gewesen. Die Therapie endet in diesem Sommer nach 4,5 Jahren. Seitdem finde ich regelmäßig gute Arbeitgeber, begleite Events und berate extern Arbeitgeber, wie sie Arbeitsbedingungen nachhaltig machen können. Generell gilt: Nachhaltigkeit ist eine Investition, die sich nicht sofort rentiert. Wenn es jedoch nur um kurzfristige und möglichst billige Erfolge geht, setzt uns das stark unter Druck. Und das schadet uns. Schlechte Softwarequalität macht krank. Auch hier spielt uns der Fachkräftemangel wirklich gut in die Karten, denn wir können uns intern und extern für bessere Rahmenbedingungen für unsere Arbeit einsetzen und natürlich auch mal konstruktiv Nein sagen.

Das hat für mich etwas mit Programmierer-Ethik zu tun. In dem Kontext entwickelt sich gerade auch der Markt der Freelancer. Auch Zusammenschlüsse mehrerer Einzelgänger zu einer Gruppe sind in der Startup-Szene zu beobachten. Denn wir sind sicher, können immer wieder zurück und werden jede Woche mehr gebraucht als davor.

Das Leben ist zu kurz für später – wir leben alle nur einmal

Der Titel dieses Absatzes „Das Leben ist zu kurz für später" *[5]* kommt aus einem Buch, das ich von meiner Schwiegermutter geschenkt bekommen habe. In dem Moment, in dem ich diesen Absatz hier schreibe, ist es Sonntag, der 20. Juni, 15:00 Uhr, und ich warte auf meinen Anschlussflug von Madrid nach Teneriffa. Ich habe bisher keinen Rückflug gebucht. Das finde ich unglaublich spannend und es motiviert mich.

Mein guter Freund Mark ist vor zwei Jahren mit seiner Frau nach Teneriffa ausgewandert. Bei den beiden werde ich zu Gast sein. Zwei Wochen Urlaub genießen und danach einmal mindestens eine Woche von dort aus arbeiten. Meine Frau und Kinder kommen vielleicht nach. Aber ich surfe sonst auch immer zwei Wochen für mich allein. Sonst werde ich leider wahnsinnig. Eine Folge meiner nervlichen Krankheit. Die behält man leider ein Leben lang und ist immer von Rückfällen gefährdet. Und doch ist dieser Trip einfach völlig anders.

Eigentlich hätte ich jetzt warten müssen. Erst mal durchziehen. Mein kleines Startup TESTIFY erst mal fertig machen – expandieren, kooperieren, Marketing, Publikationen. Es heißt doch schließlich „selbst" und „ständig". Verantwortung übernehmen. Genau darum geht es auch im oben genannten Buch. Die Autorin und Journalistin Alexandra Reinwarth wagt darin einen spannenden Selbstversuch: Sie lebt so, als hätte sie nur noch 365 Tage Zeit dafür. Sie erläutert Listen von Prioritäten, Freunden, Handlungen und Gefühlen sehr persönlich. Im Hintergrund gibt es einiges Wissenschaftliches, aber auch Beobachtungen aus dem eigenen Umfeld. Warum viele Jura studieren, sich aus dem elterlichen Einfluss ein Leben lang nicht befreien können und am Ende ein Leben führen, das für sie einfach nicht lebenswert ist, aber es „so sein muss". Wir können das aber so machen, solange wir unseren Code am Ende der Woche abliefern. Um es mit den Worten von Eminem zu sagen "This opportunity comes once in a lifetime."

Fazit: Warum ich es liebe, Programmierer zu sein – ich habe gelernt, Nein zu sagen

„Echt du arbeitest in der IT – ihr kriegt ja auch nur den ganzen Tag auf die Fresse, oder?" hat mich mal eine wirklich nette Bekannte auf einer Party gefragt. Das war mir wirklich noch nie passiert. Eigentlich ist es immer cool, „Programmier" zu sein. Ja, die Leute träumen förmlich von unseren Leben – Arbeitsbedingungen, Wertschätzungen – und finden das Wissen und den Weg dahin mystisch und nerdy.

