
Table of Contents

Klausurthemen

Introduction	1.1
Co-Economy: Wertschöpfung im digitalen Zeitalter	1.2
3 Cases	1.2.1
Die Digitalisierung der Welt	1.3
1 Industrie 4.0 oder das Industrial Internet of Things	1.3.1
3 Denken Sie in Produkten - erst danach in (IT-)Prozessen	1.3.2
Soft Skills	1.4
3 Learning	1.4.1
Gemeinwohlökonomie	1.5
1 Kurzanalyse	1.5.1
Zusammenfassung: Cloud	1.6
Einführung	1.6.1
Cloud im Detail	1.6.2
Sicherheit	1.6.3
Cloud Dienste am Beispiel	1.6.4
MS Cloud Design Patterns	1.7
Einleitung	1.7.1
Cache-Aside Pattern	1.7.2
Circuit Breaker Pattern	1.7.3
Competing Consumers Pattern	1.7.4
Compute Resource Consolidation Pattern	1.7.5
Command and Query Responsibility Segregation (CQRS) Pattern	1.7.6
Event Sourcing Pattern	1.7.7
External Configuration Store Pattern yannick kloss	1.7.8
Federated Identity Pattern	1.7.9
Gatekeeper Pattern	1.7.10
Health Endpoint Monitoring Pattern	1.7.11
Index Table Pattern	1.7.12

Leader Election Pattern	1.7.13
Materialized View Pattern	1.7.14
Pipes and Filters Pattern	1.7.15
Priority Queue Pattern	1.7.16
Queue-Based Load Leveling Pattern	1.7.17
Retry Pattern	1.7.18
Runtime Reconfiguration Pattern	1.7.19
Schedular Agent Supervisor Pattern christian holzberger	1.7.20
Sharding Pattern	1.7.21
Static Content Hosting Pattern	1.7.22
Throttling Pattern	1.7.23
Valet Key Pattern	1.7.24
Asynchronous Messaging Primer	1.7.25
Autoscaling Guidance	1.7.26
Soft Skills für Softwareentwickler	1.8
5 Konfliktmanagement	1.8.1
Soft Skills für IT-Berater	1.9
2 Kommunizieren und verstehen	1.9.1
10 Veränderungsmanagement im Überblick	1.9.2
11 Veränderungsmanagement konkret	1.9.3
SW Architecture for Developers Vol. 1	1.10
3 What is software architecture?	1.10.1

Projekte

Projekte	2.1
VR	2.2
Einführung	2.2.1
Einsatzgebiete	2.2.2
AR	2.3
Einführung	2.3.1
Grundlegende Technologien	2.3.2
Konzept	2.3.3
Zusammenfassung	2.3.4

Modell getriebene Systementwicklung	2.4
Einführung	2.4.1
Systems Engineering	2.4.2
Model Based Systems Engineering	2.4.3
Systems Modeling Language	2.4.4
Metamodeling in SysML / UML	2.4.5
Product Line Engineering	2.4.6
Model Transformation	2.4.7
Quellen	2.4.8
Anhang	2.4.9
Continous Software Engineering	2.5
Agile Vorgehensmodelle	2.5.1
Unterstützende Prozesse / Workflows	2.5.2
Unterstützende Tools	2.5.3
Quellen	2.5.4
AI	2.6
Übersicht Data Mining Algorithmen	2.6.1
Knowledge Discovery in Databases	2.6.2
Embedded Computing	2.7
IOT-Protokolle	2.7.1
MQTT	2.7.1.1
SmartHome	2.7.2
Einführung	2.7.2.1
SmartHome: Definitionen	2.7.2.2
Verschiedene Aspekte des Smart Home	2.7.2.3
Fazit	2.7.2.4
Bibliografie	2.7.2.5
Fullstack Development	2.8
NoSQL	2.8.1
Einführung	2.8.1.1
Spaltenorientierte DB	2.8.1.2
Dokumentenorientierte DB	2.8.1.3
Key-Value DB	2.8.1.4
Graphen DB	2.8.1.5

Objektorientierte DB	2.8.1.6
Objektrelationale DB	2.8.1.7
Fazit	2.8.1.8
Quellen	2.8.1.9
Single Page Application	2.8.2
Einführung	2.8.2.1
Entwurfsmuster	2.8.2.2
Quellen	2.8.2.3
Hybride App-Entwicklung	2.8.3
Unterschiede zwischen Web-Apps, Native Apps und Hybride Apps	2.8.3.1
Kommunikation in verteilten Systemen	2.8.4
Einführung	2.8.4.1
RPC	2.8.4.2
WebSocket	2.8.4.3
REST	2.8.4.4
GraphQL	2.8.4.5
Fazit	2.8.4.6
Quellen	2.8.4.7
Backend Entwicklung	2.8.5
Node.js Event-Loop	2.8.5.1
JWT vs. Session Cookie	2.8.5.2
Design	2.8.6
Einführung	2.8.6.1
Konzepte und Designansätze	2.8.6.2
Quellen	2.8.6.3
Software-Architektur	2.9

Spezielle Gebiete zum Software Engineering Script

Dies ist das gemeinsam erstellte Modul Script zu den Themen, die im Sommersemester 2017 an der FH-Bielefeld im Modul behandelt wurden.

Anleitung:

- die Texte werden mit Mardown erstellt
- pro Kapitel wird eine Datei erstellt und nicht eine Datei pro Person
- pro Buch existiert eine Datei, in der die wichtigsten Eckdaten stehen und evtl. eine kleine Einleitung oder Zusammenfassung
- daraus ergibt sich folgende Struktur pro Buch:
 - buchname.md
 - buchname (Ordner)
 - buchname/kapitelnummer_kapitelname.vorname_nachname
 - Beispiel:
 - co-economy_wertschoepfung_im_digitalen_zeitalter.md
 - co-economy_wertschoepfung_im_digitalen_zeitalter (Ordner)
 - co-
 - economy_wertschoepfung_im_digitalen_zeitalter/1_connectedness.fabian_lorenz.md
 - co-
 - economy_wertschoepfung_im_digitalen_zeitalter/2_colaboration.fabian_lorenz.md
 - co-
 - economy_wertschoepfung_im_digitalen_zeitalter/3_cases.lutz_winkelmann.md
 - ...
 - assets/bildname.png (o. *.jpg ...) (Bildordner)
- um das ganze einheitlich zu gestalten folgende Formatierung innerhalb der Kapitel:
 - Kapitelüberschrift in H1 (z. B. # 3 Cases)
 - Unterkapitel in H2 (z. B. ## 3.1 Die neue Sharing Economy)
 - Abschnittsüberschriften in H3 (z. B. ### Technologien und Innovationen)
 - Bilder einfügen (z. B. ![Alternativer Bildtext] (/assets/bildname.png) (Abbildung 1: Bildbeschriftung))
- es wird nur noch mit Github gearbeitet und nicht mehr auf dem GitBook direkt
- jeder arbeitet auf einem eigenen Branch (z. B. "florenz" und nicht master branch)

- dieser wird per pull request an github geschickt und vom Admin gemerget bzw. mit Kommentar abgelehnt, falls etwas nicht passt
- auf gitbook kann man sich die aktuelle Version ansehen (sobald der merge durchgeführt wurde)
<https://www.gitbook.com/book/fh-bielefeld-mif-sw-engineerin/script/details>
- um die Seitenzahl herauszufinden, falls jemand einen Markdown Editor verwendet, der das Dokument nicht als PDF anzeigen:
 - im Chrome-Browser rechte Maustaste
 - Drucken
 - nach PDF exportieren
- **Termin: Monday 12 Jun 2017 Inhaltliche Struktur des eigenen Themas im Gitbook (10:30 Uhr)**
 - aktuelle Version vom Masterbranchen holen
 - in die Datei summary.md im unteren Bereich "Projekte" die Struktur des Projektes nach folgendem Schema kopieren und Pull Request senden:
 - Projektname (z. B. [Fullstack Development](#))
 - individuelles Thema Student1 (z. B. [NoSQL](#))
 - Unterkapitel1 (z. B. [Einführung](#))
 - Unterkapitel2 (z. B. [ObjektbasierteDB](#))
 - Unterkapitel3 (z. B. [DokumentbasierteDB](#))
 - ...
 - individuelles Thema Student2
 - ...
 - am besten macht dies einer pro Gruppe, da alle in die selbe datei schreiben müssen, um Konflikte zu vermeiden

Co-Economy: Wertschöpfung im digitalen Zeitalter

Zusammenfassung des Buches:

Titel: Co-Economy: Wertschöpfung im digitalen Zeitalter - Netzwerke und agile Organisationsstrukturen erfolgreich nutzen

Verfasser: Claudia Pelzer, Nora Burgard

Verlag: Springer Gabler

Jahr: 2014

ISBN: 978-3-658-00954-0, 978-3-658-00955-7 (eBook)

Zusammenfassung von: Fabian Lorenz, Lutz Winkelmann

Einleitung

In der heutigen Zeit ist es dank der Digitalisierung für viele kleine Unternehmen einfacher, sich auf dem Markt gegen große Wettbewerber durchzusetzen.

Hauptgrund dafür sind Agilität, Flexibilität und Kreativität, die von großen Unternehmen, hervorgerufen durch ihre starren Strukturen, häufig nicht so gelebt werden können, wie es bei kleinen Firmen der Fall ist.

Um den Anschluss nicht zu verlieren, ist es also notwendig, diesen Strukturellen Wandel durchzuführen und die Vorteile dieser neuen "netzbasierten Wertschöpfung" zu nutzen.

3 Cases

3.1 Die neue Sharing Economy

Bei der Sharing Economy (auch Shareconomy oder Collaborative Consumption) rückt die Nutzung von Ressourcen in den Vordergrund. Um für Umweltschutz, Effizienz und Nachhaltigkeit zu sorgen wird immer öfter nicht der Besitz eines Guts sondern nur die Nutzung dieses angestrebt. Nur selten ist ein permanenter Zugriff auf eine Ressource, welche sich als Maschine, Örtlichkeit oder etwas abstraktes darstellen lässt, nötig. In diesem Fall soll durch das Teilen der Ressource für mehr Nachhaltigkeit und Kosteneffizienz gesorgt werden.

Der Gedanke der Shareconomy wurde bereits in den 70er Jahren gefordert, passte allerdings nicht zu den Gesellschaftsbedürfnissen, welche auf den Besitz von Gütern ausgelegt waren. Heutzutage hingegen wird bereits vieles in sozialen Netzwerken geteilt und somit ist der Gedanke des Teilens bereits in der Gesellschaft verankert.

In einer Statistik von 2010 aus den USA werden folgende Werte für geteilte Güter aufgezählt:

- Wohnraum (58%)
- Arbeitsraum (57%)
- Nahrungszubereitung (57%)
- Haushaltszubehör (53%)
- Kleidung (50%)

Zu den geteilten Gütern zählt auch der Kauf oder Verkauf von gebrauchten Gütern. Durch das Internet und die damit einhergehenden Möglichkeiten ist das Teilen von Ressourcen einfacher geworden. So werden öfter Immobilien, Verkehrsmittel oder selten genutzte Dinge wie Gartengeräte vermietet und geteilt.

Das momentan bekannteste Beispiel für das Verleihen von Gütern ist in der Autobranche zu finden. Vorallem in Städten wird ein Auto oftmals nichtmehr als Statussymbol, sondern als Belastung empfunden. Durch das Mieten und Teilen von Autos ist ein riesiger, neuer Markt entstanden, welcher sich Dank des Internets und mobiler Applikationen immer stärker verbreitet.

Der Grundgedanke hinter der Shareconomy ist der, ein Gut günstig zu leihen, anstatt es teuer zu kaufen. Dieser Gedanke findet sich auch immer öfter in großen Firmen und nicht nur bei Privatpersonen wieder. Um Resourcen wie zum Beispiel Speicherplatz zu sparen, werden eBooks, Musik und andere Medien digital oftmals zum Leihen angeboten. Der Konsument hat geringere Kosten für das Leihen und beansprucht nur für kurze Zeit seinen Speicherplatz.

Die Sorge, dass mit verliehenen Gütern nicht ordnungsgemäß umgegangen wird oder dass die Güter gestohlen werden können, ist momentan der stärkste Gegner der Shareconomy. Allerdings gab es ähnliche Sorgen bereits bei der Einführung des Online-Shoppings, welche sich schnell verflüchtigten.

Es gibt kritische Stimmen gegen die Shareconomy, welche behaupten, dass das Teilen ohne den eigentlichen Besitz eines Guts nicht möglich ist, wobei die Anzahl der Güter, welche sich im Umlauf befinden, allerdings verringert würde. Auch wird davon gesprochen, dass nicht die Nachhaltigkeit oder der Umweltschutz sondern lediglich der Kostenfaktor der Antrieb für die Shareconomy sei.

3.2 Crowdsourcing als neue Organisationsform

Der Begriff Crowdsourcing beschreibt den Zugriff auf ein gemeinschaftliches Wissen und personelle Ressourcen, um ein Problem zu lösen.

Unterkategorien des Crowdsourcing sind unter anderem:

- Crowdfunding (Finanzierung eines Projekts durch die Community)
- Co-Creation (Erschaffung eines Werkes durch die Community)
- Microworking (Teilaufgaben eines Projekts werden durch die Community erledigt)

Durch das Internet zählt die Crowdsourcing-Branche aktuell zu den am schnellsten wachsenden. Der Zugriff auf eine immer größer werdende Community, in welcher alle Arten von Fähigkeiten abgedeckt sind, wird durch Plattformen im Internet ermöglicht. Teams können Aufgaben zur Fertigstellung eines Projekts durch einfache Kommunikationsmöglichkeiten unabhängig von räumlichen und zeitlichen Gegebenheiten realisieren.

Die hohe Nachfrage an Crowdsourcing Dienstleistungen basiert auf folgenden sechs Grundprinzipien:

1. Die Anforderungen an technische und soziale Kompetenzen steigen.

2. Durch moderne Kommunikationstechnologien sind Personen nicht an Freizeit und Arbeitszeit gebunden.
3. Produkte werden durch Ideen, Innovationen und Informationen ersetzt.
4. Arbeitsszeiten und Löhne können flexibler an die Fähigkeiten der Personen angepasst werden.
5. Wertschöpfungsketten werden transparenter und der Konsument erlangt Eingriff in diese.
6. Durch Crowdfunding werden Produkte von Konsumenten vorfinanziert.

Im Gegensatz zu Outsourcing ist Crowdsourcing nicht an feste Arbeitszeiten und einen festen Arbeitsplatz gebunden. Außerdem sind die einzelnen Teammitglieder einfacher auszutauschen und die Bezahlung erfolgt nach dem Festpreisprinzip anstatt auf Stundenlohnbasis.

Die Motivation für Crowdsourcing erstreckt sich über viele Bereiche, wobei der materielle Ansatz ein sehr geringer ist. Ein Beispiel für Crowdsourcing ist Wikipedia, wobei Leute ihr Wissen teilen um selber von dem Wissen anderer profitieren zu können. Auch Wohltätigkeit, Spaß und Langeweile können motivieren. Geld ist nicht die beste Motivation für Crowdsourcing Projekte, da die Bezahlung oftmals nicht besonders hoch ist und die Teilnehmer dementsprechend auch nur einen Teil ihres Könnens einsetzen.

Für Unternehmen sind gibt es mehrere interessante Vorteile. Im Folgenden werden einige davon aufgelistet:

- Austausch von Wissen
- erweiterter Ideenpool
- Einbeziehung geographischer Besonderheiten
- Ressourcenersparnisse durch Spezialisierung

3.3 Beispiel: Vernetzte Arbeits- und Lebensräume

Vor allem in Städten ist der Effekt der Shareconomy deutlich zu spüren. Sei es bei dem Teilen von Büroflächen, bei Wohngemeinschaften oder bei der Autovermietung.

Der Begriff Co-Working beschreibt das Mieten eines Arbeitsplatzes mit Internetanschluss und einer Postadresse. Dieses Konzept ist vor allem bei Startups und Freelancern beliebt, da Kosten minimiert werden können, die Arbeiter nicht zu sehr abgelenkt werden wie wenn sie von zu Hause arbeiten würden und da der Austausch mit anderen Mieter möglich ist, wodurch ein professionelles Netzwerk aufgebaut werden kann.

Unter dem Begriff Co-Living wird das Teilen einer Wohnung verstanden. Vorallem für Städtetrips ist diese Hotelalternative besonders interessant, da Geld gespart werden kann und der Austausch zwischen verschiedenen Kulturen gefördert wird.

3.4 Beispiel: Vernetztes Wissen

Sogar für das Wissen an sich lassen sich Unterschiede zwischen alter und neuer Denkweise finden. Früher konnte Wissen als persönlicher Besitz angesehen werden. Oftmals blieben Arbeiter bis zur Rente nur in einer Firma, da sie sich in dieser Branche auskannten und der Umstieg in eine andere Firma eine Art neue Ausbildung mit sich ziehen würde. Heute ist dies zum Teil immernoch so allerdings wird das Wissen heutzutage vermehrt geteilt. Im Internet findet man leicht Vorlesungen von Professoren oder auch Berichte von Experten, wodurch es einfacher ist, am technologischen Wandel teilzuhaben.

Der Begriff Co-Learning stellt das Lernen geteilten Wissens in einer Gruppe unabhängig vom Ort dar. So ist es beispielsweise möglich, Nischenangebote zu erstellen, welche in einer Universität nur sehr wenige Studenten zum teilhaben bewegen können, allerdings weltweit eine große Anzahl an Interessenten finden.

Foren und Wikis dienen als Wissens- und Datensammlungen, welche auf einer technologischen Organisationskultur aufbauen, ein nötiges Maß an Vertrauen und Transparenz vermitteln und die Beitragileistenden in Kategorien einstufen.

Das Wissen, welches in diesen Sammlungen geteilt werden soll kann entweder explizit oder implizit sein.

Folgende Punkte treffen für das explizite Wissen zu:

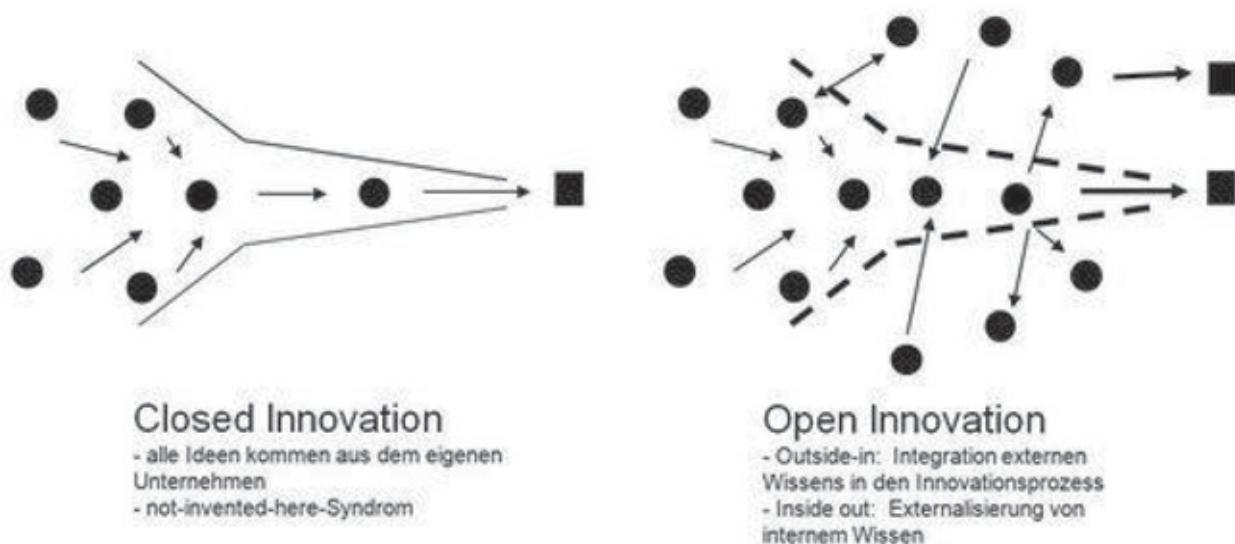
- Der Wissensinhaber kann die Information beschreiben
- Der Wissensempfänger muss Kenntnis über das Vorhandensein der Information haben

- Der Zugang zu den geteilten Informationen muss für den Wissensempfänger möglich sein
- Die Wissenssammlung muss strukturiert werden

Implizites Wissen hingegen kann vom Wissensinhaber nicht artikuliert werden. Hierunter fallen beispielsweise Fähigkeiten wie das Fahrradfahren. Der Austausch von implizitem Wissen kann somit nur in informellen Netzwerken stattfinden.

Auch innerhalb von Unternehmen ist ein Austausch von explizitem und implizitem Wissen oft nötig und kann durch ein Wiki abgewickelt werden, welches unter anderem Produktbeschreibungen, Anwenderhandbücher oder Allgemeines Markt- und Branchenwissen beinhalten kann. Hierdurch wird die Kommunikation oft vereinfacht, da der Zugang für alle betreffenden Personen möglich ist.

Durch Co-Creation und Open Innovation können auch externe Personen am Prozess der Entwicklung eines finalen Produkts in einem Unternehmen teilhaben. Impulse können hierbei von innen und von außen angestoßen und Informationen von innen nach außen weitergegeben werden. Dieser Prozess wird in der folgenden Abbildung dargestellt und wird als Crowdstorm (Brainstorming der Crowd -> Zusammensetzung von Brainstorming und Crowdsourcing) bezeichnet.



Je mehr sich die Unternehmen in diesem Prozess der Crowd öffnen, desto innovativer werden schlussendlich die Produkte, da die internen Mitarbeiter sich ansonsten in einer Art Blase befinden und sich vor externen Anreizen verschließen.

3.5 Beispiel: vernetzte Dienstleistungen

Durch die Digitalisierung konnte das Angebot und die Nachfrage von Dienstleistungen ins Internet verschoben werden und somit können Zeit- und Ortsunabhängig Spezialisten für gewisse Problemstellungen gefunden werden. Hierfür sind Online-Dienste wie Auftrags-Datenbanken oder das Micro-Payment unablässlich.

Es gibt verschiedene Tools, welche die ortsunabhängige Zusammenarbeit von Teams unterstützen wie beispielsweise Skype. Soll die Entwicklung von Projekten auch zeitunabhängig sein, so kann auf Dienste wie Google Drive zurückgegriffen werden. Diese Tools umfassen die Hauptaufgaben Online-Projektmanagement, Datentransfer und Kommunikation und werden oftmals gemeinsam eingesetzt.

Nicht nur Dienstleistungen werden vermehrt über das Internet angeboten, sondern auch Produktionsvorgänge werden immer mehr digitalisiert. Durch die Einführung des 3D-Drucks werden viele druckbare Objekte über das Internet erreichbar gemacht und können somit zu Hause erstellt werden. Natürlich ist auch das Teilen eines Objekts mit der Community möglich. Hierdurch werden Innovationen ins heimische Arbeitszimmer verlegt und somit werden Produktionsabläufe grundlegend beeinflusst.

Auch das Finanzwesen ist von der Co-Economy betroffen. Im Internet werden bereits viele Zahlungsmöglichkeiten wie Paypal angeboten. Auch Crowdfunding ist durch das Internet bekannt geworden und somit konnten bereits viele Innovationen umgesetzt werden.

Mittlerweile hat sich sogar digitales Geld in Form von Bitcoins im Finanzwesen durchgesetzt. Der Vorteil liegt hierbei darin, dass Geldbeträge banken- und zeitunabhängig den Besitzer tauschen können. Außerdem sind die Transaktionen sehr sicher und machen Rückbuchungen unmöglich.

Zur Zeit ist die Bitcoin allerdings noch sehr instabil im Kurs und der mögliche Diebstahl durch Hacker wird ebenfalls oft als ein hohes Sicherheitsrisiko angesehen.

Das Prinzip Crowdfunding ist eine Form des Crowdsourcings, in welcher Geld für Projekte gesammelt werden kann. Das Geld kommt hierbei von der Community welche automatisch einen gewissen Einfluss in die Entwicklung des Produkts bekommt.

Beim Crowdfunding kann durch die Analyse der zahlenden Community automatisch die Zielgruppe und der mögliche zu erschließende Markt erforscht werden.

Die zahlende Community kann das Geld entweder spenden, wodurch dieses nicht versteuert werden muss, oder allerdings eine materielle Gegenleistung erwarten.

Die Begriffe Crowdinvesting und Crowdloaning fallen unter den Begriff
Crowdfunding.

Das Crowdinvesting stellt hierbei die klassische Finanzierung eines Startups
dar, während beim Crowdloaning Geld an eine Privatperson in Form eines Kredits
fließt. Die Privatperson zahlt das erhaltene Geld dann verzinst an die
geldgebende Community zurück.

Die Digitalisierung der Welt

Zusammenfassung des Buches:

Titel: Die Digitalisierung der Welt - Wie das Industrielle Internet der Dinge aus Produkten Services macht

Verfasser: Peter Samulat

Verlag: Springer Gabler

Jahr: 2017

ISBN: 978-3-658-15510-0, 978-3-658-15511-7 (eBook)

Zusammenfassung von: Nils Kohlmeier, Tolga Aydemir, Yannick Kloss

1 Industrie 4.0 oder das “Industrial Internet of Things”

Industrie 4.0

Der Begriff “Industrie 4.0” bezeichnet die vierte industrielle Revolution. Bei der ersten industriellen Revolution, die Ende des 18. Jahrhunderts begann, wurden erstmals mechanische Produktionsanlagen eingesetzt, die Wasser- und Dampfkraft verwendeten. Dieser Abschnitt wird “Industrie 1.0” genannt. Die darauffolgende industrielle Revolution (“Industrie 2.0”) begann Anfang des 20. Jahrhunderts und führte die arbeitsteilige Massenproduktion ein. Maschinen wurden hierbei erstmals mit elektrischer Energie betrieben. Nachdem in den 1970er Jahren Elektronik und IT soweit waren, um in der Industrie eingesetzt zu verwenden, konnte die Produktion von Gütern (teil-)automatisiert werden. Dies wird als “Industrie 3.0” bezeichnet.

Heute soll die Industrie 3.0 zur Industrie 4.0 umgebaut werden, um die gesamte Wertschöpfungskette zu automatisieren, damit die Effizienz der Produktionsanlagen weiter gesteigert werden kann. Weiterhin soll die Qualität der Produkte verbessert und die Wettbewerbsfähigkeit der deutschen Industrie gesteigert werden. Um eine selbstorganisierende Produktion zu etablieren, ist es notwendig, dass Produktionsprozesse untereinander kommunizieren. So ist eine Unternehmen zum Beispiel in der Lage, flexibel auf Änderungen bei der Ressourcenverfügbarkeit zu reagieren.

Durch die automatisierte Organisation der Fertigung kann gut und effizient auf Kundenwünsche reagiert werden und individuelle Produkte können kostengünstig hergestellt werden.

Durch die Anbindung des Internet of Things (IoT), welches hauptsächlich die Gesamtheit der “intelligenten” elektronischen Geräte (Wearables, Haushaltsgeräte, Autos, Unterhaltungselektronik, ...) beim Endverbraucher/Kunden beschreibt, kann auf Aktionen von Endverbrauchern reagiert werden.

Die aktuelle Situation in Sachen “Industrie 4.0” sieht in Deutschland nicht gut aus. Seit einiger Zeit gehören Industrie-4.0-Produkte zwar zum Standardsortiment der Fabrikausrüster doch die Unternehmen sind bei der Umsetzung und Implementierung von Industrie-4.0-Prozessen sehr zögerlich. Einige Unternehmen setzen schon Industrie-4.0-Produkte ein, hauptsächlich aber nur mit Fokus auf Effizienzsteigerung. Die Forschung, die Arbeitsgruppen und die verschiedenen Gremien treiben dagegen die Standardisierung von Prozessen und Schnittstellen voran.

Bei der Umstellung der Industrie auf die Industrie 4.0 werden nicht nur unternehmensinterne Prozesse umgestellt, sondern auch nach außen hin werden Informationen und Dienstleistungen verknüpft. Somit kann mit den Kunden digital interagiert werden, was genutzt werden kann, um neue Produkte und Dienstleistungen zu entwickeln. Um Kosten zu sparen, werden (Teil-)Prozesse zum Kunden ausgelagert: Sie erledigen zum Teil Arbeiten, die früher das Unternehmen gemacht hat. Einige Beispiele sind zum Beispiel Selbstbedienungskassen im Supermarkt, bei denen der Kunde seine Artikel selbst einscannt, das Buchen und Einchecken von Flügen über Online-Portale sowie das Tätigen von Überweisungen bei der Bank, welches von Kunden selbst vorgenommen werden kann. Für die Umsetzung ist es notwendig, die Prozesse soweit zu vereinfachen, dass diese von jedem durchgeführt werden können.

Treiber der digitalen Transformation sind folgende: Digitalisierung, Vernetzung, Mobilität und Analytik. In allen Bereichen des Lebens werden immer mehr Dinge digitalisiert und in Software umgesetzt. So hat man zum Beispiel heute Smartphones, die mit den richtigen Apps viele verschiedene Aufgaben lösen können, für die man früher noch eigenständige Geräte brauchte. Dadurch werden aber auch neue Möglichkeiten entdeckt, die das Leben beeinflussen. Viele Dinge werden zudem heute erst im Computer designet und anschließend gefertigt. Kunden haben hiermit die Möglichkeit, ein Produkt noch vor der Fertigung zu betrachten.

Die zunehmende Vernetzung nicht nur von Menschen sondern auch von Geräten ermöglicht die Steuerung der Produktion in Echtzeit. Ein weiterer Punkt ist die Mobilität. So sind viele Menschen heute auch unterwegs online, um auch an Informationen zu kommen und austauschen zu können, wenn sie nicht zu Hause sind. Die Industrie 4.0 soll ermöglichen, auch unterwegs auf den Status der Produktion zugreifen zu können.

Die Aufgabe der Analytik ist, die gesammelten Daten zu analysieren und entsprechende Kenntnisse daraus zu gewinnen. Die Technik hierfür steht noch ganz am Anfang. Am Ende sollen die Daten Handlungsempfehlungen liefern. Ein vielversprechender Weg ist hierbei die Künstliche Intelligenz.

Auch das Thema Sicherheit ist wegen der Digitalisierung ein wichtiger Punkt bei der Industrie 4.0. Wenn immer mehr Geräte digitalisiert und vernetzt werden, müssen diese Geräte und Netze vor Angriffen geschützt werden.

Smarte Eingabegeräte

Wearables, also kleine tragbare elektronische Geräte, sind derzeit sehr beliebt und können gut in Unternehmen eingesetzt werden. Mögliche Anwendungsszenario für Wearables in Unternehmen sind zum Beispiel der Einsatz von "smarten" Handschuhen, die Monteure bei der Montage anweisen und Datenbrillen, die dem Träger wichtige Informationen einblendet.

Heutige Wearables enthalten immer mehr Sensoren, um möglichst viele Daten erfassen zu können. Auch das direkte Integrieren von Wearables in Kleidung ist derzeit sehr beliebt. Ein sehr wichtiger Faktor bei dem Erfolg der Wearables ist die Bedienbarkeit der Geräte.

Die Forschung zum Thema "Mensch-Maschinen-Kommunikation" ist sehr aktiv und es werden immer neue Eingabe- und Steuerungsmethoden entwickelt und alte ständig verbessert. So sind heute zum Beispiel schon Gestensteuerungen in Autos und Unterhaltungselektronik zu finden. Ein bekanntes Gerät ist unter anderem die Microsoft Kinect, welche die Positionen und Bewegungen von Personen mithilfe einer 3D-Kamera erkennt.

Auch beliebt sind Sprachassistenten, die per Sprache gesteuert werden und allerhand nützliche Informationen bereitstellen sowie Smart-Home-Geräte steuern. Beispiele für Sprachassistenten sind "Alexa" von Amazon und "Google Home" von Google. "Google Home" soll, um möglichst hilfreich zu sein, mit den verschiedenen Google-Diensten wie Google Mail und Google Maps verknüpft werden. Die Steuerung der Heimautomatisierung "Google Nest" mit Google Home ist (vorerst) nur in den USA verfügbar.
Die Smartphone-App "Google Assistant" verwendet zudem künstliche Intelligenz um gesprochene Wörter zu erkennen.

Neben den Sprachassistenten gibt es zudem noch die "Eye Tracker", die Augenbewegungen mithilfe von Kameras und Bildverarbeitungsalgorithmen erkennen und so eine Steuerung von Geräten über die Augen ermöglichen.

Einen anderen Steuerungsansatz untersucht das Projekt "Skin Track": Die eigene Haut soll als Touchscreen dienen. Die Technik "Thermal Touch" erkennt über Wärmebildkameras Flächen, die von einem Benutzer durch Anfassen erwärmt wurden. So können beliebige Gegenstände, die sich erwärmen lassen, zu Eingabegeräten werden.

Visualisierung

Es gibt zwei Visualisierungsarten: "Virtual Reality" (VR) und "Augmented Reality" (AR). Der Unterschied der beiden Visualisierungsarten liegt dabei an der Einbindung der Umgebung: Während bei der Augmented Reality die Umgebung mit zusätzlichen Informationen ergänzt wird bzw. reale mit virtuellen Elementen kombiniert werden, ist der Nutzer bei der Virtual Reality von der realen Umgebung komplett abgeschottet und vollständig in einer virtuellen Umgebung.

Schon Anfang der 1990er Jahre war Virtual Reality ein großes Thema. Heute helfen aber verbesserte Sensoren und eine höhere Rechenleistung dabei, Virtual Reality besser umzusetzen als vor 20 Jahren. Verschiedene empfindliche Sensoren ermöglichen es

aktuellen VR-Headsets, Kopfbewegungen genauestens zu erkennen und die Position des Benutzers zu bestimmen. Im Gegensatz zu Wearables haben VR-Applikationen andere Anforderungen an das Benutzerinterface: Die Benutzung sollte hauptsächlich mit den Händen erfolgen und so sehr intuitiv sein.

Bei der Augmented Reality wird die reale Umgebung durch virtuelle Elemente ergänzt. Die wird zum Beispiel bei den so genannten "Smart Mirrors" gemacht, die Informationen wie Nachrichten und das aktuelle Wetter direkt im Spiegel einblenden.

Neben den Smart Mirrors gibt es noch die Datenbrillen. Eine gutes Beispiel ist hierfür die "Google Glass", die in den Medien viel Aufmerksamkeit erregt hat. Die erste Version der Datenbrille wurde vom Markt genommen, aber Google arbeitet derzeit an einer "Enterprise Edition" der Brille, die zum Beispiel im Logistikbereich eingesetzt werden soll.

Die "HoloLens" von Microsoft dagegen unterstützt 3D-Projektionen, welche durch projizierte Lichtpunkte erzeugt werden.

Eine dritte Art von AR-Anwendungen sind Augmented Reality Browser, die als App auf einem Smartphone oder Tablet laufen und bestimmte Muster oder Bilder erkennen und dann entsprechende Informationen anzeigen. AR-Browser gibt es auch als "Location Based Services", welche anhand des aktuellen Standortes des Benutzers Informationen anzeigen.

Smarte Gebäude

In Smart Homes sind alle Geräte miteinander vernetzt und können automatisiert Umgebungsparameter wie etwa Temperatur und Licht steuern. So weiß das System beispielsweise, wann der Eigentümer nach Hause kommt und kann entsprechend die Temperatur in den Räumen anpassen, bevor der Eigentümer nach Hause kommt.

Neben den Smart Homes gibt es noch die sogenannten "Smart Rooms", welche von Einzelhändler eingesetzt werden, um die Bereiche im Laden zu erkennen, die von vielen Kunden aufgesucht werden. Dies wird unter anderem mithilfe von Wärmebildkameras realisiert.

Des Weiteren gibt es Bluetooth Beacons, welche sich über Bluetooth-Low-Energy (BLE) mit auf den Smartphones der Kunden installierten Laden-App verbinden und zum Beispiel Nachrichten über aktuelle Angebote senden.

Die Smart Factory ist eine Fabrik, in der alle Maschinen miteinander vernetzt sind und automatisiert alle notwendigen Arbeitsabläufe geplant und ausgeführt werden.

In intelligenten Städten ("Smart Cities") sollen Verkehr, Ver- und Entsorgung und Logistik automatisiert gesteuert werden. Ziel ist es, das Leben bequemer zu machen. Werden alle Vorgänge in einer Stadt automatisiert, gehören zum Beispiel Staus der Vergangenheit an.

3 Denken Sie in Produkten - erst danach in (IT-)Prozessen

Im Bereich der Digitalisierung stehen Führungskräfte vor neuen Herausforderungen: Es müssen neue Ausrichtungen von Unternehmensstrategien vorgenommen und neue Orientierungen vorgegeben werden. Es müssen Entscheidungen darüber getroffen werden, was zu tun ist. Dies ist ein komplexer Prozess, welcher von vielen Faktoren abhängt. Neue Geschäftsmodelle sind notwendig, um auch im internationalen Bereich marktfähig zu sein.

Zu verstehen ist, dass Technologien nicht im Vordergrund stehen: Es ist eher so, dass am Anfang Konzepte stehen, welche Dienstleistungen auf eine bessere Art und Weise erbringen. Dadurch werden Produkte und Dienstleistungen adaptiv und decken einen großen Kundennutzen ab. Es entstehen sogenannte **hybride Leistungsbündel**. Ein gutes Beispiel ist das Internet der Dinge: Produkte werden zu Services bzw. zu hybriden Leistungsbündeln.

Digitalisierung von Produkten und Dienstleistungen

Eine zentrale Fragestellung ist, wie sich traditionelle Produkte und Dienstleistungen Digitalisieren lassen. Im nachfolgenden werden zwei Muster für Geschäftsmodelle im Zeitalter der Digitalisierung vorgestellt:

- **Digitally Charged Product:** Im Internet der Dinge verbinden sich digitale Geschäftsmodellmuster mit solchen aus der nicht digitalen Welt zu einem hybriden Konstrukt. Zu den Kernbestandteilen von Digitally Charged Products gehören:
 - **Physical Freemium:** Zunächst kostenfreier digitaler Service zu einem gekauften Produkt um Interesse für kostenpflichtige Services zu wecken
 - **Digital Add-on:** Kostengünstiges physisches Gut mit zahlreichen kostenpflichtigen digitalen Services
 - **Digital Lock-in:** Nur Original-Komponenten sind kompatibel und keine Fälschungen
 - **Product as Point of Sales:** Physisches Produkt kann direkt den Verkauf eines weiteren Produkts einleiten
 - **Object Self Service:** Ausführung vom autonomen Bestellungen
 - **Remote Usage and Condition Monitoring:** Daten über Zustände und Umgebung übertragen

- **Sensor as a Service:** Sensordaten werden gesammelt, aufbereitet und zur Verfügung gestellt. Im Fokus stehen also die Daten selbst. Ein Beispiel ist die Firma Streetline: Sensoren werden auf Parkplätzen installiert um die Belegungen zu überwachen. Die ermittelten Daten werden anschließend verkauft.

Aspekte der Digitalisierung

Nachfolgend werden einige wichtige Aspekte der Digitalisierung beleuchtet, welche in der heutigen Zeit entscheidende Rollen einnehmen:

- **Digitale Verträge:** Digitale Verträge sind ein wichtiger Bestandteil der Digitalisierung. Digitale Kaufverträge führen beispielsweise selbst Aktionen aus, von der Abbuchung der Raten bis hin zur digitalen Sperrung des Produkts. Einige Vorteile von digitalen Verträgen sind unter anderem: geringere (bis gar keine) Transaktionskosten, beschleunigte Wirtschaft, Intermediäre (Banken, Börsen, Notare, ...) werden überflüssig.
- **Edison-Prinzip:** Ein bewährtes Prinzip für die Digitalisierung ist die Verwendung von vorhandenen Komponenten: Digitale Geschäftsmodelle werden durch das Zusammenfügen von bereits bekannten Technologien und Produkten entwickelt. Es sind somit nicht zwangsläufig neue Technologien o. ä. erforderlich. Oftmals ist die Verknüpfung und Vernetzung von bekannten Technologien erfolgreich.
- **Soziale Medien:** Der Einsatz von sozialen Medien wie Facebook, Instagramm, Snapchat, WhatsApp, YouTube, Twitter, Linkedin und Xing bringt Unternehmen heutzutage viele Vorteile in der externen sowie internen Nutzung: Verbesserung der Zusammenarbeit, bessere Nutzung des Mitarbeiterpotenzials, höhere Motivation und Zufriedenheit, Reduktion des Koordinierungsbedarfs, besseres Wissensmanagement, Verbesserung der Kundenkommunikation und des Unternehmensimage.
- **Customerization:** Aufgrund der hohen Transparenz durch die Digitalisierung ist der Kunde zum Mittelpunkt geworden. Die Konsequenzen für Unternehmen sind Vielfältig. Customerization hat an Bedeutung gewonnen: Ob Mitarbeiter oder Kunde, alle nehmen größeren Einfluss auf die Gestaltung von Informationssystemen, Produkten und Dienstleistungen.
- **Social Listening:** Laut einer Studie von Adobe hat sich der Beruf eines Marketingprofis stark gewandelt: Aufdringliches Marketing bzw. Push-Marketing existiert nicht mehr. Heutzutage geht es vor allem um aktives Zuhören bzw. Social Listening. Es geht darum zu wissen, worüber die Kunden reden, wo es Probleme gibt und wie das Unternehmen

diese lösen können. Unternehmen müssen folgendes tun: Großartige Kundenerlebnisse schaffen, auf viele Kanälen Kontakt zu Kunden herstellen, Kunden das Gefühl geben verstanden zu werden sowie Kundenverhalten und -bedürfnisse vorhersehen.

Digitalisierung ist die Zukunft

Die Akzeptanz digitaler Produkte ist hoch: Aus der Historie von Industrie 4.0 ist ein deutlicher Trend hin zu Produkten zu erkennen, welche auf Kundenwunsch hin gebaut wurden. Laut dem Fraunhofer IAO wollen Kunden von produzierende Firmen vermehrt Losgrößen mit einer großen Variantenvielfalt und kurzer Lieferzeit. Eine **effiziente Produktionsplanung, produktionsnahe IT- sowie geeignete ERP-Systeme, ein klares Lean Management und optimale Montageprozesse sind erforderlich**. Ein Beispiel hierfür ist das Konzept Storefactory von Adidas: Der Konzern will Produkte nach Kundenwunsch direkt in kleinen Fabriken in unmittelbarer Nähe von Geschäften fertigen.

Entsprechend werden Geschäftsprozesse auf Softwarestrukturen so angepasst, dass Services interoperabel automatisch miteinander vernetzt werden. Eine Verbindung von Verkaufsorten zu Produktionsstätten ermöglicht die Fertigung von individuellen Gütern. Unternehmenssoftware werden soweit adaptiert, dass diese in Produktionsketten integriert werden.

Unternehmen wie z. B. Uber, Spotify und car2go haben zudem neue IT-Strategien entwickelt, um gleichzeitig das Nutzungsverhalten von Kunden zu studieren damit mit diesem Wissen neue Produkte entwickelt werden können, während bestimmte Services angeboten werden.

Im Kern digitaler Geschäftsmodelle geht es um Daten, um den Erkenntnisgewinn mit dem Ziel der Transparenz über die eigenen Prozesse und Produkte, über das Nutzungs- und Kundenverhalten sowie über Kundenwünsche und -reklamationen.

Flexibilität in der technologischen Ausstattung ist entscheidend. Es ist wichtig, dass diese homogen und skalierbar sind. Die Kosten sollten im Hinblick auf den Betrieb proportional zur Last sein. Cloud-Lösungen spielen hierbei eine große Rolle. Geschwindigkeit ist entscheidend: Neue Technologien und Innovationen können den Markt schnell ändern. Profitable Geschäftsmodelle können sich innerhalb von kürzester Zeit ändern.

Soft Skills

Zusammenfassung des Buches:

Titel: Soft Skills - The software developer's life manual

Verfasser: John Z. Sonmez

Verlag: Manning

Jahr: 2014

ISBN: 978-1617292392

Zusammenfassung von: Gamze Soylev Oektem, Justin Jagieniak, Marvin Schirrmacher,
Daniel Beneker, Tim Jastrzembski

3 Lernen

Als Softwareentwickler arbeitet man in der Regel mit einer großen Anzahl an Technologien zusammen. Technologien, die heute aktuell sind, sind es morgen eventuell nicht mehr. Um auf dem neusten Stand zu bleiben, muss man sich daher ständig in neue Technologien einlesen.

Einer der wichtigsten Soft Skills ist daher, sich selbst etwas beizubringen. Doch wie lernen wir etwas am effektivsten? Es ist ein Mythos, dass wir alle auf verschiedenen Arten lernen. Wir lernen alle am besten, indem wir etwas ausprobieren oder es jemandem erklären und wir tendieren dazu einfacher zu lernen, wenn es uns interessiert. Man kann noch so viele Bücher übers Fahrradfahren lesen und noch so viele Videos schauen, wenn man es das erste Mal ausprobiert, wird es nicht sofort funktionieren. Trotzdem greifen viele Softwareentwickler zu einem Buch, um sich in eine neue Programmiersprache einzulesen. Anstatt dessen sollte man jedoch möglichst früh versuchen, etwas praktisch zu tun. Wir sind von Natur aus kreativ und neugierig. Nutzen wir diese Aspekte aus, können wir sowohl unsere Motivation, als auch unsere Lerngeschwindigkeit erhöhen.

Um eine neue Technologie zu erlernen, sind im Kern drei Punkte wichtig:

- Was benötigt man um anzufangen?
- Was kann ich grob damit tun?
- Was sind die Grundlagen? Dass heißt welche 20% muss ich lernen, um 80% meiner täglichen Aufgaben zu erledigen?

Die 10 Schritte

Dieses Kapitel soll anhand von zehn Schritten zeigen, wie wir diese Fragen beantworten können und es schaffen, möglichst schnell und effektiv etwas neues zu erlernen.

Schritt 1: Sich einen Überblick verschaffen.

Im ersten Schritt geht es darum, zu verstehen worum es grob geht. Dafür reicht in der Regel eine einfache Internetrecherche aus. Auch das Lesen von Einleitungen entsprechender Bücher kann hilfreich sein. Ziel ist es, die Größe des Themas zu bestimmen. Welche Unterthemen gehören beispielsweise zu dem Thema? In diesen Schritt sollte nicht zu viel Zeit investiert werden.

Schritt 2: Festlegen, was ich lernen will.

Man kann nicht alles lernen, daher sollte man den Fokus auf ein bestimmtes Thema richten. Möchte man beispielsweise etwas über digitale Fotografie lernen, so wäre ein Fokus alles über das Schießen von Porträt-Fotos zu lernen. Wichtig ist, dass man den Fokus auf ein einziges Thema richtet, denn wir können nicht mehrere Sachen auf einmal lernen.

Schritt 3: Das Lernziel festlegen.

Bezogen auf das Beispiel mit der digitalen Fotografie kann das bedeuten, alle Funktionen der eigenen Kamera beschrieben und nutzen zu können und erklären wann und warum man welche Funktion benutzt. Diese Lernziele sollten möglichst eindeutig sein. Denn so können wir später feststellen, ob wir dem Ziel näher kommen. Ein weiterer Vorteil ist, dass wir ein konkretes Ziel vor Augen haben, welches wir erreichen wollen.

Schritt 4: Quellen suchen.

Es ist wichtig, nicht nur mit einer Quelle zu lernen. Es gibt viele verschiedene Arten von Quellen wie Bücher, Videos, Blogs, andere Experten, Programmcode, Beispielprojekte oder Dokumentationen. In diesem Schritt sollten erstmal, ähnlich wie bei einem Brainstorming, alle Quellen, die in Frage kommen, zusammengetragen werden. Die Qualität der Quelle ist an dieser Stelle weniger relevant.

Schritt 5: Einen Lernplan erstellen.

Bei den meisten Themen bietet es sich an, bestimmte Unterthemen in einer bestimmten Reihenfolge zu lernen. Zum Erstellen dieses Planes kann man sich oft an den Inhaltsverzeichnissen der Bücher (aus Schritt 4) orientieren. Oder man schaut mit welcher Struktur andere das Thema erklären.

Schritt 6: Die Quellen filtern.

Viele Quellen werden sich thematisch überschneiden und meistens reicht auch die Zeit nicht aus, alle Quellen durchzuarbeiten. Deshalb ist es wichtig, die Quellen entsprechend zu filtern. Die ausgewählten Quellen sollten natürlich die im Lernplan ausgewählten Bereiche abdecken. In diesem Schritt sollte auch auf die Qualität der Quellen geachtet werden. Bei der Auswahl von Büchern kann es z.B. hilfreich sein, Amazon Bewertungen durchzulesen.

Die folgenden Schritte 7-10 sollten für jedes Modul aus dem Lernplan durchlaufen werden. Die Schritte folgen dem Prinzip „LDLT: learn, do, learn, teach“.

Schritt 7: Genug lernen, um anzufangen.

Wichtig ist, möglichst früh praktische Erfahrungen zu sammeln, denn wir lernen am besten, indem wir etwas tun. In diesem Schritt geht es darum, nur die grundlegenden Sachen zu lernen. Das kann beispielsweise das Durchlaufen eines „Hello-World-Beispiels“ sein oder das Einrichten der Entwicklungsumgebung. Eventuell reicht es auch schon, eine Kapitelzusammenfassung eines Buches zu lesen.

Schritt 8: Freies experimentieren.

Nutze deine Neugier und Kreativität, probiere etwas aus, bis du an einen Punkt kommst, an dem du nicht mehr weiterkommst. In diesem Schritt werden sich viele Fragen ergeben. Es kann hilfreich sein, diese aufzuschreiben.

Schritt 9: Genug lernen, um etwas sinnvolles zu tun.

Die Fragen, die sich im achten Schritt ergeben haben, sollen in diesem Schritt beantwortet werden. An dieser Stelle sollen die Quellen aus Schritt 4 intensiv genutzt werden. Der Focus sollte aber immer darauf liegen, die Fragen zu beantworten. Dass bedeutet eventuell nur einzelne Kapitel eines Buches anlesen, in denen man die Antwort auf eine Frage vermutet. Wichtig ist auch, zu prüfen ob man dem, in Schritt 3 definierten Ziel, näher kommt.

Schritt 10: Selbst erklären.

"Tell me and I forget. Teach me and I remember. Involve me and I learn." - Benjamin Franklin

Sobald wir versuchen das Gelernte jemand anderem zu erklären, werden wir merken, welche Themen von denen wir dachten, wir hätten sie verstanden, wir doch noch nicht verstanden haben. Es ist die beste Möglichkeit, das Gelernte zu überprüfen und Lücken zu füllen. Wichtig ist, das Wissen in eigene Worte zu fassen und das Ganze selbst zu Strukturieren. Möglich ist das beispielsweise, indem man ein YouTube Video erstellt, sich mit einem Freund oder Mitarbeiter unterhält, eine Präsentation erstellt oder Fragen in einem Forum beantwortet. Sobald wir selbst versuchen, etwas mit unseren eigenen Worten zu erklären, ordnen wir die unterschiedlichen Informationen in unserem Gehirn, so dass sie für uns Sinn ergeben. Erst dann können wir effektiv auf das Gelernte zurückgreifen.

Diese Schritte stellen sicherlich keine „magische Formel“ dar. Wenn man merkt, dass es so formal nicht funktioniert, sollte man die Schritte anpassen oder weglassen. Die Schritte an sich sind auch nicht wichtig, wichtig ist es, das Konzept dahinter zu verstehen. Nur dann kann man ein eigenes System entwickeln, um sich selbst effizient etwas beizubringen.

Mentor

Ein Mentor oder auch Trainer kann hilfreich sein, um neue Themen zu erlernen. Doch wie erkennt man einen geeigneten Mentor? Gute Mentoren sind meistens diejenigen, die die meisten Fehler durchlaufen haben. Allerdings muss der Mentor selbst das Thema nicht unbedingt beherrschen. Tiger Woods wird von jemandem trainiert, der selbst nicht so gut spielt wie er, aber ihm fallen Aspekte auf, die Tiger Woods nicht auffallen. Man sollte sich jemanden suchen, der bereits anderen geholfen hat, das zu erreichen, was man auch selbst erreichen möchte. Einen guten Mentor erkennt man auch oft daran, wie viele Personen er beeinflusst. Letztendlich muss man natürlich auch persönlich mit der Person zureckkommen. Doch wo findet man so eine Person? Es gibt Portale, dort kann man für verschiedene Themen Mentoren bzw. Trainer mieten, doch man sollte sich eher im eigenen Umfeld umschauen. Eventuell kann ein Freund, ein Familienmitglied, ein Freund eines Freundes, ein Arbeitskollege oder eventuell auf der eigene Chef als Mentor fungieren. Doch selbst wenn man einen Mentor findet, heißt das noch lange nicht, dass er einem auch hilft. Erfolgreiche Personen sind oft beschäftigt und haben daher wenig Zeit. Eine Möglichkeit ist daher, immer etwas im Austausch anzubieten. Das kann beispielsweise schon ein Mittagessen sein, welches man für den Arbeitskollegen übernimmt. Außerdem ist es wichtig, nicht beim ersten „Nein“ aufzugeben. Man darf an dieser Stelle nicht zu nett sein und sollte wiederholt nachhaken.

Andersrum betrachtet kann und sollte man auch selbst die Rolle eines Mentors einnehmen. Im Prinzip kann das jeder tun. Jeder weiß bereits etwas, was andere versuchen zu lernen. Als Mentor muss man, wie bereits erwähnt, nicht perfekt sein. Oft hilft es dem Lernenden schon, einen anderen Blickwinkel einzunehmen oder eine zweite Meinung zu geben. Vorteil der Mentor-Rolle ist, dass man dabei i.d.R. am meisten lernt. Dazu kommt, dass die Leute, denen man hilft sich oft an einen erinnern und einem später dafür an anderer Stelle helfen. Ein großes Problem ist jedoch, dass man irgendwann nicht mehr allen helfen kann, da auch Zeit für die eigenen Aufgaben bleiben muss. Dann ist es wichtig denen zu Helfen, die wirklich Lust haben etwas zu lernen und die entsprechende Motivation mitbringen.

Lehren

Lehren ist der beste Weg, etwas zu lernen und wahrscheinlich der einzige Weg, etwas im Detail zu verstehen. Doch wir fühlen uns meist sehr unwohl, wenn wir daran denken zu lehren. Oft liegt das nicht daran, dass wir nicht lehren bzw. erklären können, sondern daran, dass wir nicht selbstbewusst genug sind. Wir möchten i.d.R. nur die Themen lehren, in denen wir selbst Experten sind. Doch um ein Experte in einem Thema zu werden, müssen wir zuerst lehren - ein Teufelskreis. Der Trick ist, viele von uns lehren, ohne es selbst zu bemerken. Lehren bedeutet nicht nur vor Gruppe von Leuten zu stehen und Themen zu erklären. Es geht vor allem darum, das Wissen zu Teilen. Wir haben alle schon einem Arbeitskollegen oder einem Kommilitonen etwas erklärt. Auch das ist Lehren. Weiter kann

es hilfreich sein, einen eigenen Blog zu starten, Präsentationen im Unternehmen durchzuführen oder Videos bzw. Screencasts zu erstellen. Um Lehrer zu sein, braucht man keine Zertifikate und keinen Abschluss und man muss auch kein Experte sein. Wir alle sind bereits Lehrer.

Wissenslücken

Wir alle haben Wissenslücken und Schwächen, doch meistens fallen uns diese Lücken garnicht auf. Eine Möglichkeit Wissenslücken zu identifizieren, ist zu überlegen, wo man am meisten Zeit investiert. Meistens gibt es tägliche, wiederkehrende Aufgaben, die durch Wissenslücken verlangsamt werden. Ein Beispiel sind die Shortcuts der IDE. Wenn diese Shortcuts, die häufig benötigt werden, nicht bekannt sind, benötigt man für einfache Aufgaben wesentlich mehr Zeit. Eine weitere Möglichkeit um Wissenslücken aufzudecken, besteht darin, aktiv auf Verständnisprobleme zu achten und diese aufzulisten. Zusätzlich sollte man notieren, wie oft welches Verständnisproblem auftritt. Nicht jede Wissenslücke, die auftritt, muss unbedingt geschlossen werden. Aber anhand der Liste lassen sich Lücken finden, die besonders oft auftreten.

In einem zweiten Schritt geht es darum, die Wissenslücken zu schließen. Der schwerste Teil, nämlich das Identifizieren der Wissenslücken, ist bereits getan. Wichtig ist herauszufinden, was man konkret lernen muss. Es ist wenig hilfreich zu wissen, dass man beispielsweise schlecht in Physik ist. Doch wenn man weiß, dass man nich versteht, wie z.B. Federn funktionieren, lässt sich diese Lücke einfach schließen. Außerdem sollte man während eines Gespräches zeitnah nachfragen, wenn man etwas nicht versteht.

Die Gemeinwohl-Ökonomie

Zusammenfassung des Buches:

Titel: Die Gemeinwohl-Ökonomie - Eine demokratische Alternative wächst

Verfasser: Christian Felber

Verlag: Deuticke

Jahr: 2014

ISBN: 978-3-552-06291-7

Zusammenfassung von: Oliver Nagel, Jonathan Jansen und Sven Schirmer

1 Kurzanalyse

Werte bilden die Grundorientierung unseres Lebens. Sie bilden einen Leitstern in unserem Leben und prägen unsere zwischenmenschlichen Beziehungen. Zu diesen Werten gehören z.B.

Vertrauensbildung, Ehrlichkeit, Wertschätzung, Respekt, Kooperation, gegenseitige Hilfe und Teilen. Auch die Wirtschaft besitzt Werte, die jedoch bei genauerer Betrachtung konträr zu den zuvor genannten Werten sind. In der freien Marktwirtschaft gelten Prinzipien wie Gewinnstreben und Konkurrenz. Diese Prinzipien fördern jedoch Egoismus, Gier, Geiz, Neid, Rücksichtlosigkeit.

Die freie Marktwirtschaft bildet also einen weiteren Leitstern in unserem Leben. Da wir an den Werten unser Handeln orientieren, haben diese Werte fatale Folgen auf unsere Gesellschaft.

Die Gesellschaft kommt in einen Konflikt, da die Leitsterne in gegensätzliche Richtungen zeigen. Soll sich die Gesellschaft solidarisch und kooperativ verhalten oder die eigenen Vorteile vorzugsweise im Blick haben? Die Werte der Marktwirtschaft werden jedoch durch die Legislative in Form von Gesetzen und Regulieren oder durch Abkommen zwischen den Nationalstaaten weiter unterstützt.

Das Gemeinwohl in der Wirtschaft sollte durch Konkurrenz und durch die persönliche Gewinnmaximierung entstehen. Der Nationalökonom Adam Smith begründete dies vor 250 Jahren wörtlich mit:

„Nicht vom Wohlwollen des Metzgers, Bäckers, Brauers erwarten wir unsere tägliche Mahlzeit, sondern davon, dass sie ihre eigenen Interessen wahrnehmen.“ (Deuticke 2014, S. 19) Jedoch waren Unternehmen

vor 250 Jahren überwiegend klein, besaßen weniger Macht und agierten primär lokal. Oft waren die Unternehmer Gründer oder Eigentümer und bildeten mit Arbeitnehmer eine Personalunion.

In der heutigen Zeit sind jedoch immer mehr anonyme, global agierende Unternehmen zu finden. Durch die globale Aktivität erfahren diese Unternehmen mehr Konkurrenz. Die Unternehmen stehen

dadurch stärker im Wettbewerb hinsichtlich der Preisgestaltung und der Qualität ihrer Produkte. Die Konkurrenz sorgt auf der einen Seite für stärkere Leistungsanreize, jedoch hat sie

Auswirkungen auf die zwischenmenschlichen Beziehungen. Das oberste Ziel ist den eigenen Vorteil anzustreben und gegeneinander zu agieren. Das Übervorteilen wird so zur Normalität. Obwohl

die Würde der höchste aller Werte und im Grundgesetz verankert ist, sorgt die Konkurrenz dafür, dass wir Menschen nicht gleichwertig behandeln. Der Begriff Würde steht für den

"gleichen, bedingungslosen, unveräußerlichen Wert aller Menschen" (Deuticke 2014, S. 21). Daraus resultiert die Gleichheit aller Menschen. In der freien Marktwirtschaft ist jedoch üblich andere Menschen zu instrumentalisieren und übervorteilen und somit die Würde des Einzelnen zu verletzen. Wenn der eigene Vorteil unser höchstes Ziel ist, werden wir zwangsläufig Mittel für unsere Zwecke benutzen und andere übervorteilen.

Die freie Marktwirtschaft schränkt die Freiheit der Teilnehmer ein, da z.B. bei einem Tauschgeschäft eine Partei stärker abhängig ist wie die andere Partei. Derartige Tauschgeschäfte sind z.B. das Einkaufen von Nahrungsmitteln, das Anmieten einer Wohnung oder die Aufnahme eines Kredits. Dies hat zur Folge, dass z.B. ein Weltkonzern stärkeren Einfluss auf die Bedingungen eines

Liefervertrags hat als der Zulieferer. Das Ausnutzen dieser Macht sorgt erst dafür, dass die freie Marktwirtschaft effizient wird. Jedoch kann eine freie Marktwirtschaft, die durch Gewinnmaximierung und Konkurrenz gekennzeichnet ist, nicht als frei bezeichnet werden. Die ständige Angst, dass jemand von dem Nächsten übervorteilt werden kann, zerstört systematisch das Vertrauen.

Jedoch ist Vertrauen notwendig, um die Gesellschaft zusammen zu halten.

Der Wirtschaftsnobelpreisträger Friedrich August von Hayek schreibt: „Wettbewerb stellt in den meisten Fällen die effizienteste Methode dar, die wir kennen“ (Deuticke 2014, S. 24). Jedoch gibt es keine Studie, die das beweist.

Allerdings gibt es viele Studien, die untersuchen, ob Wettbewerb stärker motiviert als jede andere Methode. Eine große Mehrheit von 87% ist zu dem Entschluss gekommen, dass nicht Wettbewerb, sondern

Kooperation die effizienteste Methode ist. Anders als bei Wettbewerb motiviert Kooperation über gelingende Beziehungen, Anerkennung, Wertschätzung und gemeinsame Zielerreichung. Wettbewerb motiviert über Angst,

da viele um ihren Job, ihr Einkommen oder Status fürchten. Ein weiterer Motivationsfaktor von Wettbewerb ist die Siegeslust, also den Wunsch besser zu sein als jemand anders. Aus psychologischer Sicht spricht man

bei Menschen, die ihren Selbstwert darüber definieren, dass sie sich besser fühlen, wenn es anderen schlechter geht, von pathologischen Narzissmus. Wenn es jedoch mein Ziel ist, gute Leistungen zu erbringen

ohne das mich die Leistungen des anderen kümmern, dann brauche ich den Wettbewerb nicht. Der Wettbewerb ist aber notwendig, damit die Menschen Leistungsanreise erhalten und somit motiviert sind.

Grundsätzlich kann man zwischen der intrinsischen und der extrinsischen Motivation unterscheiden. Die intrinsische Motivation kommt von innen und wirkt stärker als die extrinsische Motivation (z.B. Wettbewerb).

Durch diese Art der Motivation entsteht die Leistung z.B. durch die persönliche Leidenschaft für eine Sache. Eine effiziente Marktwirtschaft sollte also auf einer intrinsischen Motivation aufbauen.

Durch das Verfolgen der eigenen Interessen als höchstens Ziel hat folgende Auswirkungen auf die Marktwirtschaft:

- Auf Grund des Wachstumszwangs entstehen zunehmend Großkonzerne („Global Player“), die ihre Machtposition ausspielen und Konkurrenten aufkaufen.
- Wenn im Markt nur wenige Konkurrenten vorhanden sind, werden strategische Kooperationen eingegangen, deren Ausprägung zum Teil in Form von Kartellen zu erkennen sind, da dies noch effizienter ist.
- Durch verbesserte Standortbedingungen versuchen Staaten Unternehmen anzulocken, um die Bedingungen für Gewinnmaximierung zu verbessern. Dazu zählen z.B. Lohn-, Sozial-, Steuer- und Umweltdumping.
- Die Preisgestaltung orientiert sich an der Angebot- und Nachfragemacht und spiegelt die Interessen des Mächtigen wieder.
- Je globaler der freie Wettbewerb ist, desto größer ist das Machtgefälle zwischen den Marktbeteiligen und führt zu Ungleichheiten und einer Kluft zwischen Arm und Reich.
- Das primäre Ziel des Kapitalismus ist nicht Befriedigung der Grundbedürfnisse, sondern die Vergrößerung des Kapitals. Es werden strategisch neue Bedürfnisse geweckt, hinter denen eine höhere Kaufkraft steht.
- Der Umweltschutz wird vernachlässigt, da er nicht zu der Vermehrung des Kapitals beiträgt.
- Die Anhäufung von materiellen Werten rückt in den Vordergrund und unterwirft andere Werte wie z.B. Beziehungs- und Umweltqualität. Der Konsumzwang wird zur Kaufsucht.
- Die Wirtschaft wird geprägt von Egoismus, da sie diesen durch Konkurrenzverhalten (z.B. Karriere) belohnt. Dieser Egoismus färbt auf andere Bereiche wie Politik und Medien als auch auf unsere zwischenmenschlichen Beziehungen ab.
- Die Demokratie wird schrittweise ausgeschaltet, da Wirtschaftakteure durch Lobbying, Medienbesitz oder Parteifinanzierung ihrer Interessen durchsetzen.

Zusammenfassung: Cloud

Zusammenfassung der Bücher:

Titel: Cloud Computing als neue Herausforderung für Management und IT

Verfasser: Martin Reti, Michael Pauly, Gerald Münzl

Verlag: Springer

Jahr: 2015

ISBN: 9783662458310

Titel: Cloud Computing Basics

Verfasser: S. Srinivasan

Verlag: Springer

Jahr: 2014

ISBN: 978-1-4614-7698-6, 978-1-4614-7699-3(eBook)

Titel: Cloud Computing Web-basierte dynamische IT-Services

Verfasser: Christian Baun, Marcel Kunze, Jens Nimis, Stefan Tai

Verlag: Springer

Jahr: 2011

ISBN: 978-3-642-18435-2, 978-3-642-18436-9(eBook)

Titel: Requirements Engineering for Service and Cloud Computing

Verfasser: Muthu Ramachandran, Zaigham Mahmood

Verlag: Springer

Jahr: 2017

ISBN: 978-3-319-51309-6, 978-3-319-51310-2(eBook)

Zusammenfassung von: Alexander Schwietert, Timo Rolfsmeier

Einführung

Definition

Zum jetzigen Stand gibt es keine eine einheitliche Definition des Cloud-Begriffes, noch gibt es standardisierte Gütesiegel, welches einen Rahmen für die erforderlichen Eigenschaften eines Cloud Services schaffen könnte. Der grundlegende Gedanke einer Cloud ist es, dass Ressourcen nicht mehr besessen werden, sondern temporär in Form eines Services über ein Kommunikationsnetz genutzt werden.

Viele Anbieter schreiben Cloud jedoch lediglich im Zuge des Hypes, als Marketinginstrument, in ihr Portfolio. Das klassische Outsourcing ist per se jedoch nicht mit einem Clouddienstleister gleichzusetzen.

BITKOM bietet folgende Definition an.

Cloud Computing ist eine Form der Bereitstellung von gemeinsam nutzbaren und flexibel skalierbaren IT-Leistungen durch nicht fest zugeordnete IT-Ressourcen über Netze. Idealtypische Merkmale sind die Bereitstellung in Echtzeit als Self Service auf Basis von Internet-Technologien und die Abrechnung nach Nutzung.

Eigenschaften einer Cloud

Geschäftsbezogene Merkmale	Technikbezogene Merkmale
Bereitstellung und Nutzung von IT Ressourcen als Service	Flexible Bereitstellung skalierbarer IT Ressourcen
Schnelle und flexible Ressourcen-Verfügbarkeit / On-Demand	Mandantenfähige, gemeinsam nutzbare Infrastruktur (z.B. eine Installation einer Software die mehrere Kunden mit eigenen Accounts nutzen können)
Ressourcen-Zuordnung durch den Kunden Steuerbar (Self-Service)	Hohe Automatisierung / Standardisierung des Angebots
Kurze Vertragsbindung	Logisch zentralisierte, virtualisierte IT Infrastruktur
Keine oder nur minimale Vorabinvestition für den Kunden	Zugriff via Browser oder Apps
Variable Kosten nach Nutzung der Ressourcen	Vollständige, lastabhängige Skalierbarkeit
	Messbarkeit des IT-Verbrauchs

Chancen und Risiken

Die Chancen

Grade am Konsumentenmarkt gibt es bereits viele bekannte Beispiele, wo skalierbare Ressourcen aus der Cloud eine große Rolle spielen. Unternehmen wie Amazon, Facebook oder Google können dadurch dynamisch auf die wechselnde Auslastung ihrer Infrastruktur reagieren. Auch Startups, sowie kleinere und mittelständische Unternehmen können von der Cloud stark profitieren, da sie das Investitionsrisiko für neue Unternehmungen senkt und initiale Kosten einer IT Infrastruktur wegfallen. Folgende Punkte konkretisieren die potenziellen Vorteile der Cloudnutzung.

- Durch Auslagerung in IT Services können Unternehmen sich auf ihr Kerngeschäft konzentrieren und erreichen eine geringere Fertigungstiefe in der Produktion. So kann die Qualität der Kernprozesse sowie das Wachstum gesteigert werden.
- Statt fixer Kosten werden, transparent für den Nutzer von Cloud Services, so viele Ressourcen zur Verfügung gestellt wie benötigt und entsprechend abgerechnet. Dadurch sinkt das gebundene Kapital, sowie initiale Investitionskosten die für den Aufbau einer eigenen IT Infrastruktur nötig wären.
- Risiken bezüglich Personal sowie technischen und Sicherheitsaspekten werden auf den Dienstleister übertragen.
- Die Expertise des Dienstleisters wäre nur mit erheblich höheren Eigenaufwand zu

erreichen

- Standardisierte Cloudangebote beschleunigen bereits heute die Einführung neuer Geschäftsmodelle und Produkte, da neue Modelle schnell und flexibel eingeführt und bei einem Fehlschlag auch schnell wieder vom Markt genommen werden können (kürzere Time To Market)
- Im Bezug auf die Green IT resultiert eine, nur nach Bedarf genutzte, Infrastruktur in geringeren Stromverbrauch

Die Risiken

Unternehmen haben auf der anderen Seite auch hohe Ansprüche, die die Cloud erfüllen muss. Ein Dienst muss flexibel, sicher, skalierbar, zuverlässig, hochverfügbar, kostengünstig, effizient und transparent nutzbar sein.

Das Risiko beginnt bereits mit der korrekten Einführung von Cloud Services, für die ein Unternehmen eine einheitliche Strategie braucht, damit keine Insellösung für einzelne Fachbereiche oder IT-Abteilungen entstehen.

Risiken bei unkontrollierter ungesteuerte Nutzung von Cloud-Diensten sind Folgende.

- Verlust von Unternehmensgeheimnissen oder Missbrauch von vertraulichen Kundendaten
- Compliance-Verstöße, die strafrechtlichen Konsequenzen nach sich ziehen. Können finanziell schädigen und den Ruf des Unternehmens beeinträchtigen
- Etablierung von verschiedenen Standards in verschiedenen Unternehmensbereichen kann große Integrationsaufwände in der IT-Landschaft eines Unternehmens verursachen
- Unkontrollierter Einsatz von Cloud kann Effizienzprojekte zur Konsolidierung von Hardware und Applikationen untergraben

Ein Unternehmen muss sich daher genau überlegen welche Rolle Cloud Computing im Unternehmen spielen soll, welche Daten und Prozesse sich auslagern lassen und welche Anbieter für die eigenen Ansprüche in Frage kommen.

Auch für zukünftige Entwicklungen, wie beispielsweise die Frage, ob eigenen Applikationen zukünftig direkt cloud-ready sein sollen, müssen frühzeitig betrachtet werden und in die Strategie des Unternehmens einfließen.

Integration & Strategie im Unternehmen

Auch wenn die IT-Abteilung eine tragende Rolle bei der Integration von Cloud Services in einem Unternehmen einnimmt, so sollten sie nicht alleine über die Cloud Strategie entscheiden. Diese muss auf die Unternehmensstrategie abgestimmt sein und die

entsprechenden Geschäftsziele abbilden, wie beispielsweise Umsatz, Kundenzufriedenheit oder Produktentwicklungszeiten.

Nachdem die oberste Ebene, in Form der Geschäftsziele, abgesteckt wurde, kann auf die einzelnen Geschäftsprozesse geschaut werden. Welche Prozesse können von der Cloud profitieren, können automatisiert werden, müssen dynamischer und flexibler werden? Auch müssen potenziell notwendige, neue Prozesse definiert werden.

Die festgelegten Anforderungen für die Geschäftsziele- und Prozesse müssen anschließend zu einer IT-Strategie übersetzt werden. Je nach Geschäftsmodell, Zielen und Ausrichtung des Unternehmens wird dann der Rahmen für die IT gesetzt, in welchem Ausmaß die Cloud genutzt werden soll.

Der Prozess



Abbildung 1: Prozess der Integration in die Cloud

(Quelle: *Cloud Computing als neue Herausforderung für Management und IT* von Münzl et al. 2015, S. 24)

Im ersten Schritt des Integrationsprozesses muss der Ist-Zustand der Geschäftsprozesse erfasst werden, dies geschieht in Zusammenarbeit mit den jeweiligen Abteilungen. Die Änderungsanforderungen werden vom Vergleich des Ist-Zustands mit dem Zielbild abgeleitet. Dabei wird direkt überprüft, ob die geplanten Änderungen mit Compliance und Sicherheitsvorschriften vom Unternehmen einhergehen, sowie staatliche Vorgaben eingehalten werden.

Die genaue Ausgestaltung hängt dabei vom Marktumfeld, den Kundenerwartungen, dem Geschäftsmodell und der individuellen Prozesslandkarte eines Unternehmens ab und führen entsprechend zu einer individuellen Ableitung der Strategie.

Migration

Es empfiehlt sich für ein Unternehmen eine Roadmap mit kurzfristigen, mittelfristigen und langfristigen Entwicklungszielen - und Maßnahmen anzulegen. Anhand dieser Map können Cloud-Provider gesucht werden, die die qualitativen und quantitativen Ansprüchen des Unternehmens erfüllen. Der darauffolgende Umstieg kann entweder für alle Benutzer gleichzeitig (Big Bang) oder schrittweise erfolgen. Optional wird vorher ein Pilotbetrieb eingerichtet.

Als Einstieg werden in der Regel gerne IaaS-Lösungen verwendet, da diese keine großen Anpassungen der Geschäftsprozesse mit sich bringen.

IT-Abteilungen im Wandel

Durch die Auslagerung von Kompetenzen und Ressourcen wird der IT-Manager zukünftig eine businessorientiertere Rolle einnehmen. Er ist für das Monitoring der Leistungen für Geschäftsprozesse und deren genutzte Services zuständig. Er muss als Bindeglied die Zielvorgaben und Anforderungen des Unternehmens unter den Providern delegieren. Ein weiteres Stichwort ist hier IT-Governance, wo Aspekte wie die Informationssicherheit und Compliance-Vorgaben mit den Providern abgestimmt werden müssen. Der CIO ist somit für die Gesamtorchestrierung sowie die Funktionsfähigkeit der Gesamtlandschaft aller IT-Systeme zuständig.

Die Rolle des klassischen Mitarbeiters in der IT wandelt sich vom Administrator über den Servicemanager hin zum Service Broker. Klassische Aufgaben wie Installation und Wartung fallen weg, neue Aufgaben wie das Managen und Überwachen von Cloud Services kommt hinzu.

Geschichte & Evolution

Das erste Mal wurde der Cloudbegriff in einem Businessplan der Firma compaq aus dem Jahr 1996 erwähnt, in dessen Zeitraum die Verbreitung stark zunahm. Eric Schmidt, damaliger CEO von Google, gilt als die Person die den Begriff des Cloud Computing auf einer Konferenz im Jahr 2006 populär gemacht hat. Nach einer anfänglichen Skepsis, ob der erste Hype um die Cloud wirklich gerechtfertigt war und die Unternehmen nicht nur Buzzwords hinterherlaufen, zeigte sich besonders in den letzten Jahren eine enorme Investitionssteigerung. Alleine in Deutschland wurden aus 7 Mrd. im Jahr 2014 bereits 3 Jahre später bis zu 18,5 Mrd € investiertes Kapital.

Die Cloud ist keine revolutionäre Technologie, wie es einst das Telefon, der Fernseher oder das Internet war. Diese brauchten viele Jahre bis Jahrzehnte um eine weite Verbreitung zu erreichen, doch veränderten den Alltag der Menschen in einem großen Ausmaß.

Die Cloud hingegen optimiert bereits bestehende Technologien, nutzt die bereits vorhandene Infrastruktur in Form von weit verbreiteten, schnellen Internetverbindungen und die Leistungen der großen Rechenzentren, was in einer deutlich schnelleren Verbreitung der Cloud Technologie resultiert.

Die Cloud ist die logische Fortführung des Time-Sharing Modells aus den 1970er Jahren, welches der erste konzeptionelle Ansatz für ein Mehrbenutzersystem war, wo mehrere User sich die Rechenzeit eines Prozessors teilten (siehe Abbildung 2: Evolution der Cloudtechnologie).

Die Cloud ist die logische Fortführung des Time-Sharing Modells aus den 1970er Jahren, welches der erste konzeptionelle Ansatz für ein Mehrbenutzersystem war, wo mehrere User sich die Rechenzeit eines Prozessors teilten (siehe Abbildung 2: Evolution der Cloudtechnologie).

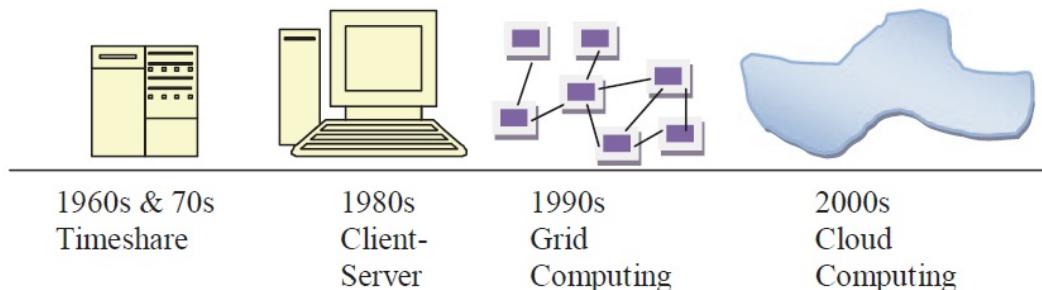


Abbildung 2: Evolution der Cloudtechnologie

(Quelle: *Cloud Computing Basics* von Sriinivasan 2014, S. 4)

Ein erster großer Schritt in der Entwicklung der Cloud wurde 1999 mit Salesforce.com getan. Die Firma schaffte ein Konzept wie man Enterprise-Anwendungen über eine einfach zu nutzende Webseite zur Verfügung zu stellen konnte.

Im Jahr 2002 kam mit den Amazon Web Services der nächste Schritt, indem der Konzern Rechenleistung und Speicher für seine Kunden anbot. Später startete Amazon zusätzlich die Elastic Compute Cloud, mit der Kunden Computer mieten konnte, um auf diesen ihre eigenen Applikationen laufen zu lassen.

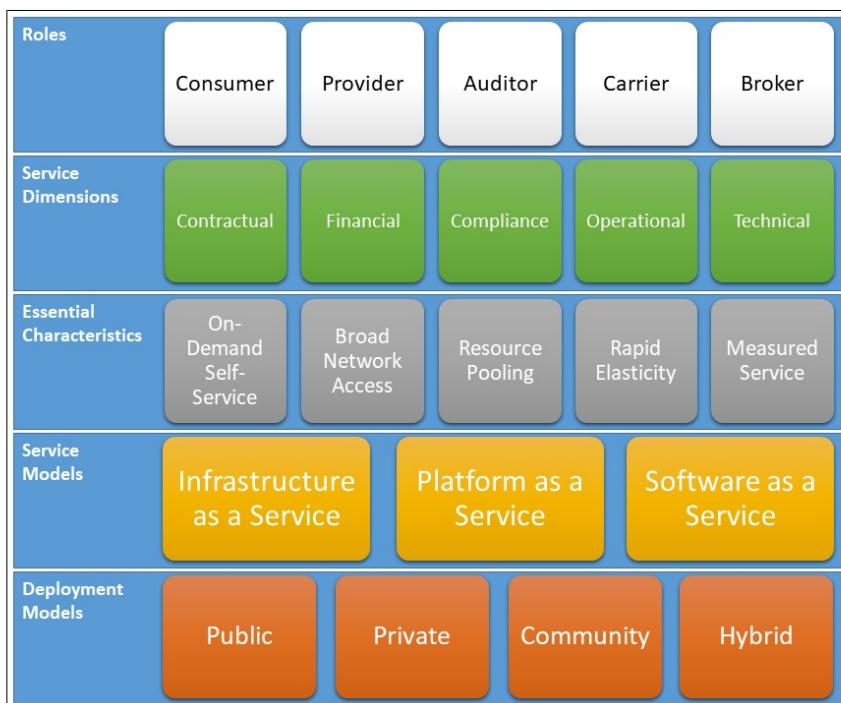
Mit der Verbreitung des Web 2.0 um das Jahr 2009 wurde ein weiterer Meilenstein erreicht. Google konnte mit seinen Apps auf neuem Wege eine Vielzahl von browserbasierten Enterprise-Applikationen anbieten und wirkte somit bei der Verbreitung des SaaS Konzeptes am Markt mit.

In den folgenden Jahren stiegen auch andere Größen wie Microsoft, IBM und Rackspace in das Cloudbusiness mit ein.

Cloud im Detail

Cloud Computing ist heutzutage ein viel benutzter Begriff, der sowohl in den Medien als auch im Unternehmensumfeld reges Interesse findet. Insbesondere die Publizität des Cloud-Ansatzes sorgt jedoch dafür, dass oft Unklarheit bezüglich seiner Bedeutung herrscht. Zwar erkennt die Mehrheit, dass Cloud Computing mit einer Auslagerung von Informationen oder Diensten gleichzusetzen ist. Eine weitere Spezifizierung ist aber nur den Wenigsten möglich.

Ziel dieses Kapitels ist es, einen einheitlichen Cloud-Begriff zu vermitteln. Hierzu wird im weiteren auf das Cloud-Computing Definition Framework eingegangen, welches vom National Institute of Standards and Technology (NIST) entwickelt und von Zalazar, Ballejos und Rodriguez um weitere Elemente erweitert wurde (siehe Abb. 1). Das Modell vollzieht eine Abstraktion der Cloud in verschiedene Schichten, welche sich teilweise bedingen. Zum einen wird auf die Teilnehmer eines Cloud-Szenarios eingegangen. Eine weitere Schicht beschäftigt sich mit den vertraglichen Bedingungen, welche zwischen den jeweiligen Teilnehmern herrschen können. Außerdem erfolgt die Aufzählung der typischen Charakteristiken einer Cloud. Abschließend wird darauf eingegangen, welche Art von Ressourcen eine Cloud bereitstellen kann bzw. wie offen eine Cloud für andere Teilnehmer ist. Eine detailliertere Beschreibung der einzelnen Schichten geschieht im Folgenden.



(Abbildung 1: Erweitertes Cloud-Computing Definition Framework)

(Quelle: in Anlehnung an *Cloud Dimensions for Requirements Specification* von Zalazar et al. 2017, S. 26)

Rollen

Das Cloud-Computing Definition Framework beinhaltet eine einheitliche Rollenbeschreibung. Hiermit ist es möglich, Zuständigkeiten und Verantwortungen in einem Cloud-Szenario exakt abzugrenzen und vertraglich festzuhalten. Generell wird zwischen fünf verschiedenen Rollen unterschieden, wobei nicht jede Rolle zwangsläufig eingenommen werden muss.

- **Consumer:** Der Consumer bzw. Konsument nimmt eine zentrale Rolle in einem Cloud-Szenario ein. Er hat das Bedürfnis, einen Cloud-Dienst wahrzunehmen. Somit kann er als Aktor angesehen werden, von dem die initiale Nutzungsanfrage ausgeht. Der Konsument hat zudem bestimmte Anforderungen im Hinblick auf die Eigenschaften des benötigten Dienstes. Beispielsweise kann er ein bestimmtes Maß an Erreichbarkeit oder Performanz einfordern bzw. hat spezifische Preisvorstellungen.
- **Provider:** Der Provider bzw. Anbieter definiert das Gegenstück zum Konsumenten. Er bietet Cloud-Dienste an und steht in der Verantwortung, die vom Konsumenten geforderte Leistung zu erbringen. Der Anbieter kann dabei eine beliebige Anzahl an Konsumenten jeweils mit gleichen oder unterschiedlichen Diensten bedienen.
- **Auditor:** Der Auditor bzw. Prüfer ist eine unabhängige Instanz, welche Gutachten bezüglich der erbrachten Leistungen seitens des Anbieters erstellt. Sind diese Leistungen unterhalb der zwischen Konsument und Anbieter vertraglich festgelegten Mindestleistungen befindlich, erstattet der Prüfer Meldung an den Konsumenten. Anschließend kann dieser beispielsweise vom Anbieter Schadensersatz verlangen.
- **Carrier:** Der Carrier bzw. Bote regelt und vollzieht die Übermittlung von Daten zwischen dem Anbieter und Konsumenten. Hierunter ist in den meisten Fällen der Internet-Provider zu verstehen. Seine erbrachten Leistungen haben erheblichen Einfluss auf die Zufriedenheit der Konsumenten. Können Daten nicht zeitnah und vollständig ausgeliefert werden, gefährdet dies ihre Wirtschaftlichkeit.
- **Broker:** Der Broker bzw. Vermittler ist ein Intermediär, welcher als Schnittstelle zwischen Konsument und Anbieter agieren kann. Ist aus spezifischen Gründen ein direkter Austausch von Leistungen zu unterlassen, trägt er dafür Sorge, dass Dienste von Teilnehmern gefunden sowie von bzw. an diese überbracht werden können.

Schlüsselmerkmale

Es existieren diverse Arten von Cloud-Computing, welche im Lauf dieses Kapitels näher erläutert werden. Allen Arten liegen jedoch im Regelfall dieselben Merkmale zugrunde, welche somit die Grundlage einer jeden Cloud bilden. Die Schlüsselmerkmale werden im Folgenden dargelegt.

- **On-demand self-service:** Jeder Konsument kann zu jeder Zeit und ohne eine weitere menschliche Interaktion seitens des Anbieters einen Dienst wahrnehmen.
- **Broad network access:** Alle Dienste sind über das Internet und unabhängig von der Plattform des Endgeräts erreichbar.
- **Resource Pooling:** Der Anbieter legt seine gesamten Hard- und Softwareressourcen zusammen. Alle Konsumenten kommunizieren hierdurch mit demselben IT-System. Da dieses System mandantenfähig ist, arbeiten die einzelnen Nutzer jedoch logisch gesehen auf verschiedenen Systemen und haben keine Einsicht in die Bereiche anderer Konsumenten.
- **Rapid elasticity:** Der Anbieter kann seinen Konsumenten dynamisch Ressourcen zuteilen und wieder entziehen. Der Bedarf des Konsumenten kann also stetig variieren, ohne dass ihm jemals Ressourcen fehlen bzw. Ressourcen von ihm ungenutzt sind. Die Ressourcenzuteilung seitens des Anbieters geschieht dabei im Idealfall automatisch.
- **Measured Service:** Die Auslastung des Cloud-Services seitens des Konsumenten ist sowohl für Anbieter als auch Konsument jederzeit einsehbar. Hierzu zählen beispielsweise der genutzte Speicher, die Anzahl der übermittelten Daten oder die Anzahl der aktiven Benutzeraccounts. Durch dieses Vorgehen besitzt der Konsument sowohl Transparenz über sein Nutzungsverhalten als auch über die daraus resultierenden Kosten.

Dienstdimensionen

Ein typisches Cloud-Szenario umfasst grob betrachtet zwei Rollen. Ein Anbieter stellt einen Dienst zur Verfügung, den ein Konsument wiederum nutzt. In dieser Konstellation hat jede Partei Anforderungen an die andere Partei. Der Konsument möchte beispielsweise eine bestimmte Performanz der angebotenen Cloud-Services gewährleistet haben, während der Anbieter auf eine pünktliche und leistungsgerechte Bezahlung Wert legt. Die Gesamtheit der Anforderungen, welche die beiden Parteien jeweils an ihr Gegenüber stellen, lassen sich fünf verschiedenen Bereichen bzw. Dimensionen zuordnen.

- **Contractual:** Contractual bzw. vertragliche Anforderungen sind die grundlegenden Richtlinien, die der Dienst-Vertrag zwischen Anbieter und Konsument umfasst. Hierin wird festgelegt, welche Parteien in welcher Rolle an dem Geschäft beteiligt sind. Die Aufführung von Vermittlern, Prüfern und Boten sei an dieser Stelle möglich. Weiterhin ist definiert, wie lang der Vertrag gültig ist bzw. wann die Nutzung des Services seitens des Konsumenten startet und endet. Letztlich kann auch spezifiziert werden, wann die Löschung der Anbieterdaten nach Ablauf des Vertrages frühestens erfolgen darf bzw. spätestens erfolgen muss. Die vertraglichen Anforderungen sind meistens statisch und ändern sich während der Vertragslaufzeit demnach nur selten.
- **Financial:** Financial bzw. finanzielle Anforderungen werden hauptsächlich vom Anbieter

gestellt und dienen seiner gerechten Entlohnung. Zum einen ist hier das Modell zu spezifizieren, nach dem die Bezahlung erfolgen soll. Beim ***tiered pricing*** erfolgt die Abrechnung anhand der Zeit, in der der Konsument den Dienst tatsächlich genutzt hat. ***Per-unit pricing*** bedeutet, dass der Konsument einen festen Preis für jede genutzte Einheit zahlt. Eine Einheit sei beispielsweise ein Gigabyte an Speicherplatz, welches der Konsument auf Seiten des Anbieters belegt hat. Beim ***subscription-based pricing*** bezahlt der Konsument ähnlich dem ersten Modell für einen Zeitraum. Der Unterschied liegt jedoch darin, dass es nicht von Relevanz ist, ob bzw. wie oft er den Service in diesem Zeitraum tatsächlich wahrgenommen hat. Dieses Bezahlmodell kann heutzutage mit einer Flatrate verglichen werden. Neben dem Bezahlmodell umfassen die finanziellen Anforderungen zudem die Höhe der Bezahlung sowie den Bezahlrhythmus. Weiterhin ist festgelegt, mit welcher Methode die Zahlung erfolgen muss, beispielsweise per Pay-Pal oder normaler Überweisung.

- **Compliance:** Compliance-Anforderungen betreffen die Einhaltung von Vorgaben. Insbesondere der Konsument muss sicherstellen, dass seine ausgelagerten Dienste bestimmte Gesetze, Normen und Standards einhalten. Andernfalls kann dies für ihn rechtliche bzw. wirtschaftliche Konsequenzen nach sich ziehen. Beispielhaft sind hierfür rechtliche Vorgaben zur Lagerung sensibler Daten bzw. strikte Zugangskontrollen zu bestimmten Ressourcen. Neben den eigentlichen Vorgaben beinhaltet diese Dimension zudem Strafen bzw. Entschädigungen, die bei Nichteinhaltung von vertraglichen Bestimmungen an den Anbieter verhängt werden können.
- **Operational:** Operational bzw. betriebliche Anforderungen sollen die Erreichbarkeit eines Cloud-Dienstes sicherstellen. Es wird initial festgelegt, wie lange der Dienst pro Zeiteinheit maximal ausfallen darf. Des Weiteren sind Maßnahmen zur Instandhaltung und Wiederherstellung der Systeme angegeben. Auch eine geeignete Bewertung bzw. Behandlung von Ausfällen wird vereinbart, um den bestmöglichen Betrieb zu gewährleisten. Letztlich sind geeignete Skalierungsvorgaben auszumachen, da die Bereitstellung von Ressourcen seitens des Anbieters ab einem bestimmten Punkt limitiert ist.
- **Technical:** Technical bzw. technische Anforderungen beziehen sich maßgeblich auf die Leistung des Dienstes. Der Konsument will sicherstellen, dass seine Bedürfnisse zu jeder Zeit in ausreichendem Maß befriedigt werden. Hierfür werden diverse Kennzahlen festgelegt, welche seitens des Anbieters zu erfüllen sind. Beispielhaft hierfür sind die Performanz und die Speicherkapazität seines IT-Systems sowie der Datendurchsatz des Netzwerks. Abhängig vom tatsächlichen Dienst sind weiterhin Schnittstellen oder vorinstallierte Betriebssysteme sowie Softwarebibliotheken aufzuführen, welche das IT-System umfassen muss.

Dienstmodelle

Der bisherige Dienst-Begriff ist allgemein formuliert und umfasst alle Arten von Leistungserbringungen. Das National Institute of Standards and Technology vollzieht hier eine weitere Spezifizierung, um verschiedenartige Dienste effektiver voneinander abgrenzen zu können. Im Folgenden werden die drei verschiedenen Dienstmodelle erläutert.

- **Infrastructure as a Service (IaaS):** Das IaaS-Modell zielt auf die Bereitstellung gesamter Rechnerinstanzen ab. Hierunter sind sowohl Server als auch Datenbanken bzw. Speichermedien im Allgemeinen zu verstehen. Der Konsument hat grundlegenden Zugriff auf die Hardware-Komponenten, die ihm der Anbieter bereitstellt. Es ist ihm somit möglich, seine eigenen Konfigurationen vorzunehmen sowie eigene Anwendungen aufzuspielen. Die Aufgabe seitens des Anbieters besteht lediglich darin, einen störungsfreien Betrieb der Hardware sicherzustellen. Hierzu zählt auch die kontinuierliche Überwachung der Performanz, Speicherbelegung sowie Datentransferrate. Das Dienstmodell bietet sich für solche Konsumenten an, die bereits über stabil laufende Anwendungen verfügen und lediglich eine flexible Skalierung der Rechenleistung benötigen.
- **Platform as a Service (PaaS):** Das PaaS-Modell baut auf dem des IaaS auf. Zusätzlich zu den Rechnerinstanzen stellt der Anbieter hier noch komplett Systemumgebungen bereit. Je nach Wunsch des Konsumenten sind auf den Hardware-Knoten bestimmte Betriebssysteme, Entwicklungsumgebungen sowie Programmbibliotheken installiert. In manchen Fällen können zudem grundlegende Prozeduren bezüglich Zugriffskontrolle, Synchronisierung oder Datenhaltung vorgefunden werden. Die bereits vorkonfigurierten Container erleichtern den Konsumenten die Software-Entwicklung, da keine zusätzlichen Anstrengungen für die Konfiguration einer Systemumgebung nötig sind.
- **Software as a Service (SaaS):** Beim SaaS-Modell gestattet der Anbieter den Konsumenten die Nutzung einer speziellen Software, welche auf seinem IT-System arbeitet. Der Konsument hat von jedem Endgerät, zu jeder Zeit und an jedem Ort Zugriff auf die Applikation. Er besitzt jedoch keinen Einfluss auf die darunterliegenden Hardware-Komponenten sowie deren Konfiguration. Dieses Recht bleibt dem Anbieter vor, der für den reibungsfreien Betrieb der Software verantwortlich ist. Die Hauptvorteile des Dienstmodells liegen in den geringeren Investitionskosten sowie in der schnelleren Einsatzfähigkeit der Software.

Deployment-Modelle

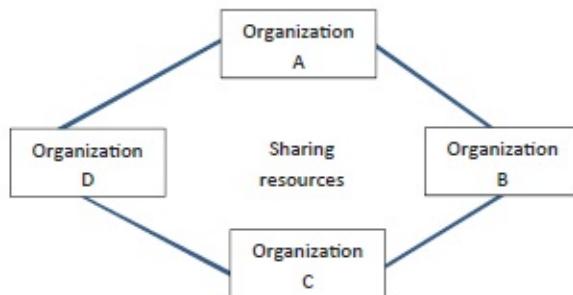
Innerhalb des Cloud-Computing existieren vier unterschiedliche Deployment-Modelle. Diese definieren jeweils, welche Zugriffsmöglichkeiten auf die Cloud-Dienste herrschen.

- **Private cloud:** Die Dienste einer privaten Cloud sind auf einzelne Organisationen bzw. Individuen beschränkt. Typischerweise hat ein Unternehmen Zugriff auf seine eigene

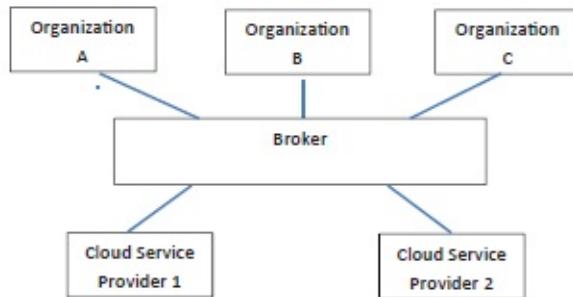
private Cloud, welche ausschließlich ihm zur Verfügung steht. Im Fall einer **typical private cloud** hostet das Unternehmen die Cloud in ihrem eigenen Rechenzentrum und ermöglicht den Zugriff per Internet. Eine Firewall trägt dafür Sorge, dass nur unternehmensinterne Mitarbeiter Zugang zu den Applikationen bzw. Informationen haben. Die **managed private cloud** ähnelt dem Vorgänger, mit Ausnahme dass das Rechenzentrum des Unternehmens von einem externen Anbieter verwaltet wird. Dies hat jedoch keinen Einfluss auf die Lokalität der Hardware. Bei einer **hosted private cloud** laufen die Cloud-Anwendungen direkt auf den Hardware-Komponenten des externen Anbieters. Weiterhin hat ausschließlich das spezifische Unternehmen Zugriff. Die letzte Variation ist die **virtual private cloud**, welche im eigentlichen Sinn eine öffentliche Cloud ist, auf die mehrere Unternehmen Zugriff haben. Durch VPN wird jedoch in dieser öffentlichen Cloud eine virtuelle private Cloud erzeugt, auf der ausschließlich das Unternehmen arbeiten kann.

- **Public Cloud:** Die öffentliche Cloud ist das Gegenstück zur privaten Cloud. Sie erlaubt verschiedenen Konsumenten den Zugriff auf ihre Funktionen. Dem Anbieter gehören dabei sämtliche Anwendungen, welche öffentlich zur Verfügung stehen. Typischerweise verfügt dieses Deployment-Modell über einfache Anmeldestrukturen und standardisierte Services, um möglichst vielen Konsumenten gerecht zu werden. Die öffentliche Cloud ist für viele Benutzer der Inbegriff von Cloud Computing.
- **Community Cloud:** In einer Community Cloud schließen sich verschiedene Organisationen zusammen, welche ein gemeinsames Ziel verfolgen. Beispielsweise sind verschiedene Krankenhäuser, die dieselben medizinischen Geräte verwenden oder ihre Erkenntnisse untereinander teilen wollen. Demnach ist die Community Cloud eine öffentliche Cloud für eine homogene Gruppe von Konsumenten. Bei einer **federated community cloud** greifen die Organisationen innerhalb der Gruppe gegenseitig auf ihre eigenen Ressourcen zu (siehe Abb. 2). Bei einer **brokered community cloud** erfolgt der Zugriff über einen Vermittler, welcher die Anfragen an einen zentralen Anbieter weiterleitet.

Federated community cloud:



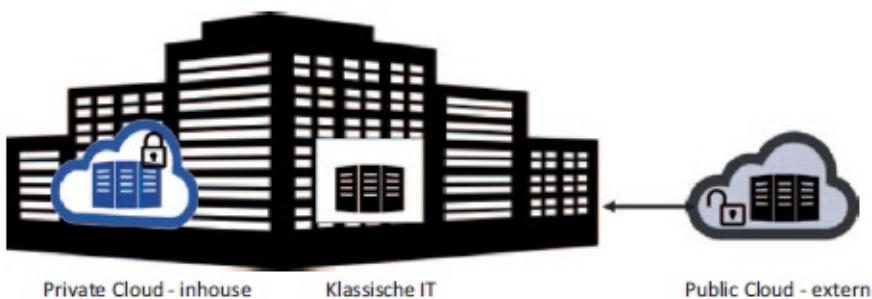
Brokered community cloud:



(Abbildung 1: Vergleich von Federated und Brokered Community Cloud)

(Quelle: *Cloud Computing Basics* von Srinivasan 2014, S. 36)

- **Hybrid Cloud:** Eine hybride Cloud ist der Zusammenschluss von mindestens zwei der vorangegangenen Deployment-Modelle. Beispielsweise kann ein Unternehmen sensible Daten in eine private Cloud auslagern, während rechenintensive und unsensible Dienste in einer öffentlichen Cloud ablaufen (siehe Abb. 3).



(Abbildung 3: Beispiel für hybride Cloud)

(Quelle: *Cloud Computing als neue Herausforderung für Management und IT* von Münzl et al. 2015, S. 13)

Das Cloud-Computing Definition Framework trägt zur Präzision und Vereinheitlichung des Cloud-Begriffs bei. Insbesondere die Kommunikation sowie Entwicklung im Cloud-Umfeld wird hierdurch erleichtert.

Sicherheit

Sicherheit ist für ein effektives Cloud Computing von existenzieller Bedeutung. Nur wenn Risiken minimiert bzw. vollkommen vermieden werden können, ist es Organisationen möglich, Potenziale im Hinblick auf Wirtschaftlichkeit und Flexibilität auszuschöpfen. Der allgemeine Begriff Sicherheit lässt sich in puncto Cloud in zwei Unterkategorien unterteilen, welche im Folgenden erläutert werden.

Compliance

Nahezu jedes Unternehmen unterliegt externen Vorgaben, welche als Compliance bezeichnet werden. Hierunter fallen zum einen staatliche Vorgaben, beispielsweise bezüglich des Datenschutzes. Zum anderen existieren brachenspezifische bzw. - unabhängige Normen und Standards, welche zur Aufrechterhaltung des Geschäfts einzuhalten sind. Mit jeder Hard- bzw. Software-Komponente, die der Konsument zu einem Anbieter auslagert, erhöht sich das Risiko, dass relevante Vorgaben fortan nicht mehr eingehalten werden. Zwangsläufig muss der Konsument sicherstellen, dass der Anbieter ohne jede Ausnahme nach seinen vorgegebenen Maßstäben arbeitet. Dieser Prozess beginnt im Idealfall schon bei der Anbieterauswahl.