Sie kennen eben nicht die Wahrheit – das Dunkle und das Schlimme. Das, was passiert, wenn du dich ausbeuten lässt. Weil du nett bist, Angst hast und nicht Nein sagen kannst. Das kann ausgenutzt werden und wir können dabei auch zerbrechen. Weil wir mit unserem Herzen und unserer Leidenschaft abliefern und Dinge zum Leben erwecken. Es sind auch weit weniger als die Hälfte der Jobs in der IT echte Office Hell. Wechseln lohnt sich und ist immer ein Schritt in die richtige Richtung. Es setzt gegenüber den Verlassenen ein Zeichen. Das kann leider auch nicht jeder tun. Bei uns hingegen wird das sogar positiv als Erfahrung gewertet. Und wenn man gefragt wird, warum man den letzten Arbeitgeber verlassen hat, dann kann man das auch gern ehrlich beantworten. Eingestellt werdet ihr sowieso sonst wärt ihr nicht beim Gespräch.

Referenzen

[1] *https://sulu.io/*

[2] „Ich arbeite in einem Irrenhaus", Martin Wehrle (ISBN 978-3-548-06133-7)

[3] „Total Recall", Arnold Schwarzenegger (ISBN 978-3-453-64058-0)

[4] *https://blog.nevercodealone.de/code-talks-softwarequalitaet-gesundheit/*

[5] „Das Leben ist zu kurz für später" Alexandra Reinwarth (ISBN 978-3-86882-916-7)

Roland Golla

Roland Golla ist PHP-Trainer für Testing und Refactoring, hat TESTIFY – Agentur für Tests gegründet und setzt sich mit Never Code Alone für Softwarequalität und soziale Projekte ein.

Mitglieder des iJUG

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

iJUG Verbund

www.ijug.eu

Impressum

Java aktuell wird vom Interessenverbund der Java User Groups e.V. (iJUG) (Tempelhofer Weg 64, 12347 Berlin, *www.ijug.eu*) herausgegeben. Es ist das User-Magazin rund um die Programmiersprache Java im Raum Deutschland, Österreich und Schweiz. Es ist unabhängig von Oracle und vertritt weder direkt noch indirekt deren wirtschaftliche Interessen. Vielmehr vertritt es die Interessen der Anwender an den Themen rund um die Java-Produkte, fördert den Wissensaustausch zwischen den Lesern und informiert über neue Produkte und Technologien.

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

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

Redaktion:
Sitz: DOAG Dienstleistungen GmbH
ViSdP: Christian Luda
Redaktionsleitung: Lisa Damerow
Kontakt: redaktion@ijug.eu

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

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

Bildnachweis:
Titel: Bild © Siarhei
https://stock.adobe.com
S. 10 + 11: Bild © Alex
https://stock.adobe.com
S. 18: Bild © profit_image
https://stock.adobe.com
S. 22: Bild © pickup
https://stock.adobe.com
S. 27: Bild © Visual Generation
https://stock.adobe.com
S. 34 + 35: Bild © VectorMine
https://stock.adobe.com
S. 40 + 41: Bild © lembergvector
https://stock.adobe.com
S. 52 + 53: Bild © oatawa
https://stock.adobe.com
S. 55: Bild © drogatnev
https://de.123rf.com
S. 58 + 59: Bild © artinspiring
https://stock.adobe.com

Anzeigen:
DOAG Dienstleistungen GmbH
Kontakt: sponsoring@doag.org

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

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

Inserentenverzeichnis

JavaLand

2022

15. - 17. März

im Phantasialand
bei Köln

Early Bird bis
19.01.2022

www.javaland.eu

Präsentiert von: DOAG Heise Medien

Community Partner: iJUG Verbund