Das wohl wichtigste Sicherheitsthema im Hinblick auf die Cloud ist der Schutz von sensiblen Unternehmensdaten. Hierunter sind zum einen Personendaten zu verstehen, wie beispielsweise Namen, Geburtsdaten und Adressen von Mitarbeitern sowie Kunden. Des Weiteren sind auch Finanzdaten hinzuzuzählen, welche den Fiskus betreffen. Bevor ein Konsument diese Daten in ein entferntes Cloud-Rechenzentrum auslagert, muss er sich zunächst über die geltenden rechtlichen Gegebenheiten informieren. In Abhängigkeit davon, in welchem Land der Konsument ansässig ist und in welchem Land sich das entfernte Rechenzentrum befindet, kann eine Auslagerung von sensiblen Unternehmensdaten von vornherein untersagt werden. Ist die Auslagerung rechtens, muss sich der Konsument sicher sein, dass der Anbieter die ihm anvertrauten Daten gewissenhaft und vertraulich behandelt. In Deutschland kann zu diesem Zweck beispielsweise eine Auftragsdatenverarbeitung beschlossen werden. Dieses Schriftstück legt fest, dass der Anbieter die sensiblen Daten nicht für seine eigenen Zwecke benutzt bzw. der Konsument das volle Recht auf die Daten beibehält.

IT-Sicherheit

Die IT-Sicherheit beschäftigt sich mit den technischen Maßnahmen, die für einen sicheren Cloud-Betrieb vorgenommen werden müssen. Schutzziele sind beispielsweise Vertraulichkeit, Integrität oder Authentizität. Eine sichere Nutzung lässt sich dabei nur gewährleisten, wenn Konsument, Anbieter und die zwischen ihnen herrschende Verbindung ganzheitlich geschützt sind.

- **Sicherheit beim Anbieter:** Der Anbieter von Cloud-Diensten steht in der Verantwortung, seine Rechenzentren gegenüber Missbrauch zu schützen. Er verfügt in vielen Fällen über sensiblen Daten diverser Organisationen bzw. Individuen und muss dafür Sorge tragen, diese vor Dritten geheim zu halten. Dabei kann er auf eine Vielzahl von Techniken zurückgreifen, die im Allgemeinen dem Schutz von Rechenzentren dienen und nicht spezifisch auf die Cloud-Architektur ausgelegt sind. Beispieldhaft sei hierfür eine sichere Authentifizierung der Konsumenten nach dem Zweifaktor-Prinzip. Weiterhin muss er Vorkehrungen treffen, welche im Detail an das Cloud-Szenario angepasst sind. Beispielsweise ist die logische Trennung von mehreren Konsumenten (Resource Pooling) abzusichern, sodass keine Zugriffe auf fremde Ressourcen möglich sind. Auch die Anwendung von speziellen Cloud-Patterns kann die Sicherheit verbessern.
- **Sicherheit beim Konsumenten:** Obwohl oft davon ausgegangen wird, dass die Verantwortung für einen sicheren Cloud-Betrieb allein beim Anbieter liegt, steht auch der Konsument in der Pflicht. Wie groß sein Beitrag zur Sicherheit dabei ausfallen kann, hängt vom Dienstmodell ab. IaaS erlaubt dem Konsumenten die ausführliche Absicherung seiner genutzten Rechenknoten. Bei SaaS fallen die Möglichkeiten deutlich geringer aus, da in der Regel keine detaillierten Konfigurationsmöglichkeiten tieferliegender Hard- bzw. Softwareschichten vollzogen werden können. Unabhängig vom Dienstmodell sollte der Konsument jedoch darauf achten, seine für den Zugriff genutzten Endgeräte frei von Schadsoftware zu halten und nur ausgewählten Benutzern zur Verfügung zu stellen.
- **Sicherheit bei der Übertragung:** Ein Schlüsselmerkmal der Cloud ist die Erreichbarkeit von Diensten über das Internet. Sensible Daten werden vom Konsumenten an den Anbieter verschickt bzw. vom Konsumenten angefragt. Dies impliziert, dass die Daten für eine bestimmte Zeitspanne auf einem öffentlichen und ggf. unsicheren Verbindungskanal befindlich sind. Zur Wahrung von Integrität und Vertraulichkeit müssen daher Mechanismen im Hinblick auf die Ver- und Entschlüsselung von Daten angewendet werden.

Cloud-Dienste am Beispiel

Die schnelle Verbreitung von Cloud Services ist vor allem durch die großen Provider wie Amazon, Google, Microsoft, Rackspace oder Terremark ermöglicht worden. Diese investieren große Summen in ihre Infrastruktur und errichten weltweit Datenzentren, die redundant und nach den Gesetzen des jeweiligen Standortes, die Daten ihrer Kunden verwalten und ihnen entsprechende Services zur Verfügung stellen.

Neben den großen Cloud Service Providern gibt es auch kleinere Angebote wie Apple, Salesforce, VMware oder Dropbox, welche sich auf bestimmte Services spezialisiert haben, z.B. die Distribution von Musik oder CRM.

Im Folgenden werden die Angebote von Amazon und Google genauer betrachtet.

Amazon Web Services

Amazon ist der größte der Clouddienstleister und übertrifft in Sachen Kapitalausstattung seine Konkurrenten um das Fünffache. Dabei bietet Amazon neben kostenpflichtigen Premiumservices auch kostenlose Angebote, mit denen Kunden die Möglichkeiten der AWS testen können. Nutzern wird ein einziger Einstiegspunkt für alle Services der Cloud geboten, was es für Einsteiger leichter macht sich zu orientieren und das genutzte Portfolio zu erweitern.

Seit dem Start der AWS im Jahr 2006 hat Amazon schätzungsweise 12 Milliarden Dollar in seine Services investiert und liegt damit hinter den Investitionen mancher Konkurrenten. Die AWS werden von Unternehmen in über 190 Ländern verwendet, entsprechend verteilen sich die Datenzentren über alle Kontinente, mit einem Fokus auf Asien und Nordamerika. Zusätzlich bietet Amazon seinen Kunden an, verschiedene Standorte je nach Service zu wählen, wo beispielsweise Daten verwaltet und von welchem Standort Rechenleistung bezogen werden soll.

AWS bietet dabei SaaS, PaaS, IaaS Services, die in Form von privaten und öffentlichen Clouds angeboten werden. Amazon garantiert dabei eine Verfügbarkeit von 99.9% der Zeit, was umgerechnet maximal 52 Minuten Ausfallzeit im Jahr bedeutet. Selbst kurze Ausfallspannen können bereits enorme Folgen aufgrund der schieren Größe und Verbreitung der AWS bedeuten, wie Vorfälle in den vergangenen Jahren zeigten.

Elastic Compute Cloud (EC2)

Mit der EC2 können Nutzer virtuelle Server verwalten, dabei kann man zwischen 4 Standorten wählen. In einem nächsten Schritt wählt der Benutzer dann eine Verfügbarkeitsregion für den gewählten Standort aus, auf der sein Server laufen wird. Weiterhin müssen Arbeitsspeicher, Prozessorleistung und Festplattenspeicher spezifiziert werden.

Die virtuellen Server werden aus Amazon Machine Images (AMI) erzeugt. Amazon bietet verschiedene vorgefertigte AMIs an, die sich in Betriebssystem und vorinstallierter Software unterscheiden. Sie stellen es dem Benutzer aber auch frei selber eine wiederverwendbare AMI zu erzeugen. Diese kann sogar veröffentlicht und vom Kunden als eigenes Produkt vertrieben werden.

Zur Identifikation gegenüber einer gehosteten Instanz wird ein Public Key Verfahren eingesetzt. Für weitere sicherheitsrelevante Einstellungen befindet sich jede Instanz in einer sogenannten Security Group, die frei konfigurierbare Regeln anbietet.

Nach erfolgreicher Einrichtung wird einer Instanz eine private sowie eine öffentliche IP eingerichtet. Über die öffentliche IP ist der Server aus dem Web erreichbar, die Private dient zur Kommunikation zwischen Instanzen in der AWS. Beide IPs sind jedoch dynamisch und werden nach Neustart der Instanz neu vergeben, daher muss man zusätzlich eine statische elastische IP Adresse reservieren.

Änderungen an der laufenden Instanz werden nicht gespeichert, dafür müssen Zustandsänderungen entweder bei großen Datenmengen Amazons S3 verwendet werden oder Amazons Elastic Block Store (EBS). EBS funktioniert in dem Fall wie eine externe Festplatte die an die Instanz angeschlossen wird.

Alle Vorgänge lassen sich über Kommandozeilenbefehle realisieren, es gibt jedoch auch weitere Werkzeuge wie die AWS Management-Konsole sowie das AWS Toolkit für Eclipse, um mit seinen Instanzen zu interagieren.

Amazon Simple Storage Service (S3)

S3 ist ein Massenspeicher mit einem eigenen Dateisystem. Die bis zu 5 GB großen Web-Objekten liegen hier in sogenannten Buckets und können über einen Bucket-Namen angesprochen werden. Der Zugriff auf die Objekte erfolgt standardmäßig über Web Services und ist über eine SOAP oder REST API möglich. Hierfür stehen ebenfalls komfortable, grafische Werkzeuge zur Verfügung. Zusätzlich gibt es auch Tools von Drittanbietern, die einen Zugriff auf den Speicher, wie auf weit entfernte Festplatten ermöglichen.

Amazon Elastic Block Store (EBS)

Mit dem Speicherdiensst können Datenspeicher von 1GB bis 1TB erzeugt werden, sogenannte Volumes. Ein Volumen kann ein beliebiges Dateisystem beinhalten und verhält sich dabei wie ein USB Stick. Ein Volumen kann lediglich an einer Instanz (AMI) angehängt werden, die in derselben Verfügbarkeitszone angelegt wurde wie das Volumen. Der Speicher ist persistent.

Amazon Simple Queue Service (SQS)

Die Nachrichtenwarteschlange SQS wird zur Skalierung von Anwendungen in der AWS eingesetzt. Sender können hier Nachrichten in die Warteschlange einstellen, welche von registrierten Empfängern dann abgearbeitet werden. Eine dienstnehmende Komponente kann ihre Arbeitsaufträge in die Warteschlange einstellen, wo sie eine dienstgebende Komponente abholt. So können bei Engpässen der kritischen Komponenten, diese parallel auf mehreren EC2 Instanzen laufen.

Amazon SimpleDB

SimpleDB ist auf eine einfach strukturierte, dafür hoch effiziente und zuverlässige Datenhaltung ausgelegt. Die transaktionalen Möglichkeiten sind hierbei stark eingeschränkt, werden jedoch für ein breites Spektrum an Anwendungen als ausreichend erachtet.

Amazon Relational Database Service

Als Gegenwurf zur SimpleDB steht das Amazon RDS, ein Plattformdienst zur einfachen Einrichtung, Betreiben und Skalieren von relationalen Datenbanken in der Cloud. Dabei gewährt Amazon Zugriff über die bekannten Funktionen einer MySQL Datenbank. Bei der Einrichtung wählt RDS automatisch die optimalen Parameter, unter Berücksichtigung von Rechenressourcen und Speicherkapazitäten. Diese lassen sich jedoch im Nachhinein auch entsprechend anpassen.

RDS stellt elastische Kapazitäten zur Verfügung und erledigt gleichzeitig zeitraubende Datenbankverwaltungsaufgaben. Automatisierte Backups und Snapshots können in individuell definierten Intervallen durchgeführt werden und wodurch eine hohe Zuverlässigkeit erreicht wird.

Bei Bedarf kann die Datenbank vertikal skaliert werden, bei sehr aufwendigen Leseoperation kann durch sogenannte Read-Replika Instanzen horizontal repliziert werden. Ein kostenloses Hochverfügbarkeitsangebot bietet die Möglichkeit synchron replizierte Instanzen in mehreren Verfügbarkeitszonen bereit zu stellen. Bei Ausfällen oder Wartungsfenstern in einem Standort wird automatisch zwischen den replizierten Instanzen gewechselt. Über Amazon CloudWatch kann man schließlich die Auslastung der Datenbank überwachen.

Google Apps

Das App-Angebot von Google ist stark gewachsen in den vergangenen 5 Jahren. Das erste Angebot war Google Docs, was zum direkten Konkurrenten Microsoft Office, die Probleme von verschiedenen Versionen und inkompatiblen Formaten untereinander umgehen sollte. Zusätzlich wurden Features wie das kollaborative Arbeiten an einem Dokument hinzugefügt, sowie die Möglichkeit geboten Dokumente in verschiedenen Formaten jederzeit hoch und runterzuladen.

In den nächsten Jahren wurden die Services mit Gmail, Google Drive, Google Talk, Google Calendar, Google Video, Google Labs und Google Play erweitert. Da alle diese Services kostenfrei verfügbar sind und Drive beispielweise großzügige 30 GB zur Verfügung stellt, verbreiteten sie sich entsprechend schnell.

Google hat schätzungsweise bis zu 21 Milliarden in ihre Services investiert und somit deutlich mehr als Amazon, bietet seine spezialisierten Services jedoch kostenlos an und finanziert diese über Werbeangebote.

App Engine

Googles App Engine ist ein PaaS Service und dient als Programmierumgebung und Ausführungsumgebung für neu entwickelte Apps. Unterstützt werden Java und Python. Zum Testen kann die Anwendung auf einer lokalen Laufzeitumgebung gestartet werden, schließlich kann man eine fertige Version auf die Google Infrastruktur übertragen und ausführen lassen.

Zusätzlich zu den Funktionalitäten der Java Bibliotheken bietet die Plattform weitere Services, die über die Standardschnittstellen von Java angesprochen werden können und die Portabilität sowie die Benutzbarkeit des Codes erweitern.

- Zur Speicherung von Daten verwendet die Plattform den sogenannten DataStore, eine schemalose objektorientierte Datenbank. Angesprochen werden kann sie mit Hilfe von JDO und JPA.
- Des Weiteren können über die JavaMail Schnittstelle Funktionen zum Senden und Empfangen von Emails via Google-Mail Konten implementiert werden.
- Zur Authentifizierung werden Google Konten verwendet, welche mit Hilfe eines rudimentären Rechte-Managements zwischen Administratoren und Nutzern der App unterschieden werden können.

Für Entwickler gibt es ein kostenloses Tageskontingent ans Nutzbarer CPU Leistung und Speicher, welches für die meisten kleineren Applikationen ausreichend ist. Erst bei größeren Projekte werden teurere Premiummodelle notwendig.

Google Storage

Der Speicherdienst funktioniert ähnlich dem S3 Service von Amazon und ermöglicht das Speichern von Web-Objekten in den Datenzentren von Google, die über REST abgerufen werden können. Auch hier werden Objekte in sogenannte Buckets gespeichert. Mit Hilfe des Kommandozeilenwerkzeugs GSUtil können Objekte hochgeladen und Buckets angelegt werden., sowie Zugriffsrechte verwalten.

Google Cloud Print

Durch den Cloud Print Service können sich netzwerkfähige, kompatible Drucker in der Cloud registrieren lassen und anschließend von jeder Applikation, die Google Cloud Print unterstützt, über das Internet angesprochen werden. So fallen die Installation von einzelnen Treibern und eventuelle Kompatibilitätsprobleme weg. Drucker können mit anderen Google Konten geteilt werden und so den Zugriff gestatten oder gleich als Dienstleister auftreten.

MS Cloud Design Patterns

Zusammenfassung des Buches:

Titel: MS Cloud Design Patterns - Prescriptive Architecture Guidance for Cloud Applications

Verfasser: Alex Homer, John Sharp, Larry Brader, Masashi Narumoto, Trent Swanson

Verlag: Microsoft

Jahr: 2014

ISBN: 978-1-62114-036-8

Zusammenfassung von: Nils Kohlmeier, Daniel Beneker, Jonas Wiese, Malte Berg, Tolga Aydemir, Sven Schirmer, Yannick Kloss, Niklas Harting, Timo Rolfsmeier, Alexander Schwietert, Lukas Taake, Fabian Lorenz, Philipp Viertel, Benjamin Schmidt, Lutz Winkelmann, Tim Jastrzembski, Gamze Soylev Oektem, Justin Jagnieniak, Christian Holzberger, Marcel Dzaak, Andrei Günther, Oliver Nagel, Marvin Schirrmacher, Jonathan Jansen

Einleitung

Dieses Dokument stellt eine Zusammenfassung von Design-Pattern im Hinblick auf Cloud Computing dar und richtet sich dabei an das Buch "Cloud Design Patterns - Prescriptive Architecture Guidance for Applications" (ISBN: 978-1-62114-036-8). Im Buch werden acht grundlegende Aspekte für Probleme in der Entwicklung im Bereich Cloud Computing betrachtet:

- **Verfügbarkeit:**

Die Verfügbarkeit ist definiert als die Zeit, indem das System funktions- und arbeitsfähig ist. Für gewöhnlich wird diese Zeit prozentual angegeben und mit Benutzern in Form eines Service-Level-Agreements (SLA) vereinbart.

- **Datenmanagement:**

Datenmanagement ist eines der Schlüsselaspekte im Bereich des Cloud-Computings. Im Hinblick auf Anforderungen bezüglich Performance, Skalierbarkeit oder Verfügbarkeit liegen Daten an mehreren Orten und auf mehreren Servern gestreut vor. Hier sind bspw. Datensynchronisation und Datenintegrität gefordert.

- **Design und Implementierung**

Entscheidungen in der Design- und Entwicklungsphase haben immense Auswirkungen auf die Qualität und die Kosten einer Applikation. Das Design hat bspw. Einfluss auf die Konsistenz und Kohärenz zusammenhängender Komponenten, die Wartbarkeit eines Systems und die Wiederverwendbarkeit einzelner Komponenten.

- **Benachrichtigungen**

Die dezentrale Natur von Cloud-Applikationen erfordert eine Infrastruktur für Benachrichtigungen, um verteilte Komponenten und Dienste koppeln zu können.

- **Management und Überwachung**

Cloud-Applikationen werden in Datenzentren betrieben, dadurch besteht oft keine volle Kontrolle über die Infrastruktur. Somit gestaltet sich das Management und die Überwachung oft schwieriger als bei einem "on-premises" Betrieb. Applikationen müssen zur Laufzeit Informationen freigeben, um Administratoren die Möglichkeit zu bieten das System zu verwalten und zu überwachen.

- **Performanz und Skalierbarkeit**

Performanz beschreibt die Fähigkeit eines Systems eine Aktion in einem bestimmten Zeitintervall ausführen zu können, während die Skalierbarkeit die Fähigkeit eines Systems beschreibt erhöhte Lasten verwalten zu können, ohne dass sich Auswirkungen in der Performanz oder bei anderen verfügbaren Ressourcen bemerkbar machen.

- **Elastizität**

Elastizität beschreibt die Fähigkeit eines Systems auf Fehler reagieren zu können, sowie sich von Fehlern erholen zu können. Da typischerweise mehrere Kunden einen Dienst auf einem Server in Anspruch nehmen, besteht eine erhöhte Fehlerwahrscheinlichkeit.

- **Sicherheit**

Sicherheit beschreibt die Fähigkeit eines Systems böswillige oder unbeabsichtigte Aktionen zu vermeiden und nur Aktionen innerhalb des System-Designs zu genehmigen. Cloud-Applikationen sind oft öffentlich zugänglich und müssen deshalb vor böswilligen Angriffen und unautorisiertem Zugriff geschützt werden.

Im Folgenden werden diverse Design-Pattern vorgestellt, welche Hilfestellungen für Herausforderungen in der Entwicklung cloudbasierter Systeme bieten und dabei die oben genannten Aspekte einbeziehen.

Cache-Aside Pattern

Das Cache-Aside Pattern beschreibt ein Verfahren, nach dem häufig benötigte Daten für den schnellen Zugriff in einem Cache zwischengespeichert und bei Bedarf aktualisiert werden. Der Ablauf ist wie folgt: Das System greift zuerst auf den Cache zu und prüft, ob die gewünschten Daten im Cache vorhanden sind. Ist dies nicht der Fall, so werden die Daten aus dem eigentlichen Speicher (Datebank, etc.) geladen und im Cache für weitere Zugriffe gespeichert. Werden Änderungen an den Daten durchgeführt, so werden die Änderungen im Speicher übernommen und der entsprechende Cache-Eintrag wird als ungültig markiert, damit bei einem weiteren Zugriff die Daten neu aus dem Speicher geladen werden.

Beim Umsetzen dieses Patterns sollte eine Gültigkeitsdauer der Daten im Cache festgelegt werden. Daten, die eine längere Zeit nicht abgerufen wurden, müssen aus dem Cache entfernt werden, um Platz für Daten zu schaffen, die häufiger benötigt werden. Diese Gültigkeitsdauer sollte nicht zu kurz gewählt werden, sonst werden die Daten ständig neu aus dem Datenspeicher geladen. Bei der Wahl der Gültigkeitsdauer sollte aber auch die Zeit, die benötigt wird, um ein bestimmtes Datum aus einem Datenspeicher zu holen, beachtet werden. So sollte zum Beispiel ein Datum, dass in einer externen Datenbank liegt, die sehr langsam ist, länger im Cache verbleiben als ein Datum, dass in einer Datenbank auf demselben Server liegt und somit schneller geladen werden kann. Ein sehr wichtiger Punkt ist, dass Daten im Datenspeicher durch eine externe Anwendung geändert werden können und die Kopie der Daten im Cache damit nicht mehr gültig ist. Die Anwendung, die den Cache verwendet, merkt die Änderung erst beim erneuten Laden der Daten aus dem Datenspeicher. Verwenden mehrere Anwendungen die gleichen Daten, so sollte man sich die Verwendung von geteilten Caches überlegen, bei der mehrere Anwendungen gemeinsam auf einen Cache zugreifen. Beim Start der Anwendung kann außerdem der Cache mit Daten gefüllt werden, die wahrscheinlich während des Startens benötigt werden.

Das Cache-Aside-Pattern sollte eingesetzt werden, wenn der Bedarf von bestimmten Daten nicht vorhersagbar ist und quasi zufällig ist. Für statische Daten oder für Session-Daten in Client-Server-Anwendungen sollte das Pattern nicht eingesetzt werden.

Circuit Breaker Pattern

Mit diesem Pattern lässt sich auf Fehler, die unterschiedlich lange dauern, angemessen reagieren. Die richtige Anwendung des Pattern kann die Stabilität und Elastizität eines Programms verbessern.

Kontext und Problem

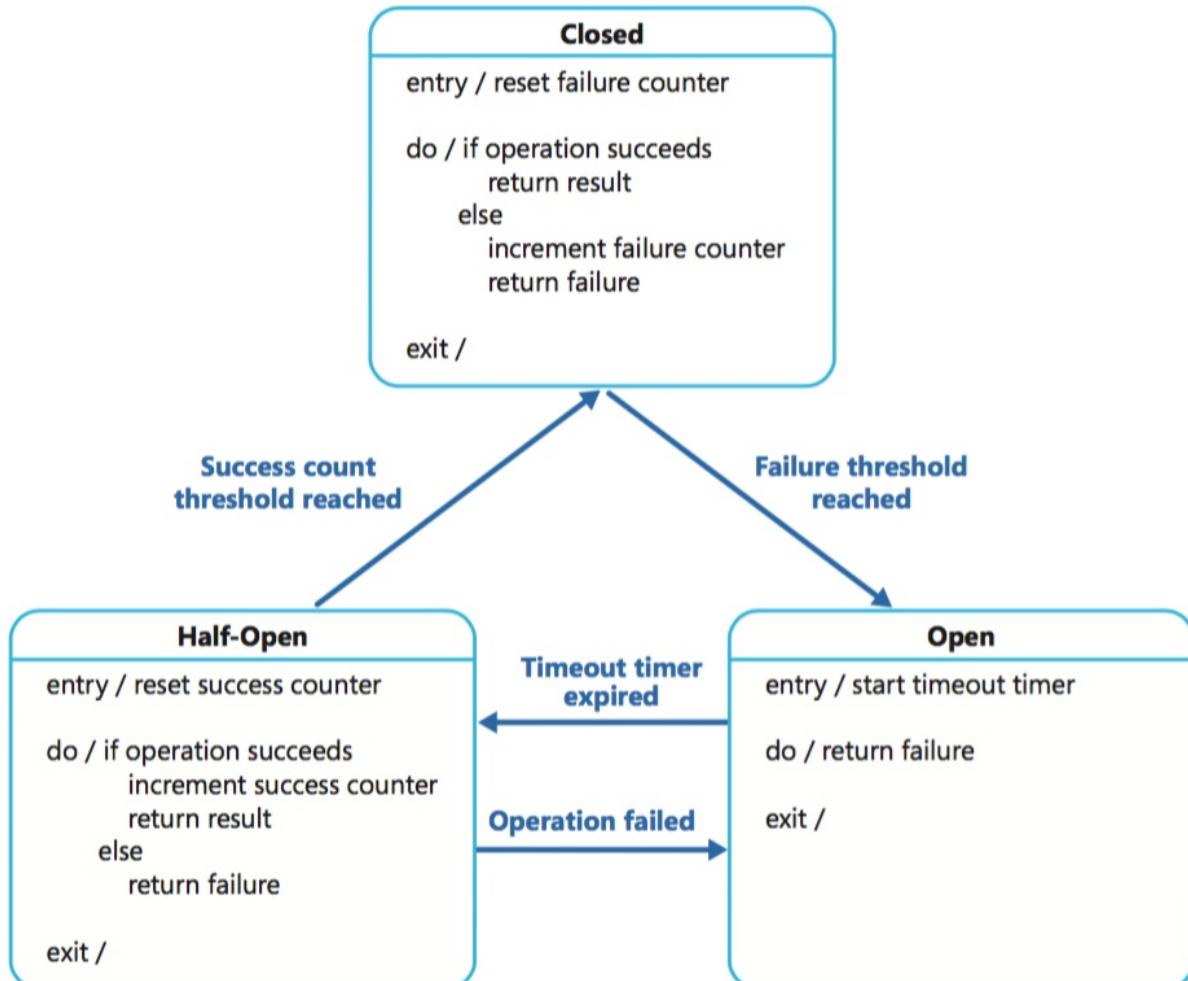
Es gibt zwei Arten von Fehlern: Kurzlebige Fehler, die sich oft nach kurzer Zeit auflösen, wie beispielsweise ein kurzzeitig langsames Netzwerk und größere Fehler, die aus unerwarteten Ereignissen resultieren. Im letzteren Fall ist es sinnlos den ausgefallenen Service weiter anzusprechen. In diesem Fall sollte das Programm den Fehler schnell akzeptieren und entsprechend damit umgehen.

Lösung

Die Idee ist, das Programm davon abzuhalten Operationen, die sehr wahrscheinlich fehlschlagen, (erneut) auszuführen. In Abgrenzung dazu führt das „[Retry Pattern](#)“ Operationen erneut aus, wenn man davon ausgeht, dass sie funktionieren.

Das Circuit Breaker Pattern ist ein Proxy für Operationen die fehlschlagen können. Der Proxy misst die Anzahl vorheriger Fehler und entscheidet auf der Grundlage, ob weitere Operationen zugelassen werden oder nicht.

Implementieren lässt sich dieses Verhalten als State Machine. Die folgende Abbildung stellt die State Machine mit den drei Zuständen Closed, Open und Half-Open dar.



Status: Closed

Dies ist der initiale Zustand. Alle Anfragen durchlaufen den Proxy und werden zugelassen. Kommt es bei einer Anfrage zu einem Fehler, wird er mit Hilfe eines Zählers registriert. Erreicht dieser Zähler einen Schwellwert, wird in den Status „Open“ gewechselt. Der Zähler sollte periodisch zurück gesetzt werden, damit gelegentliche Fehler nicht dazu führen, dass immer in den „Open“ Status gewechselt wird.

Status: Open

Beim Wechsel in diesen Status wird ein Timeout-Timer gestartet. Der Proxy blockiert alle Anfragen und gibt entsprechende Exceptions zurück. Die Länge desTimeouts entscheidet darüber, wie lange alle Anfragen blockiert werden sollen. Ist der Timer abgelaufen, wird in den Status „Half-Open“ gewechselt.

Status: Half-Open

In diesem Status wird ein Teil der Anfragen zugelassen. Mit diesem Status wird versucht zu erkennen, ob der entsprechende Service wieder erreichbar ist oder nicht. Es wird nur ein Teil der Anfragen zugelassen, um zu verhindern, dass ein Service, der wieder erreichbar ist, plötzlich mit so vielen Anfragen überflutet wird, dass er sofort wieder abstürzt. Alle erfolgreichen Anfragen werden gezählt. Schlägt eine Anfrage fehl, wird sofort wieder in den Status „Open“ zurück gewechselt. Wurde eine bestimmte Anzahl von Anfragen erfolgreich durchgeführt, ist davon auszugehen, dass der Fehler behoben ist. Es wird wieder in den Status „Closed“ gewechselt.

Weitere Aspekte

- Exception Handling: Falls vom Circuit Breaker eine Exception geworfen wird, sollte der Empfänger entsprechend reagieren, wie beispielsweise dem Benutzer eine Meldung anzuzeigen, die Oberfläche für eine gewisse Zeit zu deaktivieren oder alternative Operationen anbieten.
- Verschiedene Arten von Exceptions: Circuit Breaker selbst kann verschiedenste Exception erhalten, wenn er eine Anfrage ausführt und diese fehlschlägt. Auf verschiedene Exceptions könnte unterschiedlich reagiert werden. So könnten beispielsweise mehr Timeout Fehler eintreten, bis in den „Open“ Status gewechselt wird, als bei einer allgemeinen Exception, die auf einen schwerwiegenderen Fehler hinweist.
- Logging: Fehlgeschlagene Anfragen sollten gespeichert werden. Außerdem sollte die State Machine an sich überwacht werden. Bei einem Wechsel in den Status „Open“ kann beispielsweise der Systemadministrator benachrichtigt werden.
- Fehlgeschlagene Operationen testen: Statt eines Timeout-Timers im „Open“ Status, kann der nicht erreichbare Service testweise angesprochen werden, um festzustellen, wann der Service wieder erreichbar ist. Dies kann mit einem einfachen Ping-Befehl oder über einen speziellen Health Endpoint realisiert werden.
- Manuelle Steuerung: Es sollte möglich sein, dass beispielsweise der Systemadministrator manuell einen bestimmten Status setzen kann.
- Overhead: Ein Circuit Breaker kann Ziel sehr vieler Anfragen werden und sollte daher nur einen kleinen Overhead erzeugen.
- Ressourcen differenzieren: Angenommen in einem Data-Store-Service ist eine Datei nicht mehr erreichbar, dann könnte es sein, dass der Circuit Breaker in den Status „Open“ wechselt und alle Anfragen blockiert, obwohl alle anderen Dateien des Data-Stores problemlos erreichbar sind.
- Fehlgeschlagene Anfragen erneut ausführen: Statt in einem Fehlerfall direkt eine Exception zu werfen, können auch alle Anfragen gesammelt werden und ausgeführt werden, sobald der Service wieder erreichbar ist.

Competing Consumers Pattern

Falls eine Cloud basierte Anwendung viele Anfragen bearbeiten muss, die Anzahl der Anfragen aber stark schwankt, kann es sinnvoll sein, eine „Message Queue“ zu verwenden. Mit Hilfe dieser Queue kann die Anwendung (Producer) die Nachrichten zu anderen Anwendungen (Consumer Services) umleiten. Diese Consumer Services können die Nachrichten dann Asynchron und frei Skalierbar abarbeiten.

Vorteile

Da die Anzahl der Consumer Services variierbar ist, ermöglicht dieses Pattern eine inhärente Last Verteilung. Falls viele Anfragen eintreffen können weitere Instanzen des Consumer Service gestartet werden. Diese können, wenn die Anzahl der Anfragen abnimmt, abgeschaltet werden. Da die Message Queue als Buffer dient, ist eine Synchronisierung zwischen Producer und Consumer nicht nötig. Die Queue führt außerdem dazu, dass jede Anfrage mindestens einmal bearbeitet wird. Falls ein Consumer während der Bearbeitung einer Anfrage abstürzt, kann die Queue so konfiguriert werden, dass eine andere Instanz die Anfrage erneut bearbeitet.

Nachteile

Durch die Verwendung einer Message Queue ist nicht garantiert, dass die Nachrichten in der gleichen Reihenfolge, wie sie eingetroffen sind bearbeitet werden. Da die Queue nur dafür sorgt, dass jede Nachricht mindestens einmalig bearbeitet wird, kann es vorkommen, dass eine Nachricht mehrfach abgearbeitet wird. Da durch die Queue keine direkte Kommunikation zwischen Producer und Consumer erfolgt, sie also füllig entkoppelt sind, ist es schwierig die Ergebnisse des Consumers zurück an den Producer zu liefern. Auch die Message Queue kann bei zu vielen Nachrichten zu Flaschenhals werden.

Fazit

Das Pattern sollte verwendet werden, wenn die Anzahl der Anfragen stark variiert und die Aufgaben der Anwendung gut Asynchron zu verarbeiten sind. Außerdem müssen die Aufgaben unabhängig voneinander sein und parallel verarbeitet werden können. Falls die Aufgaben nicht gut voneinander getrennt werden können oder die Reihenfolge der Aufgaben wichtig ist, ist das Pattern nicht sinnvoll zu verwenden.

Compute Resource Consolidation Pattern

Dieses Pattern kann verwendet werden, um Berechnungen mehrerer Operationen oder Aufgaben in einer zusammenhängenden Einheit durchzuführen. Hierdurch kann die Ressourcennutzung optimiert und die Kosten gesenkt werden.

Hintergründe und Probleme

Eine Cloud Anwendung enthält häufig viele Operationen und Aufgaben, welche meist standartmäßig in getrennten Einheiten, beispielsweise auf verschiedenen virtuellen Maschinen, verarbeitet werden. Dieses Konzept verursacht eine große Menge an verarbeitenden Einheiten, welche das System sehr komplex machen. Außerdem verursachen diese Einheiten Kosten, auch wenn sie inaktiv oder unausgelastet sind. Ein weiteres Problem sind die erschwerten Kommunikationswege, welche durch die Aufteilungen entstehen.

Lösung

Um den genannten Problemen entgegenzuwirken, können mehrere Aufgaben und Operationen in eine verarbeitende Einheit zusammengelegt werden. Diese müssen dafür zuallererst gruppiert werden. Eine verbreitete Möglichkeit ist es zu prüfen, welche Aufgaben und Operationen ähnliche Anforderungen und Verarbeitungszeiten besitzen.

Zusammengefügt können diese Einheiten je nach Auslastung skalieren und somit mehr Rechenleistung, in Form von weiteren berechnenden Einheiten, hinzufügen oder bei geringer bzw. gar keiner Nutzung Rechenleistung abtreten. Wichtig ist es jedoch darauf zu achten, wie die Aufgaben und Operationen, welche gruppiert werden sollen während der Nutzung skalieren. Werden beispielsweise Operationen gruppiert, welche teilweise viele neue Einheiten für die Verarbeitung benötigen und der andere Teil nur auf bestimmte Ereignisse wartet, könnten unnötigerweise bei Ressourcenanforderungen auch dem Warteprozess erhöhte Ressourcen zugeschrieben werden, obwohl diese nicht benötigt werden. Verarbeitende Einheiten können unterschiedliche Rechenleistungen wiederspiegeln. Daher sollte darauf geachtet werden, dass Einheiten mit hohen Rechenleistungen, welche dementsprechend teurer sind, eine durchgängige und nicht zu schwache Auslastung haben. Daher sollten Operationen und Aufgaben welche viel Leistung in kleinen Schüben benötigen zusammengelegt und einer leistungsstarken Einheit zugewiesen werden. Dagegen sollten langwierige und viel Ressourcen benötigende

Aufgaben und Operationen nicht zusammengefasst werden, da es zu einer Überlastung der Einheit führen kann. Eine kontinuierliche Analyse der Gruppierungen hilft dabei diese optimal anzutragen.

Berücksichtigungen

Bei der Implementierung des Patterns sollten unbedingt vorher bestimmte Aspekte berücksichtigt werden. Hierzu zählt wie bereits beschrieben die Entscheidung der Gruppierung anhand ähnlicher Skalierungsanforderungen und Verarbeitungszeiten. Jedoch sollten Operationen mit hohen Ressourcenanforderungen nicht zusammen in eine Einheit gruppiert werden. Ebenfalls sollte darauf geachtet werden, dass wenn eine Operation oder Aufgabe regelmäßig gewartet wird und dies ein Neuaufsetzen der verarbeitenden Einheit erfordert, alle gruppierten Operation der Einheit ebenfalls gestoppt und neu geladen werden müssen. Hinzukommend kann es sein, dass die Operationen sich denselben Kontext in der Einheit teilen und somit auf Ressourcen des anderen zugreifen können. Die hierdurch mögliche Sicherheitslücke weitet sich mit der Anzahl der sich teilenden Operationen in einer Einheit aus. Ebenso können Fehler die von einer Operation ausgelöst wurden, das Verhalten anderer Operationen in derselben Einheit einschränken. Die Komplexität steigt durch das Gruppieren von Operationen zu einer Einheit, was die Wartbarkeit und Testbarkeit verschlechtert. Außerdem ist zu berücksichtigen, dass die Cloud Infrastruktur regelmäßige Aufräumarbeiten tätigt. Einheiten mit mehreren lang laufenden Operationen sollten daher so konfiguriert werden, dass diese nicht vorzeitig unterbrochen werden. Beim Entwickeln einer Operation sollte darauf geachtet werden, diese so zu entwerfen, dass eine Änderung der Umgebung keine Auswirkung auf die Operation hat und weiterhin funktioniert.

Anwendungsfälle

Dieses Pattern sollte für wenig bzw. regelmäßig keine Leistung benötigende Operationen und Aufgaben verwendet werden, welche in einer eigenen Einheit nicht kosteneffektiv sind. Sobald eine Operation Ausfallsicher sein soll oder strikten Sicherheitsanforderungen entsprechen muss ist von einer Anwendung des Pattern abzusehen und eine eigene Einheit zu verwenden.

Command and Query Responsibility Segregation (CQRS) Pattern

Das CQRS-Pattern führt eine Teilung zwischen den `data transfer objects` (DTO) ein, sodass das Schreiben von Daten (Command) und Lesen von Daten (Query) über zwei verschiedene Interfaces - respektive DTOs - geschieht.

Problemstellung

Klassisch geschehen alle Schreib- und Leseoperationen auf ein Entitiy (in der Datenbank) über ein und das selbe DTO und dabei wird immer mit ein und der selben Datenbank kommuniziert. Das heißt, alle CRUD-Operationen haben das selbe Zielobjekt. Zum Beispiel wird ein Kunden-DTO über den `data access layer` (DAL) angefordert und auf dem Bildschirm angezeigt. Ein Nutzer der Applikation aktualisiert nun einige Felder des Kunden-DTO und speichert. Die Applikation wird nun diesen DTO über den DAL auf die Datenbank speichern. Ein DTO wurde also sowohl für das Lesen, als auch für das Schreiben genutzt.

Diese Art und Weise mit den CRUD-Operationen und DTOs umzugehen, funktioniert gut, solange die Business Logik, die auf diese Daten und Operationen angewandt werden soll, gering ist.

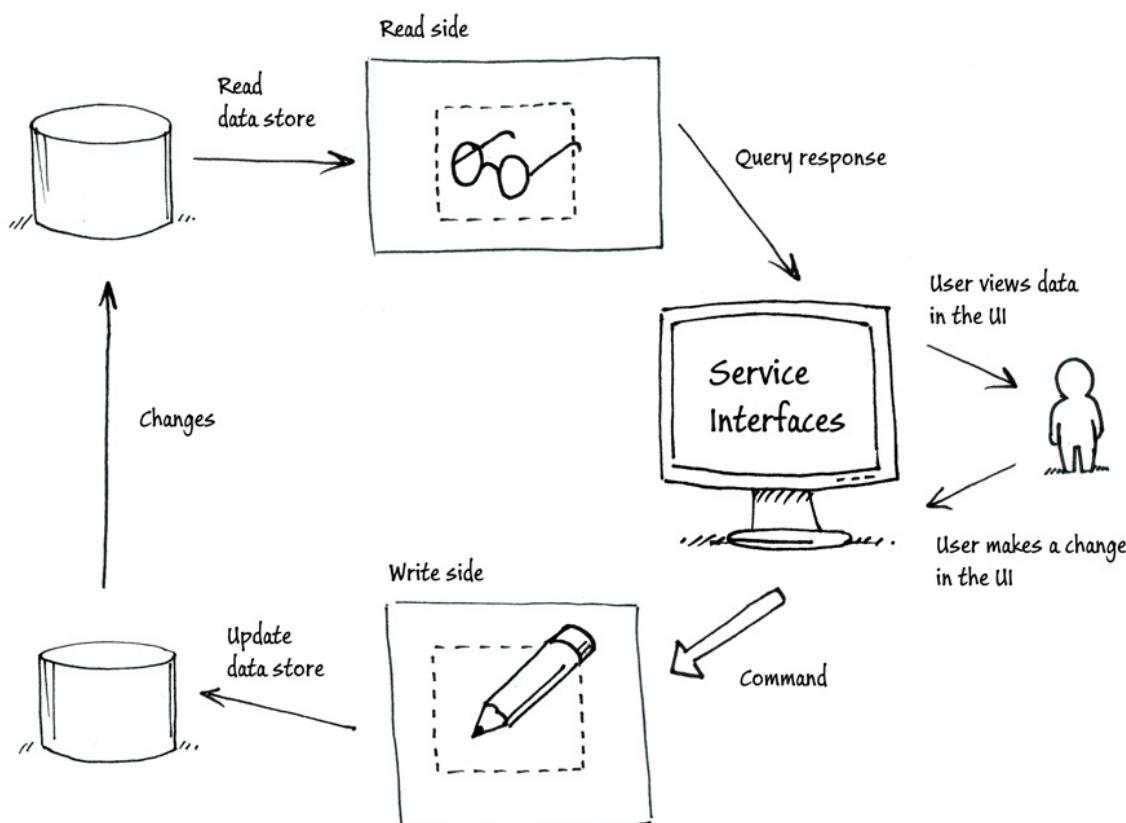
Die Nachteile:

- Oft kommt es zu Fehlanpassungen zwischen den Lese- und Schreibdarstellungen der Daten; zum Beispiel zusätzliche Spalten oder Eigenschaften, die korrekt aktualisiert werden müssen, obwohl sie nicht als Teil einer Operation benötigt werden.
- Viele Nutzer, die parallel auf die Datenbank zugreifen, steigern die Gefahr, dass die Integrität der Daten nicht gewährleistet werden kann. Die ausgiebige Nutzung von Transaktions-Mechanismen des RDBMS, sind unumgänglich. Die Performance der Applikation wird dadurch nicht steigen und die Komplexität der Querys nimmt zu.
- Die Verwaltung der Sicherheit und Berechtigungen werden umständlicher, da jede Entität sowohl Lese- als auch Schreiboperationen unterliegt, die versehentlich Daten im falschen Kontext ausliefern können.

Lösung

CQRS trennt die Lese- und Schreiboperationen und nutzt dafür unterschiedliche Interfaces. Dadurch kann sogar eine Trennung der `data models` erfolgen (nicht verwechseln mit den DTOs). Durch Datenmodelle, die für Lese- bzw. Schreiboperationen optimiert wurden, kann die Komplexität der Implementierung reduziert werden. Ein Nachteil für einen Entwickler ist, dass mit diesen CQRS-Modellen nicht einfach Code generiert werden kann (getter/setter).

Diese Datenmodelle können die selbe physische Datenbank ansprechen, jedoch macht es aus Sicht der Sicherheit, der Performance und der Skalierbarkeit Sinn, auch die Datenbanken zu trennen. Die Lese-Datenbank kann dann ein `read-only`-Duplikat der Schreib-Datenbank sein. Dadurch kann die Skalierung der Infrastruktur deutlich vereinfacht werden, da (meistens) deutlich mehr Leseoperationen in einer Applikation stattfinden, als Schreiboperationen. Das heißt, es können mehrere Lese-Datenbanken aufgesetzt werden, um so die Arbeitsgeschwindigkeit der Applikation zu erhöhen. Außerdem kann die Lese-Datenbank und die Schreib-Datenbank dahingehend optimiert werden, dass diese verschiedene Normalisierungsgrade haben. Die Schreib-Datenbank kann zum Beispiel nach der dritten Normalform optimiert werden. Die Lese-Datenbank kann wiederum eine denormalisierte Datenbank verwenden, um möglichst schnelle Abfragen zu ermöglichen.



(Abbildung 1: Eine mögliche Implementierung des CQRS-Pattern. Quelle: <https://msdn.microsoft.com/en-us/library/jj591573.aspx>)

Wann sollte CQRS verwendet werden

- Wenn die Applikation den parallelen Zugriff mehrerer Nutzer auf die selben Daten ermöglicht und es häufig vorkommt.
- Wenn bei den Schreiboperationen viel Business-Logik auf die Daten angewandt wird und diese Daten immerzu validiert werden, um die Konsistenz der Daten zu gewährleisten. Das Lese-Modell gibt nur eine einfache DTO wieder und besitzt keinerlei Business-Logik in diesem Fall.
- Wenn die Leseoperationen deutlich öfter stattfinden als Schreiboperationen und das Ziel der schnellere Zugriff auf die Daten ist.
- Wenn bei dem Entwicklungsprozess Entwickler-Teams mit unterschiedlichen Erfahrungswerten arbeiten. So kann das erfahrenere Team die komplexen Schreib-Modelle implementieren und das weniger erfahrenere Team implementiert die Lese-Modelle und kümmert sich um das User Interface.

Wann sollte CQRS nicht verwendet werden

- Wenn die Business-Logik relativ einfach ist.
- Wenn eine einfache Bedienoberfläche des CRUD-Typs und die damit verbundenen Datenzugriffsoperationen ausreichen.
- Wenn es sich um die Umsetzung durch das gesamte System handelt. Es gibt Komponenten in einem System, da kann die Implementierung des CQRS leichter sein. Diese sollten bevorzugt umgesetzt werden. Unnötige Komplexität sollte vermieden werden.

Event Sourcing Pattern

Bei dem Event Sourcing Pattern werden alle Veränderungen des Zustands eines Systems oder der Daten als Sequenz von Events gespeichert.

Problematik

Das herkömmliche [create, read, update, and delete \(CRUD\)](#) Model birgt einige Nachteile: Zunächst werden bei einem Update die alten Werte mit neuen Werten überschrieben. Das konventionelle Update impliziert demnach eine Löschung (Delete) der alten Daten, sofern kein Logfile existiert, welches alle Änderungen aufzeichnet. Bei einem Updatevorgang werden die Daten meist gesperrt, wodurch andere Benutzer und Anwendungen keine Änderungen an den Daten machen können. Des Weiteren werden bei einem CRUD System die Operationen direkt auf dem Data Store durchgeführt. Hieraus resultierte eine schlechtere Performance und Reaktion, sowie Skalierbarkeit des Systems.

Lösung

Die Anwendung beschreibt die Änderungen der Daten als Events. Auch das Löschen der Daten ist ein Event. Diese Events werden asynchron an den Event Store verschickt und dort persistent abgelegt. Eine Löschung der Events im Event Store ist nicht vorgesehen, die Events sind immutable. Der Event Store publishes die Events an einen Topic, sodass alle Consumer die den Topic abonniert haben, die Events bekommen und verarbeiten können. Typische Abonnenten sind z.B. Controller für die Materialized View, die daraufhin die View aktualisieren. Für die Vereinfachung der Präsentationsschicht wird meist für jede Entität der derzeitige Datenstand als materialized View gespeichert.

Das Event Sourcing Pattern wird meist zusammen mit dem [CQRS-Pattern](#) eingesetzt.

Vorteile

Der Prozess, der die Events verarbeitet, kann von dem Prozess, welcher das Event erstellt hat, entkoppelt werden, sodass die Verarbeitung später erfolgen kann. Außerdem entstehen durch die Entkopplung neue Skalierungsmöglichkeiten. Der gesamte Systemänderungsverlauf bzw. die Datenhistorie kann durch eine erneute Betrachtung der

Events erfolgen. Dies ist hilfreich z.B. für das Debugging und Testen der Systeme. Zudem kann hier auch die Business Logik einer Historisierung der Entitäten erfolgen. Generell hat der Event Log einen hohen Business Value.

Berücksichtigungen

Durch eine fehlende Allokation der Daten, kann es zu Misständen kommen. Besonders wenn mehrere verschiedene Anwendungen oder auch Instanzen Events publishen wollen. Die korrekte Reihenfolge kann so auch nicht immer gewährleistet werden. Ein Timestamp der Events oder eine autoinkrementelle ID-Nummer kann hier für eine ACID-Fähigkeit Abhilfe schaffen. Außerdem ist immer eine Verzögerung der zwei entkoppelten Prozesse zu bemerken, wenn das Event gepublished wurde wartet es noch auf seine Verarbeitung.

Der aktuelle Datenstand wird durch die Verarbeitung aller Events bestimmt. Bei einer Vielzahl an Events empfiehlt es sich Snapshots zu machen, um eine Verarbeitung aller Events zu umgehen, sodass nur die Events ab dem Snapshot verarbeitet werden müssen.

Empfehlungen für die Benutzung des Patterns

Die Vorteile des Event Sourcing Patterns überwiegen, wenn

- die beiden Prozesse (Datenveränderung, Datenverarbeitung) entkoppelt werden sollen, Flexibilität des Systems
- Datenzustandverläufe von Interesse sind.
- eine hohe Skalierbarkeit erwartet wird.
- schnelle und einfache Anpassung an neue Business Logik gefordert wird.

Eine Verwendung des Patterns ist nicht empfehlenswert, wenn

- die traditionellen CRUD Mechanismen ausreichend sind, Historisierung der Daten nicht benötigt wird.
- ACID Fähigkeit vorhanden sein muss.
- Echtzeitupdates der Views gefordert werden.
- Veränderungen der Datenbestände nicht erfolgen.

External Configuration Store Pattern

Definition

Mit dem *External Configuration Store Pattern* werden Konfigurationsinformationen an einer zentralen Stelle verwaltet. Diese werden von allen Anwendungen und den Instanzen von Anwendungen verwendet. Dieses Pattern soll für ein vereinfachtes Management und eine bessere Steuerung der Konfigurationen beitragen.

Probleme mit herkömmlichen Lösungen

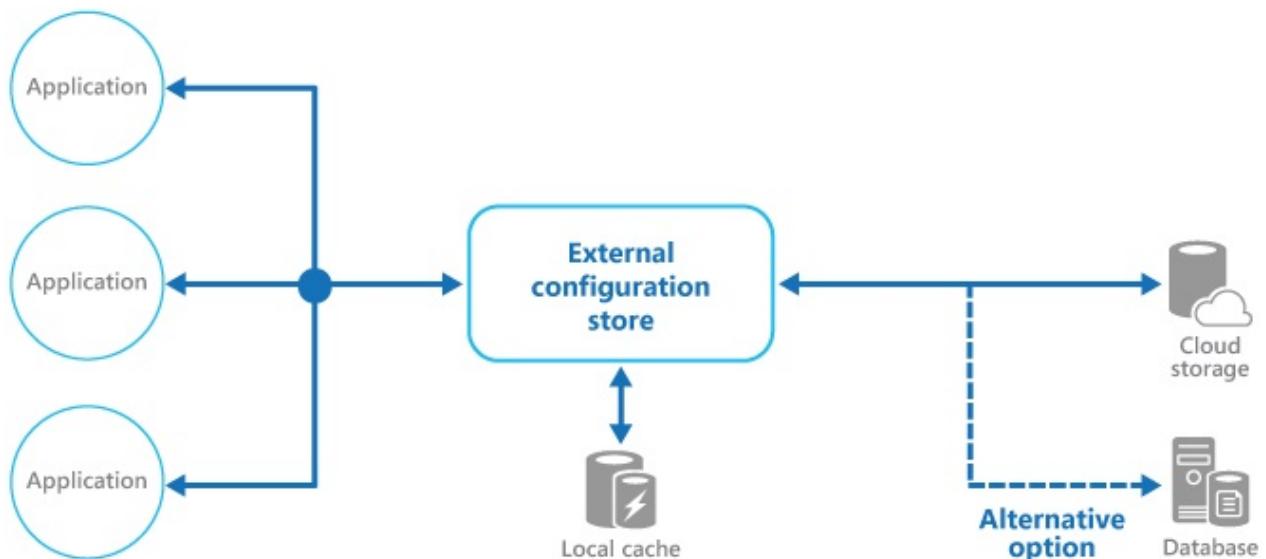
Eine Vielzahl von Anwendungen haben Konfigurationsinformationen in Dateien gespeichert, welche in Laufzeitumgebungen einer Cloud ausgeführt werden. Bei manchen Anwendungen lassen sich diese verändern und die Anwendung reagiert sofort darauf. In anderen Fällen müssen diese jedoch neu gestartet werden, was die Anwendung in dieser Zeit unerreichbar macht und zusätzlichen Administrationsaufwand erfordert. Ein weiteres Problem ist, wenn in einer Cloud alle Instanzen einer Anwendung geupdated werden und diese während dem Vorgang unterschiedliche Konfigurationen verwenden. Die Authentifizierung von Benutzern muss berücksichtigt werden.

In vielen Anwendungsfällen ist es von Vorteil, wenn Konfigurationen von einer zentralen Stelle für Anwendungen gemeinsam genutzt werden.

Das Pattern

Das *External Configuration Store Pattern* sieht es vor, dass Konfigurationsinformationen an einer externen, zentralisierten, Stelle von einem nicht-flüchtigen Speichertyp gespeichert werden. Über definierte Schnittstellen können Konfigurationen gelesen und verändert werden. In einem typischen Cloud Szenario könnte der Speicher beispielsweise ein Cloud-basierter Speicherservice sein. Die Schnittstellen müssen konsistent und einfach zu verwenden sein. Die Konfigurationsinformationen sollten in einem strukturierten Format wiedergegeben werden.

Je nach Typ des Speichers kann es von Vorteil sein, wenn die Softwarekomponente, welche für Konfigurationsspeicher zuständig ist, die Konfigurationsinformationen in einen lokalen Cache einliest um die Performance zu erhöhen.



Überblick des External Configuration Store pattern mit optionalem lokalen Cache. (Quelle: <https://docs.microsoft.com/en-us/azure/architecture/patterns/external-configuration-store>)

Wann das Pattern verwendet werden kann

- Wenn sich Anwendungen und Instanzen einer Anwendung Konfigurationsinformationen teilen oder wenn Standardeinstellungen auf mehreren Anwendungen und Instanzen einer Anwendung erzwungen werden müssen
- Wenn das Standard Konfigurationssystem nicht alle geforderten Konfigurationseinstellungen unterstützt
- Wenn ein ergänzender Speicher für einige Konfigurationen benötigt wird
- Wenn ein vereinfachter Mechanismus für die Administration von mehreren Anwendungen benötigt wird

Federated Identity Pattern

Durch die Verwendung dieses Patterns, kann die Entwicklung sowie die Administrierung von Anwendungen vereinfacht werden, indem die Authentifizierung an einen externen Anbieter weitergereicht wird.

Problematik

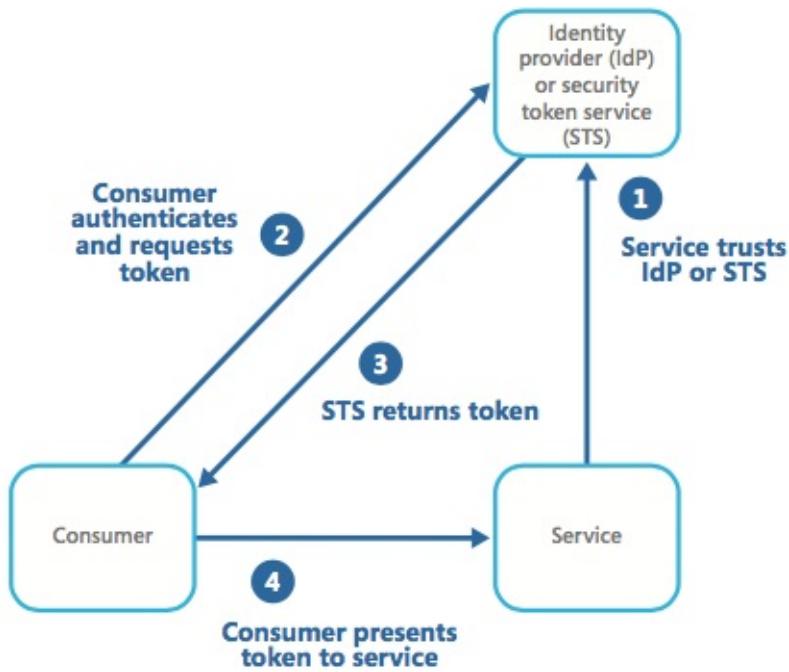
Anwender müssen im Alltag mit vielen verschiedenen Anwendungen arbeiten, für welche sie unterschiedliche Anmeldeinformationen haben. Dies führt zu negativen Erfahrungen für die Anwender, da sie sich viele Passwörter merken müssen. Des Weiteren ist es für die Administratoren ein sehr hoher Aufwand alle Anmeldeinformationen zu verwalten. Neben dem Punkt des hohen Verwaltungsaufwands, führt es auch zu einem Sicherheitsrisiko falls ein Mitarbeiter die Firma verlässt, müssen viele einzelne Anmeldeinformationen gelöscht werden.

Lösung

Um der vorangegangenen Problematik entgegenzuwirken, sollte eine Authentifizierungsfunktion implementiert werden, welche Federated Identity's unterstützt. Das bedeutet, dass die Authentifizierungsfunktion vom Anwendungscode getrennt wird und die Authentifizierung an einen Identity Provider (IdP) wie z.B. Microsoft, Google, Yahoo! oder Facebook weitergereicht wird.

Dies vereinfacht die Entwicklung sowie die Verwaltung der Nutzerdaten. Außerdem hat der Benutzer dadurch die Möglichkeit, sich neben der Standard Authentifizierung über einen frei wählbaren IdP zu authentifizieren. Dadurch muss sich der Anwender keine neuen Anmeldeinformationen merken. Des Weiteren wird durch die Verwendung eines IdP die Authentifizierung von der Autorisierung entkoppelt.

Abbildung 1 veranschaulicht die Funktionsweise des Federated Identity Pattern, wenn eine Anwendung auf einen Dienst zugreifen muss, der eine Authentifizierung mittels eines IdP erfordert.



(Abbildung 1: Übersicht über

eine Authentifizierung)

Berücksichtigungen

Wenn man dieses Pattern verwendet, sollte man verschiedene Aspekte beachten:

- Wenn die Anwendung auf mehrere Rechenzentren verteilt werden soll, muss der Identitätsmanagement Mechanismus in den Rechenzentren ebenfalls bereitstehen, um die Zuverlässigkeit und Verfügbarkeit der Anwendung zu gewährleisten.
- Wenn man einen Identity Provider bereitstellt muss beachtet werden, dass in der Regel keine Informationen über den authentifizierten Benutzer außer einer E-Mail-Adresse und einem Namen zur Verfügung stehen.
- Wenn mehr als ein Identity Provider für die Anwendung bereitsteht muss erkannt werden, zu welchem Identity Provider der Anwender weitergeleitet werden muss.

Anwendungsfälle

Dieses Pattern wird sehr häufig in SaaS (Software as a Service) Anwendung verwendet, um den Nutzern neben der Standard Anmeldung, eine Anmeldung über ihre Accounts für die Sozialen Netzwerke bereitzustellen.

Weitere Anwendungsfälle sind Enterprise Anwendungen, Business-to-Business Anwendungen, Anwendungen welche mit dritt Anbieter Software interagieren sowie Unternehmen, die ihre IT-Systeme zusammengefasst haben.

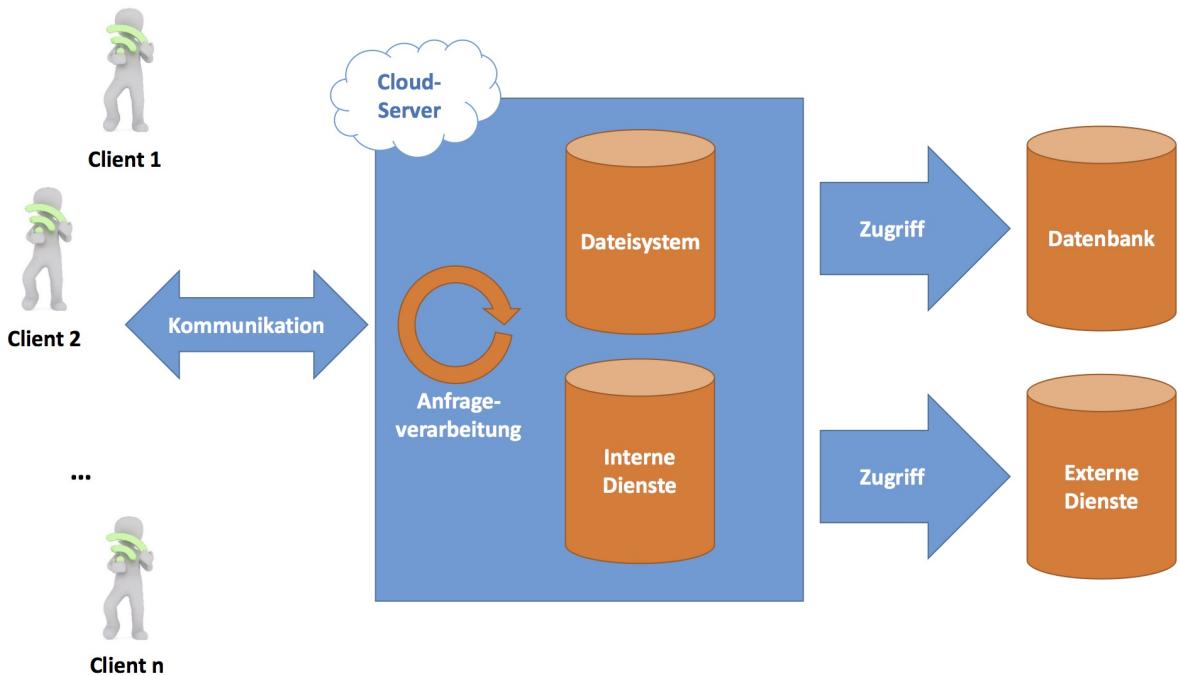
Gatekeeper Pattern

Cloud Computing erfreut sich zunehmender Beliebtheit. Das Auslagern von Diensten und Daten an einen externen Ort bietet diverse Vorteile im Hinblick auf Kosteneffizienz und Skalierbarkeit. Auch die zentrale Zugänglichkeit ist ein entscheidender Faktor, der die fortlaufende Verbreitung des Trends begünstigt. Jeder Benutzer kann zu jeder Tages- und Nachtzeit und an jedem Ort der Erde auf die ihm zugehörigen Informationen und Dienste zurückgreifen. Alles, was er dazu benötigt, ist eine Netzwerkverbindung sowie ein netzwerkfähiges Endgerät. Die zentrale Lage von Ressourcen macht jedoch auch einen potentiellen Schwachpunkt der Cloud-Architektur aus.

Problematik

Ein cloud-typisches Interaktionsszenario beinhaltet zwei verschiedene Akteure. Vom Client bzw. Endnutzer geht eine initiale Anfrage aus. Beispielsweise möchte er sich authentifizieren, vertrauliche Auskünfte beantragen oder eine Anwendung nutzen. Die Anfrage wird ohne Umwege an einen zentralen Server gesendet und anschließend von diesem verarbeitet. In Abhängigkeit von der Forderung des Clients führt der Server verschiedene Aktionen aus. Beispielhaft sei hierfür die Kommunikation mit Datenbanken oder das Anstoßen von externen Diensten. Die gesammelten Ergebnisse werden letztlich vom Server wieder an den Client übermittelt, sodass die Anfrage abgeschlossen werden kann.

Eine Problematik, die sich durch diese Kommunikationsarchitektur ergibt, beruht auf der fehlenden Trennung der Programmlogik innerhalb des Servers (siehe Abb. 1). Dieser dient zum einen als direkter Kontaktspunkt für eine beliebige Anzahl an Clients. Hierunter können sich sowohl rechtmäßige Benutzer, welche einen tatsächlichen Anspruch auf ihre jeweilige Forderung haben, als auch unrechtmäßige Benutzer befinden, denen keine Leistung erbracht werden darf. Der Server ist dementsprechend öffentlich und von jedem Endgerät aus kontaktierbar. Zum anderen ist derselbe Rechenknoten für die Orchestrierung und Instanziierung von Diensten sowie das Auslesen und die Übermittlung von vertraulichen Daten zuständig. Hier lässt sich ein Widerspruch erkennen. Auf derselben Rechnerinstanz findet gleichzeitig die Verarbeitung von unbekannten Anfragen sowie der Umgang mit sensiblen Informationen bzw. Diensten statt.



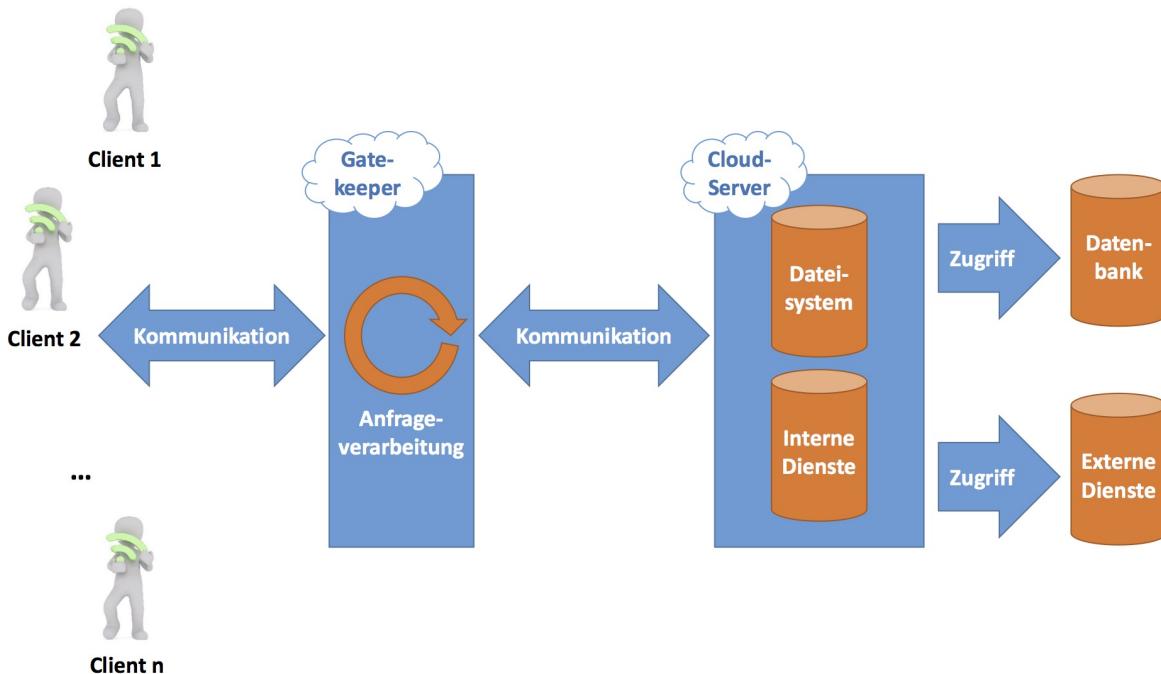
(Abbildung 1: Klassische Cloud-Architektur)

Potentielle Sicherheitslücken, die die beschriebene Architektur beinhaltet, lassen sich im Wesentlichen dem Bereich Security zuordnen. Aufgrund der direkten Kommunikationsmöglichkeit wird es Angreifern erleichtert, sich in das System einzuschleusen. Anfragen können beispielsweise so formuliert werden, dass der Zugriff auf eine dahinterliegende Datenbank ohne Umwege möglich ist. Das Auslesen oder Manipulieren von gesamten Informationsbeständen wäre eine denkbare Folge. Auch das Anstoßen sicherheitskritischer Dienste seitens eines Eindringlings kann ernsthafte Konsequenzen für unschuldige Benutzer sowie den Cloud-Betreiber nach sich ziehen. Neben der Sicherheit wird auch die Verfügbarkeit von Cloud-Diensten durch die direkte Kommunikationsmöglichkeit zwischen Client und Server gefährdet. Das kontinuierliche Senden von Anfragen mit dem Ziel einen Serverausfall herbeizuführen (Denial of Service-Angriff), ist effektiv möglich.

Lösung

Zur Lösung der vorangegangenen Problematik dient das sogenannte Gatekeeper Pattern. Die Idee, die diesem Architekturmuster zugrunde liegt, ist keinesfalls neu, sondern beruht auf zwei bestehenden Techniken. Zum einen wird auf das Prinzip der Firewall zurückgegriffen, welche Netzwerkzugriffe auf Systeme beschränkt. Des Weiteren dient das Fassade-Pattern aus der objektorientierten Programmierung als Grundlage. Der Kerngedanke dieses Entwurfsmusters besteht darin, eine Vielzahl von möglicherweise komplexen Schnittstellen vor dem Benutzer zu verbergen und ihm einen unkomplizierten Zugang zu einem System bereitzustellen.

In der Praxis kann der Gatekeeper als eine zusätzliche Serverinstanz verstanden werden, welche eine Vermittlerfunktion zwischen den Clients und dem Cloud-Server einnimmt (siehe Abb. 2). Anfragen seitens der Benutzer werden initial an den Gatekeeper geleitet und von diesem verarbeitet. Anschließend stellt der Gatekeeper eine Anfrage an den Cloud-Server, der ihm im Bestfall ein passendes Ergebnis liefert. Dieses übermittelt der Gatekeeper letztlich an den Benutzer und schließt somit die Anfrage ab.



(Abbildung 2: Anwendung des Gatekeeper Pattern)

Die Aufgabe des Gatekeepers besteht darin, die direkte Kommunikation von Client und Server zu unterbinden. Der Benutzer soll demnach keine Kenntnis vom Cloud-Server besitzen, sondern lediglich von der zwischengelagerten Schnittstelle. Des Weiteren trägt der Gatekeeper für eine geeignete Verarbeitung der Benutzeranfragen Sorge. Idealerweise werden nur solche Anfragen an den Cloud-Server weitergeleitet, die von rechtmäßigen Benutzern stammen bzw. keine Gefahr für die Sicherheit des Systems sowie die Integrität der Daten darstellen. Ggf. kann auch eine Überarbeitung der Anfragen dabei helfen, dass keine Sicherheitsrisiken eingegangen werden. Es ist hervorzuheben, dass der Gatekeeper ausschließlich Anfragen behandelt und übermittelt. Er hat beispielsweise keinen direkten Zugriff auf die Daten bzw. Dienste des Servers. Hierin liegt zugleich die Kernidee des Patterns. Sollte der Intermediär von Angreifern befallen werden, resultiert keine Gefahr für die Benutzer bzw. den Cloud-Betreiber. Der Eindringling besitzt mithilfe des übernommenen Gatekeepers keine Kontrolle über den Cloud-Server und hat demnach auch keinen Zugriff auf vertrauliche Ressourcen.

Nachteile

Die Anwendung des Gatekeeper Patterns hat positive Auswirkungen auf die Sicherheit von Cloud-Diensten. Jedoch ergeben sich auch Nachteile im Bezug auf Performance und Erreichbarkeit. Die Kommunikation über einen Vermittler hat zwangsläufig einen negativen Einfluss auf die Übermittlungsgeschwindigkeit von Informationen. Es ist nicht mehr ausreichend, Daten einmalig zu übermitteln. Dieser Vorgang, der auch das ver- und entpacken von Dateien beinhaltet, muss nun zweimal vollzogen werden. Insbesondere bei zeitkritische Anwendungen kann dies zu Problemen führen. Ein weiterer Nachteil ist die geringere Ausfallsicherheit, die sich durch den Einsatz eines zusätzlichen Rechenknotens ergibt. Sollte der Gatekeeper versagen, kann auch der Cloud-Server nicht mehr angesprochen werden. Um dieser Gefahr entgegenzuwirken, ist es sinnhaft, mehrere Instanzen des Gatekeepers zu betreiben. Hierdurch kann auch gleichzeitig ein erhöhtes Lastaufkommen bewältigt werden.

Health Endpoint Monitoring Pattern

Ziel des Patterns ist es funktionale Checks innerhalb einer Applikation auszuführen. Diese werden von externen Kontroll-Tools über entsprechende Zugangspunkte angestoßen und in bestimmten Intervallen abgerufen. Das Pattern hilft somit die korrekte Ausführung verschiedener Applikationen und Services zu gewährleisten.

Einsatzgebiet

- Webseiten und Webapplikationen auf Verfügbarkeit und korrekte Funktionsweise überwachen
- Middle-tier oder geteilte Services überwachen, um Fehlerquellen zu isolieren, die andere Applikationen beeinflussen könnten
- Zum Erweitern von bereits vorhandenen Überwachungselementen wie dem Logging und protokolierte Performanzkennzahlen einer Applikation

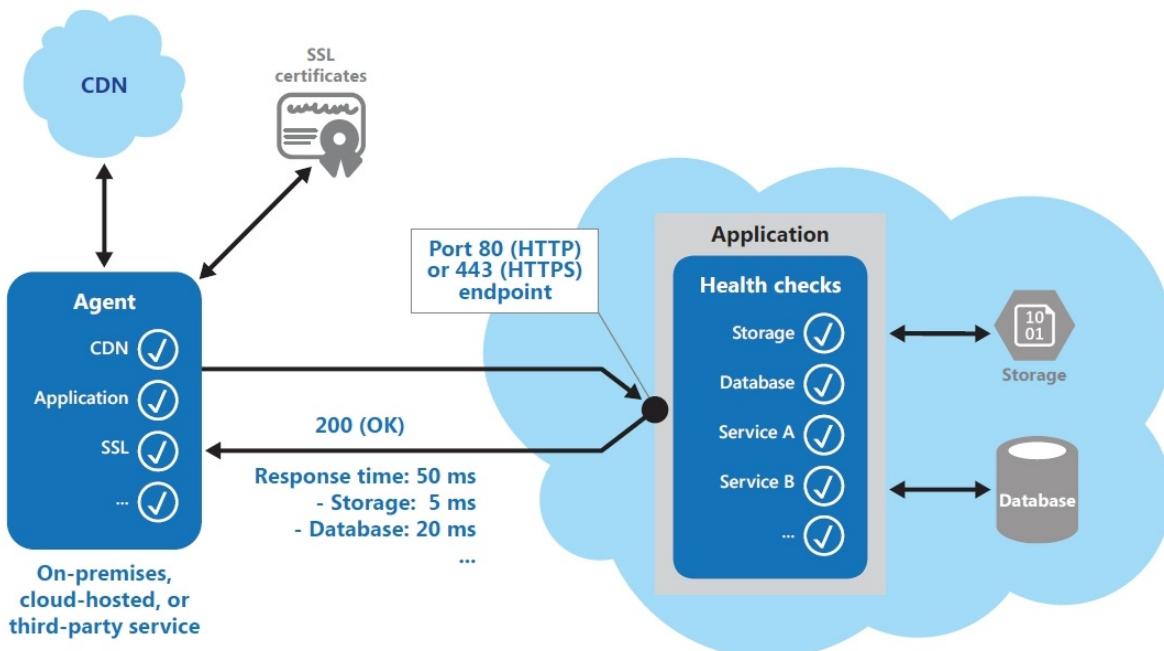
Problematik

Es ist üblich bzw. in vielen Fällen eine feste Voraussetzung, dass Webapplikationen oder Services auf ihre korrekte Ausführung überprüft werden. Bei einem Service in der Cloud kommen jedoch neue Abhängigkeiten und potenzielle Problemquellen dazu. Der eigene Service hängt potenziell von weiteren Services des Plattformanbieters oder anderen ab und über die Hostingumgebung hat man keine volle Kontrolle.

Auch hängt die Performance und Verfügbarkeit letzten Endes von der Hardware und der Netzwerkverbindung des Clouddienstleisters ab. Es ist also zwingend notwendig in regulären Abständen die Verfügbarkeit der eigenen Services in der Cloud zu überprüfen.

Das Pattern

Das Pattern besteht aus 2 wesentlichen Vorgängen bzw. Elementen. Zum einen aus dem Kontroll-Tool, das die Kontrollanfrage an einen frei konfigurierbaren Endpunkt einer Cloud Applikation stellt. Zum anderen die Cloud Applikation, die auf Anfrage ihre jeweiligen Checks durchführt. Diese können Kontrollen von weiteren genutzten Services inkludieren, sowie Kontrollen der Datenbankanbindungen und des Cloudspeichers. Ein entsprechender Response Code wird an das Kontroll-Tool zurückgesendet (siehe Abbildung 1).



(Abbildung 1: Übersicht des Patterns)

Das Kontroll-Tool analysiert anschließend den gesendeten Response Code gegen ein Set von frei konfigurierbaren Regeln, die festlegen wie welche Ergebnisse zu bewerten sind.

Zusätzlich können weitere Checks durchgeführt werden.

- Detaillierte Analyse des Inhalts der Antwort abseits des Codes, der z.B. weitere Information über teilweise fehlgeschlagene Tests geben könnte, selbst wenn der gegebene Response Code 200(OK) war
- Das Messen der Antwortzeit zur Überprüfung der Performance der Applikation und der Netzwerkgeschwindigkeit
- Auslaufende SSL Zertifikate abfragen
- Messen eines DNS Lookups auf die URL der Applikation, sowie Kontrolle der zurückgegebenen URL des DNS Lookups auf ihre Korrektheit

Es ist empfehlenswert die Tests von verschiedenen Kontroll-Tools von verschiedenen Standorten abfragen zu lassen um evtl. Unterschiede in der Verbindung ausfindig zu machen. Dies kann potenziell auch die Wahl beeinflussen wo eine Applikation deployt wird. Die Tests sollten auch gegen die Service-Instanzen von Kunden laufen, um zu überprüfen ob die Applikation für alle Kunden korrekt funktioniert. Wenn also ein Kunde seinen Cloudspeicher auf mehrere Standorte verteilt hat müssen alle Standorte kontrolliert werden.

Implementierung: Was man beachten sollte

- Wie genau sollten die **Antwort der Cloud Applikation** aussehen? Der minimalste Ansatz, ein simpler HTML Response Code, könnte potenziell nicht genug Informationen übermitteln.

- **Mehrere Endpunkte** für unterschiedliche Services innerhalb der Applikation konfigurieren, so sollte jeder Check (Datenbank, Storage..) einzeln angesprochen werden können oder über einen weiteren Endpunkt übergreifend. Für den jeweiligen Endpunkt können unterschiedliche Regeln (z.B. erwartete Antwortzeiten) definiert werden.
- Man kann den gleichen Endpunkt benutzen, der auch für den generellen Zugriff genutzt wird um über die jeweiligen Pfade auch direkt **funktionale Tests** auszuführen (z.B. einen User anlegen mit /applikation-url/create & /applikation-url/healthcheck/id für den regulären Check)
- Sollten zu viele **Ressourcen der Cloud Applikation** für die Checks verwendet werden, könnte dies die Usererfahrung beeinflussen. In der Regel können Logs über Fehler und Performanzzähler bereits genug aussagen und machen ausgiebige Performance Checks daher unnötig.
- **Sicherheit** der Endpunkte: Zugriff nur mit Authentifikation / 'versteckte' Endpunkte über unübliche Ports & verschiedene IP Adressen / Endpunkte so konfigurieren das sie bestimmte Informationen über die gewünschten Tests benötigen, andernfalls werden Anfragen abgewiesen
 - Zugriff auf gesicherte Endpunkte: Nicht alle Tools Unterstützen in ihre eingebauten health-verification Features auch Authentifizierung. Drittparty Anbieter wie Pingdom, Panopta, NewRelic oder Statuscake helfen hier.
- Auch die **Kontrolltools müssen getestet werden**, z.B. indem eine Cloud Applikationen ein festen OK Response auf einem Endpunkt sendet, den das Kontrolltool auslesen korrekt auslesen sollte.

Microsoft Azure

Applikationen die auf einer Microsoft Azure Umgebung gehostet werden, können bereits eingebaute Services der Plattform nutzen.

Der **Azure Management Service** bietet dabei die Möglichkeit bis zu 10 Regeln festzulegen, die durch einen Grenzwert definiert werden. Dieser kann sich z.B. auf die CPU Auslastung oder die Fehler pro Sekunde bei Anfragen beziehen. Bei Überschreitung der festgelegten Werte wird Alarm gegeben und der Nutzer per Email informiert.

Welche Konditionen überwacht werden können hängt von der Applikation und der Hostumgebung ab, nutzen jedoch in jedem Fall vom Nutzer definierte Endpunkte um die Kontrollen abzufragen.

Für virtuelle Maschinen oder Webapplikationen kann ein sogenannter **Traffic Manager** zusätzlich HTML Request stellen, die z.B. die Verfügbarkeit von Webseiten checken.

Index Table Pattern

Das Index Table Pattern dient zur Performance-Optimierung in Anwendungen, die mit großen Mengen an persistierten Daten arbeiten.

Da insbesondere Abfragen anhand anderer Werte als des Primärschlüssels besonders langsam werden können, bietet es die Möglichkeit, weitere Schlüssel anlegen zu können.

In relationalen Datenbanken kann dieses Verhalten meist mittels zusätzlicher Index-Spalten gelöst werden.

Das Index Table Pattern dient in anderen Fällen als eine Möglichkeit, diese Funktionalität nachzubilden.

Bei der Umsetzung gibt es drei unterschiedliche Strategien:

- **Vollständige Duplizierung**

Sämtliche Daten pro zusätzlichem Schlüssel in einer zusätzlichen Tabelle abgelegt. Dabei findet eine *vollständige Denormalisierung* statt. Problematisch bei häufigen Änderungen und im Bezug zur Speichermenge.

- **Normalisierte Index Tabellen**

Die neuen Tabellen enthalten jeweils nur den zusätzlichen Schlüssel und eine Referenz auf die ursprüngliche Tabelle, den *Fact Table*. Problematisch, da in jedem Fall zwei Abfragen erfolgen müssen.

- **Teilnormalisierte Index Tabellen**

Ähnlich den normalisierten Tabellen, allerdings werden häufig verwendete Daten dupliziert und nur für die fehlenden eine Anfrage an den *Fact Table* benötigt. Durch dieses Vorgehen erreicht man eine Balance zwischen den beiden anderen Fällen.

Darüber hinaus können auch Variationen umgesetzt werden, wie ein Schlüssel, der aus mehreren Feldern des Fact Table besteht, ein Ersatz für *Composite Keys*.

Ein Index Table kann auch verwendet werden, um den Zugriff auf Daten in Shards zu optimieren. Dabei wird der für das Sharding gehashte Primär Schlüssel als Referenz auf den geshardeten Fact Table verwendet. Der Index Table bleibt dabei zusammenhängend.

Probleme und Nachteile dieses Patterns sind zum einen der zusätzliche Aufwand, die Daten über mehrere Tabelle hinweg konsistent zu halten, bzw. die Notwendigkeit zweier Abfragen, um einen Datensatz auszulesen; je nach verwendeter Strategie.

Darüber hinaus muss auch bei Datenänderungen Konsistenz gewährleistet sein.

Sinnvoll ist die Anwendung des Patterns, wenn häufig Daten anhand von Nichtschlüsselattributen abgefragt werden. Dabei bestehen allerdings die folgenden Ausnahmen:

- **Häufig ändernde Daten**, durch die der Mehraufwand zum Pflegen der Index Tabellen die Performance-Gewinne überschreitet.
 - **Kleine Wertemenge** als neuer Schlüssel, wie z.B. Geschlechterinformationen (m/w)
 - **Unbalancierte Daten**, bei denen z.B. in 90% der selbe Wert enthalten ist. Eine Ausnahme dieser Ausnahme ist es, wenn häufig die verbleibenden 10% abgefragt werden.
-

Verwandte Entwursmuster sind:

- Data Consistency Primer
- Sharding Pattern
- Materialized View Pattern

Leader Election Pattern

Das Leader Election Pattern hat zum Ziel, die Aktionen mehrerer verteilter Komponenten, die an einer gemeinsamen Aufgabe arbeiten, zu koordinieren. Dafür wird aus allen Komponenten eine Komponente ausgewählt, die nun die Arbeit der anderen Komponenten managt. Somit wird verhindert, dass sich die Komponenten in ihrer Arbeit gegenseitig behindern, beispielsweise wenn versucht wird auf dieselbe Ressource zur gleichen Zeit zuzugreifen.

Problemhintergrund

Typischerweise ist es bei verteilten- oder Cloud-Anwendungen so, dass es mehrere Komponenten/Instanzen gibt, die beispielsweise denselben Programmcode beinhalten und auf gemeinsame Ressourcen zugreifen müssen. Dies kann bei skalierbaren Anwendungen der Fall sein, wenn beispielsweise jede Komponente für genau einen Benutzer zuständig ist, aber auf gemeinsame Ressourcen zugegriffen werden muss. Damit beim Zugriff auf diese gemeinsamen Ressourcen nicht unter anderem die Änderungen von einer Ressource von einer anderen stumpf überschrieben werden, muss dieser Zugriff koordiniert werden. Auch ist es möglich, dass jede Komponente nur einen kleinen Teil einer großen Aufgabe bearbeitet und am Ende alle Teilergebnisse zusammengeführt werden müssen. Da in beiden Fällen jede Komponente gleichgestellt ist (Peer), gibt es keine direkte Komponente die speziell für die Koordination zuständig ist.

Lösung

Um dieses Problem nun zu lösen, muss eine Instanz aus allen Instanzen gewählt werden, die nun die Leitung der anderen übernimmt. Da alle Instanzen dieselbe Code-Basis aufweisen, ist es somit möglich, dass jede beliebige Instanz zum Leiter ernannt wird. Der Prozess zur Wahl des Leiters muss also genau koordiniert werden, damit es nicht zu Fehlern kommt und es beispielsweise zwei oder mehr Instanzen gibt, die die Leitung übernehmen. Des Weiteren muss darauf geachtet werden, dass im Fehlerfall, beispielsweise bei einem Netzwerkausfall, ein neuer Leiter ernannt wird, wenn der alte seine Aufgabe nicht mehr absolvieren kann. Häufig wird dies so gelöst, dass jede Instanz den Leiter überwacht, beispielsweise durch einen Herzschlag-Mechanismus oder einfachem Abfragen des Leiters. Tritt dabei ein Fehler auf, wählen die verbleibenden Instanzen unter sich einen neuen Leiter aus. Die Wahl eines Leiters kann dabei beispielsweise folgendermaßen ablaufen:

- Es wird die Instanz mit der niedrigsten Prozess ID gewählt.
- Es existiert ein Mutex, der von allen versucht wird zu erreichen. Die Instanz die den Mutex zuerst bekommt wird zum Leiter. Dabei muss allerdings sichergestellt werden, dass der Mutex freigestellt wird, wenn der Leiter beispielsweise durch einen Fehler vom Netzwerk getrennt wird.
- Implementierung verschiedenster Leader Election Algorithmen, beispielsweise Bully- oder Ring-Algorithmus. Diese setzen voraus, dass jede Instanz eine eigene einzigartige ID besitzt und das die Instanzen untereinander kommunizieren können.

Probleme und Überlegungen

Folgende Punkte müssen bei der Anwendung dieses Patterns beachtet werden:

- Der Prozess zum Wahl eines Leiters muss Fehlerresistent sein.
- Es muss möglich sein den Ausfall eines Leiters festzustellen und entsprechend zu handeln, beispielsweise indem ein neuer Leiter gewählt wird.
- In skalierbaren Systemen ist es möglich, dass die Instanz die zum Leiter gewählt wurde beendet wird, wenn weniger Ressourcen benötigt werden und das System runter skaliert.
- Wird ein Mutex zur Wahl eines Leiters genutzt, ist der gesamte Prozess auch abhängig davon. Fällt der Zugriff zu dieser Mutex-Ressource aus, so kann auch kein neuer Leiter gewählt werden und das gesamte Peer System kann zusammenbrechen.
- Die manuelle Implementierung von einem der vorhandenen Leader Election Algorithmen bietet die größte Flexibilität und Anpassbarkeit.

Wann sollte es genutzt werden

Dieses Pattern sollte genutzt werden, wenn es in einem Verteilten (Cloud) System mehrere gleichgestellte Instanzen/Komponenten gibt, deren Zusammenarbeit sorgfältig koordiniert werden muss und es keinen direkten eigenständigen Leiter gibt. Der gewählte Leiter sollte dabei nur die Koordination der Arbeit der anderen Instanzen übernehmen und nicht noch selbst diese Arbeit ausführen, um zu verhindern, dass die Kommunikation verlangsamt wird. Dieses Pattern sollte nicht verwendet werden wenn:

- Es einen speziellen Prozess gibt, der die Aufgabe der Koordination übernimmt und somit immer als Leiter fungiert. Dies könnte beispielsweise durch ein Singleton Prozess umgesetzt werden, welcher im Fehlerfall neugestartet werden kann.
- Die Koordination durch deutlich einfachere Verfahren abgewickelt werden kann. Beispielsweise kann im Fall, dass mehrere Instanzen den Zugriff auf eine gemeinsame Ressource benötigen ein optimistischer oder pessimistischer Kontrollzugriff (Locking)

realisiert werden.

- Es entsprechende Drittanbieter Lösungen/Frameworks gibt, die diese Koordination der einzelnen Prozesse übernehmen können.

Materialized View Pattern

Context und Problem

Die Speicherung von Daten geschieht oft unter der Perspektive der reinen Datenspeicherung und berücksichtigt dabei nicht die Frage, wie die Daten gelesen werden sollen. Die Daten werden gemäß ihrer Typen gespeichert (z.B. Zahlen, Strings, usw.) und nach dem Schema des Datenmodells wie z.B. NoSQL oder relational.

Bei einer Datenabfrage ("Query"), welche nur eine Teilmenge der Daten benötigt, die jedoch über verschiedene Entitäten hinweg gespeichert wurden, müssen all diese Entitäten ausgelesen werden, um die gewollte Information zu erlangen.

Lösung

Das Materialized View Pattern beschreibt die Erzeugung von im Voraus erzeugten vorgefüllten Views. So eine View "materialisiert" die Daten in einer Form, welche am besten geeignet ist für das Ergebnis der Datenabfrage.

Eine materialisierte View ("materialized View") enthält nur die relevanten Daten für die entsprechende Datenanfrage, was einen schnelleren und einfacheren Informationsgewinn bedeutet. Sie können auch aktuelle, berechnete Werte enthalten, welche in der Abfrage definiert sind. Diese Views und die Daten die sie enthalten, können jederzeit auf Basis der Quelldaten wieder neu erstellt werden. Sie wird daher in keinem Fall direkt von einer Applikation aktualisiert, sondern eher als Cache behandelt. Wenn sich die Quelldaten ändern, muss die View jedoch aktualisiert werden, was entweder vom System automatisiert durchgeführt wird oder manuell.

Probleme und Überlegungen

Folgendes muss bei der Implementierung dieses Pattern beachtet werden:

- Wann und wie wird die View aktualisiert? (z.B. reagieren auf Veränderungen)
- Manche Systeme benötigen Materialized View Patterns. (z.B. bei Event Sourcing Patterns)
- Die Konsistenz der Daten bei Generierung und Aktualisierung der View
- Wo soll die View gespeichert werden? Sie muss nicht am selben Ort wie die

eigentlichen Daten liegen

- Ist die View nur zur Leistungsverbesserung da, kann sie z.B. im Cache gespeichert werden
- Materialized Views sollten ausgereizt werden (Berechnungen in die Datenabfrage einbinden)
- Wenn es möglich ist, Indexierung der View vornehmen um die Leistung zu steigern in relationalen Datenbanken

Wann wird dieses Pattern verwendet?

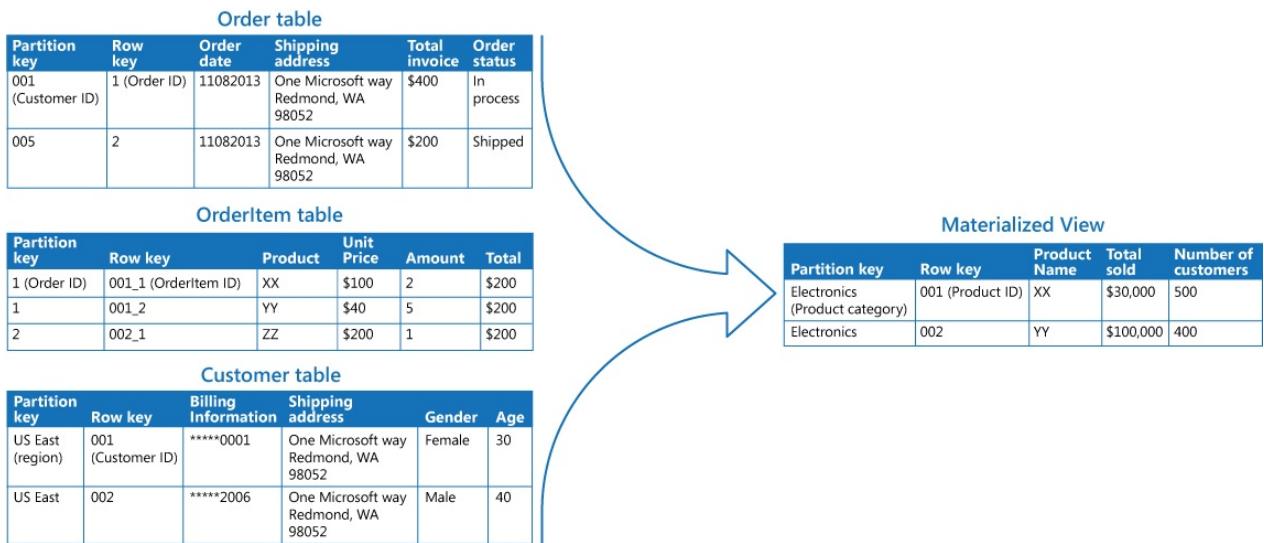
- Wenn Datenabfragen sehr komplex sind oder eine direkte Abfrage zu schwierig ist (z.B. unstrukturierte Daten, keine oder wenig Struktur vorhanden)
- Temporäre Views um die Leistung zu verbessern
- Lokal verfügbare Views, falls die Verbindung zu den Daten nicht besteht
- Vereinfachte Datenabfragen und Daten-Views zum experimentieren
- Sicherheit: Nur bestimmte Daten bestimmten Nutzern zur Verfügung stellen
- Als Brücke bei der Nutzung unterschiedlicher Datenspeichermethoden (z.B. eine Cloud zum Schreiben und eine relationale Datenbank für schnelle Datenabfragen)

Wann wird dieses Pattern nicht verwendet?

- Die Daten sind einfach abzufragen
- Die Quelldaten ändern sich sehr oft
- Konistenz ist ein wichtiger Faktor

Beispiel

Im folgenden Beispiel wird mittels anspruchsvollen Datenabfragen eine "materialized View" erstellt. Über verschiedene Entitäten wird eine View erstellt, die die gesamten Verkaufszahlen elektronischer Produkte enthält, zusammen mit der Anzahl der Kunden. Nutzer können mit Hilfe dieser View einfach bestimmte Resultate abfragen oder in andere Datenabfragen miteinbeziehen.



(Abbildung 1: Beispielhafte Anwendung um eine Zusammenfassung der Verkäufe darzustellen. Quelle: <https://docs.microsoft.com/en-us/azure/architecture/patterns/materialized-view>)

Ähnliche Patterns und Richtlinien

Folgende Patterns und Richtlinien könnten auch von Interesse sein:

- Data Consistency Primer
- Command and Query Responsibility Segregation (CQRS) Pattern
- Event Sourcing Pattern
- Index Table Pattern

Pipes and Filters Pattern

Eine Aufgabe sollte aus mehreren einzelnen Schritten bestehen. Diese einzelnen Schritte sollten wiederwendbar sein. Diese Wiederverwendbarkeit verbessert die Leistungsfähigkeit, Skalierbarkeit und Wiederverwendbarkeit der einzelnen Schritte deutlich, da die einzelnen Schritte unabhängig und skalierbar untereinander sind.

Problem

Für die Entwicklung einer Anwendung kann man einzelne Verarbeitungsschritte zu einem monolithischen Modul zusammenfassen. Aber wenn die gleichen Verarbeitungsschritte in mehreren verschiedenen Modulen vorkommen, dann verbessert sich der Code nicht. Der Code wird dupliziert statt reduziert, da der Code nicht wiederverwendet wird, sondern nur kopiert wird. Der Code wird nicht gut optimiert, da der Code in verschiedenen Modulen dupliziert wird und keine Regeln der Wiederverwendbarkeit genutzt wurden. Die Anwendung ist dadurch nicht besonders skalierbar. Wenn sich einige Verarbeitungsschritte in den Modulen ändern z.B. der Ablauf der Schritte ändert sich oder weitere Schritte werden hinzugefügt, dann muss der Code an mehreren Stellen überarbeitet werden.

Lösung

Eine Lösung für diese Probleme wäre, wenn man die Verarbeitungsschritte zu einem Prozess zusammenfasst. Dieser Prozess verarbeitet die einzelnen Schritte linear. Die Verarbeitungsschritte erledigen immer eine Aufgabe. Jeder Verarbeitungsschritt empfängt Daten und sendet Daten an den nächsten Verarbeitungsschritt weiter. Die einzelnen Verarbeitungsschritte sind Komponenten. Diese Komponenten können einfach ausgetauscht werden, wenn sich der Prozess verändert. Außerdem wird duplizierter Code vermieden, da der Code in einer Komponente gekapselt ist. Ändert man den Code in der Komponente, so würgt sich diese Änderung auf alle Prozesse aus, wo diese Komponente verwendet wird. Das Prinzip führt zu einer besseren Wiederverwendung des Codes (siehe Abbildung 1).

Die Verarbeitungsgeschwindigkeit des Prozesses hängt von den einzelnen Komponenten ab. Wenn einige Komponenten zu langsam sind, dann kann das zu einer Verlangsamung des ganzen Prozesses führen. Aus diesem Grund sollte man parallele Prozesse verwenden, um die langsamen Komponenten zu verteilen. Diese Verteilung erhöht die Verarbeitungsgeschwindigkeit der Anwendung. Gerade bei Serveranwendungen führt, dass zu einer besseren Performance für das gesamte System. Die Komponenten (oder Filters)

sind untereinander unabhängig und können beliebig skaliert werden. Sie eignen sich gut für Cloud-Server, da die Komponenten auf verschiedene Maschinen verteilt werden können. Rechenintensivere Komponenten können auf leistungsfähigeren Maschinen verteilt werden als weniger rechenintensivere Komponenten, dass spart Kosten und führt zu einer besseren Performance für die Serveranwendung. Man kann von überall auf der Welt auf die Komponenten und Prozesse (oder auch Pipeline) zugreifen, da sich diese in der Cloud befindet.

Der vorige Filter kann seine Ergebnisse an den nächsten Filter in der Pipeline weitergeben, bevor der vorige Filter zu Ende ist. Es können so mehrere Filter parallel arbeiten. Sollte ein Filter oder eine Maschine von der Cloud ausfallen, dann kann die Pipeline diesen Fall kompensieren. Die Pipeline leitet die Verarbeitungsschritte an einem anderen Filter weiter. Dieser Filter erledigt dann diese Aufgaben und gibt das Ergebnis an der Pipeline zurück. Das System ist dadurch sehr gut gegen Systemausfälle geschützt, da bei Ausfällen einer oder mehrerer Filters nicht diese ganze Pipeline mit ausfällt. Wenn man verteilte Transaktionen entwickeln will, dann kann man das „Pipes and Filters Patterns“ mit dem „Compensating Transaction Pattern“ kombinieren. Die verteilte Transaktion wird in mehrere Aufgaben geteilt und jede Aufgabe bekommt einen Filter. Der Filter benutzt das „Compensating Transaction Pattern“.

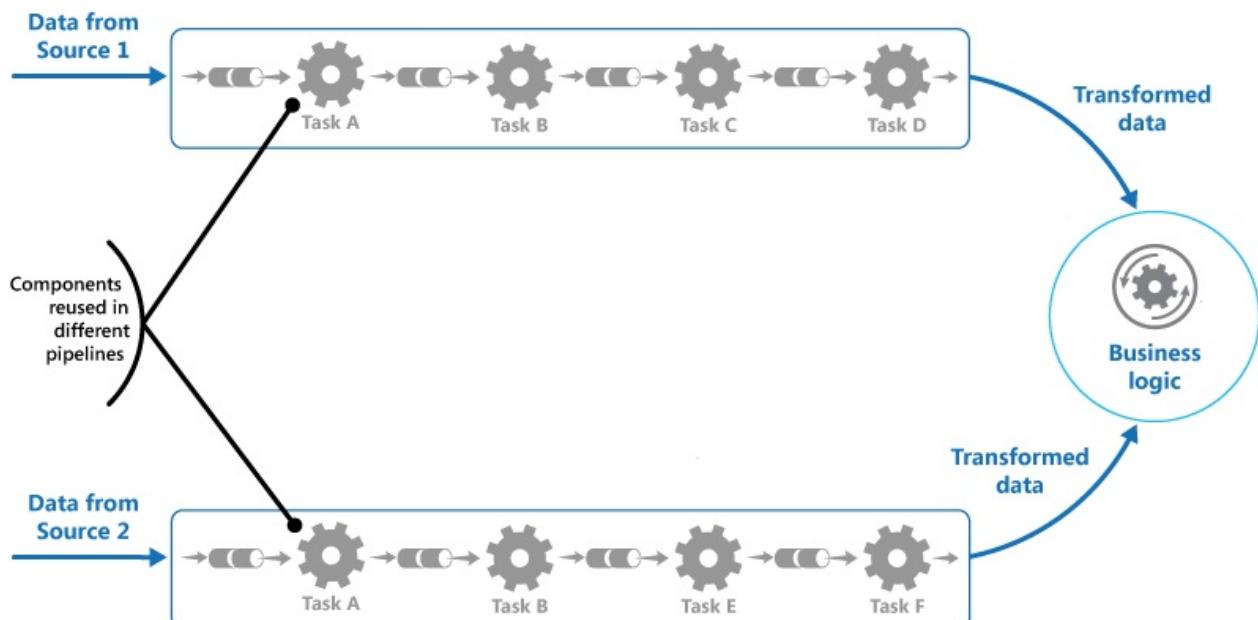


Abbildung 1: Grafische Darstellung des Patterns.

Benutzung

Wenn man das „Pipes and Filters Patterns“ benutzen will, dann sollte man darauf achten, dass die Anwendung nicht komplexer dadurch wird, immer noch zuverlässig läuft, das keine Idempotenz entsteht, keine wiederholten Nachrichten verschickt werden und der Zustand

und Kontext der Filter erhalten bleibt. Eine Komplexität kann dadurch entstehen, wenn man die Aufgaben an verschiedene Server verteilt. Die Zuverlässigkeit muss sicherstellen, dass keine Daten zwischen den einzelnen Filters verschwinden. Filters sollten Idempotenz sein, da sonst Ergebnisse von verschiedenen oder gleichen Filtern mehrfach auftreten könnten. Diese Daten würden dann mehrfach verarbeitet werden, dass führt zu Leistungseinbrüchen der Serveranwendung. Die Pipeline muss überprüfen, ob doppelte Nachrichten an die Filters geschickt werden und gegebenenfalls diese entfernen. Jeder Filter muss seinen Zustand speichern und seinen eigenen Kontext besitzen, da sie getrennt voneinander arbeiten.

Anwendungsbeispiele

Zum Beispiel geeignet für folgende Aufgaben:

- Ein Problem in mehreren unabhängigen Schritte lösen
- Die Verarbeitungsschritte der Serveranwendung sind unterschiedlich skalierbar
- Wenn etwas Flexibel ist, dann kann man die Verarbeitungsschritte anders anordnen und untereinander verändern
- Die Verteilung der Serveranwendung und der Verarbeitungsschritte auf viele verschiedenen Servers verbessert die Leistung für die Serveranwendung
- Wenn ein System vor Ausfällen geschützt werden soll

Es ist nicht so gut geeignet für folgende Aufgaben:

- Wenn die Verarbeitungsschritte nicht unabhängig voneinander sind, sondern zu einem großem Zusammenhang gehören.
- Wenn die Speicherung der Zustandsinformation Probleme mit der Datenbank verursachen würde

Bei der Implementierung sollte eine Warteschlange für die Pipeline verwendet werden. Die Filter bekommen die Daten und verarbeiten die Daten danach. Wenn die Daten verarbeitet wurden, dann werden die Daten an dem nächsten Filter in der Warteschlange weitergegeben. Bis das Ende der Warteschlange erreicht wurde. Der Anfang macht das erste Objekt der Filter Klasse in der Warteschlange.

Priority Queue Pattern

Das Priority Queue Pattern arbeitet Abfragen mit einer höheren Priorität schneller ab, als Abfragen mit einer geringen Priorität.

Konzept und Problem

Beispielsweise bei der Nutzung einer Cloud werden Abfragen an den Server über eine Message-Queue verwaltet. Da einige Abfragen so zeitnah wie möglich behandelt werden müssen, um dem Nutzer das Gefühl von "Echtzeit" zu vermitteln müssen diese durch Priorisierung vorgeschaltet werden.

Lösung

Um das Konzept von First-in First-out zu modifizieren bekommt jede eingehende Abfrage eine Priorität zugewiesen. Die neue Abfrage wird dann hinter das letzte Element der gleichen Priorität gesetzt. Somit hat man mehrere First-in First-out Queues innerhalb einer nach Prioritäten sortierten Queue.

Sollte ein System das Prinzip der Priority Queue nicht unterstützen, so kann durch die Erstellung mehrerer Queues für jede einzelne Priorität Abhilfe geschaffen werden.

Demnach werden erst alle Abfragen aus der Queue mit der höchsten Priorität abgearbeitet. Ist diese leer, so wird auf die nächsthöhere Priorität eingegangen.

Der hierbeschriebene single-pool-Ansatz führt dazu, dass eine Abfrage mit einer niedrigen Priorität womöglich nie abgearbeitet wird, da neue Abfragen mit einer höheren Priorität dazu kommen. Abhilfe soll hier der multi-pool-Ansatz sein, welcher die niedriger priorisierten Abfragen in Abhängigkeit seiner verfügbaren Ressourcen auf jeden Fall abarbeitet.

Die Nutzung der Priority Queue hat den Vorteil Business-Anforderungen welche die Priorisierung betreffen, wie beispielsweise die Bevorzugung von Pro-Kunden gegenüber normalen Kunden durchsetzen zu können.

Probleme und Überlegungen

Die Prioritäten müssen in Abhängigkeit zur Lösung implementiert werden. Wenn alle Abfragen die höchste Priorität haben, so muss unter Umständen auch hierbei lange auf eine Antwort gewartet werden.

Bei einer hohen Auslastung der Queue muss für den single-pool-Ansatz eine Lösung für die niedrig Priorisierten Abfragen gefunden werden.

Beim multi-pool-Ansatz auf einer single-pool-Basis muss ein Algorithmus zum Abrufen der korrekten Abfragen in den unterschiedlich priorisierten Queues implementiert werden.

Die Raten der behandelten niedrig- und hochpriorisierten Abfragen müssen überwacht und verglichen werden, um die Priorisierungen möglichst effizient anzupassen.

Abschließend kann man sagen, dass die Priority Queue bei Abfragen mit verschieden gewichteter Priorität oder bei Abfragen von Kunden mit verschieden gewichteter Priorität zum Einsatz kommen sollte.

Queue-Based Load Leveling Pattern

Das **Queue-Based Load Leveling Pattern** besteht darin, dass man Services um eine asynchrone Queue erweitert, welche die Anfragen der Tasks entgegennimmt und dem Service weiterreicht. Das soll verhindern, dass bei zu vielen Anfragen der Tasks an den Service dieser überlastet und seine Funktionalität verzögert oder gar nicht erfüllen kann. Das Pattern erhöht somit die Verfügbarkeit und Verlässlichkeit der Services.

Kontext und Problem

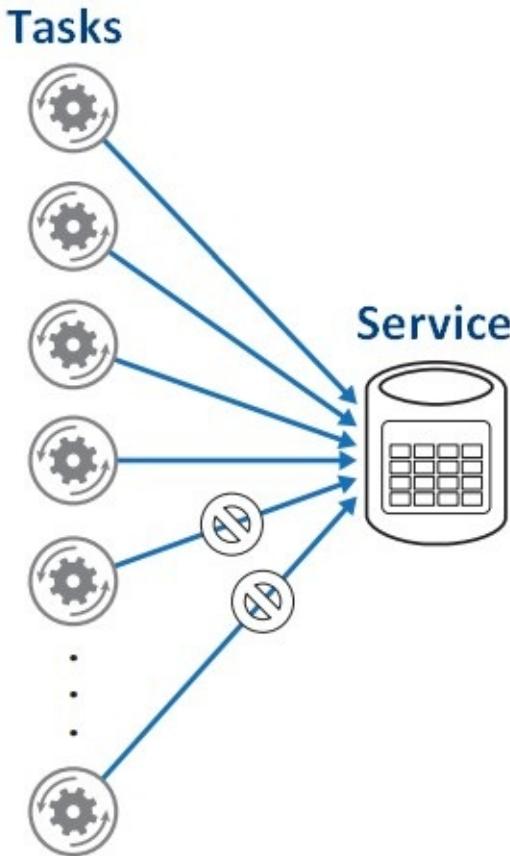
Viele Programme in der Cloud basieren darauf, dass sie Tasks ausführen, welche Services aufrufen. Wenn aber zu viele Tasks auf denselben Service zur gleichen Zeit zugreifen wollen, kann dies zur einer Überlastung des Service führen, was sich in Verzögerungen oder nicht Erreichbarkeit äußern kann (siehe Bild).



(Quelle: Cloud Design Patterns)"

Lösung

Der Einsatz einer asynchronen Queue kann zwar die Verzögerung der Anfrage der Tasks nicht vollständig verhindern, sorgt aber dafür, dass die Anfragen nicht unbeantwortet bleiben.



(Quelle: Cloud Design Patterns)"

Die Anfragen der Tasks werden in die Queue gesteckt und geordnet nach Zeitpunkt des Eintreffens abgearbeitet.

Das bietet folgende Vorteile:

- Die Verfügbarkeit der Services wird maximiert, da Verzögerungen weniger Auswirkungen auf die Funktionalität haben. Selbst wenn der Service temporär nicht erreichbar ist, können die Anfragen nachgearbeitet werden.
- Der Service kann skaliert werden, da je nach Anforderung die Anzahl der Queues und Services angepasst werden können.
- Es unterstützt bei der Kostenkontrolle bzgl. Ressourcen, da durch die Queue die Anzahl an Services anstatt für den Worst Case nur ausreichend für den Durchschnittsarbeitsfluss sein muss. Das Pattern eignet sich ideal für Applikationen, bei denen hohe Zugriffszahlen in kurzer Zeit entstehen könnten. Es wäre entsprechend nicht effektiv, dies bei Applikationen einzusetzen, bei denen das Gegenteil der Fall ist.

Man sollte folgende Punkte beachten, sofern man dieses Pattern implementieren will:

- Es ist von Vorteil, eine Logik für die Applikation zu entwickeln, welche die Abarbeitungsrate der Services kontrolliert, damit der Service nicht überlastet. Es sollten grundsätzlich Arbeitsflussspitzen vermieden werden. Zur Sicherstellung kann das System unter realistischen Arbeitsfluss getestet werden, um bei Unstimmigkeiten die

Anzahl der Queues und Services anzupassen.

- Die Queue funktioniert nur in einer Richtung. Wenn der Task eine Rückmeldung erwartet, müssen weitere Mechanismen implementiert werden, die dies sicherstellen (z.B. [Asynchronous Messaging Primer](#)).

Retry Pattern

Dieses Muster (Retry Pattern) ermöglicht, dass eine Anwendung erwartete und temporäre Fehler behandelt. Wenn eine Anwendung versucht, sich mit einem Service oder Netzwerk zu verbinden, wiederholt sie die Operation, weil die Ursache des Fehlers vorübergehend (transient) sein kann.

Problem

Wenn eine Anwendung mit Elementen in einer Cloud kommuniziert, können vorübergehende Fehler auftreten: der kurzzeitige Untergang der Netzwerkkonnektivität, die kurzzeitige Nichtverfügbarkeit eines Services... Die Aktion, die diese Fehler auslöst, kann erfolgreich sein, wenn sie mit einer Zeitverzögerung wiederholt wird.

Lösung

Eine Anwendung kann diese Fehler folgendermaßen behandeln:

- Wenn der Fehler nicht vorübergehend ist (z.B. Authentifizierungsfehler), sollte die Anwendung die Operation abbrechen und eine 'Exception' werfen.
- Wenn der bestimmte Fehler ungewöhnlich oder selten ist, sollte die Anwendung die Operation sofort wiederholen. (Wahrscheinlich wird der Fehler nicht erneut auftreten.)
- Wenn der Fehler ein 'busy' Fehler ist, sollte die Anwendung warten, bevor die Operation wiederholt wird. Wenn die Operation nach einer bestimmten Anzahl von Versuchen immer noch nicht erfolgreich ist, sollte die Anwendung den Fehler als eine Ausnahme (Exception) behandeln. Es ist hilfreich Fehler, die mit Retry lösen kann, in einem log detailliert aufzuzeichnen.

Was es zu beachten gilt

- Die Retry policy sollte an die Art der Anwendung und des Fehlers angepasst werden. Wenn die Operation für die Anwendung nicht essentiell ist, kann es manchmal für die Anwendung besser sein nicht die Operation zu wiederholen, sondern abzubrechen.
- Wenn man zu viel und zu schnell erneut versucht, kann man sowohl den 'busy' service, als auch die Anwendung überlasten.
- Nach einer Vielzahl von Wiederholungen kann man erstmal einen Fehlerbericht

ausgeben, dann eine Weile warten und erst dann ein oder zwei Mal erneut die Operation versuchen.

- Es ist wichtig, dass die Zeitverzögerung zwischen erneuten Anfragen an die Art des Fehlers angepasst ist.
- Retry Code muss die Zuverlässigkeit und Performance des gesamten Codes nicht beeinträchtigen. Man muss den Retry Code vielfältig testen.
- Es soll keine verschachtelten Retry Codes geben. Ein Retry Code ist z.B. verschachtelt, wenn eine Aufgabe, die Retry Code hat, eine andere Aufgabe, die auch Retry Code hat, aufruft.
- Man sollte versuchen herauszufinden, ob die Fehler lang andauernd sind. In diesem Fall sollte man den Fehler als Ausnahme behandeln.

Wann sollte man Retry Pattern benutzen?

Retry Pattern sollte benutzt werden:

- Wenn bei einer Anwendung, die mit einem fernen Service interagiert, vorübergehende Fehler auftreten können, sollte man Retry Pattern benutzen.

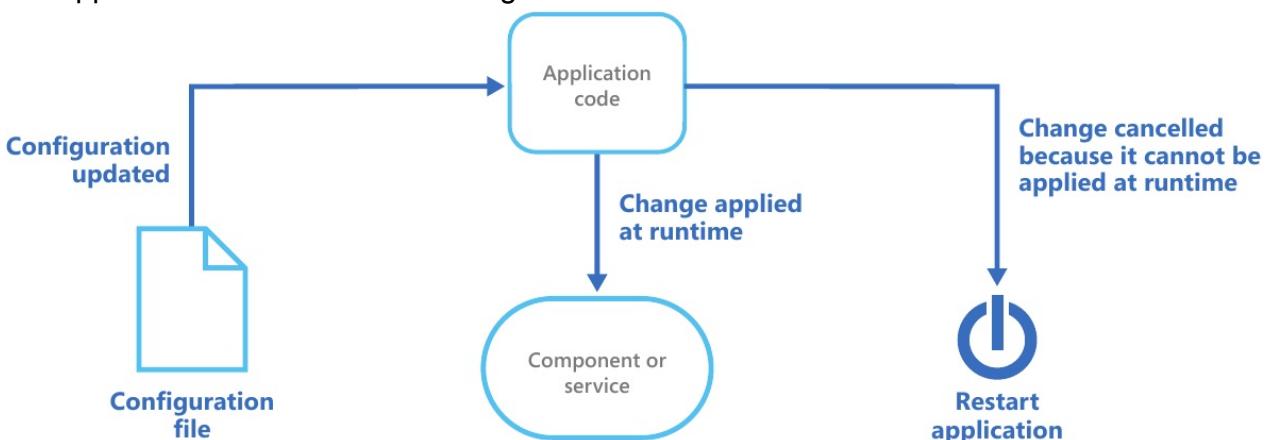
Retry Pattern sollte nicht benutzt werden:

- Wenn ein Fehler lang andauernd ist.
- Für nicht vorübergehende Fehler
- Wenn ein 'busy' Fehler bei einem bestimmten Service häufig ist.

Runtime Reconfiguration Pattern

Das Runtime Reconfiguration Pattern soll eine Rekonfiguration einer Applikation während der Laufzeit ermöglichen, so dass der Nutzer oder Kunde nur eine minimale Downtime oder daraus resultierende Unterbrechung der Applikation erfährt.

Dieses Pattern ist erstmal abhängig von den Features des Hosts der Applikation. Diese könnten aus dem Hochladen einer Konfigurationsdatei oder dem Zugriff über ein Administrationsportal oder einer API bestehen. Der Applikationscode kann dabei die Änderung an der Konfiguration untersuchen und sie auf die dazu gehörigen Komponenten der Applikation anwenden. Die Komponenten müssen dabei die Änderung erkennen und sie übernehmen. Wenn die Komponenten das nicht während der Laufzeit können, ist ein Neustart der Applikation dafür notwendig. Dies kann dadurch geschehen, dass der Host eine Änderung erkennt und der Applikation mitteilt sich neu zu starten oder es muss ein Code implementiert werden, der Änderungen an der Einstellung erkennt und gegebenenfalls die Applikation zum Neustarten zwingt.



(Abbildung 1 - Schema eines Runtime Reconfiguration Patterns)

Viele Umgebungen arbeiten mit Events als Antwort auf eine Konfigurationsänderung. Die Umgebungen, die es nicht tun, erfordern ein zyklisches Absuchen nach Änderungen der Konfiguration und eine Anwendung dessen falls notwendig. Es kann dabei auch nötig sein, das Datum und die Zeit einer Konfigurationsdatei zu vergleichen, um eine eventuelle neue Version festzustellen. Alternativ kann die Umgebung auf bestimmte Änderungen reagieren. Zum Beispiel könnte beim Auftreten eines Runtime-Error ein automatisches Sammeln von zusätzlichen Informationen erfolgen.

Folgende Punkte sollte man beim Implementieren dieses Patterns beachten:

- Die Konfigurationsdateien sollten abgetrennt von einer entwickelten Applikation gespeichert werden.
- Wenn Konfigurationsänderungen nicht automatisch erkannt werden, sollte man einen

alternativen Mechanismus zum Erkennen und Anwenden von Änderungen erstellen.

- Wenn das zyklische Absuchen nach Änderungen erforderlich ist, sollte man abwägen in welchen Abständen man dies tut.
- Wenn es mehr als eine Instanz der Anwendung gibt, sollte man beachten, dass die Änderung an alle Instanzen der Anwendung nach einer gewissen Zeit kommuniziert wird.
- Wenn eine Konfigurationsänderung einen Neustart des Hostsystems erfordert, sollte man die Art der Änderung identifizieren. Es könnte eine Änderung sein, die einen automatischen Neustart des Systems erfordert oder eine, die die Verantwortung dem Administrator zuschiebt, um die Applikation je nach Zeit oder Performance des Systems neu zu starten.
- Es ist abzuwägen wie man die Konfiguration zurücksetzen will, für den Fall, dass die Änderung Fehler verursacht hat.
- Auch die Herkunft der Konfigurationseinstellung sollte beachtet werden, weil es die Performance der Applikation beeinflussen könnte. Zum Beispiel ein Fehler, der aufgetreten ist, wenn ein externer Speicher nicht verfügbar ist.
- Das Cachen kann dabei helfen Verzögerungen zu reduzieren, wenn Komponenten wiederholt auf Konfigurationen zugreifen müssen.

Dieses Pattern ist hilfreich für:

- Applikationen, welche unnötige Downtimes vermeiden müssen.
- Umgebungen, die automatisch Events auslösen, wenn die Hauptkonfiguration sich geändert hat.
- Applikationen bei denen sich die Konfigurationen oft ändern und die Änderung nicht den Neustart der Applikation benötigen sollte.

Nicht hilfreich wäre es für Laufzeitkomponenten, die so aufgebaut sind, dass sie die Konfiguration nur in ihrer Startzeit laden und der Aufwand diese Komponenten anzupassen im Vergleich zum Neustarten und einer kurzen Downtime nicht gerechtfertigt ist.

Schedular Agent Supervisor Pattern

Das Pattern dient dazu Aufgaben und Abfolgen von Aufgaben in der Cloud auszuführen und zu Koordinieren.

Cloud bedeutet dabei, dass mehrere Systeme am Ausführen von Aufgaben beteiligt werden. Aufgaben sind dabei Prozesse die auf andern Systemen ausgeführt werden. Die Aufgaben werden nicht direkt innerhalb der Komponenten des Patterns eingebettet sondern durch z.B. URLs in externen System aufgerufen.

Durch den Einsatz des Patterns soll insbesondere im Fehlerfall, beim Absturz des Supervisors oder beim Fehlschlagen einzelner Aktionen, ein definierter Zustand wiederhergestellt werden (Self-Healing).

Das Pattern beinhaltet 3 Zuständigkeiten (Actors):

- Scheduler
- Agent
- Supervisor

Durch den **Scheduler** wird die Reihenfolge der Ausführung der Aufgaben festgelegt. Weiterhin wird der derzeige Zustand der Aufgabe festgehalten, welche Teilschritte ausgeführt wurden und in welchem Zustand sich die derzeitige Teilaufgabe befindet. Der Supervisor speichert den Zustand der aktuellen Ausführung in einer Datenbank dem "State-Store".

Dieser State-Store bietet ein Protokoll über den Zustand der der aktuellen Aufgabe, er bietet aber auch die Möglichkeit die Ausführung bestimmter Aufgaben nach einem Absturz oder Neustart fortzusetzen.

Der **Agent** koordiniert den Aufruf der konkreten Aufgabe. Er Prüft den Zustand einer Aufgabe und kann den Scheduler anweisen bestimmte Aktionen erneut auszuführen. Er entspricht im wesentlichen dem Proxy-Pattern mit der Erweiterung von Timeouts und Kommunikation mit einem bekannten Scheduler. Der Agent ist möglich generisch zu halten er sollte keine Kenntnis des aktuellen Geschäftsvorgangs haben.

Der **Supervisor** benutzt den Agent um den Status einer Teilaufgabe abzufragen und dem Scheduler zugänglich zu machen.

Im Kontext von Cloud-Anwendungen ist die Koordinierung der Aufgaben besonders wichtig, insbesondere ist es wichtig, dass Aufgaben nicht mehrfach ausgeführt werden. Um dies sicherzustellen muss zusätzlich zum Scheduler Agent Supervisor Pattern auf das Leader

Election Pattern zurückgegriffen werden. Pro Aufgabe muss ein Scheduler ausgemacht werden der für diese spezifische Aufgabe verantwortlich ist.

Um korrekte Ergebnisse der ausgeführten Aktionen zu erhalten sollten diese Indempotent sein. Sollte dies nicht möglich sein muss ein Mechanismus für Transaktionen (Ein Rollback im Fehlerfall muss gewährleistet sein) innerhalb der Aufgaben oder der Aktoren integriert sein.

Sharding Pattern

Die Idee ist es große Datenmengen in mehrere horizontale "Splitter"(Shards) aufzuteilen, um die Skalierbarkeit zu verbessern.

When to use

Wenn die Daten die Ressourcen einer einzelnen Datenbank übersteigen werden. Um die Performance zu steigern, indem die Zugriffskonflikte minimiert werden.

Problemstellung

Eine Datenbank auf einem einzigen Server wird sich früher oder später mit den Folgenden Themen befassen müssen: Speicherplatz Rechenleistung Netzwerkbandbreite Datenverteilung

Diese Punkte durch "aufrüsten" auszubauen, verschiebt lediglich das Problem in die Zukunft.

Sharding als Lösung

Die Datenbank wird in mehrere Splitter aufgeteilt. Jeder Splitter verfügt über das gleiche Schema, enthält jedoch nur eine individuelle Menge der Gesamten Datenmenge.

Für das Aufteilen von Daten auf die einzelnen Splitter werden "shard keys" oder "partition keys" verwendet. Dieser Schlüssel muss statisch sein.

Sharding organisiert die Daten, die Logic sorgt bei einem Zugriff der Applikation dafür, dass die angeforderten Daten aufgefunden werden.

Vorteile

Jederzeit erweiterbar, durch das Hinzufügen von weiteren Splittern. Zugriffskonflikte können reduziert und Performance gesteigert werden, durch ein Ausbalancieren des Workloads der einzelnen Splitter. In einer Cloudlösung können die Splitter "nah" am User platziert werden.

Nachteile

Daten der Splitter müssen ausgeglichen werden, um für eine hohe Performance zu sorgen.

Sharding Strategies

Für die Vergabe von "shard keys" und das Verteilen der Daten, gibt es 3 gängige Strategien, welche im Folgenden skizziert werden.

The Lookup strategy

Hier liegen alle benötigten Daten (für einen Teil einer Abfrage) auf einem einzigen Shard.
Vorteile: Mehr Kontrolle über die Konfigurationen der Shards. Nachteile: Overhead durch das aufsuchen mehrere Speicherorte(Shards)

The Range strategy

Häufig zusammen abgefragte Daten, bspw. in einem Query, werden zusammen in einem Splitter gespeichert. Vorteile: Einfach zu implementieren. Einfaches Datenmanagement
Nachteile: Keine optimale Balance zwischen den Shards. Rebalancing ist schwer.

The Hash strategy

Aus mehreren Attributen werden Hashes gebildet, welche abgefragt werden können. Durch diese Strategie soll ein Balance der Zugriffe unter den Shards erreicht werden. Dies ist vor allem sinnvoll in Umgebungen, mit sehr vielen Zugriffen. Vorteile: Hashermittlung kann Overhead erzeugen. Rebalancing ist schwer.

Static Content Hosting Pattern

Im Folgenden wird das Static Content Hosting Pattern beschrieben, welches dazu dient Ressourcen effizienter zu verwalten, wenn statische Elemente von Servern zur Verfügung gestellt werden.

Problemstellung

Web-Applikationen beinhalten im Normalfall einige Elemente mit statischem Inhalt. Dieser statische Inhalt kann in Form von HTML-Seiten und anderen Ressourcen, wie Bildern oder Dokumenten vorliegen. Clients können entweder über den Aufruf der HTML-Seite oder über einen direkten Download auf diese statischen Elemente zugreifen. Dies führt dazu, dass der Webserver Anfragen zum Bereitstellen statischer Inhalte verarbeiten muss und somit Rechenzeit nutzt, um die Inhalte zu instanziieren. Diese Rechenzeit lässt sich minimieren, indem statische Inhalte ausgelagert werden.

Abhilfe

In vielen cloudbasierten Systemen ist es möglich die zur Laufzeit erstellten statischen Instanzen zu minimieren, indem statische Instanzen auf einem Speicherdiensst zum Abruf bereitgestellt werden. Die Kosten für Speicherdiensste sind im Gegensatz zu den Kosten für Rechenzeit auf cloudbasierten Systemen wesentlich geringer. Das folgende Schaubild bietet einen Überblick in das Konzept des Static Content Hosting Pattern:

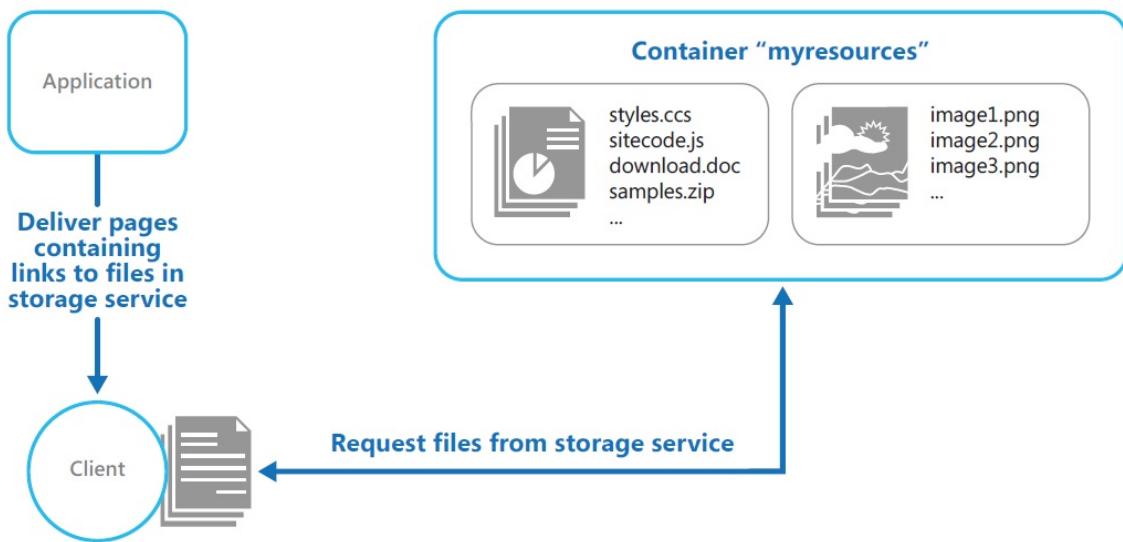


FIGURE 1
Delivering static parts of an application directly from a storage service

Herausforderungen und Überlegungen

Beim Implementieren dieses Design-Patterns ist folgendes zu beachten:

- Der gehostete Speicherdiensst muss einen HTTP Endpunkt auflösen können, damit Benutzer auf den statischen Inhalt zugreifen können. Und Einige Speichergeräte unterstützen HTTPS, falls statische Ressourcen SSL erfordern.
- Für eine maximale Performance und Verfügbarkeit sollte ein "Content Delivery Network" in Betracht gezogen werden. Dieses nutzt eine Zwischenspeicherung der Inhalte an mehreren Datenzentren und ist dadurch allerdings mit zusätzlichen Kosten verbunden.
- Durch das Duplizieren der Daten an verschiedenen Datenzentren kann sich die IP-Adresse ändern, die URL bleibt jedoch erhalten.
- Die Komplexität der Applikation steigt, da Daten gestreut vorzufinden sind und damit ist auch ein erhöhter Aufwand für das Verwalten der Daten verbunden.
- Speicherdiensste
- Die Rechte für den öffentlichen Zugriff in "Storage Container" müssen korrekt verwaltet werden um bspw. unbefugte Uploads zu verhindern. Um anonyme Zugriffe zu verhindern sollte ein Valet Key oder ein Valet Token genutzt werden: [Valet Key Pattern](#).

Wann sollte das Pattern genutzt werden?

Das Pattern ist ideal geeignet, um:

- Kosten für das Hosting von Webseiten und Applikationen mit statischen Inhalten zu minimieren.

- Kosten für das Hosting von Webseiten zu minimieren, wenn diese ausschließlich aus statischen Inhalten bestehen.
- statische Inhalte für Applikationen bereitzustellen, welche auf anderen Hosts (oder "on-premises") betrieben werden.
- Inhalte an mehreren geographisch verteilten Standorten mit Hilfe eines "Content Delivery Networks" zu speichern.
- Kosten und Nutzung der Bandbreite zu überwachen, da sich Host- und Laufzeitkosten von Speicherkosten der statischen Inhalte trennen lassen.

Das Pattern ist nicht geeignet, wenn:

- Applikationen die statischen Inhalte manipulieren, bevor die Inhalte an Clients ausgeliefert werden. Beispielsweise wenn Applikationen einen Zeitstempel auf ein Dokument aufsetzen, direkt bevor es heruntergeladen wird.
- nur sehr wenig statischer Inhalt zu verwalten ist.

Throttling Pattern

Problemstellung

Die Auslastung einer Cloud unterliegt zeitlichen Schwankungen. Sie ist abhängig von der Anzahl der aktiven Benutzer und deren Tätigkeiten. Daher kann es zu plötzlichen und unerwarteten Anstiegen der Auslastung kommen, die das System an die Grenzen seiner Ressourcen bringt. Die vereinbarten Service-Level können auf Grund der resultierenden Performanceeinbußen nicht mehr eingehalten werden. Um schwankende Auslastungen zu bewältigen, können Strategien wie Autoscaling oder Throttling angewendet werden.

Strategien

Beim Autoscaling kann die Cloud die Bereitstellung weiterer Ressourcen anfragen, wenn die eigenen Ressourcen nicht ausreichen. Bei einem unerwartet starken Anstieg der Auslastung kann es dennoch zum Defizit kommen, da diese Bereitstellung nicht unmittelbar erfolgt. Bei der Throttling-Strategie dürfen Anwendungen die Ressourcen bis zu einem Softlimit verbrauchen. Wenn dieses Softlimit überschritten wird, werden Anwendungen gedrosselt, sodass das System weiterhin seine Funktion erbringen kann und die vereinbarten Service-Level eingehalten werden. Das Drosseln kann durch Ablehnen von Benutzeranfragen über eine Zeitperiode oder durch das Deaktivieren / Einschränken von Funktionalität bestimmter nicht essentieller Services erreicht werden (z.B. Begrenzung der Auflösung bei Videostreaming). Dabei können die Ressourcen aller Benutzer gleichermaßen (Queue-based Load Leveling Pattern) oder bei Benutzer mit einem höheren Service-Level geringer eingeschränkt werden (Priority Queue Pattern). Im Folgenden soll der Ablauf der zwei Strategien Throttling und Autcoscaling in Kombination gezeigt werden. Die Abbildung 1 zeigt dazu den Ressourcenverbrauch aller Anwendungen über die Zeit an. Bis zu dem Zeitpunkt T1 ist der Ressourcenverbrauch unterhalb des Softlimits. Ab dem Zeitpunkt T1 wird dieses Limit überschritten und das System fragt die Bereitstellung weiterer Ressourcen an (Autoscaling). Bei einem schnellen Anstieg des Ressourcenbedarfs kann das System die maximale Belastung erreichen, da das Autoscaling Zeit in Anspruch nimmt. Um dies zu verhindern, kann eine Throttling-Strategie zwischen der Zeitspanne T1 und T2 eingesetzt werden. Das Throttling kann zum Zeitpunkt T2, wenn die angefragten Ressourcen zur Verfügung stehen, wieder verringert werden.

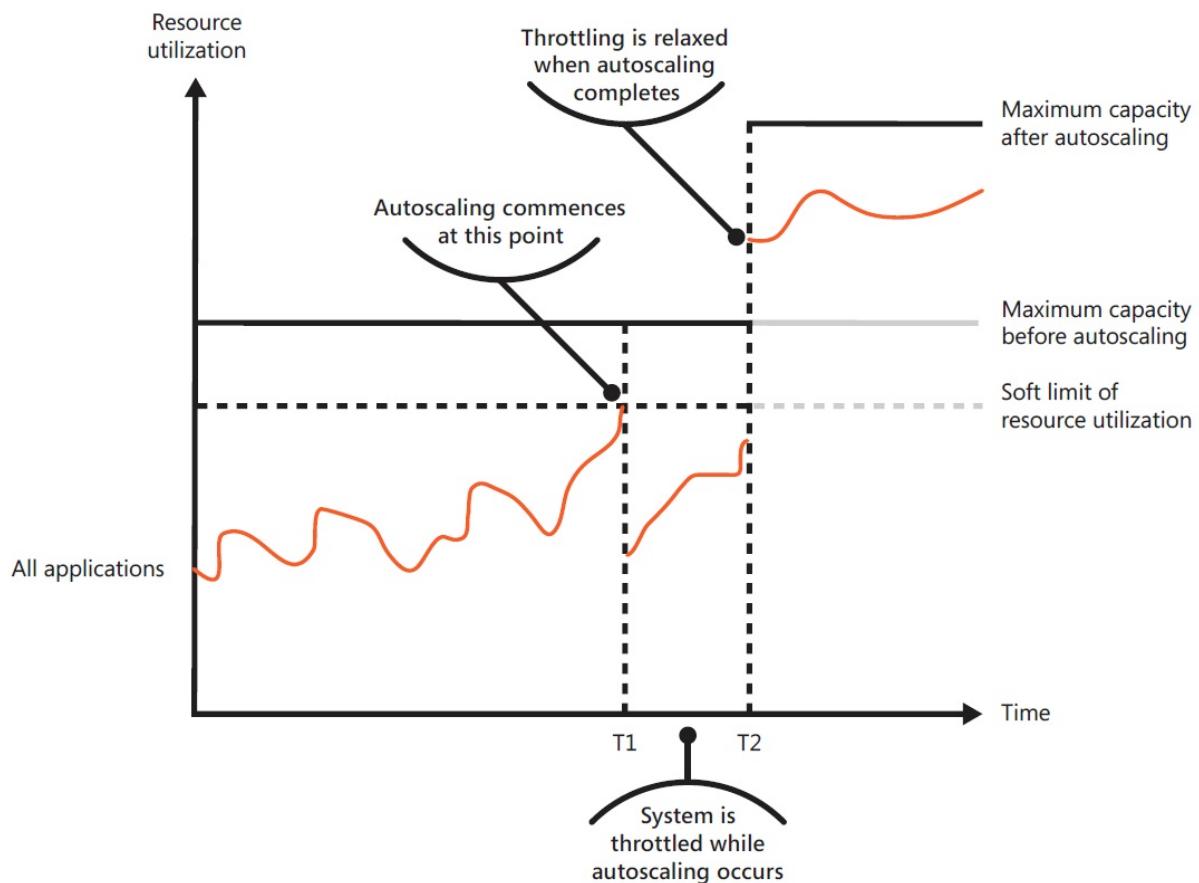


Abbildung 1: Kombination der Throttling- und Autoscaling-Strategie [Cloud Design Patterns, S. 157].

Weitere Betrachtungen

- Eine Throttling-Strategie hat Auswirkungen auf das Systemdesign und muss daher früh betrachtet werden.
- Das System muss das Drosseln schnell umsetzen können.
- Das System muss einer gedrosselten Client-Applikation eine aussagekräftige Fehlermeldung zurückgeben, sodass diese entsprechend reagieren kann.
- Bei kurzen Anstiegen des Ressourcenbedarfs sollte auf Autoscaling verzichtet werden, da diese Strategie kostspielig ist.
- Eine aggressivere Throttling-Strategie oder eine größere Ressourcenreserve sollte betrachtet werden, wenn das System trotz einer aktiven Drosselung an die Grenzen geht und dies nicht tolerierbar ist.

Einsatz des Pattern

- Einhaltung von Service-Level-Vereinbarungen.
- Verhindern, dass eine Teilnehmer-Applikation große Mengen an Ressourcen für sich

selbst beansprucht.

- Abfangen von Spitzen des Ressourcenverbrauchs.

Valet Key Pattern

Das Valet Key Pattern wird bei Server-Client-Systemen angewendet, um Clients den direkten, aber eingeschränkten, Zugriff auf dedizierte Ressourcen zu ermöglichen. Das soll den Datenumfang der Serverapplikation reduzieren, ohne die Datensicherheit zu gefährden. Das Pattern ist besonders bei Applikationen nützlich, die Cloud-basierte Speichersysteme nutzen, da so Kosten minimiert und die Skalierbarkeit und Performanz des Gesamtsystems maximiert werden können.

Kontext und Problemstellung

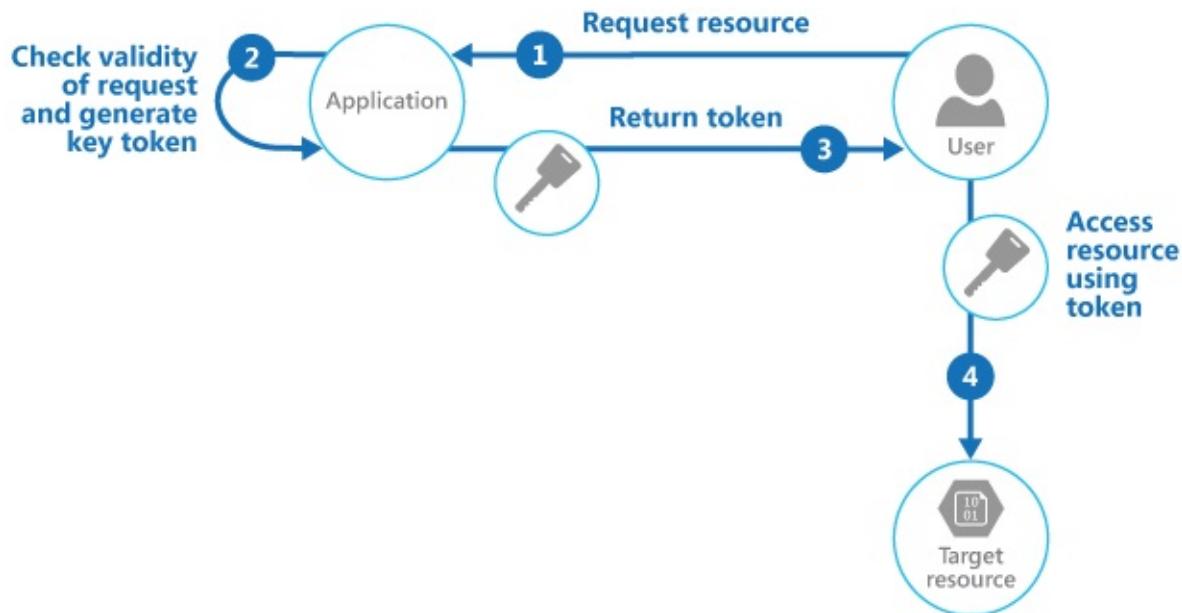
Bei Server-Client-Systemen ist typischerweise die Server-Applikation dafür zuständig Daten zwischen Clients und dem eigenen Datenspeicher hin- und herzutransferieren. Dieses Vorgehen bindet jedoch wertvolle Ressourcen, wie Rechenzeit, Arbeitsspeicher und Bandbreite. Dabei sind Datenspeicher eigentlich in der Lage den Datenaustausch mit dem Client direkt zu regeln, ohne die Applikation zu belasten.

Dafür braucht ein Client Zugriff auf die Sicherheitsschlüssel des Datenspeichers. Sobald der Client diesen Zugriff hat, hat die Applikation jedoch keine Handhabe mehr über den Datenaustausch zwischen Client und Datenspeicher; sie kann also nicht mehr als Gatekeeper agieren. Dieser direkte und damit unkontrollierte Zugriff ist bei verteilten Systemen, die nicht-vertrauenswürdige Clients ausschließen können müssen, kritisch und deshalb nicht gangbar. Applikationen müssen stets in der Lage sein den direkten Datenzugriff durch den Client granular zu steuern und ggf. zu verwahren, um Datensicherheit zu garantieren und gleichzeitig die eigenen Ressourcen zu schonen.

Lösungsansatz

Es bedarf der dedizierten Zugriffskontrolle auf den Datenspeicher. Letzterer kann die Authentifizierung und Authorisierung der Clients aber nicht selber durchführen. Eine typische Lösung für dieses Problem ist es, dem Client, durch die Verwendung von Schlüsseln oder Tokens, nur eingeschränkten Speicherzugriff zu geben. Diesen Schlüssel nennt man Valet Key, welcher dem Client temporären Zugriff ausschließlich auf benötigte Speicherbereiche gewährt. Die Ausstellung der Valet Keys zur Laufzeit durch die Applikation ist dabei schnell und einfach, wordurch die Auslastung von Applikation und Server durch Datentransferprozessen effektiv gesenkt wird.

Nach der Ausstellung eines Tokens benutzt der Client diesen, um auf eine dedizierte Ressource zuzugreifen; und zwar nur für einen beschränkten Zeitraum und mit restriktiven Zugriffsrechten (vgl. Abbildung). Zum einen kann so nur auf bestimmte Speicherbereiche zugegriffen werden (Tabellen, Tabellenzeilen, Container, Container-Elemente). Zum anderen verliert der Token nach einer bestimmten Zeit seine Gültigkeit und dem Client wird der Ressourenzugriff wieder verwehrt. Zusätzlich kann die Applikation die Gültigkeit vorzeitig beenden, wenn z.B. der Client den erfolgreichen Abschluss eines Datentransfers gemeldet hat.



Probleme und Überlegungen

Wenn es um die Implementierung des Patterns geht, sind folgende Aspekte zu berücksichtigen:

- Verwaltung des Gültigkeitsstatus' und -zeitraums: Für den Fall, dass ein Sicherheitsschlüssel durchgesickert ist und die Gefahr der boshaften Verwendung besteht, gibt es verschiedene Gegenmaßnahmen. Der Schlüssel kann als ungültig markiert, die Server-Rechte angepasst oder der korrespondierende Server-seitige Schlüssel deaktiviert werden. Präventiv kann bereits bei der Schlüsselgenerierung die Gültigkeitsdauer möglichst kurz gesetzt werden; dabei darf sie aber nicht so kurz gewählt werden, dass der Client nicht genügend Zeit zur kompletten Datenübertragung hätte.
- Kontrolle des Zugriffsevels: Der Sicherheitsschlüssel sollte dem Nutzer ausschließlich den Ressourenzugriff ermöglichen, der für die aktuelle Operation notwendig ist. Soll der Nutzer nichts hochladen dürfen, genügt einfacher Leserzugriff. Soll er lediglich Daten hochladen, genügt ausschließlich Schreibzugriff in einen bestimmten leeren Bereich.

- Lenken des Benutzerverhaltens: Die granulare Ressourcenkontrolle ist durch die Funktionalitäten der Applikationsdienste und des Datenspeichers beschränkt, weshalb die Client-Operationen nur eingeschränkt lenkbar sind. So hat man keine Handhabe über die Anzahl der Uploads oder die Dateigrößen, wodurch große unerwartete Kosten entstehen können. Um die Anzahl der Uploads zu kontrollieren, unterstützen Datenspeicher jedoch oft die Möglichkeit erfolgreichen Uploads zu melden, was die Applikation zur Überwachung nutzen kann.
- Validierung und optionale Zensierung hochgeladener Daten: Ein boshafter Nutzer kann kompromitierte Daten hochladen oder ein autorisierter Nutzer lädt versehentlich defekte oder manipulierte Daten hoch. Deshalb müssen alle Uploads validiert und überprüft werden.
- Alle Operationen können durch Protokollfunktionen der Schlüsselbasierten Mechanismen protokolliert werden, was für die Prozessoptimierung oder Rechnungsstellung gegenüber dem Nutzer verwendet werden kann.
- Sichere Übermittlung des Schlüssels durch Einbettung in die URL, automatischen Download und der ausschließlichen Übermittlung via HTTPS.
- Sensitive Daten bei der Übermittlung schützen durch die verpflichtende Verwendung von SSL oder TLS.

Anwendungsszenarien

Nützlich bei folgenden Anwendungen:

- Minimierung der Ressourcenlast und Maximierung der Performanz und Skalierbarkeit, da die Schlüsselerzeugung typischerweise eine einfache kryptografische Operation ist.
- Minimierung der Operationskosten, da der direkte Datenzugriff die Netzwerklast reduzieren kann.
- Wenn Clients regelmäßig Daten hochladen, besonder bei großen Speichern oder großen Dateien.
- Wenn die Applikation nur limitierte Hardwareressourcen zur Verfügung hat.
- Wenn Daten dezentral gespeichert sind und die Applikation häufig vermitteln muss.

Nicht nützlich bei folgenden Anwendungen:

- Wenn die Applikation Daten bearbeiten oder validieren muss, bevor sie an den Client gesendet werden können.
- Wenn das Design der Applikation die Implementierung des Patterns schwierig macht; das Valet Key Pattern sollte bei der initialen Architektur berücksichtigt werden.
- Wenn die Anzahl der Up- und Downloads genau überwacht werden muss und der Datenspeicher über keinen Mitteilungsmechanismus verfügt.
- Wenn die Größe der Uploads begrenzt werden muss. Alternativ kann nach dem Upload

die Dateigröße kontrolliert werden oder zyklisch während des Uploads.

Asynchronous Messaging Primer

Der Nachrichtenaustausch ist einer der Schlüsselfunktionen von verteilten Systemen. Dieser ermöglicht es Applikationen und Diensten miteinander zu Kommunizieren und zu kooperieren und hilft dabei skalierbare und belastbare Systeme zu entwerfen. (vgl. S. 166)

Queue-Grundlagen

Innerhalb von Cloud-Technologien ist der Nachrichtenaustausch häufig in Form von Queues (Warteschlagnen) implementiert. Diese unterstützen meist folgende drei Operationen: Ein Absender kann eine Nachricht in die Queue legen, ein Empfänger kann diese Nachricht untersuchen oder sie aus der Queue entnehmen. (vgl. S. 166)

Senden und Empfangen von Nachrichten mithilfe einer Queue

Eine Queue kann man sich tatsächlich wie eine Warteschlange vorstellen. Der Absender hängt hinten an die Warteschlange neue Nachrichten an, die der Empfänger von der anderen Seite abarbeitet (FIFO-Prinzip). Versucht ein Empfänger aus einer leeren Queue eine Nachricht zu entnehmen, so wird diese Operation geblockt. Häufig ist es den Empfängern auch möglich zu Prüfen ob Nachrichten zur Verfügung stehen, sodass das Blocken umgangen werden kann. (vgl. S. 166)

Die Infrastruktur der Queue muss dafür sorgetragen, dass eine einmal hineingelegte Nachricht nicht verloren geht. (vgl. S. 166)

Queues sind ideal für das asynchrone Arbeiten zweier oder mehrerer Dienste. Nach dem Ablegen einer Nachricht in einer Queue, kann sich der Absender anderen Aufgaben widmen. Zusätzlich werden Queues häufig in Verbindung mit mehreren Sendern und Empfängern eingesetzt.

Grundlegende Message-Queue-Patterns

One-way messaging ist die einfachste Form der Implementierung. Absender legen Nachrichten in eine Queue die anschließend von Empfängern bearbeitet werden. Es findet keine Überprüfung statt, ob die Nachrichten tatsächlich irgendwann von einem Empfänger bearbeitet werden. (Vgl. S. 168)

Beim **request/response messaging** erwartet der Absender eine Rückmeldung des Empfängers, dass die Nachricht verarbeitet wurde. Die Antwort der Empfänger wird in eine Absender-Spezifische Queue gelegt (jeder Absender hat eine eigene Queue für die Antworten). Erfolgt innerhalb einer bestimmten Zeit keine Antwort, kann ein Empfänger entsprechende Maßnahmen einleiten (z.B. erneutes Senden der Nachrichten). (Vgl. S. 168)

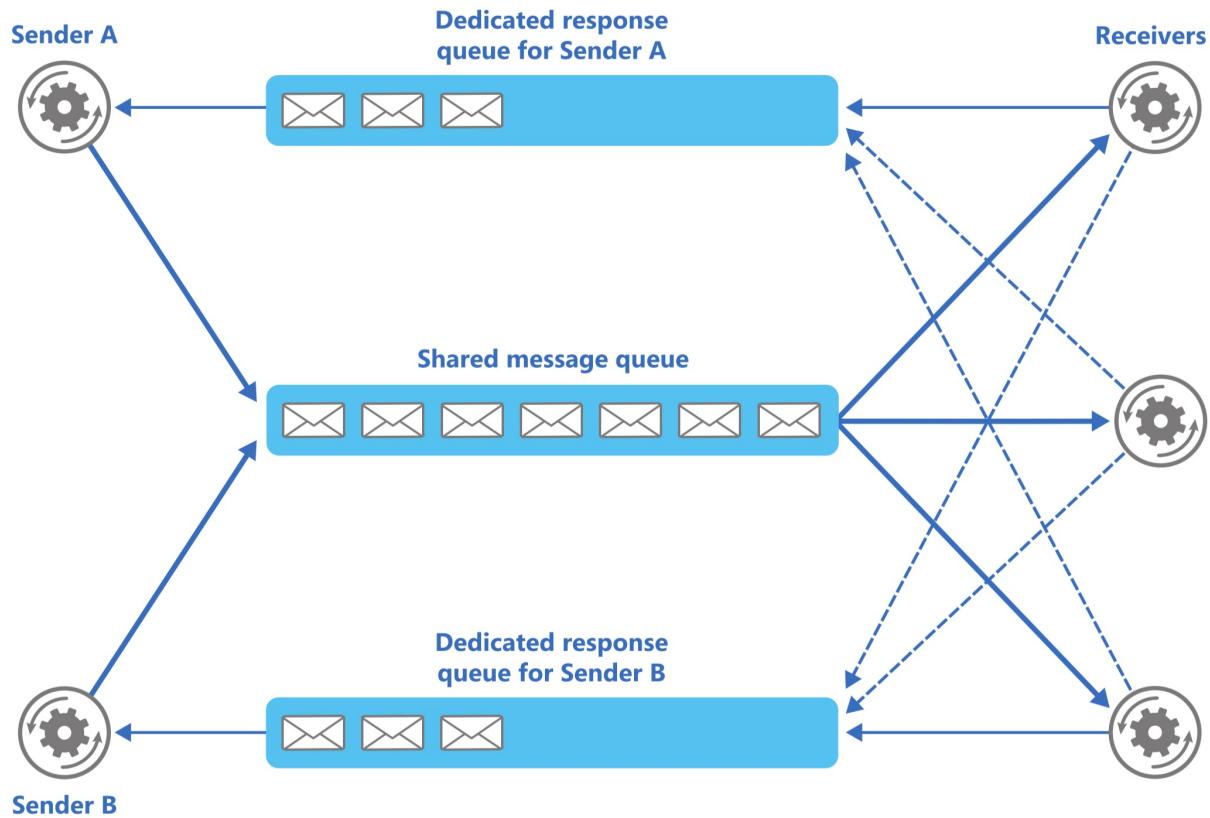


Abbildung 1: Request/Response Messaging [Cloud Design Patterns, S. 168].

Broadcast messaging wird eingesetzt wenn mehrere Empfänger die gleiche Nachricht erhalten sollen. Der Absender legt seine Nachrichten in eine Haupt-Queue, von der diese Nachrichten dann in Empfängerspezifische Queues umkopiert werden.

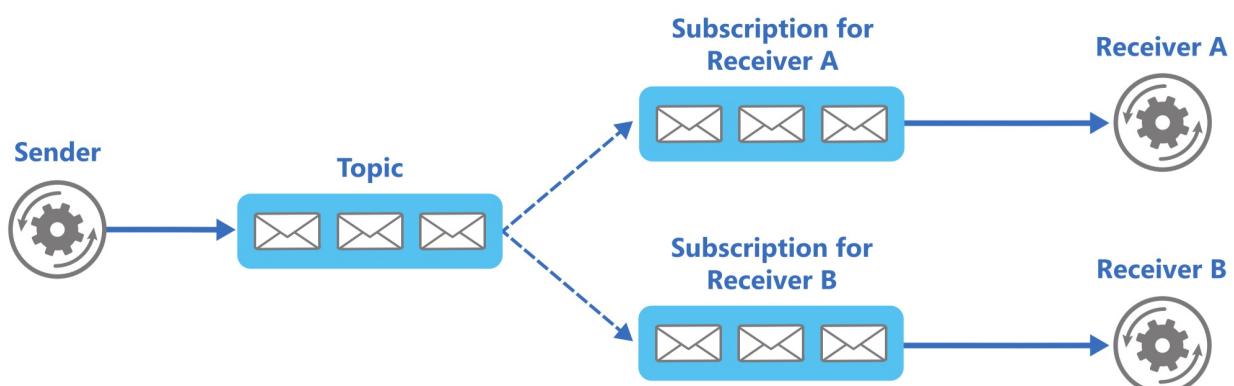


Abbildung 2: Broadcast Messaging [Cloud Design Patterns, S. 169].

Scenarien für asynchronen Nachrichtenaustausch (Auswahl)

Decoupling workloads: Das Trennen des Aufrufens einer Aktion von dessen tatsächlicher Durchführung (Beispiel: Ein Anwender ruft eine Aktion über eine Website auf, dessen Durchführung erfolgt jedoch auf einem Application-Server). (Vgl. S. 169)

Temporal decoupling: Sender und Empfänger können zu unterschiedlichen Zeiten arbeiten. Ein Empfänger kann Nachrichten auch verarbeiten, wenn der Sender nicht zur Verfügung steht. (Vgl. S. 169)

Load balancing: Es kommen verschiedene Empfänger auf mehreren Servern zum Einsatz. Die Architektur ist durch das hinzufügen weiterer Server skalierbar. (Vgl. S. 169)

Load leveling: Beim Erhalten vieler Nachrichten zur gleichen Zeit wird das System nicht direkt in die Höchstleistung getrieben. Der Queue-Mechanismus sorgt für ein langsames Steigen der Anfragen, sodass dem System die Möglichkeit geboten wird, sich den Gegebenheiten anzupassen. (Vgl. S. 170)

Cross-platform integration: Das Zusammenspiel verschiedener Software-Systeme kann über eine Queue erleichtert werden. Eine Queue kann so implementiert werden, dass die Interaktion vollkommen System- und Sprachenunabhängig ist. Es bedarf lediglich einer Definition von Schnittstellen und des Formats der Nachrichten. (Vgl. S. 170)

Überlegungen zur Implementierung von asynchronem Nachrichtenaustausch (Auswahl)

Message ordering: Die Reihenfolge der Nachrichten kann in machen Software-Systemen eine Rolle spielen. Nicht in allen Queues wird eine bestimmte Abarbeitungs-Reihenfolge garantiert. Das "Priority Queue Pattern" realisiert einen Mechanismus der das Abarbeiten bestimmter Nachrichten vor anderen umsetzt (Priorisierung). (Vgl. S. 171)

Message grouping: Typischerweise sollen Nachrichten innerhalb einer Queue unabhängig sein. In manchen Fällen lassen sich die Abhängigkeiten allerdings nicht vollkommen eliminieren. In diesem Fall sorgt "Message Grouping" für das Abarbeiten zusammengehöriger Nachrichten durch den gleichen Empfänger. (Vgl. S. 171)

Idempotency: (Idempotenz) beschreibt, dass die mehrfache Durchführung einer Operation mit den gleichen Daten immer zum gleichen Ergebnis führt. Im Bezug auf den Nachrichtenaustausch ist gemeint, dass auch wenn eine Nachricht fehlerhafterweise mehr als einmal empfangen und bearbeitet wird, die mehrfache Ausführung keine Auswirkungen auf das System hat. (Vgl. S. 171)

Verwandte Entwurfsmuster und Empfehlungen (Auswahl)

Autoscaling Guidance: Sobald die Anzahl der Nachrichten innerhalb der Queue eine bestimmte Schwelle unter- oder überschreitet, kann es sinnvoll sein einen Teil der Empfänger ab- bzw. hinzuzuschalten. (Vgl. S. 172)

Circuit Breaker Pattern: Wenn ein Empfänger oder Sender sich wiederholt nicht mit der Queue verbinden kann, ist es Sinnvoll weitere Versuche zu unterbinden, bis der Fehler gefunden respektive behoben wurde. Dieser Mechanismus wird mit dem "Circuit Breaker Pattern" realisiert. (Vgl. S. 172)

Scheduler Agent Supervisor Pattern: Der Nachrichtenaustausch ist häufig Teil eines größeren Workflows. Das Scheduler Agent Supervisor Pattern beschreibt, wie der Nachrichtenaustausch genutzt werden kann um Aktionen innerhalb von verteilten Systemen zu koordinieren, sowie andere verteilte Ressourcen sinnvoll zu nutzen. (Vgl. S. 173)

Autoscaling Guidance

Was ist Autoscaling?

Autoscaling ist ein dynamischer Prozess der Ressourcen so ver-/ bzw. einteilt, dass eine Applikation die Anforderungen an Performance und die SLAs (service level agreements) erfüllt. Häufig ist er ein automatisierter Prozess, der kontinuierlich die Performance überprüft und entscheidet, ob Ressourcen hinzugefügt oder entfernt werden sollen. Autoscaling bestimmt sämtliche Ressourcen, nicht nur Rechenressourcen.

Scaling Arten

Vertikales Skalieren umfasst das Umstrukturieren der Lösungen durch den Einsatz verschiedener Hardware. In einer Cloud stellt vertikales Skalieren bessere Ressourcen zur Verfügung und überträgt das System auf diese. Durch vertikales Skalieren wird ein System häufig zeitweise nicht erreichbar sein.

Horizontales Skalieren setzt das System auf zusätzliche Ressourcen auf, dabei kann das System ohne Unterbrechung weiterlaufen. Elemente die das System mit einbeziehen, können auf diesen zusätzlichen Ressourcen verfügbar gemacht werden und je nach Bedarf wieder runtergefahren werden. Viele Cloud Systeme nutzen diese Art des Skalierens.

Implementierung einer Autoskalierungsstrategy

Umfasst typischerweise die folgenden Prozesse und Komponenten:

- Einfangen der Schlüssel Performance und Skalierungsfaktoren wie z.B. Antwortzeiten, Warteschlangenlänge, Speichernutzung, etc
- Überwachungskomponenten
- Entscheidungslogik die anhand der überwachten Skalierungsfaktoren entscheidet ob skaliert wird oder nicht. Hierbei spielt die Zeit eine wichtige Rolle.
- Ausführungskomponenten die für Aufgaben verantwortlich sind. Sie nutzen typischerweise Tools und Skripte für:
 - Versorgung mit Ressourcen
 - Rekonfigurierung des Systems
- Testen und Validieren der Strategy

Immer häufiger werden Cloud Systeme mit einem Build-in tooling ausgestattet, um bei der Implementierung des Autoscalings zu helfen.

Gesichtspunkte für das Implementieren von Autoscaling

Autoscaling bietet keine sofortige Lösung, einfach nur Ressourcen oder mehr Instanzen eines Prozesses hinzufügen garantiert keine Performancesteigerung. Folgende Punkte sollten in Betracht gezogen werden:

- Das System sollte horizontal Skalierbar sein. Vermeiden von Instanzähnlichkeiten, keine Codelösungen, die immer in einer spezifischen Instanz eines Prozesses laufen. Anfragen derselben Quelle werden nicht immer der selben Instanz zugeteilt. Autoscaling kann zusätzliche Instanzen eines Dienstes hinzufügen!
- Langanhaltende Aufgaben sollten horizontale und vertikale Skalierung unterstützen. Ohne entsprechende Beachtung könnten Instanzen von Prozessen nicht sauber heruntergefahren werden oder sogar Daten verlieren. Idealerweise wird bei solch langanhaltenden Aufgaben die Abwicklung in kleinere Stücke aufgeteilt. Das "Pipes and Filters Pattern" zeigt ein Beispiel für die Umsetzung.
- Beinhaltet die Lösung mehrere Elemente wie Web Rollen, Arbeiter Rollen und andere Ressourcen, dann sollten diese Elemente als Einheit skaliert werden. Es ist wichtig, die Beziehung der Elemente zu verstehen und sie in Gruppen aufzuteilen.
- Um exzessives horizontales Skalieren zu vermeiden, sollten man dem Autoscaling Grenzen setzen. Autoscaling ist nicht der beste Mechanismus um plötzlichem Arbeitsaufwand entgegenzuwirken, da es Zeit braucht um Instanzen zu starten oder Ressourcen hinzuzufügen. (Throttling Pattern)
- Überwachen und Aufzeichnen von Details jedes Autoscaling Prozesses. Mithilfe dieser Informationen kann die Effektivität der Strategy gemessen werden.

Autoscaling in einer Windows Azure Lösung

- Das *Windows Azure Autoscaling* unterstützt die üblichen Skalierungsszenarien und man kann mit Hilfe des Windows Azure Management Portal Lösungen konfigurieren.
- Die *Microsoft Enterprise Library Autoscaling Application Block* ermöglicht eine Skalierungslösung basierend auf spezifischen Regeln und Performancedaten. Dieser Ansatz ist flexibler und komplexer, braucht aber Code um Performancedaten zu bekommen.
- Die *Microsoft Windwos Azure Monitioring Services Management Library* stellt den Zugang zu den *Windows Azure Monitoring Service* Operationen bereit. Sie beinhaltet eine vereinheitlichte API für das Abrufen und Konfigurieren von Metriken, Warnungen und Autoscaling Regeln der Windows Azure Dienste.

Benutzen von Windows Azure Autoscaling

Es gibt die folgenden zwei Möglichkeiten um Windows Azure für das Autoscaling zu konfigurieren:

- basierend auf Metriken wie die durchschnittliche CPU Leistung der letzten Stunde oder das Backloggen von Elementen in einer Mitteilungswarteschlange. Die Parameter des

Windows Azure Autoscaling können der Skalierung des Systems angepasst werden. Autoscaling ist kein sofortiger Prozess, es braucht Zeit um auf Metriken zu reagieren. Windows Azure setzt Regeln, die z.B. nur eine Skalierung in einem 5 Minuten Intervall zulässt.

- zeitbasierte Autoskalierung um sicherzustellen, dass Instanzen auch verfügbar sind wenn sie gebraucht werden.

Verwandte Muster und Richtlinien

Bei der Verwendung von Autoscaling sind außerdem die Muster "Throttling Pattern", "Competing Consumers Pattern" und die Richtlinien "Instrumentation and Telemetry Guidance" relevant.

Soft Skills für Softwareentwickler

Zusammenfassung des Buches:

Titel: Soft Skills für Softwareentwickler

Verfasser: Uwe Vigenschow, Björn Schneider, Ines Meyrose

Verlag: dpunkt.verlag

Jahr: 2014

ISBN: 978-3-86490-190-4, 978-3-86491-529-1 (eBook)

Zusammenfassung von: Jonas Wiese, Daniel Beneker, Fabian Lorenz

5 Konfliktmanagement

Konflikte analysieren

Konfliktbeschreibung

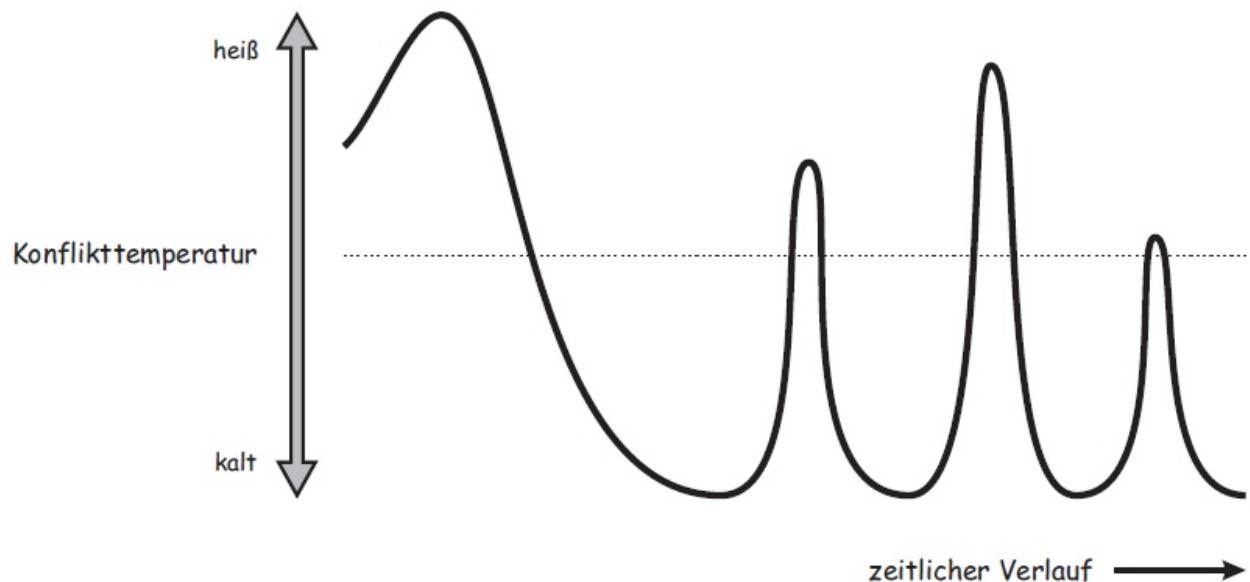
Ein Konflikt im IT-Bereich lässt sich als spezielle soziale Beziehung definieren, die folgende Eigenschaften besitzt:

- Es gibt entgegengesetzte, widersprüchliche Ziele, Interessen oder Handlungsweisen
- Es sind zwei oder mehr voneinander abhängige Personen daran beteiligt
- Die Beteiligten erkennen zum Zeitpunkt des Konflikts keine sofortige Lösung

Innerhalb von IT-Projekten kommt es sehr häufig zu Konflikten. Einschränkende Rahmenbedingungen, eintretende Risiken, Zeitdruck und gegebenenfalls unterschiedliche Zielvorstellungen von Stakeholdern führen zu Konflikten im Team, sowie mit bzw. zwischen den Stakeholdern. Auch die Auswirkungen eines Projekts sind Auslöser von Konflikten, da beispielsweise die Arbeitsbedingungen der Anwender verändert werden, oder im Extremfall durch neue Software sogar Arbeitsplätze wegfallen. Da für den Erfolg eines Projekts die Zusammenarbeit mit den Anwendern extrem wichtig ist, sind solche Konfliktpotentiale äußerst risikoreich. Konflikte helfen aber auch dabei, Unterschiede offenzulegen. Ein konstruktiver Umgang mit Konflikten ist daher essenziell für den Projekterfolg.

Heiße und kalte Konflikte

Konflikte lassen sich in zwei Formen einteilen: heiße und kalte Konflikte. Heiße Konflikte sind laut, die Konfliktparteien sind (über-)motiviert und suchen die direkte Konfrontation untereinander. Kalte Konflikte sind dagegen unterschwellig, es herrscht Frustration und Enttäuschung und der direkte Kontakt wird von den Konfliktparteien gemieden.



Die Temperatur eines Konflikts verändert sich im Laufe der Zeit. Von außen sichtbar ist er dabei nur oberhalb der gestrichelten Linie, wenn er auflodert.

Konflikte im kalten Zustand sind schwer zu erkennen und zu lösen. Es empfiehlt sich daher zu warten bis der Konflikt wieder in eine heiße Phase übergeht, eventuell ihn sogar selbst anzuhühen. Im heißen Zustand ist der Konflikt dann wieder zugänglicher, man kann ihn angehen und letztendlich lösen.

Konfliktarten

Sachkonflikt

Unterschiedliche Meinungen treffen im Rahmen einer sachlich geführten Diskussion aufeinander.

- Ursache: Wegen fehlender Informationen, unterschiedlicher Kenntnisse und Erfahrungen der beteiligten Personen oder durch Missverständnisse werden unterschiedliche Wege beim Lösen eines Problems gesehen.
- Beispiel: Designentscheidungen können häufig wegen fehlender Informationen nur auf Basis der eigenen Erfahrung getroffen werden. Dabei kann es beispielsweise bei der Auswahl eines geeigneten Patterns zu einem Sachkonflikt kommen.
- Lösungsweg: Das Informationsdefizit muss abgebaut werden. Es müssen Informationen gesammelt werden und alle Beteiligten auf den gleichen Stand gebracht werden. Dazu werden offene Fragen gestellt und aktives Zuhören genutzt. Häufig ist auch das prototypische Ausprobieren ein erfolgreicher Lösungsweg im IT-Bereich. Somit können konkrete Informationen schnell beschafft werden und alle Szenarien, die auf ungenauen Informationen beruhen, können eliminiert werden.

Beziehungskonflikt

Die Beziehung zwischen zwei Personen ist gestört und führt zu einer einseitigen oder gegenseitigen Abwertung.

- Ursache: Vorurteile, Ängste, mangelnder Respekt und nicht ausreichende Wertschätzung.
- Beispiel: Das teilweise immer noch existierende Vorurteil gegenüber Frauen und Technik oder Vorurteile aufgrund von Altersunterschieden.
- Lösungsweg: Aufbau von gegenseitigem Respekt, Wertschätzung und Verständnis.

Interessenkonflikt

Dem Verhalten der Konfliktpartner liegen unterschiedliche Interessen zugrunde, die nicht direkt geäußert werden, sondern hinter Positionen versteckt sind.

- Ursache: Es bestehen unterschiedliche Interessen, Diese werden jedoch nicht offen geäußert, sondern vordergründige Positionen werden ausgetauscht.
- Beispiel: Bei Gehaltsverhandlungen werden immer nur Forderung und Gegenposition genannt und wiederholt: "Ich will 300€ mehr im Monat!" "Ich kann Ihnen aber nur 50€ bieten!".
- Lösungsweg: Über das Erkennen und Verstehen der tatsächlichen Interessen wird versucht ein Lösungsweg zu finden. Eine offene und vertrauensvolle Umgebung kann dabei hilfreich sein, diese Interessen zu erkennen. Im Beispiel der Gehaltsverhandlung muss erkannt werden, warum der Arbeitnehmer mehr Gehalt möchte, beispielsweise vielleicht weil er mehr Geld für seinen Lebensunterhalt braucht, oder weil seine Kollegen mehr verdienen und er sich ungerecht behandelt fühlt. Beim Arbeitgeber können begrenzte Mittel eine Rolle spielen oder vielleicht hängt seine Prämie daran, möglichst niedrige Gehaltsabschlüsse zu erzielen. Sind die tatsächlichen Interessen identifiziert, kann ein Interessenausgleich verhandelt werden.

Wertekonflikt

Als Wert wird bezeichnet, was eine einzelne Person als wichtig oder lohnend einzuschätzen gelernt hat. Dies können Lebensprinzipien sein, oder Ziele die man erreichen möchte. Ein Wert ist also recht unabhängig von konkreten Situationen oder anderen Personen. Auch ändern sich die Werte einer Person oder einer Gesellschaft mit der Zeit.

- Ursache: Unterschiedliche, dauerhafte Wertvorstellungen treffen aufeinander
- Beispiel: Zwei Entwickler teilen sich einen Arbeitsplatz, z.B. weil sie beide halbtags arbeiten. Der eine möchte den Schreibtisch stets frei und aufgeräumt haben, der andere möchte seine Arbeitsmittel verteilt und stets zugreifbar auf der Schreibtischfläche

verstreut haben.

- Lösungsweg: Die gegenseitigen Wertvorstellungen müssen von der Beteiligten offengelegt und anerkannt werden. Ziel ist dabei die Unterschiedlichkeit zu erkennen und zu akzeptieren. Da Wertvorstellungen nicht kurzfristig verändert werden können, ist es nur möglich Regeln aufzustellen, die von den Konfliktparteien akzeptiert und eingehalten werden müssen, sodass eine Zusammenarbeit weiterhin erfolgen kann. Im Beispiel könnten sich die beiden Personen darauf einigen, dass jeder einen bestimmten Teil des Schreibtisches bekommt. Ist dies nicht dauerhaft möglich, können nur noch andere Wege gefunden werden, in denen dieser Wertkonflikt nicht mehr auftritt.

Rollenkonflikt

Rollenkonflikte entstehen durch unterschiedliche Anforderungen, die von verschiedenen Seiten an einer Rolle gestellt werden. Sie sind damit unabhängig von der tatsächlichen Person, die diese Rolle einnimmt.

- Ursache: Eine Person hat eine Rolle inne, an die unterschiedliche Erwartungen gestellt werden.
- Beispiel: Mittlere Führungskräfte bekommen Anforderungen von der Geschäftsführung, sowie von den ihr unterstellten Mitarbeitern. Beide haben dabei meist unterschiedliche Erwartungen.
- Lösungsweg: Rollenkonflikte können meist nicht vollständig aufgelöst werden. Stattdessen sollten die unterschiedlichen Erwartungen offengelegt und transparent priorisiert werden. Somit werden die Entscheidungen zumindest nachvollziehbar. Auch kann es oft helfen die Kompetenzen der beteiligten Rollen zu verteilen bzw. zu optimieren.

Ein weiteres bekanntes Beispiel für Rollenkonflikte in der IT sind dabei die Konflikte zwischen Entwicklern und der (externen) Qualitätssicherung. Beide verfolgen zwar dasselbe Ziel, die Software zu verbessern, allerdings kann beispielsweise durch schlechte Kommunikation von Fehlern die Situation schnell verschlechtert werden. Entwickler können sich stark mit ihren geschriebenen Quellcode identifizieren und sich durch Kritik am Quellcode persönlich angegriffen fühlen.

Systemkonflikt

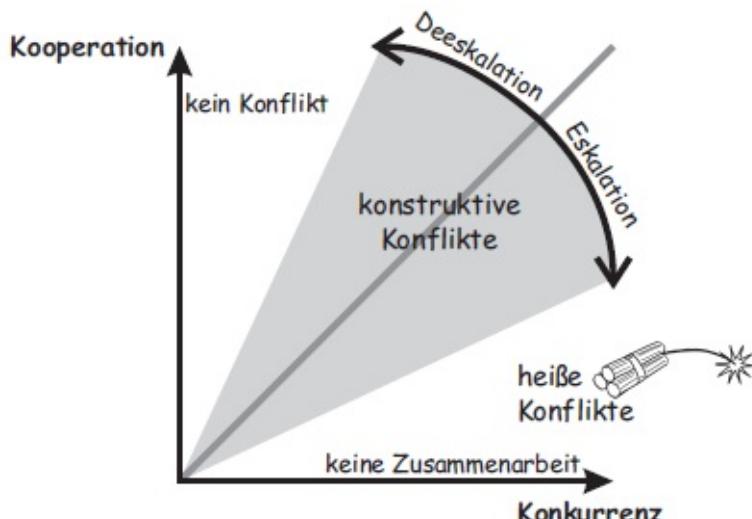
Bei einem Systemkonflikt wirken Konflikte von außerhalb der eigentlichen Gruppe, also aus dem umgebenden System auf die Gruppe ein. Ein Systemkonflikt kann daher nicht innerhalb der eigentlich betrachteten Gruppe aufgelöst werden.

- Ursache: Aufgrund äußerer Engpässe oder anderer, widersprüchlicher Rahmenbedingungen kommt es zu Konfliktsituationen innerhalb einer Gruppe.

- Beispiel: In der Entwicklungsabteilung haben zwei Mitarbeiter gekündigt, das derzeit laufende Projekt ist jedoch von äußerster Wichtigkeit für die Existenz des Unternehmens. Aufgrund einer Unternehmensweiten Umstrukturierung herrscht jedoch ein genereller Einstellungsstopp für neue Mitarbeiter und Mittel für die Beauftragung von externen Mitarbeiter steht auch nicht zur Verfügung.
- Lösungsweg: Die Situation kann im Rahmen des Projektmanagements nur selten aufgelöst werden, da die vorhandenen Reserven meist nicht ausreichend für solche Extremsituationen sind. Es kann daher nur Klarheit über die Zusammenhänge geschaffen werden und der Konflikt an die Stelle delegiert werden, die ihn auflösen kann. Für die Akzeptanz der übergeordneten Instanz ist es wichtig, im Rahmen der Eskalation auch mehrere, alternative und konkrete Lösungsszenarien zu benennen.

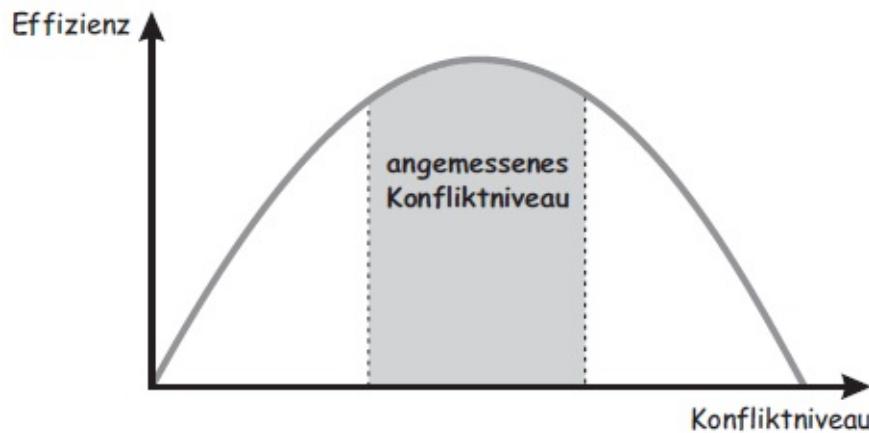
Konfliktdynamik

Konflikte können selbst in den bestmöglich funktionierenden Gruppen entstehen. Dies muss jedoch nicht negatives sein, denn Konflikte können dabei helfen die Entwicklung weiterzubringen und kreative Lösungen hervorzubringen. Dafür müssen sie jedoch konstruktiv sein. Zu viel Kooperation und damit ein nicht vorhandenes Konfliktpotential führt zu einfallslosen Lösungen. Zu viel Konkurrenz führt dazu, dass jeder nur noch für sich selbst arbeitet und somit nie das bestmögliche Potential der Gruppe ausgeschöpft werden kann.



Konstruktive Konflikte entstehen aus dem Wechselspiel von Kooperation und Konkurrenz der beteiligten Personen

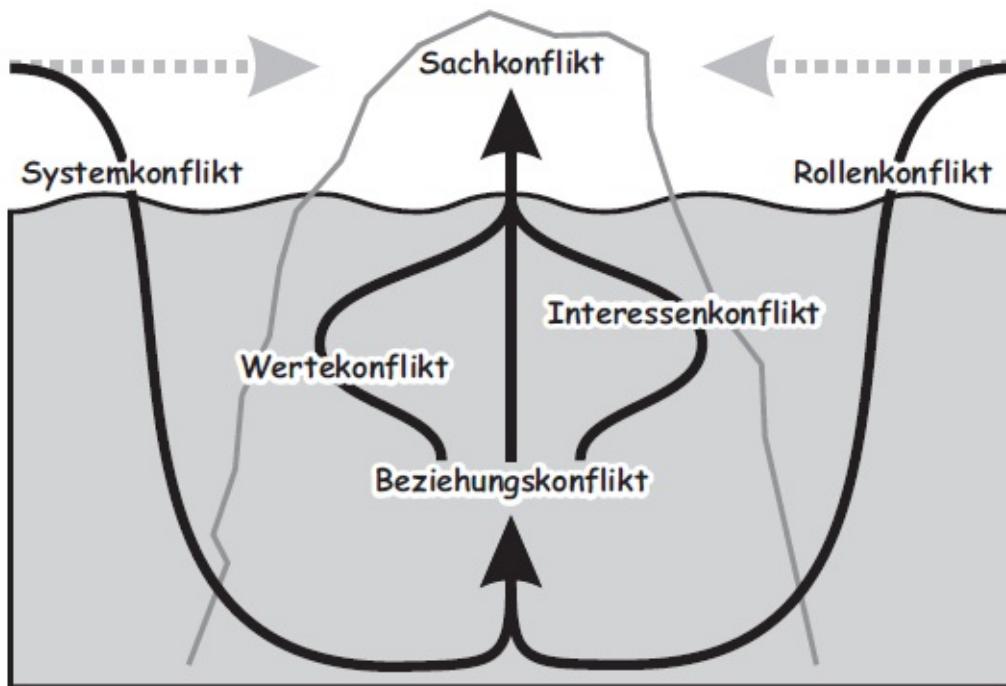
Es muss also ein angemessenes, konstruktives Konfliktniveau erreicht werden, um die bestmöglichen Lösungen zu finden. Dafür ist es notwendig, dass die Zusammenarbeit mit gegenseitiger Wertschätzung und einem gemeinsamen Zielkonsens stattfindet.



Konflikte sind notwendig, um uns zu hinterfragen und Lösungen zu optimieren. Die Kunst ist es, dafür ein angemessenes Konflikt niveau zu finden

Grundsätzliche Konfliktlösungsstrategie

Ein realer Konflikt besteht meist aus einer Vermengung von verschiedenen Konfliktarten. Jede einzelne Art kann dabei entweder gerade heiß oder erkaltet sein. Hinter einem Sachkonflikt kann beispielsweise auch ein Beziehungskonflikt stehen, der vor längerer Zeit seinen Anfang nahm, jedoch immer noch unterschwellig eine Rolle bei dem derzeitigen



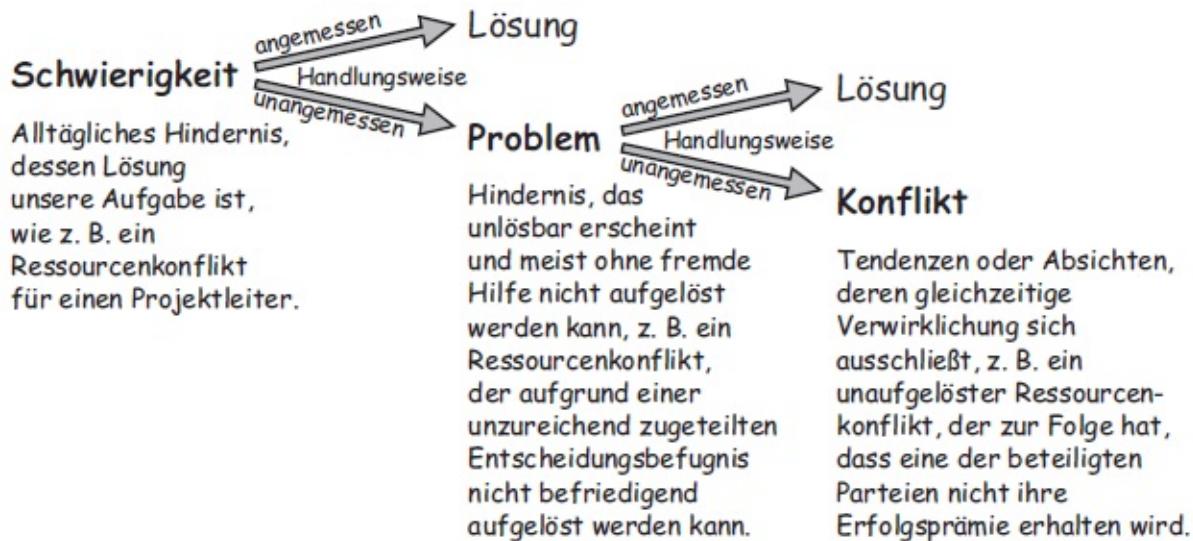
Eine sinnvolle Reihenfolge, in der die einzelnen Konfliktanteile nach den Konfliktarten angegangen werden.

Die verschiedenen Konflikte werden dabei als Eisbergmodell dargestellt. Der Sachkonflikt ist dabei meist nur die Spitze des Eisbergs, darunter liegen aber deutlich mehr, meist nicht sichtbare andere Konflikte. Statt also den Sachkonflikt direkt anzugehen, sollten zuerst die zugrunde liegenden Konflikte gelöst werden.

Konfliktmuster rechtzeitig erkennen

Für ein erfolgreiches Konfliktmanagement ist es notwendig aufkommende Konflikte frühzeitig wahrzunehmen. Nicht jedes auftretende Hindernis in einem Projekt wird sich direkt zu einem Konflikt entwickeln. Zu Beginn sind es meist nur Schwierigkeiten, alltägliche Hindernisse, die meist die betroffene Person selbst lösen kann. Wird angemessen auf die Schwierigkeit reagiert und eine Lösung gefunden kann kein Konflikt mehr darauf entstehen. Eine unangemessene Behandlung der Schwierigkeit führt jedoch zu einem Problem. Ein Problem ist dabei ein Hindernis, das für eine einzelne Person meist unlösbar erscheint und nur mit Hilfe von anderen überwunden werden kann. Wird auf ein Problem wieder angemessen reagiert, kann eine Lösung gefunden werden und es entsteht kein Konflikt. Wird jedoch auf das Problem auf unangemessene Weise behandelt, entwickelt sich das Problem zu einem Konflikt. Um Konflikten somit vorzubeugen, ist es also wichtig, dass bereits Schwierigkeiten gut behandelt werden und es erkannt wird, wenn sie zu einem Problem werden.

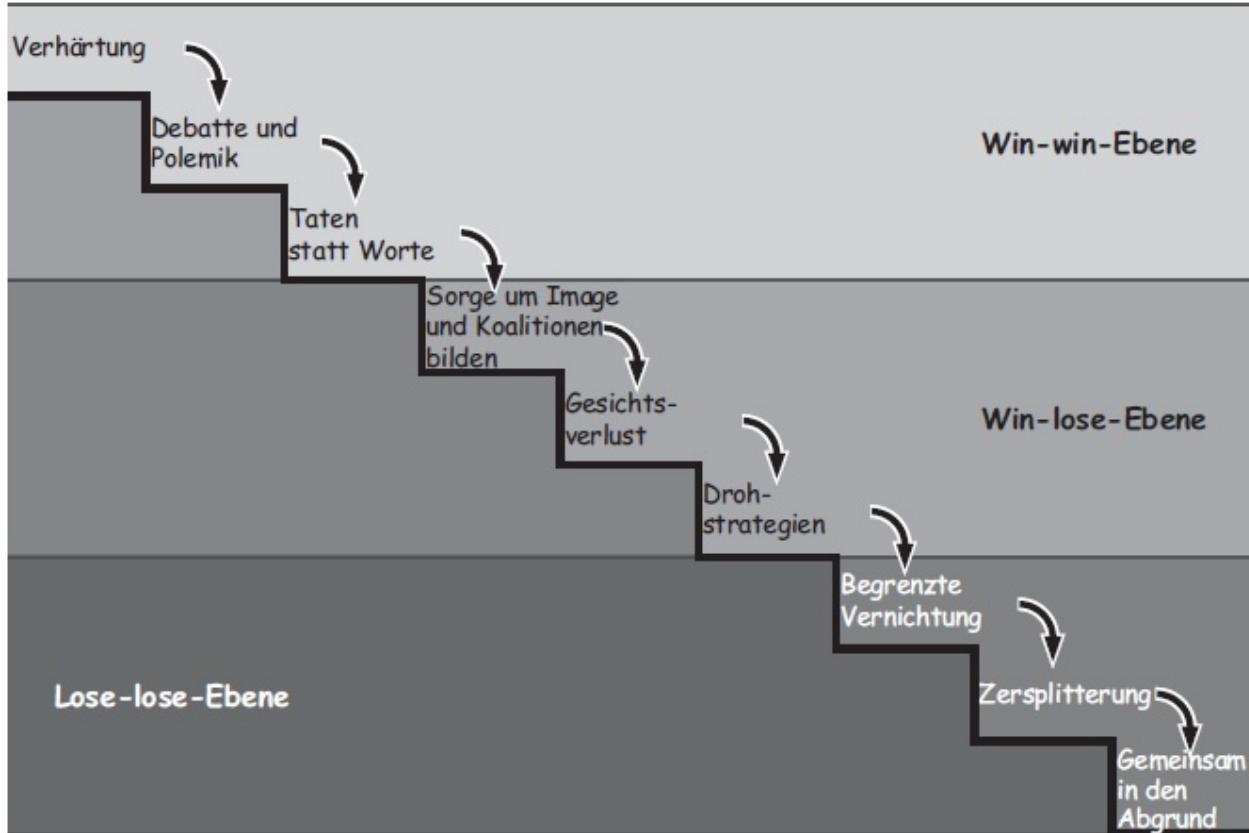
angemessen reagiert, kann eine Lösung gefunden werden und es entsteht kein Konflikt. Wird jedoch auf das Problem auf unangemessene Weise behandelt, entwickelt sich das Problem zu einem Konflikt. Um Konflikten somit vorzubeugen, ist es also wichtig, dass bereits Schwierigkeiten gut behandelt werden und es erkannt wird, wenn sie zu einem Problem werden.



Aus alltäglichen Schwierigkeiten werden durch unangemessene Lösungsversuche Probleme. Werden diese Probleme nicht gelöst, entstehen daraus Konflikte.

Konflikteskalationsstufen

Die Entwicklung eines Konflikts kann auch anhand eines neunstufigen Phasenmodells dargestellt werden. Jeweils drei Phasen sind dabei zu einer Ebene zusammengefasst. In der ersten Ebene, der Win-Win Ebene, kann dabei noch jeder Beteiligte am Konflikt einen Gewinn bei der Lösung des Konflikts gewinnen. In der Win-lose Ebene gibt es jedoch mindestens einen Verlierer, beispielsweise weil jemand anderes seine Meinung durchgesetzt hat, oder es wird ein Kompromiss gefunden. Dann hat zwar keiner einen vollständigen Gewinn gemacht, aber auch nicht komplett verloren. Die Lose-lose Ebene hat zur Eigenschaft, dass alle Beteiligten nur noch Verlieren können. Hier wird entweder der eigene Verlust in Kauf genommen, solange die andere Partei noch größere Verluste hat, oder er wird mit allen Mitteln versucht die andere Partei vom Gewinnen abzuhalten.



Konflikte eskalieren häufig nach dem neunstufigen Phasenmodell der Eskalation von Friedrich Glasl. Jeweils drei Stufen bilden eine Ebene, die die Möglichkeiten der beteiligten Parteien beschreibt.

1. Stufe: Verhärtung

Alltägliche Schwierigkeiten lassen Meinungsverschiedenheiten aufeinanderprallen. Da Meinungsverschiedenheiten recht alltäglich sind, werden sie häufig auch nicht als Beginn eines Konflikts wahrgenommen.

2. Stufe: Debatte und Polemik

Konfliktpartner überlegen sich Strategien, um andere von ihren Argumenten zu überzeugen und unter Druck zu setzen.

3. Stufe: Taten statt Worte

Konfliktpartner erhöhen durch Taten den Druck auf die anderen um ihre jeweilige Meinung durchzusetzen. Gespräche werden abgebrochen, es findet keine direkte Kommunikation mehr statt.

4. Stufe: Sorge ums Image und Koalitionsbildung

Konfliktparteien versuchen Unterstützer für ihre Sache zu finden, Gegner werden denunziert. Konflikt verschärft sich so stark, dass es nicht mehr um die eigentliche Sache geht, sondern nur noch darum zu gewinnen.

5. Stufe: Gesichtsverlust

Mit Gesichtsverlust ist der Verlust der moralischen Glaubwürdigkeit der Gegner gemeint. Sie sollen in ihrer Identität vernichtet werden, unter anderem durch Unterstellungen und politischen Machtspielen. Verlust des kompletten gegenseitigen Vertrauensverhältnisses.

6. Stufe: Drohstrategien

Konfliktparteien versuchen die Situation durch Drohungen zu kontrollieren. Zwar können auch in den vorherigen Phasen schon Drohungen ausgesprochen werden, in dieser Stufe sind sie jedoch sehr ernst gemeint und auch umgesetzt.

7. Stufe: Begrenzte Vernichtung

Konfliktparteien versuchen sich mit allen Tricks gegenseitig zu schaden. Der Gegner wird dabei nicht mehr als Mensch wahrgenommen und solange sein Schaden größer ist, wird auch ein eigener kleiner Schaden als Gewinn angesehen.

8. Stufe: Zersplitterung

Konfliktparteien versuchen sich mit Vernichtungsstrategien gegenseitig zu zerstören.

9. Stufe: Gemeinsam in den Abgrund

Eigene Vernichtung wird einkalkuliert, um den Gegner zu besiegen.

Kommunikationsmuster in Konflikten

Häufig werden in Konflikten destruktive Kommunikationsweisen genutzt. Dabei wird dies nicht bewusst gemacht, sondern man fällt aufgrund von Stress unbewusst in bestimmte Kommunikationsmuster zurück.

Um diese Muster zu verdeutlichen wird die Dimension eines Konflikts auf drei Aspekte reduziert.

Selbst: Die eigene Person, meine Bedürfnisse, Interessen, Anliegen, sowie mein Selbstwertgefühl

Andere: Die anderen Personen im direkten Umfeld und deren Bedürfnisse, Interessen und Anliegen

Kontext: Die konkrete Sache um die es geht und alle direkt damit im Zusammenhang stehenden fachlichen Aspekte.

Es können nun vier Kommunikationsmuster anhand daraus abgeleitet werden, wie eine Person in ihrer Kommunikation diese Aspekte gewichtet. Je nach Muster können dabei manche Sachen gut funktionieren, andere wiederum nicht. Es kann daher für die Konfliktbewältigung von Vorteil sein manche Muster durchzusetzen, oder zu verlassen.

Beschwichtiger: Fokussierung auf Kontext und Andere, Vernachlässigung des Selbst.

- **Motto:** Nur wenn ich für andere Sorge, geht es mir gut
- **Charakteristika:** Macht sich viele Gedanken und ist anspruchslos für sich selbst
- **Typische Syntax:** häufiger Gebrauch von Einschränkungen und Konjunktiven
- **Was funktioniert gut?** Harmonie schaffen, vermitteln im Team, Sympathie erzeugen, anpassungsfähig, hohes Risikobewusstsein, zuverlässig sein und sehr genau hinschauen (allerdings dafür langsamer)
- **Was funktioniert nicht?** Etwas durchsetzen, standfest sein, streiten (gibt schnell nach)

und aktive Führung

Rationalisierer: Fokussierung auf Kontext, Vernachlässigung von Selbst und der Anderen.

- **Motto:** Nur wenn wir ganz sachlich und logisch vorgehen, kommen wir zu einem guten Ergebnis
- **Charakteristika:** Fakten dominieren, zeigt keine Emotionen, detailverliebt
- **Typische Syntax:** Substantivierung, tilgen von Verben, Formulierungen mit "man"
- **Was funktioniert gut?** Forschen, Ziele und Maßnahmen finden, Wissen sammeln, Einzelarbeit, Regeln und Grenzen einhalten, Auswirkungen betrachten und Neutralität im Sinne eines Schiedsrichters
- **Was funktioniert nicht?** Unkonventionelle Wege gehen, Kompromisse erarbeiten, Konflikte und die dahinter stehenden Bedürfnisse erkennen

Ankläger: Fokussierung auf Kontext und Selbst, Vernachlässigung der Anderen.

- **Motto:** Nur wenn ich kämpfe, bekomme ich etwas
- **Charakteristika:** lebhaft, laut, schnell, aggressiv
- **Typische Syntax:** Formulierungen mit "andere" oder Universalquantoren wie "immer" oder "jeder"
- **Was funktioniert gut?** Etwas vorantreiben, effizient arbeiten, etwas optimieren, Klarheit schaffen, Sicherheit geben, schnell Ergebnisse liefern und Führung im Sinne von Orientierung
- **Was funktioniert nicht?** Kompromisse erarbeiten, andere einbinden, zuhören, genau arbeiten, Umwelt wahrnehmen, Fingerspitzengefühl und Zusammenarbeit

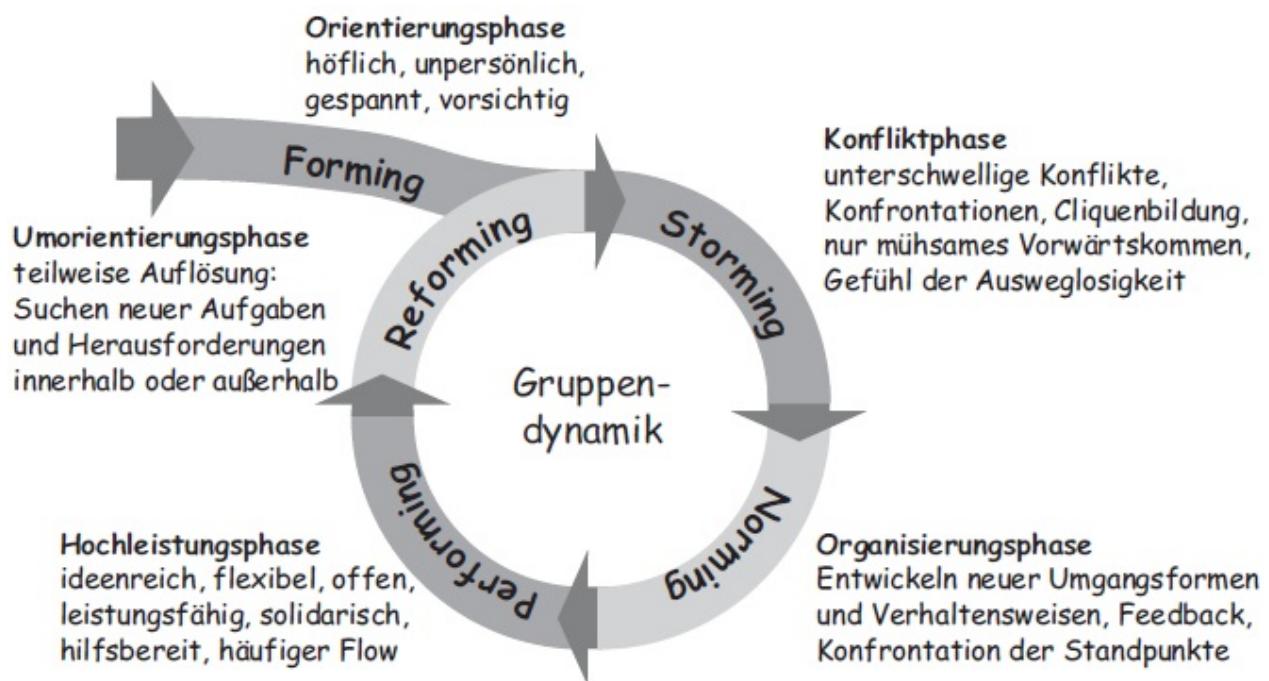
Ablenker: Fokussierung auf nichts, Vernachlässigung des Kontext, des Selbst und der Anderen

- **Motto:** Nichts ist wirklich wichtig
- **Charakteristika:** unstrukturiert bis chaotisch, unverbindlich, kreativ, immer gute Stimmung
- **Typische Syntax:** rascher Wechsel der Wortwahl, eher generelle und zukunftsorientierte Formulierungen, Worte ohne Beziehung zum Gesprächspartner und dadurch kaum konkret greifbar
- **Was funktioniert gut?** Neue Ideen einbringen, Begeisterung, Vision entwickeln und Schwung einbringen, Motivation sowie Innovation
- **Was funktioniert nicht?** Konflikte austragen, Verbindlichkeit schaffen und Dinge zu Ende bringen

Gruppendynamik

Konflikte zwischen Menschen können nur entstehen, wenn Menschen aufeinander treffen, oder in irgendeiner Beziehung zueinander stehen. Es wird also stets eine Gruppe von Menschen betrachtet. Jede Gruppe hat dabei eine eigene Dynamik und es kann innerhalb einer Gruppe zu Konflikten kommen. Des Weiteren besteht auch eine Beziehung zu anderen Gruppen und auch in dieser Beziehung kann es zu Konflikten kommen.

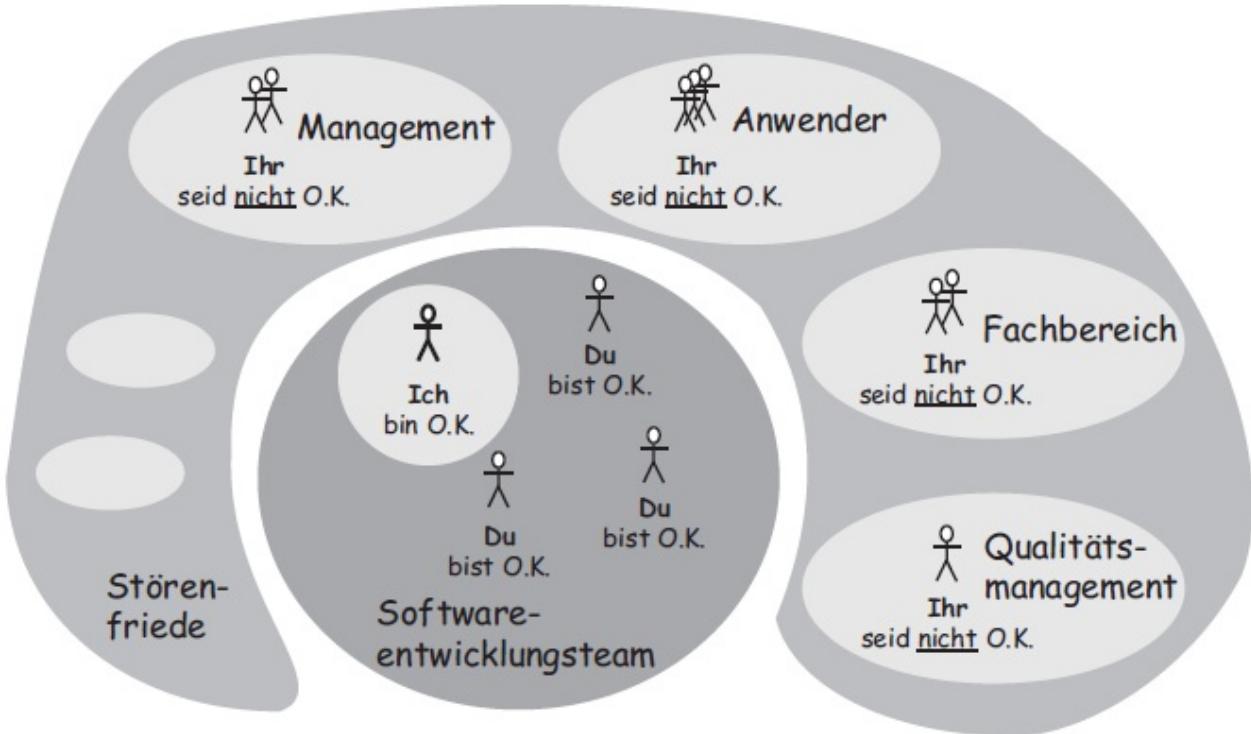
Die Struktur einer Gruppe ist ständigen Veränderungen unterworfen und sie entwickelt sich somit weiter. Diese Entwicklung kann dabei in mehreren Phasen erfolgen, dargestellt in einem Phasenmodell.



Jede Gruppe hat einen Lebenszyklus und obliegt damit einer inneren Dynamik. Damit verbunden ist eine Konfliktphase in der Findung einer Gruppe (Storming)

Zu Beginn des Lebenszyklus einer Gruppe steht dabei die Orientierungsphase, hier herrscht noch Unsicherheit und ein distanziertes Verhalten unter den Gruppenmitgliedern. In der anschließenden Konfliktphase beginnt der Machtkampf der Gruppenmitglieder untereinander. Jeder versucht seine gewünschte Rolle in der Gruppe einzunehmen, es kommt zu Konflikten und die Gruppe selbst kommt nur mühsam voran. Diese Phase ist die Härteprobe einer jeden Gruppe, ist sie überstanden geht es in die Orientierungsphase. Hier entwickeln sich die Umgangsformen und Verhaltensweisen, die Gruppenmitglieder haben ihren Platz gefunden und der Konkurrenzkampf weicht einer intensiven Zusammenarbeit. Die nächste Phase ist dann die Hochleistungsphase, die Gruppe ist eingespielt, kann optimal arbeiten und ihr Potential voll zur Geltung bringen. Nach dieser Phase erfolgt die Phase der Umorientierung. Einzelne Personen verlassen die Gruppe, andere kommen hinzu und es werden neue Aufgaben gesucht. Dabei kommt es wieder zu Spannungen und es tritt erneut die Konfliktphase ein.

Neben Konflikten in der Gruppe selbst, kann es auch zu Konflikten mit anderen Gruppen kommen. Als Entwickler ist man beispielsweise häufig den anderen Mitgliedern des Entwicklungsteams positiv gegenüber eingestellt. Ausstehenden Gruppen, wie beispielsweise der Qualitätssicherung, ist man jedoch negativ gegenüber eingestellt. Sie werden als Störenfriede gesehen, die die eigene Gruppe negativ beeinflussen. Diese Haltung gegenüber anderen Gruppen ist jedoch langfristig schädlich und von Konflikten geprägt.

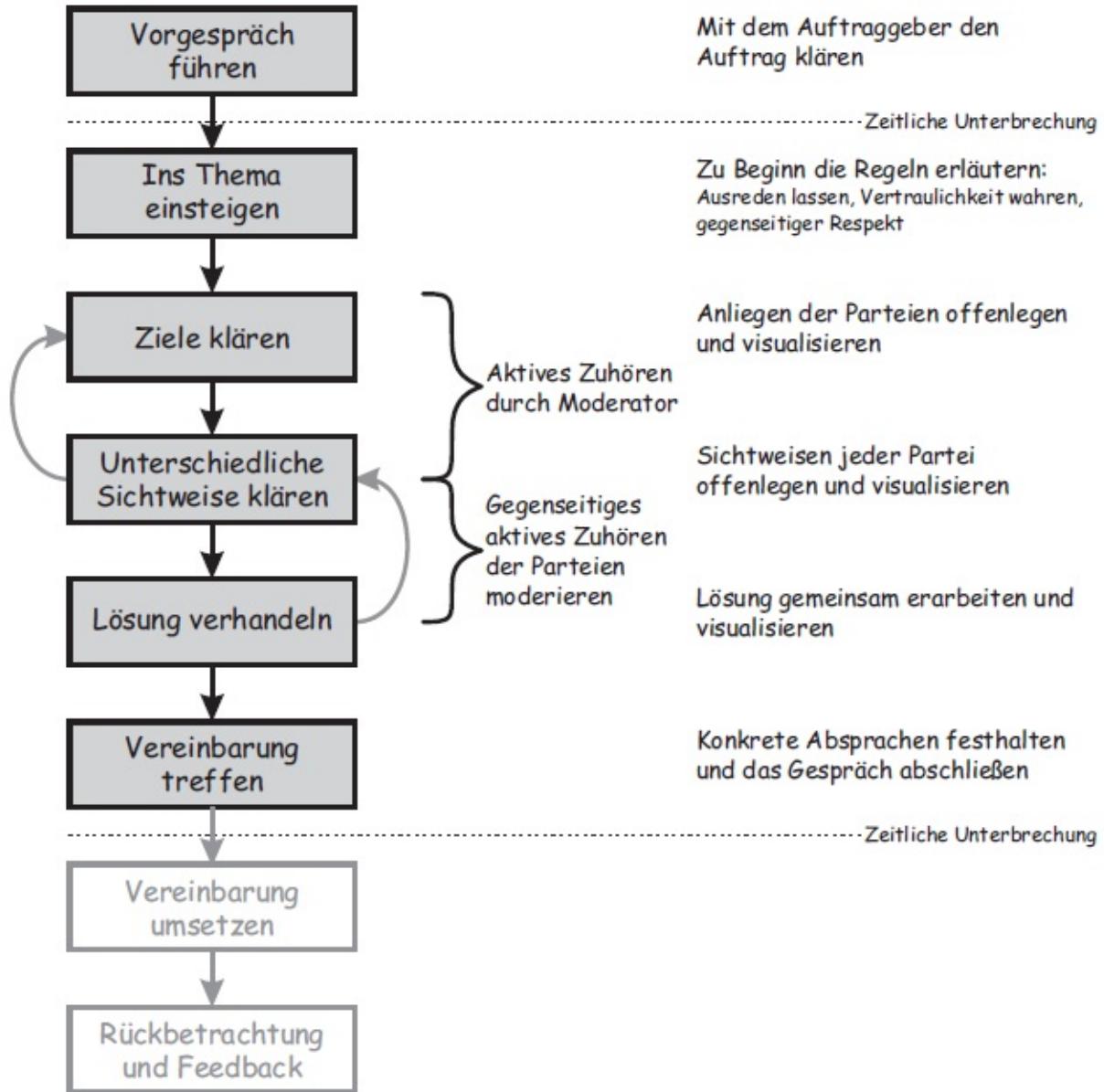


Die Nicht-O.K.-Einstellung zu Gruppen außerhalb der eigenen führt leicht zu Konflikten mit diesen Menschen.

Konflikte managen

Natürlich ist es nicht nur notwendig Konflikte rechtzeitig zu erkennen, sondern auch angemessen mit ihnen umzugehen. Häufig hilft dabei ein Kritikgespräch. Bei einem Kritikgespräch setzen sich zwei Personen zusammen und stellen ihre jeweils unterschiedlichen Sichtweisen dar. Dabei wird stark auf aktives Zuhören gesetzt, sobald eine Person ihre Sichtweise dargestellt hat, gibt die andere wieder, was sie verstanden haben. Dadurch wird ein gemeinsames Verständnis aufgebaut, auf dessen Grundlage nun konkrete Handlungen vereinbart werden können. Wie bei vielen anderen Konfliktaspekten, ist auch hier die gegenseitige Wertschätzung vonnöten: Kritik soll nicht dazu dienen die Person zu verletzen, sondern ihr zu helfen. Eine andere Art Konflikte zu managen besteht in der Konfliktmoderation. Dabei gibt es einen allparteilichen Moderator, der keiner Seite einen Vorzug gibt und allen Lösungsmöglichkeiten offen gegenübersteht. Dieser leitet die

Diskussion der Konfliktparteien und legt dabei den Fokus auf den Inhalt der Auseinandersetzung und dem Empfinden und Sichtweisen der Beteiligten. Er achtet darauf, dass ein aktives Zuhören stattfindet sowie darauf, dass sachliche Aussagen und positive Ergebnisse visualisiert werden. Der Ablauf einer Moderation kann dabei in acht Phasen erfolgen, dargestellt in der folgenden Abbildung.



Struktur einer Konfliktbearbeitung mit Moderation.

Hilft auch die Konfliktmoderation nicht mehr, so gibt es die Möglichkeit der Konfliktmediation. Dabei gibt es einen neutralen außer parteilichen Moderator, der den beteiligten Parteien dabei hilft, selbstständig eine Lösung zu finden. Die Konfliktparteien müssen dabei freiwillig an der Mediation teilnehmen, bereit sein fair und offen zu kommunizieren und für alle Lösungsmöglichkeiten offen sein. Der Ablauf einer Mediation ist dabei ähnlich wie bei der Konfliktmoderation, sie ist jedoch etwas förmlicher. Außerdem besitzt sie eine gesetzliche Grundlage, Mediatoren dürfen eine Mediation nur durchführen, wenn sie die gesetzlichen Voraussetzungen erfüllen.

Erfolgreich Verhandlungen führen

Die Fähigkeit, erfolgreich Verhandlungen zu führen, ist für Softwareentwickler von Vorteil, beispielsweise um wichtige Entscheidungen bei der Projektgestaltung mitzubestimmen.

Meist trifft man bei Verhandlungen auf zwei Verhandlungsstrategien, zum einen dem harten Verhandeln, bei dem der Sieg und somit der eigene Vorteil im Mittelpunkt steht, zum anderen das weiche Verhandeln, wo versucht wird eine möglichst konfliktfreie Übereinkunft zu finden.

	Hartes Verhandeln	Weiches Verhandeln
Ziel:	Sieg	Übereinkunft
Haltung:	Hart zu Menschen und Positionen, Verhandlungspartner sind Gegner, Misstrauen	Weich zu Menschen und Positionen, Verhandlungspartner sind Freunde, Vertrauen
Strategie:	Drohen, beharren auf Positionen, Zugeständnisse der anderen Seite sind der Preis für die Übereinkunft, Druck ausüben	Angebote machen, ändern von Positionen, einseitige Zugeständnisse, bestehen auf Übereinkunft, Druck nachgeben
Linie:	Verdeckt	Offen

Hartes und weiches Verhandeln in der Gegenüberstellung bzgl. Ziel, innerer Haltung, Strategie und Verhandlungslinie.

Beide Varianten haben jedoch Nachteile, harte Verhandlungen führen langfristig zu schlechten Geschäftsbeziehungen, weiche Verhandlungen führen dazu, dass die eigenen Ziele und Wünsche nicht beachtet werden.

Um erfolgreiche Verhandlungen zu führen, ist es daher hilfreich sich am Harvard Konzept zu orientieren. Hier wird eine dritte Möglichkeit des Verhandelns benutzt, das Sachorientierte Verhandeln.

	Sachorientiertes Verhandeln
Ziel:	Dauerhafte, vernünftige Übereinkunft
Haltung:	Teilnehmer sind Problemlöser bzw. Partner, offen und weich zu den Menschen, hart bei den eigenen Interessen
Strategie:	Menschen und Probleme getrennt behandeln, beiderseitige Interessen herausarbeiten, Optionen finden, Bewertungskriterien entwickeln
Linie:	Vermeiden, um flexibel einen Interessenausgleich erreichen zu können

Kennzeichen sachorientierten Verhandelns bzgl. Ziel, innerer Haltung, Strategie und Verhandlungslinie.

Hier wird beim Verhandeln zwischen den Sach- und Beziehungsebene unterschieden. Geht es um die Sache werden harte Argumente benutzt, gleichzeitig drücken wir jedoch in der Beziehungsebene unsere gegenseitige Wertschätzung aus. So können Lösungen gefunden werden, die von allen beteiligten Parteien vernünftig akzeptiert werden.

Soft Skills für IT-Berater

Zusammenfassung des Buches:

Titel: Soft Skills für IT-Berater

Verfasser: Uwe Vigenschow, Björn Schneider

Verlag: dpunkt.verlag

Jahr: 2012

ISBN: 978-3-89864-780-9, 978-3-86491-203-0 (eBook)

Zusammenfassung von: Tolga Aydemir, Justin Jagieniak, Niklas Harting, Malte Berg, Oliver Nagel, Jonathan Jansen, Sven Schirmer

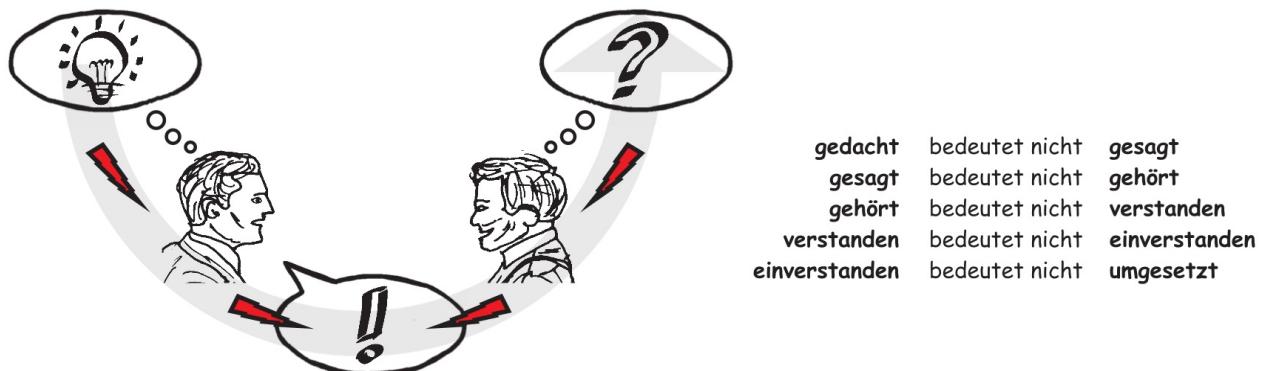
2 Kommunizieren und verstehen

Um den Prozess der Beratung effizient zu gestalten, gibt es Methoden und Hilfestellungen im Bereich der Kommunikation, die in diesem Kapitel vorgestellt werden. Dabei gibt es eine grobe Aufteilung in: Fragen stellen und miteinander reden; Auf Einwände angemessen reagieren; Inhalt, Form und Beziehung

Fragen stellen und miteinander reden

Die Beratung eines Unternehmens hat auch zur Folge, dass der Berater auf die Wünsche und Probleme der einzelnen Mitarbeiter eingehen muss. Dies gilt umso mehr in einer methodischen Beratung. Die Einarbeitung in diesen Abteilungen mit den jeweiligen Mitarbeitern läuft im Idealfall über ein gemeinsames Gespräch ab.

Dabei werden jedoch Probleme auftreten:



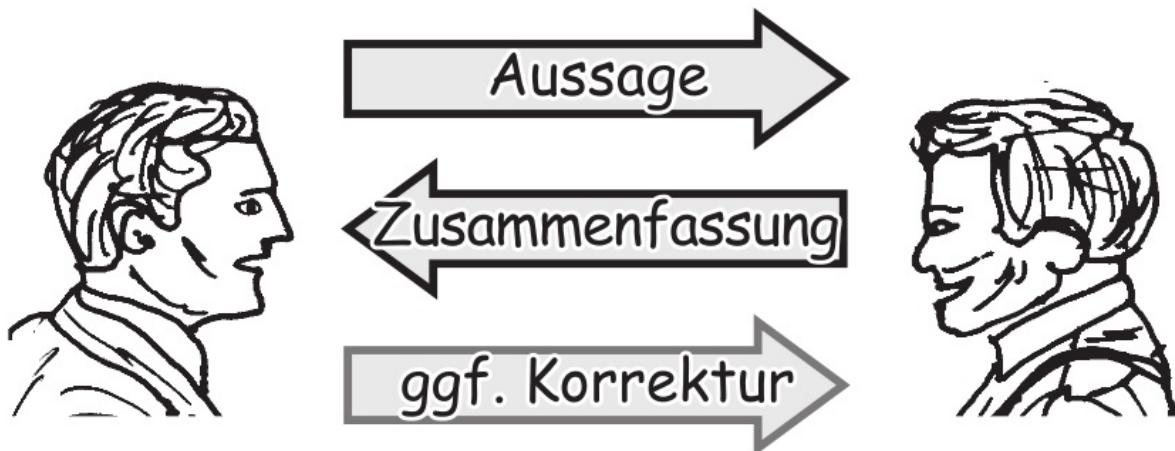
(Abbildung: Mögliche Probleme in der Kommunikation)

Wenn das Gespräch in einer Fremdsprache für einen der beteiligten Personen abläuft, dann nehmen diese Kommunikationsprobleme zu.

Der Berater muss ein gemeinsames Bild in den Gedanken der beteiligten Personen erschaffen, damit die Kommunikationsprobleme minimiert werden.

Aktives Zuhören

Ein Werkzeug, um diese Kommunikationsprobleme zu minimieren ist das aktive Zuhören. Person A erzählt etwas. Person B fasst das erzählte zusammen und gibt dadurch Person A die Möglichkeit das Bild in den Gedanken von Person B zu prüfen und ggf. zu korrigieren. Außerdem wird dadurch das Gesprächstempo verlangsamt, sodass auch Fremdsprachler die Details des Gesprächs mitnehmen können.



(Abbildung: Aktives Zuhören)

Mit Fragen ein Gespräch führen

Während der Beratungsgespräche ist die Aufgabe des Beraters das Gespräch zu führen, um so auf Problemstellungen aufmerksam zu machen. Die sog. offenen Fragen (auch W-Fragen genannt) können dabei helfen. Die geschlossenen Fragen (mit ja/nein zu beantworten) sind eher dazu geeignet, um Zustimmung oder Ablehnung innerhalb der Gesprächsgruppe abzufragen.

Wichtig werden die Fragen, wenn die Mitglieder der Gesprächsgruppe sich gegenseitig blockieren oder Probleme auftreten. Dann ist es die Aufgabe des Beraters mit den richtigen Fragen aus dieser Blockade zu manövrieren.

Sechs-Stufen-Fragetechnik

Die Sechs-Stufen-Fragetechnik kann dabei helfen Informationen zu erlangen, die noch gar nicht erwähnt wurden, jedoch wichtig für die Analyse des Problems sind. Dadurch entsteht ein Gesprächsleitfaden und unterstützt den Berater dabei auch an Informationen zu gelangen, die für den Gesprächspartner (interne Mitarbeiter) so selbstverständlich sind, dass diese gar nicht erwähnt werden.

- Prozesswörter überprüfen:
Identifizieren Sie Verben und substantivierte Verben wie z. B. melden, Auslastung, erfassen. Stellen Sie zu jedem dieser Prozesswörter die W-Fragen: Wer? Was? Wann? Wie? Wo? Warum?
- Komparative und Superlative überprüfen:
Bestimmen Sie die Bezugspunkte: Worauf bezieht sich der Vergleich oder die Steigerung? Bringen Sie die Messmethode in Erfahrung, über die die geforderten Eigenschaften nachgeprüft werden können.

- Universalquantoren überprüfen:
Identifizieren Sie die Universalquantoren wie alle, keiner, immer, nie, jeder, stets usw. Hinterfragen Sie darüber die Ausnahmen sowie die impliziten Annahmen.
- Bedingungen überprüfen:
Gibt es noch andere Varianten? Sind alle Möglichkeiten vollständig aufgezählt? Sind alle Entscheidungs- und Verzweigungsbedingungen genannt?
- Konstanten und konfigurierbare Werte überprüfen:
Identifizieren Sie feste sowie konfigurierbare Größen und Werte und geben Sie ihnen sprechende Namen wie z. B. Rechnungsschwellwert oder Volljährigkeit. Tragen Sie die Namen mit ihren aktuellen Werten in ein tabellarisches Glossar an zentraler Stelle z. B. im Intranet ein.
- Abkürzungen und Fachbegriffe im Glossar definieren:
Identifizieren Sie alle Abkürzungen, Akronyme und Fachbegriffe und definieren Sie diese in einem Glossar. Prüfen Sie dabei auf Widersprüche oder unterschiedliche Sichten in den beteiligten Fachbereichen. Der Begriff Kunde könnte z. B. im Marketing oder Produktmanagement unterschiedliche Aspekte haben.

Auf Einwände angemessen reagieren

Wenn Einwände seitens des Gesprächspartner geäußert werden, dann muss der Berater die jeweilige Person zunächst analysieren, um angemessen zu reagieren. Eine falsche Reaktion, kann zu nachhaltigen Kommunikationsproblemen führen und die Tätigkeit des Berates negativ beeinflussen. Um die Gesprächspartner analysieren zu können, gibt es eine grobe Einteilung der Persönlichkeiten in vier Quadranten und wie diese zu erkennen sind:



(Abbildung: Das Vier-Quadranten-Modell)

Zum Beispiel können Aussagen wie "Wozu das alles?" dem Warum-Quadranten zugeordnet werden. Diese Person hat eine skeptische Haltung und tendiert zu philosophischen Fragen. Das kann allerdings ein Zeichen für den Rückzug dieser Person aus dem Gespräch sein. Die Erkennungsmerkmale und typischen Aussagen für diese Quadranten sind in den folgenden Abbildungen dargestellt:

Warum	Was	Wie	Wohin noch
Skeptische Haltung	häufiges Rückfragen	kompromissbereit	visuelle, schnelle Sprache
Philosophische Fragen	prüfende Fragen	schnelle Entscheidung	in die Zukunft gerichtete Fragen
Provokation, Unterstellung	keine emotionale Übertreibung	situativ aufmerksam	unverbindlich, unabhängig von anderen
Rückzug oder Aggression	auditive Neigung	schnell, kurz	ignorant

(Abbildung: Erkennungsmerkmale der vier Grundtypen)

Warum	Was	Wie	Wohin noch
Wozu das alles?	Das ist ein Problem!	Das ist doch ganz einfach!	Dann könnte man ja auch ...
Das ergibt für mich keinen Sinn!	Das ist eine gute Frage!	Wozu lange reden?	Ich sehe da für uns noch ...
Das ist doch alter Wein in neuen Schläuchen!	Können Sie mal sagen ... Erklären Sie mir genauer ...	Ich mache das mal eben!	Das schränkt uns zu stark ein!

(Abbildung: Typische Aussagen oder Fragen der vier Grundtypen)

Nach der Einordnung eines Einwands in eines der vier Quadranten, kann nun angemessen darauf reagiert werden. Auch dazu gibt es Empfehlungen für den Berater:

- Warum
 - Einwand wertschätzen
 - Persönlichen Nutzen, Sinn bzw. Zweck herausstellen
 - Das dem Einwand zugrunde liegende Problem herausarbeiten und dessen Lösung als zentralen Erfolgsfaktor für eine angestrebte Veränderung erkennen und würdigen
- Was
 - Informationen und Details geben
 - Weiterführende Quellen und Literaturhinweise nennen
 - Konkrete Fakten und Detailtiefe bieten

- Etwas demonstrieren
- Wie
 - Konkrete Handlungsanweisungen
 - Muster und Beispiele
 - Ergebnistypen und Checklisten
 - Selbst ausprobieren lassen
- Wohin noch
 - Möglichkeiten, Optionen und Zusammenhänge herausstellen
 - Lösungen anbieten
 - Stufenpläne oder schrittweise Lösungskonzepte erarbeiten, um die Freiheitsgrade für zukünftige Entscheidungen lange offen halten zu können

Die Einteilung einer Präsentation diesen vier Quadranten entsprechend, kann den Zuhörern dabei helfen, die Präsentation besser zu verstehen. Dazu gibt es mit dem runden Pfeil in der Abbildung "Das Vier-Quadranten-Modell" eine Empfehlung für die Reihenfolge der in einer Präsentation zu beantwortenden Fragen.

10 Veränderungsmanagement im Überblick

10.1 Warum fallen uns Veränderungen so schwer?

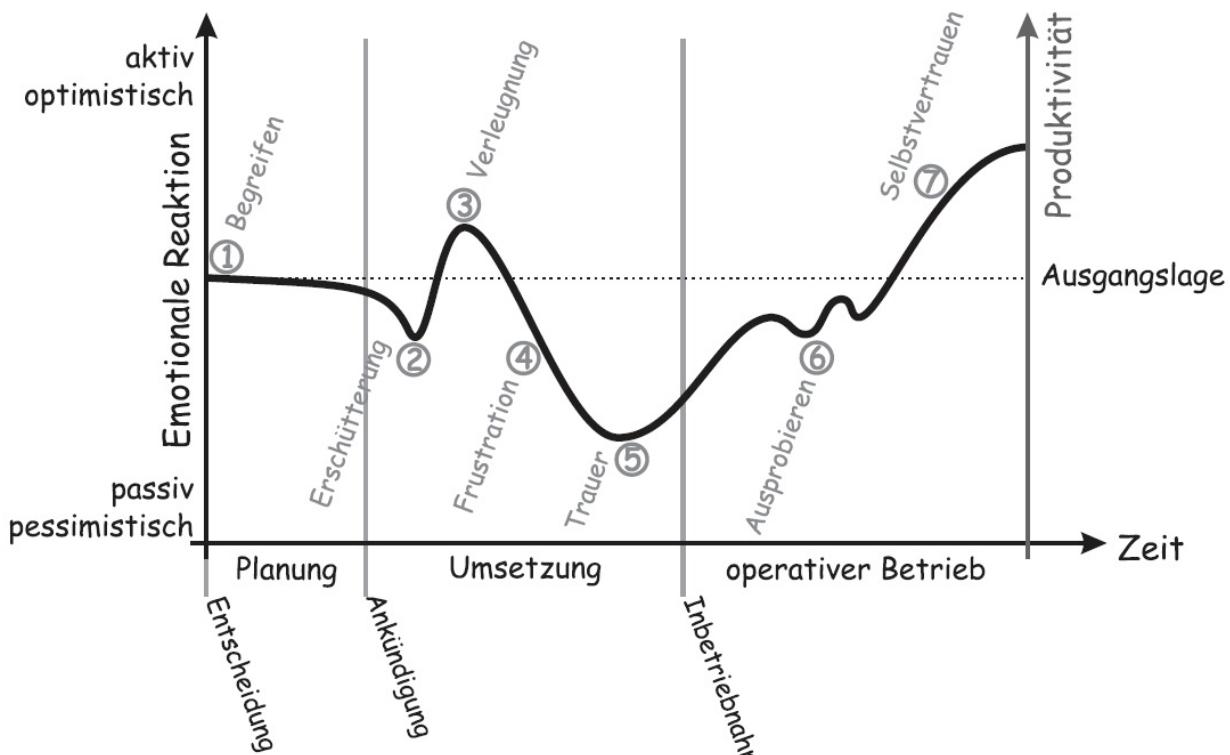
Ein Veränderungsmanagement wird definiert als die kontinuierliche Begleitung von grundsätzlichen Veränderungen. Die notwendigen Maßnahmen im Laufe des Prozesses werden als Projekte durchgeführt.

Jedes IT-Projekt benötigt ein Veränderungsmanagement, da durch neue Geschäftsprozess-Software die Arbeitsweisen der Mitarbeiter oder durch neue Produktionssteuerungen die Produktionsabläufe sich ändern. Beratungsprojekte werden daher fast auch immer von einem Veränderungsmanagement begleitet.

IT-Beratungen sind aufwendig und teuer. Sie lohnen sich aus Kundensicht nur, wenn Themen aus eigener Kraft nicht angegangen werden können oder auf Grund von Rahmenbedingungen nicht in der gewünschten Zeit umsetzbar sind.

10.1.1 Der einzelne Mitarbeiter

Obwohl Weiterentwicklungen unseren ganz normalen Alltag bestimmen, fehlt es oft an einem langfristig erfolgreichen Veränderungsmanagement. Die Folge ist, dass Mitarbeiter teilweise überfordert und frustriert sind, da ihre Erwartungen an die erhofften Verbesserungen nicht erfüllt werden. Die betroffenen Mitarbeiter erleben ein Wechselbad der Gefühle. Dieser Gefühlsverlauf von Mitarbeitern wurde in Form einer Kurve von der Boston Consulting Group veröffentlicht.



(1) Begreifen - Der Mitarbeiter sieht die eigenen Erwartungen bei einer Veränderung als nicht erfüllbar an.

(2) Erschütterung - Der Mitarbeiter ist erschüttert bzw. geschockt.

(3) Verleugnung - Der Mitarbeiter verneint die Veränderung, um sich selbst zu schützen.

(4) Frustration - (5) Trauer - Es setzt ein Verstehensprozess ein und der Mitarbeiter akzeptiert die Lage. Er erkennt, dass er alte Verhaltensmuster aufgeben muss.

(6) Ausprobieren - Der tiefen Trauer entkommt der Mitarbeiter nur durch Ausprobieren neuer Verhaltensweisen. Durch das Aufbauen von neuem Wissen über die Abläufe gewinnt der Mitarbeiter wieder an Sicherheit.

(7) Selbstvertrauen - Der Mitarbeiter gewinnt durch Erfolge wieder Selbstvertrauen.

Der Kurvenverlauf offenbart, dass die Produktivität über die Dauer der Veränderung schwankt und ein dauerhafter Mehrwert sich erst spät einstellt. Der Ablauf in der Veränderungskurve ist unumkehrbar. Die Mitarbeiter reagieren auf diese Unumkehrbarkeit sehr unterschiedlich.

10.1.2 Die Organisation und das Timing

Eine IT-Beratung wird meist auf Managementebene initiiert. Jedoch haben Veränderungsprozesse eine eigene Dynamik, da diese Prozesse Konsequenzen für Mitarbeiter, Linienführungskräfte und weitere Rollen haben. Die Veränderungskurven finden

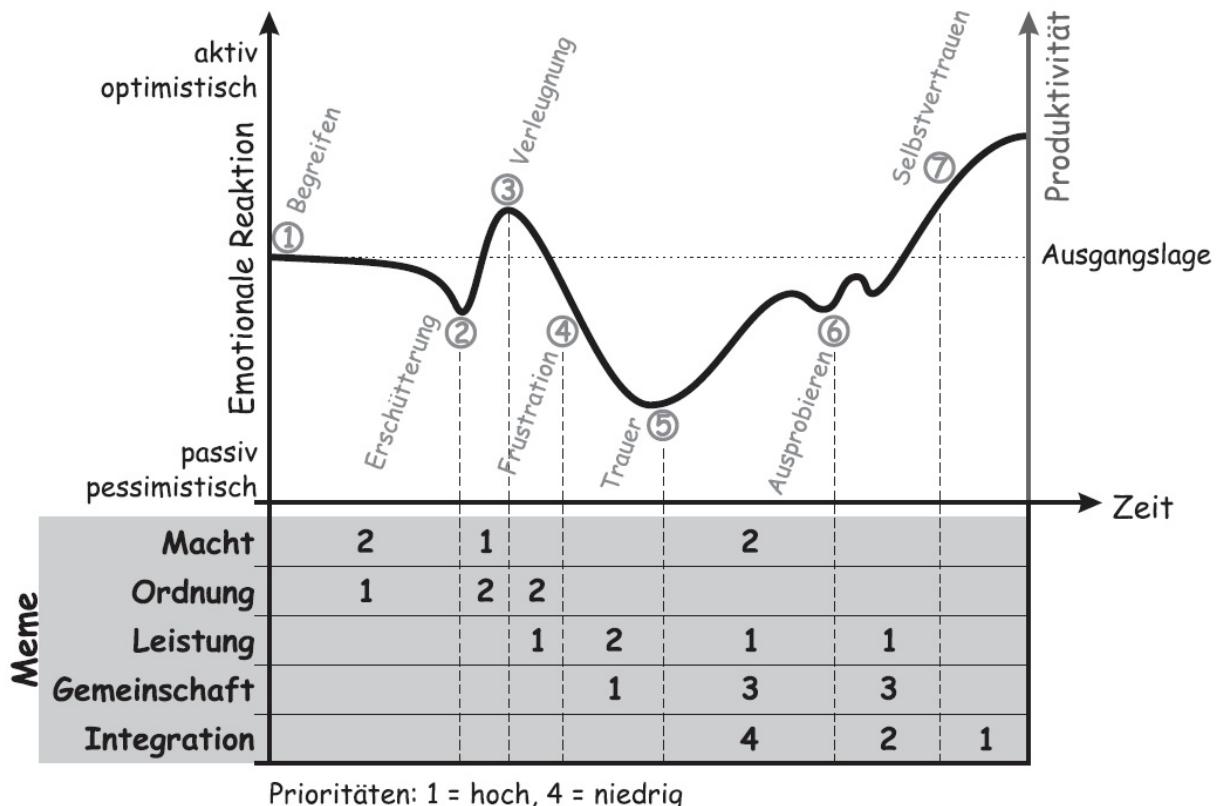
auf verschiedenen Organisationshierarchieebenen phasenversetzt statt. Ein Veränderungsprozess auf einer Ebene kann einen Veränderungsprozess zu einem späteren Zeitpunkt auf einer anderen Ebene initiiieren, um eine Anpassung des betroffenen Aspekts umzusetzen. Die Phasenverschiebung kann zu Problemen der internen Kommunikation eines Unternehmens führen, wenn z.B. das Management die Veränderung abgeschlossen hat und die Mitarbeiter sich jedoch noch in der Trauerphase befinden. Die Kommunikation ist dann oft nicht empfängerorientiert und verfehlt somit sein Ziel.

10.1.3 Was kann Veränderungsmanagement leisten?

Ein gutes Veränderungsmanagement setzt Aktionen erfolgreich um. Bei einer Veränderung durchleben dabei die Mitarbeiter zwar genau die gleichen Phasen des Gefühlverlaufs, jedoch kann die Dauer der Phasen verkürzt und die emotionalen Reaktionen verringert werden.

10.1.4 Veränderungen unterstützen

Um als Berater den Veränderungsprozess zu unterstützen, muss er die aktuelle Phase einer Veränderung bestimmen können und sein Verhalten so anpassen, dass es die Menschen bei der Veränderung bestmöglich unterstützt. Zusätzlich besitzt er die Aufgabe den Informationsfluss zwischen Führungskraft und Mitarbeitern sicherzustellen. Die nachfolgende Abbildung zeigt die Veränderungskurve und den notwendigen Memen.



Vom Begreifen zur Erschütterung - Die Ordnung muss wie vor Beginn der Veränderung beibehalten werden. Die Führungskräfte müssen ihren Mitarbeitern aktiv zuhören und die Beweggründe für die Veränderung erläutern.

Von der Erschütterung zur Verleugnung - Die Mitarbeiter lehnen auf Grund von Angst die Veränderung ab. Die Führungskräfte müssen fest an der Veränderung festhalten und keinen Zweifel auftreten lassen.

Von der Verleugnung zur Frustation - Die Mitarbeiter merken, dass sie ihr Verhalten ändern müssen. Die Führungskräfte müssen sie dabei so gut es geht unterstützen.

Von der Frustation zur Trauer - Führungskräfte können zugeben, dass sie keine detaillierte Lösung haben, und können dies nutzen, um eine engere Kommunikation mit den Mitarbeitern aufzubauen und sie zum Durchhalten zu ermutigen.

Von der Trauer zum Ausprobieren - Die Mitarbeiter verstehen, welches Verhalten ziieldienlich ist. Die Führungskräfte stehen den Mitarbeitern mehr beratend und reflektierend zur Seite. Spitzenleistungen werden honoriert und kommuniziert.

10.2 Veränderungsmanagementmodelle

Analyse des Problems

Bei anstehenden Veränderungen ist es zunächst essentiell wichtig konkrete Informationen zu sammeln und die Punkte zu klären, die zu beachten sind. Es ist immer wieder zu erkennen, dass zu vorschnell über Lösungen nachgedacht wird. Dies führt dazu das Lieblingslösungen einer Person (z.B. zu einer Organisationsstruktur oder bestimmte agile Techniken) gewählt werden, die sich am Ende als unpassend erweisen. Weiterhin folgt aus dieser Lösungsprägung, dass nur noch Informationen betrachtet werden, die zu dem Bild einzelner Personen passen und wichtige Informationen übersehen werden. Ein weiteres Problem ist, dass die Fähigkeiten der betroffenen Personen oder der Einsatz von konkreten Lösungen als viel zu optimistisch angesehen werden. Es kommt folglich zu Verzögerungen bis hin zum Scheitern der Veränderung.

Frühe Kommunikation

Anstehende Veränderungen sollten bereits früh kommuniziert werden, damit sich durch die umfangreiche Kommunikation kein unvollständiges Bild ergibt. Ein vollständiges Bild ist wichtig, damit die Konzeption keine Lücken aufweist und alle wichtigen Stakeholder abgeholt werden. Durch den intensiven Austausch mit relevanten Stakeholdern können Konflikte rechtzeitig aufgedeckt und konstruktiv gelöst werden.

Faule Kompromisse

Faule Kompromisse können zu einem Scheitern des Veränderungsprozesses führen. Häufige Symptome sind, das Bestehen lassen von alten Abläufen oder Rollen oder das Hinausschieben von Entscheidungen, die jedoch jetzt benötigt werden.

Ausprobieren

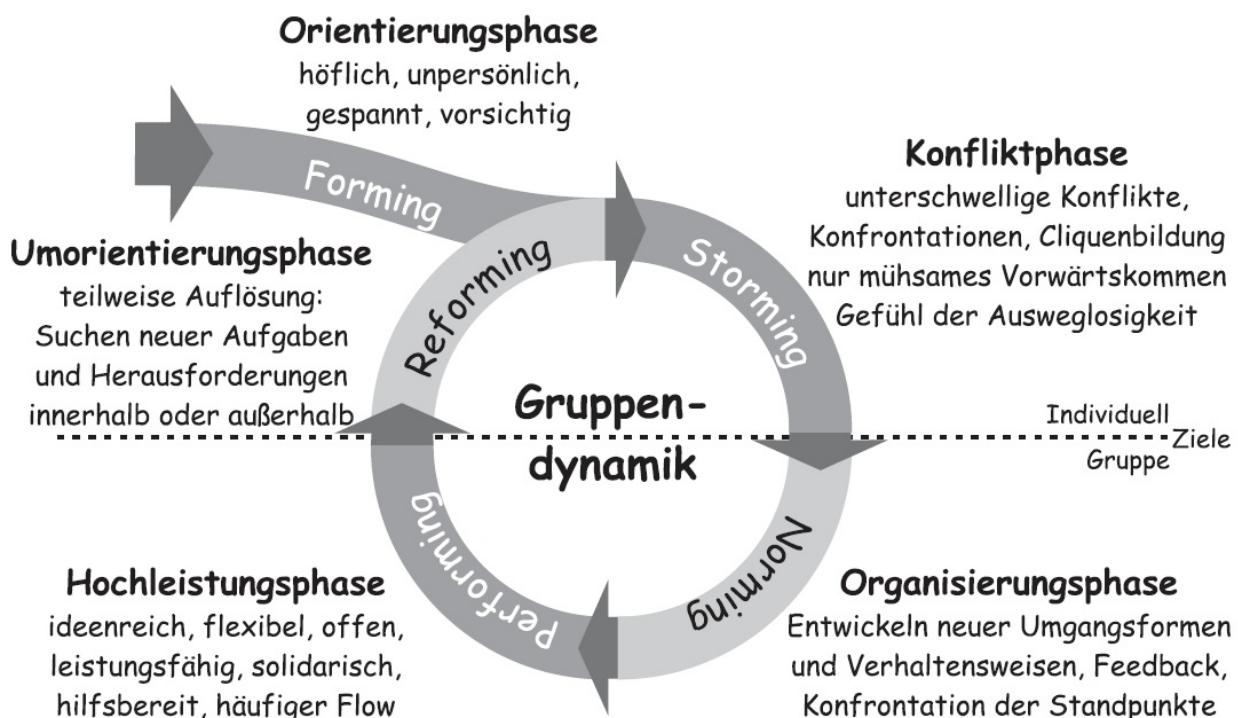
Es wird häufig vergessen, dass Veränderungen viel mit Ausprobieren zu tun haben. Die Folge ist, dass der Druck entsteht, alle Entscheidungen zu einem frühen Zeitpunkt treffen zu müssen. Da jedoch im frühen Stadium die Unsicherheit sehr hoch ist, werden Entscheidungen lieber gar nicht getroffen.

10.3 Dynamik in Gruppen

Die Teamdynamik ist bei Veränderungen ein mächtiger Aspekt. Sie kann Veränderungsprozesse unterstützen, aber auch blockieren. Ein Verständnis über diese Dynamik ist daher für IT-Berater unbedingt notwendig.

10.3.1 Die Teamuhr

Die nachfolgende Abbildung zeigt die Phasen, die jedes Team zyklisch durchläuft.



Forming: Diese Orientierungsphase ist geprägt von Unsicherheit und distanzierten Verhalten. Die Teammitglieder versuchen sich aneinander zu gewöhnen und verhalten sich daher meist vorsichtig.

Storming: In dieser Konfliktphase kommt es vermehrt zu Machtkämpfen. Es bilden sich Cliques und es entstehen Konflikte. Das Team kommt nur mühsam voran.

Norming: In dieser Organisierungsphase entwickeln sich gruppeneigene Umgangsformen und Verhaltsweisen. Die Teammitglieder intensivieren ihre Zusammenarbeit. Jedes Mitglied hat seine eigenen Aufgaben und wird innerhalb des Teams akzeptiert.

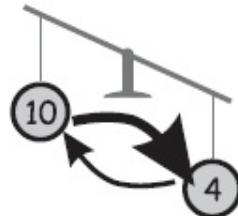
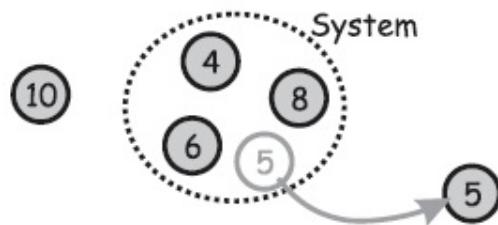
Performing: Die Gruppe ist äußerst leistungsfähig. Die Teammitglieder sind offen, ideenreich, flexibel und offen für Neues.

Reforming: Einige Teammitglieder wagen den Aufbruch zu neuen Ufern. Dadurch haben einzelne Gruppenmitglieder die Chance nachzurücken. Die Gruppenstruktur ändert sich deutlich, sodass es zu Unruhe und Unzufriedenheit kommt.

10.3.2 Systemische Betrachtungen und Grundregeln

Bei einer systemischen Organisationsberatung wird das Zusammenspiel mehrerer Elemente und im Kontext des Systems betrachtet, um eine Lösung für ein Problem zu finden. Die Lösung kommt dabei aus dem System, da die betroffenen Personen die Experten des Problems sind.

In einem System gibt es drei Ebenen, um die eigene Position innerhalb einer Gruppe zu bewerten. Diese Ebenen stehen in Wechselwirkung, da z.B. ein Problem auf einer Ebene Wirkung auf andere Ebenen haben kann.



Bindung: Auf dieser Ebene geht es um die Frage, ob ich neben der formalen Zuordnung zu einer Gruppe oder eines Projektteams, mich auch tatsächlich als Mitglied fühle und von den Kollegen im Team akzeptiert werde.

Ordnung: Die nächste Frage nach der Klärung der Zugehörigkeit ist, ob ich mich selbst im Team am richtigen Platz sehe. Fühle ich mich in meiner Rolle im Team wohl?

Ausgleich: In dieser Ebene geht es um die richtige Balance zwischen Geben und Nehmen. Bin ich der einzige der Überstunden macht und bekomme ich etwas von den Kollegen zurück?

10.3.3 Systemische Ordnung in Gruppen

Als IT-Berater für ein bestehendes Team gilt es die systemische Ordnung in der Gruppe zu verstehen. Ein erster Ausgangspunkt ist dabei die Eintrittsreihenfolge. Weitere Fragen, wie z.B. "Wer bildet den Kern?", "Wer wird akzeptiert und von anderen Gruppenmitgliedern zu Entscheidungen hinzugezogen?", geben ein detailliertes Verständnis über diese Ordnung.

11 Veränderungsmanagement konkret

Zwei Dinge sind mit Veränderungen immer einhergehend: Risiken die bei der Veränderung eingegangen werden müssen und Widerstand, der durch das Verändern hervorgerufen wird. Diese beiden Themen werden in diesem Kapitel behandelt.

Jede Veränderung ist ein Projekt

Da Veränderungen im Kontext von Organisationen durchgeführt werden, sind diese meist mit einer gewissen Komplexität behaftet. Des Weiteren sind verschiedenste Personen an Veränderungen beteiligt - es ergibt also Sinn, Veränderungen als Projekte zu betrachten.

Häufig wird der Fehler gemacht das Hauptaugenmerk auf den Inhalt der Veränderung zu legen. Also zu betrachten, welche Änderungen notwendig sind, welche Techniken hierbei evtl. eingesetzt werden müssen und wie Mitarbeiter geschult werden müssen. Ein vernachlässigter Aspekt sind hierbei die Mitarbeiter selbst. Es ist überlegenswert einen Plan aufzustellen, wann welche Mitarbeiter über die anstehenden Veränderungen informiert werden sollen und in welchem Rahmen diese Informationsverteilung stattfinden soll. Bei größeren Veränderungen hat es sich bewährt einen Veränderungsmanager einzusetzen, der für den notwendigen Informationsfluss sorgt.

Widerstand

Bei jeder Veränderung trifft man auf Widerstand. Oft wird Widerstand von Führungskräften fehlinterpretiert als ein tatsächliches "dagegenstellen" der entsprechenden Mitarbeiter. Häufiger ist jedoch anzutreffen, dass diese Mitarbeiter nur nicht wissen, wie sie sich bei den Veränderungen einbringen können. In der Akzeptanzmatrix werden vier Reaktionstypen gezeigt:

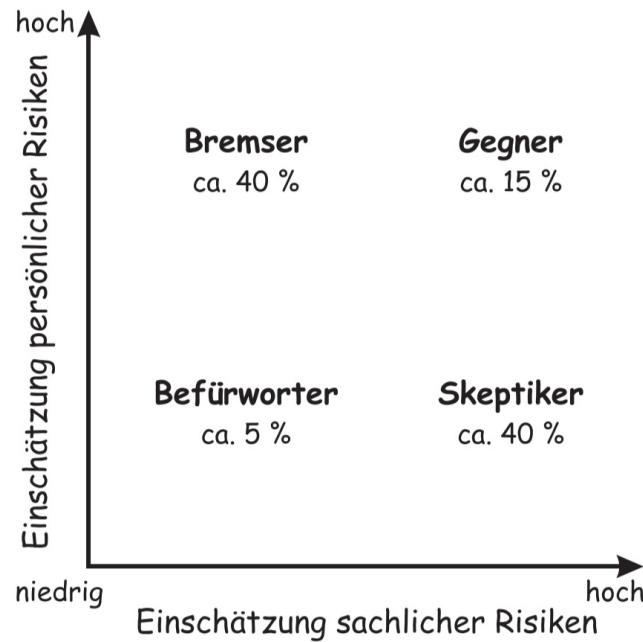


Abbildung 1: Reaktionstypen (S. 187)

Befürworter sind Personen die für jegliche Veränderungen offen sind. Diese Personengruppe hat erkannt, dass Veränderungen notwendig sind und können bei der Durchführung der Veränderungen als Multiplikatoren dienen. (Vgl. S 187)

Skeptiker sind Personen, die (persönliche) Nachteile durch die Veränderungen befürchtet. Diesen müssen fehlende Informationen gereicht werden. Sind bestimmte Informationen nicht vorhanden, können diese gemeinsam mit den Skeptikern erarbeitet werden. Hierbei wird auch eine gemeinsame Vertrauensbasis geschaffen. Einwände der Skeptiker sollten genauer betrachtet werden. In ihnen liegt ein großes Potential die Veränderungen sinnvoll zu verbessern bzw. anzupassen. Durch die Berücksichtigung der Einwände, können aus Skeptikern sogar Befürworter werden. (Vgl. S. 188)

Bremser sind Personen die nicht ausreichend Informationen über die geplanten Veränderungen haben und ebenfalls Angst vor persönlichen Risiken haben. Diese Personengruppe muss mit weiteren Informationen versorgt werden - zu klären ist hier die Frage, welche Informationen genau fehlen. Durch das Beschaffen der Informationen oder das gemeinsame Erarbeiten dieser, können weitere Verfeinerungen an den Veränderungen vorgenommen werden und evtl. sogar aus Bremsern Befürworter gemacht werden. (Vgl. S. 187)

Gegner sind Personen die ausdauernd gegen die Veränderungen sind. Auch bei weiterem Investieren von Energie zur Überzeugung ist keine Einsicht vorhanden. Dennoch sind Einwände der Gegner eine wichtige Informationsquelle. einige der Einwände können durchaus berechtigt sein und verdienen eine genauere Betrachtung. Es muss abgewogen werden, ob ein Einwand ignoriert werden kann oder weiter betrachtet werden muss. Die

Wahrscheinlichkeit, aus dieser Personengruppe Befürworter zu machen ist jedoch gering. Es sollten keine großartigen Energien in die Überzeugungsarbeit gesteckt werden. (Vgl. S. 188)

Stabilität und Veränderungen

Veränderungen sollten keineswegs so häufig wie möglich durchgeführt werden. Damit Mitarbeiter produktiv arbeiten können, muss ein stabiler Zustand erreicht werden. Eines der wichtigsten Ziele von Veränderung ist es, einen besseren stabilen Zustand so schnell wie möglich zu erreichen. Ist dieser erreicht, wird die Funktionsoptimierung der Prozesse angestrebt. Veränderungen sind wieder notwendig, wenn sich die Rahmenbedingungen (also die Umwelt) des Systems verändert haben - wenn sich also beispielsweise der Kundenkreis bzw. das Kundenalter ändert.

Ein System kann verschiedene Systemverhalten aufweisen:

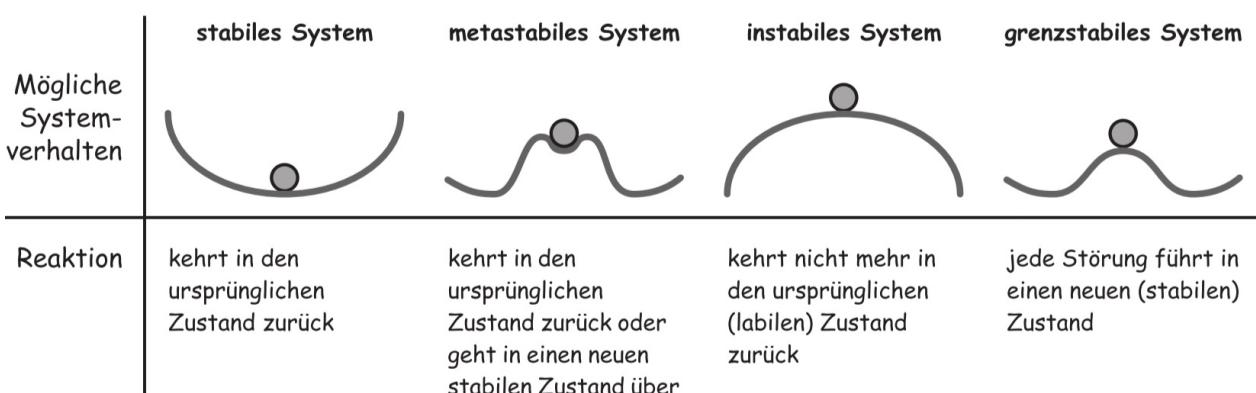


Abbildung 2: Systemverhalten (S. 190)

Stabiles Systemverhalten: jegliche Änderungen (extern, intern) führen zum gleichen stabilen Systemzustand.

Beispiel: Der Projektleiter ist im Urlaub. Während seiner Abwesenheit passt sich das System den neuen Gegebenheiten an. Nach seinem Urlaub haben alle Prozesse die gleiche Gültigkeit. (Vgl. S. 189)

Metastabiles Systemverhalten: das System bleibt bei Änderungen entweder im gleichen stabilen Zustand, oder wechselt in einen anderen, neuen stabilen Zustand.

Beispiel: Der Projektleiter wird ausgetauscht. Der neue Projektleiter hat die Möglichkeit entweder alle bestehenden Prozesse beizubehalten, oder neue einzuführen bzw. bestehende zu ändern. (Vgl. S. 190)

Instabiles Systemverhalten: bei Änderungen oder Störeinflüssen wird das gesamte System zerstört.

Beispiel: In wirtschaftlich schwierigen Zeiten sind die Mitarbeiter stark verunsichert. Ein Mitarbeiter kündigt, woraufhin weitere wichtige Mitarbeiter gehen. (Vgl. S. 190)

Grenzstabiles Systemverhalten: jede Störung führt zwingend zu einer Anpassung des Systems, welche in einem stabilen Zustand endet.

Beispiel: Anpassung von Anforderungen innerhalb eines laufenden Projektes. (Vgl. S. 190)

Störungen und Änderungen der Rahmenbedingungen können also unterschiedliche Auswirkungen auf das System haben. die Wirkung verschiedener Störungen kann jedoch nicht vorausgesagt werden.

Erfolgsschlüssel Veränderungsfähigkeit

Es gibt vier Arten von Veränderungsmanagement, aufgetrennt auf Wirkungszeitpunkt und Fokus (Vgl. S. 191):

Strategisches Veränderungsmanagement: Langfristige strukturelle Vorbereitung auf die Veränderungen

Projektveränderungsmanagement: Projektmanagement von Veränderungsprozessen

Kulturelles Veränderungsmanagement: Veränderungskompetenzen der Mitarbeiter werden erhöht. Die Mitarbeiter stellen sich auf die Veränderungen ein.

Führungsveränderungsmanagement: Mitarbeiter und Führungskräfte werden bei dem Veränderungsprozess geführt und begleitet.

Die letzten beiden Arten von Veränderungsmanagement haben die Vorbereitung der Personen eines Unternehmens im Fokus, wohingegen die ersten beiden die Prozesse genauer betrachten.

Die Ausrichtung der Personen auf Veränderungen sind ein wichtigerer und häufig weniger betrachteter Aspekt bei der Einführung von Veränderungen.

"Viele Organisationen sind aber auf ein kulturelles und Führungsveränderungsmanagement gar nicht ausgerichtet (...)." (S. 191)

Dies stellt Veränderungen vor eine Hürde, kann aber auch die Chance sein, eine neue Veränderungskultur im Unternehmen einzuführen.

Störungen und Identität von Mitarbeitern

Veränderungen können bei zu großem Einwirken in die Abläufe als Störend empfunden werden. Jede Person / Gruppe hat eine individuelle Störungsschwelle. Ein Ziel von Veränderungen ist stets unterhalb dieser Störungsschwelle zu agieren. (Vgl. S. 192,193)

Die Störungsschwelle kann schlagartig durch zu große Veränderungen überschritten werden, oder durch viele kleine Veränderungen, die in Summe über die Störungsschwelle gelangen. Letztere sind wesentlich schwerer nachzuvollziehen, da die einzelnen Veränderungen als solche nicht als Störend empfunden werden.

Als störend werden Veränderungen empfunden, die den Mitarbeiter betreffen - ihn also subjektiv nach der Veränderung schlechter dastehen lassen, bezogen auf Gehalt, Arbeitsinhalte, Arbeitszeiten etc. (Vgl. S. 194)

"Nach Frederick S. Perls (1893 – 1970) wird unsere Identität von fünf Säulen getragen (...):" (S. 195)

- Leiblichkeit (Gesundheit)
- Soziales Netz und soziale Umwelt (Freunde, Familie)
- Arbeit, Leistung, Freizeit (Beruf, Hobbys)
- Materielle Sicherheit (Besitz, Vermögen)
- Werte (Denkweise)

(Vgl. S. 195)

Die Identität wird von allen diesen Säulen gemeinsam getragen. Wird eine Säule geschwächt, so gerät das gesamte Konstrukt der Identität ins wanken. Je stärker die Auswirkungen von Veränderungen auf eine oder mehrere dieser Säulen ist, desto stärker wird auch der Widerstand der einzelnen Person gegen diese Veränderungen sein.

Software Architecture for Developers: Volume 1

Zusammenfassung des Buches:

Titel: Software Architecture for Developers: Volume 1 - Technical leadership and the balance with agility

Verfasser: Simon Brown

Verlag: Leanpub

Jahr: 2016

ISBN:

Zusammenfassung von: Gamze Soeylev Oektem, Lutz Winkelmann, Yannick Kloss

Was ist Software-Architektur?

Applikationsarchitektur

Bei der Applikationsarchitektur geht es um die niedrigeren Stufen (lower levels) des Softwareentwurfs. Meistens gibt es nur eine Technologie (Java, Microsoft .Net, etc.). Die Bauteile beinhalten Programmiersprache, Bibliotheken, Frameworks, etc. Bei der Applikationsarchitektur geht es um die Software und die Organisation des Codes.

Systemarchitektur

In einem Softwaresystem gibt es meistens mehrere Applikationen. Diese müssen zusammenarbeiten. Die Mehrheit der Softwaresysteme kommunizieren mit der Außenwelt, daher sind Interoperability und Integration mit den anderen Systemen sehr wichtig. Zusätzlich ist die Hardware auch wichtig. Die Bauteile beinhalten sowohl Software als auch Hardware (z.B. Programmiersprache, Frameworks, Servers, Infrastruktur). Systemarchitektur ist abstrakter als Applikationsarchitektur.

Softwararchitektur

Softwarearchitektur ist die Kombination der Applikation- und Systemarchitektur. Hier geht es um:

- Logging und Exception Handling
- Sicherheit
- Performance, scalability, availability
- Real-world constraint of the environment
- Interoperability/ Integrität,
- Operational, support and maintenance requirements
- ... Wir müssen also einen allgemeinen Blick auf die Software werfen.

Enterprise-Architektur - Strategie statt Code

Bei der Enterprise-Architektur geht es nicht darum, wie Technologie funktioniert, sondern wie Technologie in der Organisation besser benutzt werden kann. Enterprise-Architektur guckt wie man Menschen, Prozesse und Technologien am besten organisiert und einsetzt, damit

die Organisation besser und effizienter funktionieren kann.

Softwareprojekte der Studenten

Projektname: **VR** Projektleiter: Tim Jastrzembski Projektteam: Tim Jastrzembski Github-Repo: <https://github.com/tjastrzembski/ViReCo.git>

Projektname: **AR** Projektleiter: Andrei Güter Projektteam: Andrei Güter Github-Repo:

Projektname: **Modell getriebene Systementwicklung** Projektleiter: Projektteam: Jonathan Jansen, Oliver Nagel Github-Repo:

Projektname: **Continous Software Engineering** Projektleiter: Projektteam: Lukas Taake, Marcel Dzaak, Marvin Schirrmacher Github-Repo:

Projektname: **AI**

Projektleiter:

Projektteam: Daniel Beneker, Sven Schirmer, Yannick Kloss

Github-Repo: [Twitter Miner](#)

Projektname: **Embedded Computing** Projektleiter: Gamze Söylev Öktem Projektteam: Nils Kohlmeier, Wladimir Streck, Gamze Söylev Öktem, Jonas Wiese, Justin Jagieniak Github-Repo: <https://github.com/nkohlmeier/Spezielle-Gebiete-zum-Softwareengineering.git>

Projektname: **Fullstack Development** Projektleiter: Projektteam: Timo Rolfsmeier, Niklas Harting, Alexander Schwietert, Tolga Aydemir, Malte Berg, Lutz Winkelmann, Fabian Lorenz, Benjamin Schmidt

Pflichtenheft: [PDF](#) GitLab-Group: [ShareBase](#)

Projektname: **Cloud Security** Projektleiter: Philipp Viertel Projektteam: Philipp Viertel, Andre Kaleja Github-Repo: <https://github.com/pviertel/cloudsecurity.git>

Projektname: **Software-Architektur** Projektleiter: Christian Holzberger Projektteam: Christian Holzberger Github-Repo: <https://github.com/cHolzberger/2017-Sw-Eng-Projekt.git>

VR

Projektleiter: Tim Jastrzembski

Projektteam: Tim Jastrzembski

Github-Repo: <https://github.com/tjastrzembski/ViReCo.git>

Einführung

Das Thema VR gewinnt in der heutigen Zeit immer mehr an Beliebtheit. Aber was ist VR?

Definition



(Quelle: <http://static.giga.de/wp-content/uploads/2015/05/The-Witcher-3-einstieg.jpg>)")

Der Begriff **Virtual Reality** (kurz: **VR**) ist eine vom Computer generierte (alternative) Realität, welche fiktiv und bzw. oder unsere Wirklichkeit annähern kann. Dabei sind zahlreiche Computerspiele ein gutes Beispiel dafür: Sei es ein Rollenspiel (*Final Fantasy*, *The Witcher*, *World of Warcraft (WoW)*), ein Abenteuerspiel (*Tomb Raider*), Actionspiel (*Half-Life*), Egoshooter (*Counter Strike*) oder Simulationen für verschiedene Berufe (*Euro Truck Simulator 2*, *Microsoft Flight Simulator X*, *Landwirtschafts-Simulator 17*), jedes dieser Spiele basiert auf einer Welt, die entweder frei erfunden (z.B. die Welt von Azeroth in **WoW**) oder an unsere Wirklichkeit angelehnt ist (z.B. die genannten Simulationsspiele). In der heutigen Zeit wird ferner die sogenannte Immersion (engl.: immersive - eintauchen, eindringen) im Zusammenhang mit der **VR** gebracht. Mit Geräten wie dem **Head-Mounted Display** (Datenhelm, kurz: **HMD**), haptische Controller (z.B. Datenhandschuhe) und Sensoren zur Bewegungsverfolgung wird versucht, der **VR** ein noch realistischeres Gefühl zu geben, indem man einerseits die Wahrnehmung der Wirklichkeit reduziert (siehe Bild) und andererseits die Möglichkeiten an Interaktionen mit der virtuellen Umgebung so einfach und natürlich wie möglich gestaltet. Dadurch wird nicht nur eine Immersion, sondern auch

eine Präsenz erzielt. Denn im Gegensatz zu Bildschirm, Maus und Tastatur, die einem nur Einblick von außerhalb in die virtuelle Realität ermöglichen, geben die **HMDs** einem das Gefühl, mittendrin zu sein. [1] [2]

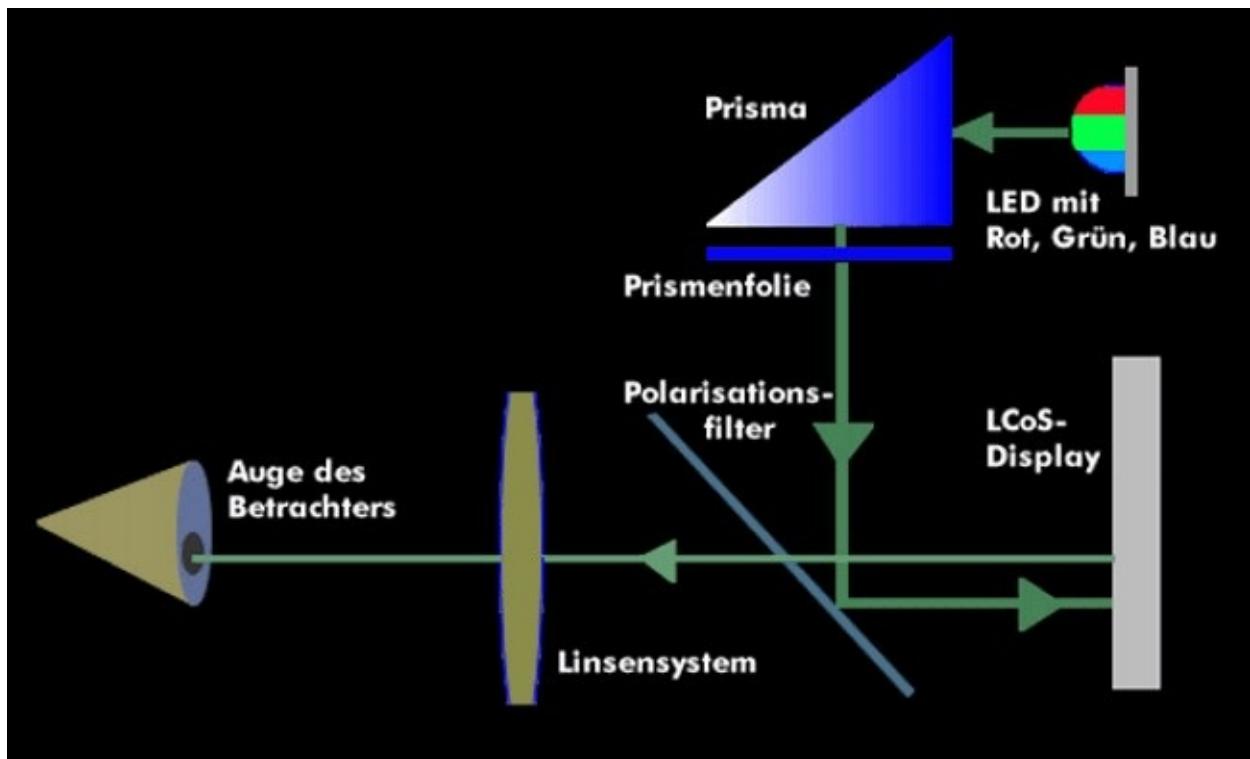
Head-Mounted Display



(Quelle: <https://www.theverge.com/2016/4/5/11358618/htc-vive-vr-review>)")

Wie der Name vermuten lässt, sind Head-Mounted Displays (**HMD**) Projektionseinheiten, welche auf dem Kopf getragen werden. Sie fallen je nach Einsatzgebiet in verschiedenen Größen aus (z.B. Brillenform oder Helmform) und haben verschiedene Darstellungsweisen. Dementsprechend können **HMDs** sowohl für Augmented Reality (siehe **AR**) als auch für **VR** genutzt werden. **HMDs** nutzen speziell für **VR** die Eigenschaften des stereoskopischen Sehens [3]: Indem sie auf zwei Microdisplays, welche sich nah an den Augen befinden, das gleiche Bild, jedoch mit geringfügig unterschiedlichen Blickwinkeln projizieren, entsteht eine Tiefenwahrnehmung der abgebildeten Szene, welche die Szene realistischer erscheinen lässt. [4]

Funktionsweise



(Quelle: <http://www.itwissen.info/lex-images/prinzip-der-hmd-projektion.png>)")

Wie die Abbildung zeigt, erfolgt bei dieser Projektionsweise in 4 Schritten [4]:

1. Drei Leuchtdioden für das RGB-Spektrum erzeugen sequentiell Licht.
2. Die Farben werden separat nacheinander über Polarisationsfilter auf das LCoS-Display geworfen.
3. Das Display reflektiert das eintreffende Licht auf die Linse.
4. Das Licht wird durch die Linse je nach Einfallswinkel gebrochen und auf das Auge geworfen.

Quellen:

[1] <http://www.giga.de/konsolen/oculus-rift/specials/virtual-reality-was-ist-das-definition-brillen-games-technologie/>

[2] http://www.ard.de/home/ard/Was_ist_Virtual_Reality/3364362/index.html

[3] https://de.wikipedia.org/wiki/Stereoskopisches_Sehen

[4] <http://www.itwissen.info/Datenhelm-head-mounted-display-HMD.html>

Einsatzgebiete



(Quelle: <https://cdnwebqastorage.azureedge.net/dimensions/gb-reservations.jpg>)")

Kommerziell gesehen tritt **VR** meist in Form von Spielen auf. Im privaten Haushalt können diese durch Einsatz von **HMDs** immersiv verstärkt werden. Alternativ kann man in bestimmten Regionen **VR**-Spielhallen besuchen.[15]

In der Medizin kann sie eine sichere Alternative zur Behandlung von Phobien dienen: Die Konfrontationstherapie würde rein virtuell verlaufen, wobei der Patient sich selbst in einer sicheren Umgebung befindet. [17]

Im Militär findet sie unter anderem in einer Fallschirmspring-Simulation (*DoDAAM*) bereits Verwendung: Der Anwender kann den Sprung ohne gefährliche Konsequenzen lernen. [17]

Aber auch im Berufswesen findet sie zahlreiche Verwendung: *Arch Virtual* soll ein unterstützendes Tool zur 3D-Planung für Architekten sein. [17]

Bildungstechnisch kann jedes Simulationsspiel gezählt werden. Dabei können solche als *Serious Game*[16] aufgefasst werden. Beispielsweise kann *Microsoft Flight Simulator X* in Verbindung mit *VATSIM Germany*[19] eine gute Grundlage zum Pilotentraining sein. Als weiteres *Serious Game* kann man *ChernobylVR*[18] nennen. Hier kann man die Gegend um Chernobyl erkunden, ohne sich selbst den gefährlichen Strahlungen auszusetzen.[17]

Quellen:

[16] http://www.seriousgames.de/?page_id=160

[17] <https://www.androidpit.de/die-verschiedenen-einsatzgebiete-fuer-virtual-reality?nocol=1>

[18] <https://survivethis.news/chernobyl-vr-serious-gaming-fuer-guten-zweck-spenden/>

[19] http://www.vacc-sag.org/?PAGE=first_steps_pilot

Augmented Reality

1 Einführung

2 Grundlegende Technologien

3 Konzept

4 Zusammenfassung

Projektleiter: Andrei Günter

Projektteam: Andrei Günter

Github-Repo: <https://github.com/aguenter/artranslator>

1 Augmented Reality

Dieses Dokument zeigt die Vorteile von Augmented Reality (AR) auf und beschreibt die bisherigen Fortschritte auf diesem Gebiet in Wissenschaft und Industrie. Weiterhin werden die fundamentalen Elemente eines AR-Systems dargestellt und zu einem Konzept zusammengefasst. Das Konzept spiegelt die wesentlichen Herausforderungen zur Realisierung eines AR-Systems wider und kann somit als generelles Framework für die Architektur eines AR-Systems betrachtet werden.

2 Einführung

Dieser Abschnitt leitet die Thematik AR ein, indem ein Beispiel einer AR-Anwendung einige Vorteile dieser Technologie aufzeigt. Im Anschluss werden bestehende AR-Systeme durch eine Unterteilung in verschiedene Aspekte in einen Überblick gebracht.

2.1 Motivation

Seit dem letzten Jahrzehnt erweckt AR großes Interesse in Wissenschaft und Industrie. AR stattet eine reale Umgebung mit virtuellen digitalen Informationen aus. Somit können Benutzer zusätzliche Informationen zu einem Gegenstand oder zu einer Umgebung abrufen.

Beispielsweise könnte eine AR-Anwendung bisherige Anleitungen zur Montage von Maschinen ersetzen und bereichern. Ein Problem bisheriger Anleitungen ist, dass eine Projektion der Inhalte auf die Realität nicht möglich ist. Somit sind Leser der Anleitung dazu gezwungen die Anweisungen über ihre Vorstellungskraft auf die reale Umgebung zu übertragen. Weiterhin können genannte Komponenten der Anleitung nicht direkt in der Realität auffindbar sein oder es fehlt eine geeignete Visualisierung, um diese Komponenten exakt und schnell zu lokalisieren.

Eine AR-Anwendung könnte Arbeiter in der Montage von Maschinen unterstützen, indem ein Monteur Projektionen von virtuellen Komponenten auf einer realen Maschine angezeigt bekommt. Darüber hinaus könnten diese virtuellen Komponenten so animiert werden, dass sie die Art und Weise der Anbringung repräsentieren. So lassen sich die Inhalte der Anleitung für die Montage direkt in der realen Umgebung visualisieren. Dem Monteur wird das Arbeiten durch entsprechende Projektionen erleichtert und der Prozess der Montage wird durch ein besseres Verständnis beschleunigt.

Weiterhin können neue Mitarbeiter durch intuitive AR-Anwendungen besser und schneller geschult werden. Neben den oben genannten animierten Visualisierungen können AR-Anwendungen jegliche Form von virtuellen Informationen annehmen, zum Beispiel auch reine textuelle Informationen. Dadurch lassen sich Begrifflichkeiten leicht mit den entsprechend visualisierten Prozessen verknüpfen. Folglich ist das Erlernen der Begrifflichkeiten für neue Mitarbeiter intuitiver und direkt mit der praktischen Erfahrung verbunden.

Das obige Beispiel durchleuchtet nur grob einige Vorteile von AR-Systemen und dient zur Veranschaulichung. Im Folgenden werden verschiedene Aspekte von bestehenden AR-Systemen beleuchtet.

2.2 Stand der Technik

AR wird besonders in den Bereichen Gaming, Navigation, Guiding und in interaktiven Medien verwendet [1, 2]. Bestehende AR-Systeme unterscheiden sich grundsätzlich in der Verwendung von Geräten zur Anzeige von virtuellen Inhalten sowie in der Darstellung der virtuellen Inhalte. Wie in Abbildung 1 dargestellt, unterscheiden sich die Geräte wie folgt [2]:

- *Hand-Held*: Die virtuellen Inhalte werden über ein Gerät dargestellt, welches der Benutzer in der Hand hält (z.B. Smartphone oder Tablet).
- *Head-worn*: Die virtuellen Inhalte werden über ein Gerät dargestellt, welches der Benutzer auf dem Kopf trägt (z.B. Smart-Glass oder Head-Mounted-Display).
- *Spatial*: Die virtuellen Inhalte werden direkt in die reale Welt projiziert (z.B. Beamer), hierbei ist der Benutzer nicht auf das Tragen eines Gerätes angewiesen.

Weiterhin werden in Abbildung 1 verschiedene Formen der Darstellung von virtuellen Inhalten angeführt [2]:

- *Video*: Die realen Objekte und die virtuellen Inhalte sind miteinander verschmolzen und die Sicht des Benutzers findet komplett auf einer digitalen Ebene statt.
- *Optical*: Über eine Perspektive auf die reale Welt werden virtuelle Inhalte auf reale Objekte überlagert.
- *Retinal*: Virtuelle Inhalte werden über einen schwachen Laser direkt auf die Netzhaut des Benutzers projiziert.
- *Projector*: Virtuelle Inhalte werden direkt auf reale Objekte projiziert.
- *Hologram*: Virtuelle Inhalte werden als Hologramm dargestellt.

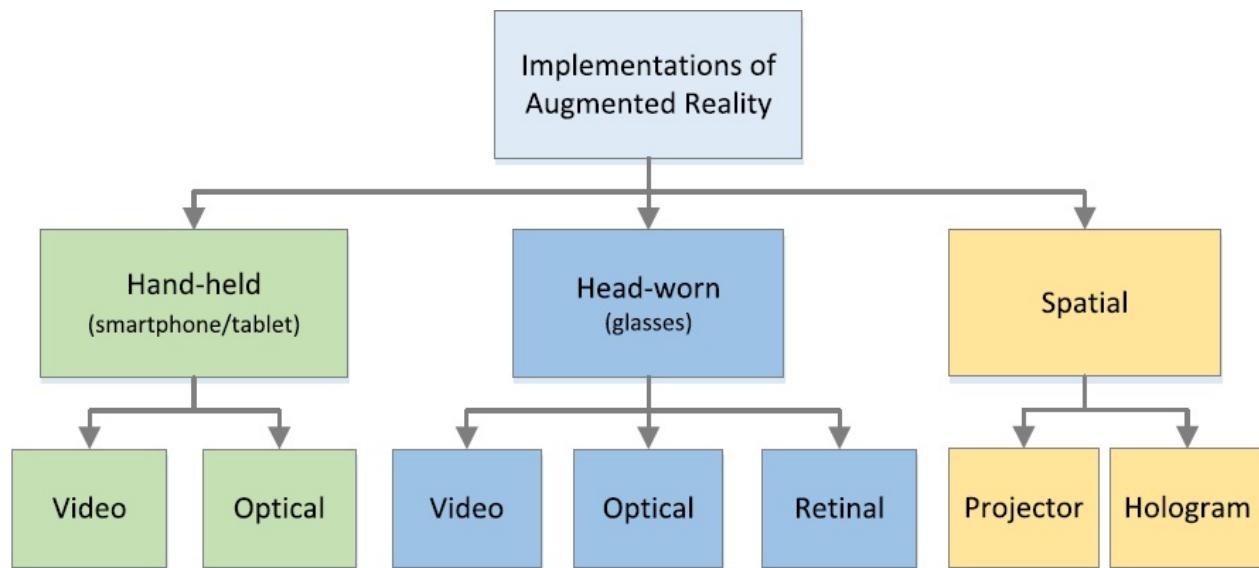


Abbildung 1: Geräte zur Anzeige und zur Darstellung von virtuellen Inhalten in AR-Systemen [2]

3 Grundlegende Technologien

Dieser Abschnitt erläutert die für AR notwendigen Technologien im Detail. Dazu wird zu Beginn eine Definition für AR erarbeitet. Anschließend folgt eine Beschreibung drei fundamentaler Bausteine von AR-Systemen: Tracking, Darstellung und Interaktion [3].

3.1 Augmented Reality

Es ist allgemein anerkannt, dass T. P. Caudell und D. W. Mizell den Begriff Augmented Reality mit ihrer Arbeit für ein Head-Mounted-Display begründeten [4]. Das Head-Mounted-Display wurde als Prototyp eines Assistenzsystems für Monteure im Flugzeugbau konzipiert.

Während der letzten Jahrzehnte haben sich weitere Ansätze etabliert und damit hat sich auch die Definition von AR stetig verändert. Daher wird für dieses Dokument eine Abgrenzung von herkömmlichen AR-Systemen zu pervasiven AR-Systemen vorgenommen. Herkömmliche AR-Systeme sind Prototypen für speziell zugeschnittene Szenarien und auch nur in diesem Kontext verwendbar, während pervasive AR-Systeme ein breites Einsatzgebiet abdecken und je nach Kontext die Sichtweise und den virtuellen Inhalt für den Benutzer anpassen [1]. Für diese Abgrenzung wird folgende Definition herangezogen:

- **Pervasive AR:** Pervasive Augmented Reality ist eine kontinuierliche und allgegenwärtige Benutzerschnittstelle, welche eine dreidimensionale physikalische Welt mit digitalen Informationen erweitert, während der Kontext des Benutzers beobachtet wird und der virtuelle Inhalt vom Kontext abhängig angepasst wird [1].

3.2 Tracking

Um virtuelle Inhalte in einer dreidimensionalen realen Umgebung an den passenden Positionen darstellen zu können, nutzen AR-Systeme unterschiedliche Tracking-Methoden. Die einfachste Form des Trackings nutzt auf realen Objekten angebrachte Marker, z.B. in Form von QR-Codes. Ein großer Nachteil ist jedoch, dass der virtuelle Inhalt vordefiniert auf die Marker angepasst werden muss. Darüber hinaus ist die Anbringung von Markern auf realen Objekten nicht immer möglich.

Bei der Verwendung eines mobilen Gerätes (bspw. hand-held oder head-worn) schafft hier das 3D Tracking Abhilfe. Hierbei ist das Tracking für die Verfolgung der Trajektorie des Gerätes und zur Bestimmung der Pose des Gerätes zuständig. Wenn gleich aussehende

Objekte unterschieden werden müssen, ist die Trajektorie und die Pose des Gerätes von großer Bedeutung. Am Beispiel einer Maschine lässt sich dies mit mehrfach vorkommenden Komponenten verdeutlichen. Wenn eine AR-Anwendung einen Mitarbeiter bei der Wartung dieser Komponenten unterstützt, so ist es von immenser Bedeutung, dass keine Komponenten verwechselt werden.

Insbesondere bei der Verwendung von mobilen Geräten, haben sich das Tracking mit GPS [5] und das kamerabasierte Tracking etabliert [6, 7].

3.3 Darstellung

Die Darstellung ist ein weiteres Kernelement in AR-Systemen. Die richtige Kalibrierung der Position und die richtige Ausrichtung von virtuellen Inhalten bestimmt die Zufriedenheit des Endanwenders.

Virtuelle Inhalte können wie im Abschnitt 2.2 beschrieben dargestellt werden. Im letzten Jahrzehnt wurden überwiegend Head-mounted-Displays verwendet [3]. Jedoch sind hand-held- und head-worn-basierte Ansätze oftmals unkomfortabel. Ein weiterer Nachteil bei head-worn-basierten Ansätzen liegt in den geringen Akkulaufzeiten der aktuell verfügbaren Geräte.

Ansätze über Projektionen oder Hologramme sind sehr benutzerfreundlich, jedoch nur schwer in dynamischen Umgebungen umzusetzen und mit hohen Kosten verbunden.

3.4 Interaktion

Der virtuelle Inhalt passt sich dem Kontext des Benutzers an, es findet also eine Interaktion mit dem System statt. Ob das System diese Anpassung automatisch vornimmt oder der Benutzer eine Eingabe tätigen muss ist hier nicht relevant. Bereits eine Bewegung eines Benutzers oder das Verändern der Umgebung kann zu einem neuen Kontext führen, diesen Kontext muss das AR-System beobachten und entsprechend reagieren. Ein Beispiel einer Interaktion kann ein von einem Projektor angezeigter virtueller Schalter sein, der sich betätigen lässt wenn der Benutzer seine Hand über den Schalter hält.

Das kontinuierliche Beobachten und das Reagieren auf den Kontext des Benutzers wird als Ubiquitous Computing bezeichnet, während die Interaktion aus obigen Beispiel die Bezeichnung tangible bits trägt [3].

Ein weiterer Schlüssel für eine benutzerfreundliche Interaktion und somit auch für die Akzeptanz von AR-Systemen, stellt das Verständnis für soziale, psychologische, kulturelle und organisatorische zwischenmenschliche Interaktionen dar [3]. So könnte bspw. eine AR-Anwendung entwickelt werden, mit der es möglich ist aus sprachlichen Informationen emotionale Informationen zu gewinnen und diese in einer für den Menschen verständlichen Form darzustellen [3].

Auch in der Interaktion führen hand-held-basierte Ansätze zu Herausforderungen in Bezug auf die Benutzerfreundlichkeit. Die Darstellung von virtuellen dreidimensionalen Objekten auf einer zweidimensionalen Schnittstelle (bspw. Tablet) kann in einigen Fällen die Benutzer verwirren.

4 Konzept

Dieser Abschnitt bringt die zuvor erläuterten Technologien in eine konzeptionelle Übersicht. Das hier vorgestellte Konzept kann als Framework zur Entwicklung von Architekturen für AR-Systeme dienen und wird als *AR-Basis-Zyklus* bezeichnet. Die Abbildung 2 veranschaulicht die notwendigen Herausforderungen, um ein AR-System zu realisieren. Hierbei greifen die drei dargestellten Basis-Elemente in einem sich wiederholenden Zyklus ineinander. Das Tracking bestimmt die für die Darstellung notwendigen Positionen von virtuellen Inhalten. Während diese Prozesse erfolgen, kann sich der Kontext für die virtuellen Inhalte durch eine Interaktion verändern. Diese Interaktion hat unmittelbaren Einfluss auf die Umgebung des AR-Systems und erfordert ein kontinuierliches Tracking.

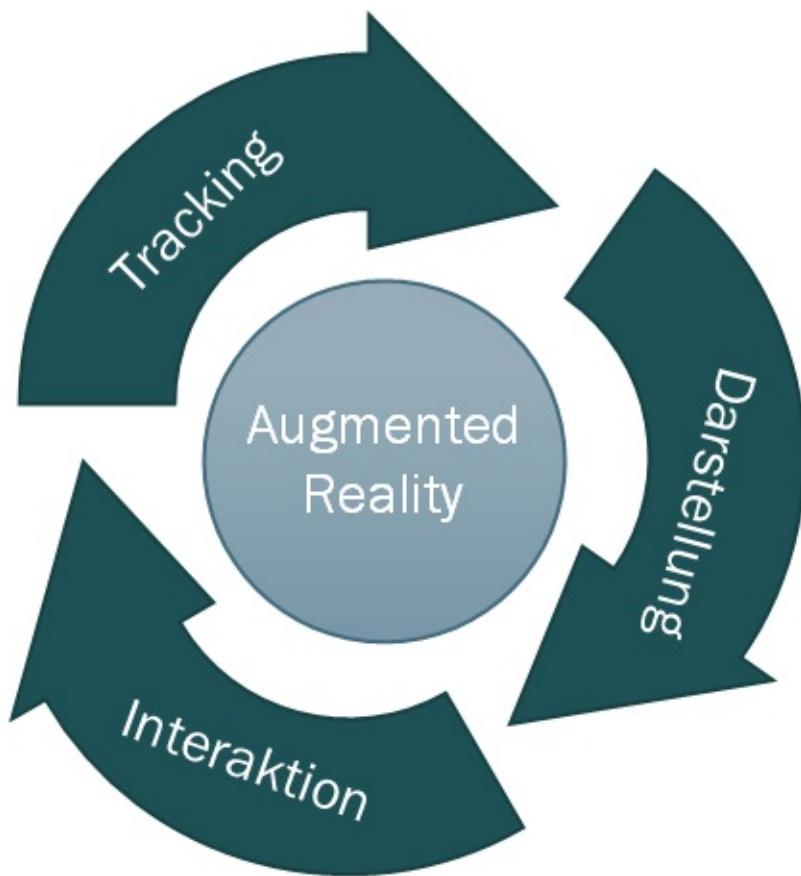


Abbildung 2: AR-Basis-Zyklus

Zur Umsetzung dieses Konzepts stehen frei verfügbare sowie kommerzielle Werkzeuge zur Verfügung. Die folgende Übersicht stellt einige bekannte AR Software Development Kits (SDK) in Relation zu folgenden Aspekten:

- Tracking: siehe Abschnitt 3.2
- Darstellung: siehe Abschnitt 3.3

- Interaktion: siehe Abschnitt 3.4
- Lizenz: stellt dar, ob eine freie oder kommerzielle Lizenz zur Verfügung steht
- Dokumentation: Bewertung der Ausführlichkeit der Dokumentation

Für jeden Aspekt wird ein Symbol vergeben, welches darstellt wie gut das SDK den Entwickler im entsprechenden Aspekt unterstützt:

- „+“: vorhanden
- „-“: eingeschränkt vorhanden
- „x“: nicht vorhanden
- „?“: keine Informationen vorhanden

SDK Name	Tracking			Darstellung	Interaktion	Lizenz		Dokumentation
	Marker	3D Tracking	GPS			frei	kommerziell	
Metaio SDK	+	+	+	+	+	x	x	+
ARToolkit	+	x	x	-	-	+	x	-
Vuforia	+	-	+	+	+	-	+	+
Wikitude	+	+	+	+	+	-	+	+
DroidAR	+	+	+	?	+	x	+	?
D'Fusion	+	x	+	+	+	x	+	+

Tabelle 1: Übersicht AR-SDK's

Modell getriebene Systementwicklung

Projektleiter:

Projektteam: Jonathan Jansen, Oliver Nagel

Github-Repo:

Einführung

Die aktuelle Marktentwicklung stellt Unternehmen vor immer größere Herausforderungen, da zunehmend qualitativ hochwertigere und komplexere Systeme zu geringeren Kosten und kürzeren time-to-market Zeiten gefordert werden. Zusätzlich steigt die Erwartung des Kunden, dass Systeme kundenspezifisch angepasst werden können (mass customization) und über den Produktlebenszyklus hinweg mit kontinuierlichen Systemverbesserungen versorgt werden.

Diese Anforderungen sorgen besonders in einer dokumentenbasierten Entwicklung für große Probleme. Die Pflege von Dokumenten eines komplexen Systems ist sehr zeitintensiv und fehleranfällig. Eine Änderung am System wird oft nicht konsistent in allen relevanten Dokumenten übernommen, da oft unklar ist, welche Dokumente oder Stellen im Dokument geändert werden müssen. Infolgedessen werden die Dokumente nicht mehr gepflegt und das Wissen bleibt im Kopf der einzelnen Entwickler. Weitere Änderungen sind dann oft risikoreich und zeitaufwendig, da ein ganzheitliches Verständnis des Systems nicht mehr existiert und die Informationen mühsam von verschiedenen Personen zusammengetragen werden müssen. Ein weiteres Problem ist, dass diese Dokumente nicht zentral gepflegt werden, sondern in der Realität oft verteilt auf verschiedenen Netzlaufwerken oder Computern der Entwickler liegen. Es fehlt der Überblick über die relevanten Dokumenten.

Eine weitere Herausforderung stellt das Variantenmanagement für Unternehmen dar. Es fehlt den Unternehmen oft an einer effizienten Methodik, die Komplexität der Varianten zu beherrschen und Varianten innerhalb einer Produktlinie effizient zu entwickeln.

Um den Anforderungen gerecht zu werden, versuchen Unternehmen zunehmend ihre Entwicklung von einer dokumentbasierten auf eine modellbasierte Systementwicklung umzustellen. Die modellbasierte Systementwicklung wird auch als Model-Based-Systems-Engineering (MBSE) bezeichnet. MBSE beschreibt einen interdisziplinären Ansatz des Systems Engineering, welche bewährte Vorgehensweisen des traditionellen Systems Engineering mit Modellierungstechniken verbindet. Der Entwurf, die Spezifikation, sowie die Verifikation und Validierung eines komplexen Systems erfolgt in einem Systemmodell. Dieses Systemmodell begleitet den kompletten Entwicklungsprozess und stellt die Quelle der wesentlichen Entwicklungsartefakte dar.

Der Übergang zu MBSE stellt für viele Unternehmen jedoch eine Herausforderung dar, da die modellbasierte Entwicklung eine bedeutsame Veränderung für die Belegschaft in der Entwicklungsorganisation ist. Die nachfolgenden Kapitel sollen einen Einstiegspunkt in die Thematik liefern und einen Überblick über relevante Themen von MBSE und Variantenmanagement geben. Die ersten beiden Kapitel geben eine Einführung in das

Systems Engineering und MBSE. Im dritten und vierten Kapitel wird die SysML vorgestellt, welche eine standardisierte Notation für die Modellierung von Systemarchitekturen liefert und domainspezifisch angepasst werden kann. Im fünften Kapitel wird eine Methodik zur effizienten Entwicklung von Produktlinien beschrieben. Das letzte Kapitel stellt die Modelltransformation vor. Diese wird benötigt um Modelle einer Disziplin (z.B. Systemarchitektur) konsistent in Modelle einer anderen Disziplin (z.B. Software, Hardware) zu überführen.

Systems Engineering

Das Systems Engineering (SE) ist ein interdisziplinärer Ansatz mit dem Ziel Systeme erfolgreich umzusetzen. Dabei werden alle spezialisierten Teams in ein gemeinsames Team integriert und ein strukturierter Entwicklungsprozess geformt, der die Systemkonzeptionierung bis hin zur Inbetriebnahme umfasst. SE weist der Definition der Kundenbedürfnisse und der geforderten Funktionalität von Beginn an einen hohen Stellenwert zu, um qualitativ hochwertige Produkte zu entwickeln, die im Einklang mit diesen Bedürfnissen stehen. [INC17a](#)

Systems Engineering basiert auf System Thinking. Das System Thinking ist eine spezielle Sichtweise auf die Realität. Sie schärfst die Sicht auf das Ganze und wie die Teile innerhalb des Ganzes zusammenwirken. Im Systems Engineering wird System als das Ganze und die einzelnen Systemelemente als Teile betrachtet. [INC17b](#)

Ein System kann als eine Kombination aus Systemelementen betrachtet werden, die zusammen Ergebnisse liefern, die sie alleine nicht erzielen könnten. Unter Ergebnissen eines Systems werden Eigenschaften, Funktionen, Charakteristiken und Verhalten verstanden, die das System aufweist. Die Elemente können dabei durch Menschen, Hardware, Software, Hilfsmittel, oder ähnliches repräsentiert werden. Alle Dinge, die benötigt werden, damit ein System Ergebnisse liefern kann. [INC17a](#)

Beispielsweise kann ein Systemelement durch das Wartungspersonal repräsentiert werden. Das Wartungspersonal ist Teil eines größeren Systems und wird benötigt, damit ein Systemelement (z.B. eine Anlage) über den Lebenszyklus hinweg eine bestimmte Funktion erfüllen kann.

Der Ansatz des System Thinking hilft ein besseres Verständnis von einem System zu erlangen. Ein System Thinker weiß,

- wie das System sich in einen größeren Kontext einzuordnen ist,
- wie es sich verhält und
- wie mit dem System umzugehen ist [INC17b](#).

Eine wichtige Norm im Bereich des Systems Engineering ist die ISO/IEC 15288 – „System- und Software-Engineering - System-Lebenszyklus-Prozesse“ und bildet den Einstiegspunkt für ein ernsthaftes Systems Engineering.

Die ISO/IEC 15288 ist internationale Norm, welche die Lebenszyklusprozesse eines Systems beschreibt. Sie hat das Ziel diese Prozesse bewertbar zu machen und zu verbessern. Die Lebenszyklusprozesse lassen sich in folgende übergeordnete Kategorien

unterteilen:

Agreement Prozesse

Die *Agreement Prozesse* definieren die notwendigen Aktivitäten für das Erstellen von Verträgen oder Vereinbarungen zwischen zwei Organisationen.

Technical Management Prozesse

Die *Technical Management Prozesse* umfassen die Erstellung, Entwicklung und Ausführung von Plänen, der Abgleich des aktuellen Fortschritts und der Zielerreichung mit diesen Plänen, sowie die Kontrolle der Ausführung, um das Ziel zu erreichen.

Technical Prozesse

Die *Technical Prozesse* umfassen die Definition der Anforderungen, die Umwandlung der Anforderungen in ein konkretes Produkt, die Sicherstellung der Reproduktion eines Produktes, die Benutzung des Produktes, sowie die zuverlässige Gewährleistung der Betriebsbereitschaft und die Entsorgung des Produktes.

Organizational Project-Enabling Prozesse

Die *Organizational Project-Enabling Prozesse* stellen sicher, dass das Unternehmen Produkte liefern kann. Sie stellen die Ressourcen und Infrastruktur bereit, damit Projekte gestartet, unterstützt und kontrolliert werden können.

Die in der Norm beschriebenen Prozesse und Lebenszyklusmodelle müssen an das Unternehmen und den Projekten spezifisch angepasst werden. Diesen Schritt der Anpassung wird auch als *tailoring* bezeichnet. [INC15]

Das Lesen der Norm kann ein zäher Akt sein. Die Incose hat aus diesem Grund das SE-Handbook veröffentlicht, welches eine hohe Konformität zu der Norm aufweist. Das SE-Handbook kann zusätzlich als Schulungsmaterial für die Zertifizierung des „Certified Systems Engineer“ genutzt werden.

Model Based Systems Engineering

Dieses Kapitel beschäftigt sich mit dem Model Based Systems Engineering (MBSE). Anhand von Definitionen und Erläuterungen wichtiger Fachbegriffen, die in Verbindung mit MBSE stehen, soll der Leser ein besseres Verständnis erlangen.

Das Model Based Systems Engineering setzt sich aus den Begriffen „Systems Engineering“ und „Model“ zusammen. Das Modell wird im Kontext von MBSE auch als Systemmodell bezeichnet. Dieses Systemmodell ist eine Abstraktion eines komplexen Systems. Die Abstraktion hilft die Komplexität eines Systems zu reduzieren, sodass Vorgänge und Funktionen einfacher dargestellt, verstanden und gestaltet werden können. [THM17](#)

Das Systemmodell enthält alle relevanten Informationen eines Systems für die verschiedenen Stakeholder (z.B. Projektleiter, Entwicklungsleiter, Testingenieur) und ist die primäre Informationsquelle während der kompletten Entwicklung. Die unterschiedliche Darstellung dieser Informationen wird als Sicht (engl. View) bezeichnet und richtet sich nach den Interessen der Stakeholder. Weiterhin kann das Systemmodell wichtige Informationen für Entscheidungen liefern oder für frühe Analysen (z.B. FMEA, Risikoanalyse) genutzt werden. Ein entscheidender Vorteil, der sich durch die Verwendung eines Systemmodells ergibt, ist jedoch die Wiederverwendung von Elementen (z.B. Komponenten). Diese sind im Modell einmal hinterlegt und können in unterschiedlichen Systemarchitekturen visualisiert werden. Eine Änderung an diesem Element sorgt für eine Aktualisierung in allen Systemarchitekturen, die dieses Element verwenden.

Ein Systemmodell zeichnet sich durch folgende Eigenschaften aus:

- Das Modell darf sich aus mehreren Repositories zusammensetzen, solange diese in sich konsistent sind und sich nach außen wie ein einzelnes Modell repräsentieren.
- Das Modell erlaubt unterschiedliche Sichten auf die Informationen.
- Das Modell ist maschinell auswertbar und liegt in einer abstrakten Syntax vor, die explizit MBSE-Konzepte wie Anforderungen oder Systemarchitekturen unterstützt.

[THM17](#)

Die nachfolgenden Abbildung zeigt mögliche beteiligte Rollen in einem Systems Engineering Ansatz, die an einem gemeinsamen Systemmodell arbeiten. Die Rollenbezeichnungen und Prozesse können von Unternehmen zu Unternehmen variieren.

Der Produktmanager oder Kunde auf der linken Seite der Abbildung übergibt dem Systemanalytiker die Anforderungen in einem Lastenheft oder über ein ReqIF-File. ReqIF steht für Requirements Interchange Format und ist ein standardisiertes XML-Dateiformat von der Object Management Group, mit welchem Anforderungen inkl. ihrer Metadaten

zwischen verschiedenen Requirements Management Tools ausgetauscht werden können. Der Systemanalytiker hat die Aufgabe diese Anforderungen zu analysieren, abzustimmen und zu verfeinern. Zusätzlich muss er die Anforderungen von weiteren Stakeholder (z.B. Gesetzgebung) berücksichtigen. Die Analyse der Anforderungen erfolgt durch die Modellierung von Use Cases bis hin zu einem umfangreichen Domänenmodell. Die textuellen Anforderungen in dem Requirements Management Tool können dazu als grafische Elemente in ein Modellierungstool importiert werden. Für die Modellierung von Systemen eignet sich die Notation SysML der Object Management Group, welche Teile der UML-Notationselemente übernimmt und zusätzlich um domänenspezifische Sprachelement erweitert.

Auf Basis der verfeinerten Anforderungen kann der Testingenieur seine Testfälle erstellen und der Systemarchitekt das System genauer spezifizieren. Sowohl die Beschreibung der Testfälle als auch die Spezifizierung des Systems erfolgt erneut in SysML. Die einzelnen Komponenten, die während der Spezifizierung vom Systemarchitekt identifiziert wurden, werden anschließend jeweils an die Experten-Teams für Software, Hardware und Mechanik / Konstruktion weitergegeben.

Die Teams haben die Aufgabe diese Komponenten mit ihren Werkzeugen weiter zu detaillieren. Damit die Teams mit ihren Werkzeugen an einer Komponente weiterarbeiten können, kann es notwendig sein, dass das SysML-Modell durch Modelltransformation in ein anderes Modell übersetzt werden muss.

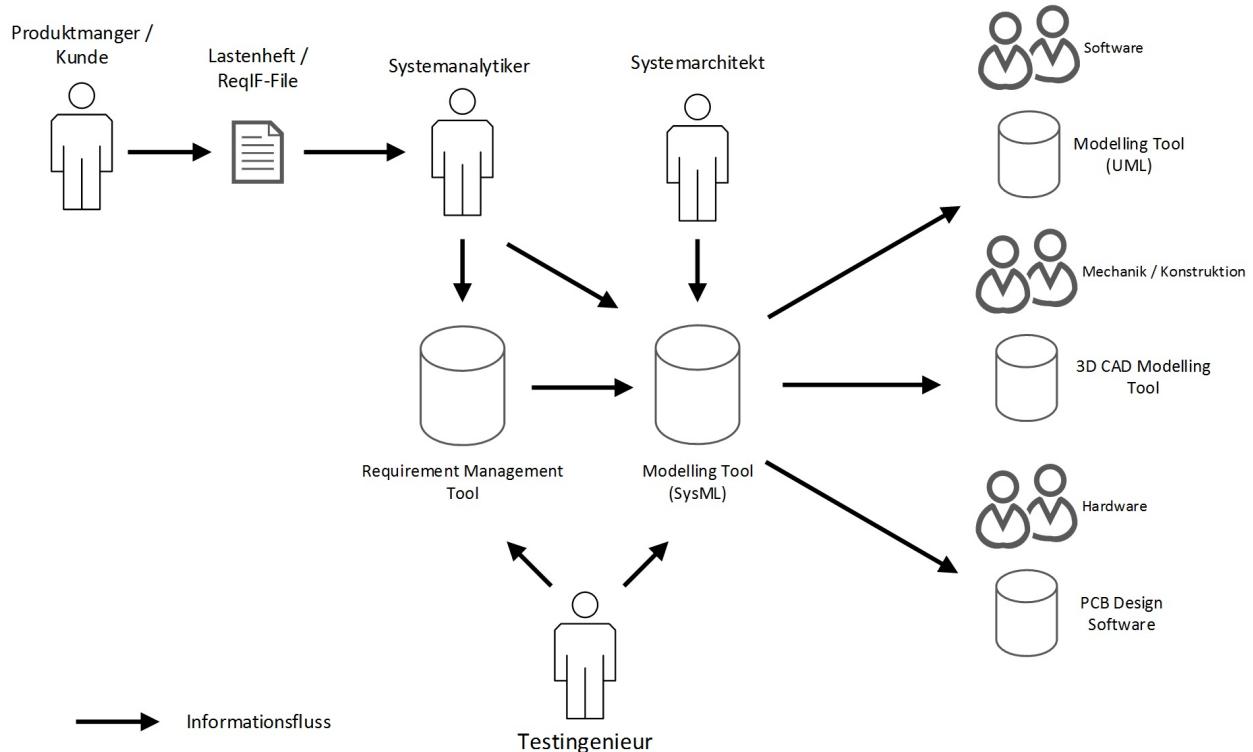


Abbildung 1: Vereinfachte Darstellung der beteiligten Rollen

Obwohl die vorgestellte Vorgehensweise stark an das Wasserfallmodell erinnert, handelt es sich in der Realität um einen stark iterativen Entwicklungsprozess mit vielen Feedbackzyklen. Sobald z.B. die Anforderungen in ausreichender Qualität vorliegen, können erste Entwürfe der Systemarchitektur entstehen und Lösungsideen mit Fachexperten abgestimmt werden. In den Experten-Teams kann ein agiles Vorgehen angestrebt werden.

Systems Modeling Language

Die Systems Modeling Language (SysML) ist eine allgemeine grafische Modellierungssprache um komplexe Systeme zu spezifizieren, analysieren, designen und verifizieren. Die erste Version von SysML wurde von der Object Management Group 2007 angekündigt und liegt bis dato in der Version 1.5 vor. Komplexe Systeme umfassen in der Regel Hardware, Software, Informationen, Personal, Abläufe und technische Hilfsmittel. Die SysML liefert eine grafische Repräsentation mit einer Semantik um Systemanforderungen, Verhalten, Struktur und Zusicherungen eines Systems zu modellieren. Sie enthält eine Teilmenge der UML-Elemente und zusätzlich Erweiterungen, um den Anforderungen des System Engineering gerecht zu werden (siehe Abbildung 2).

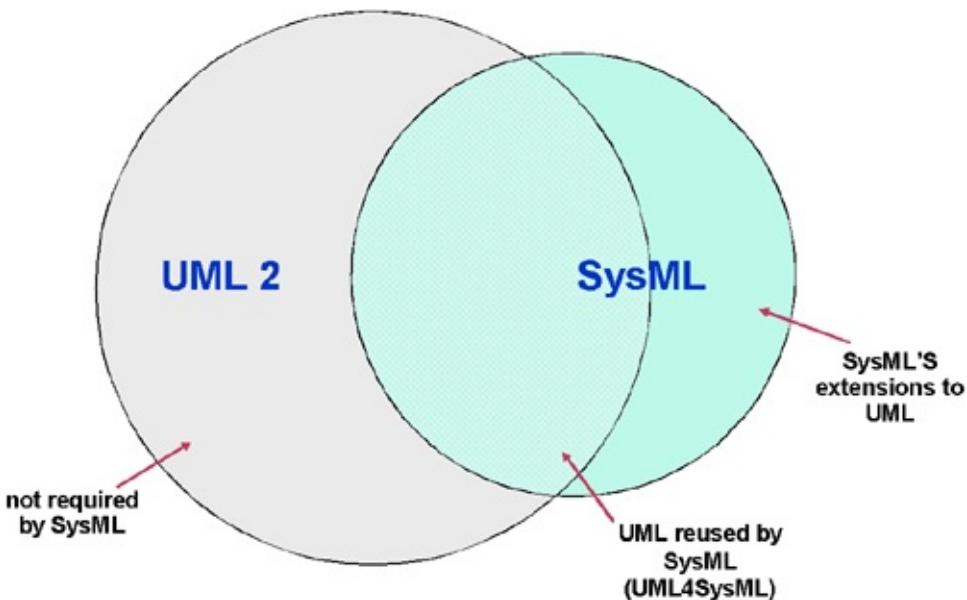
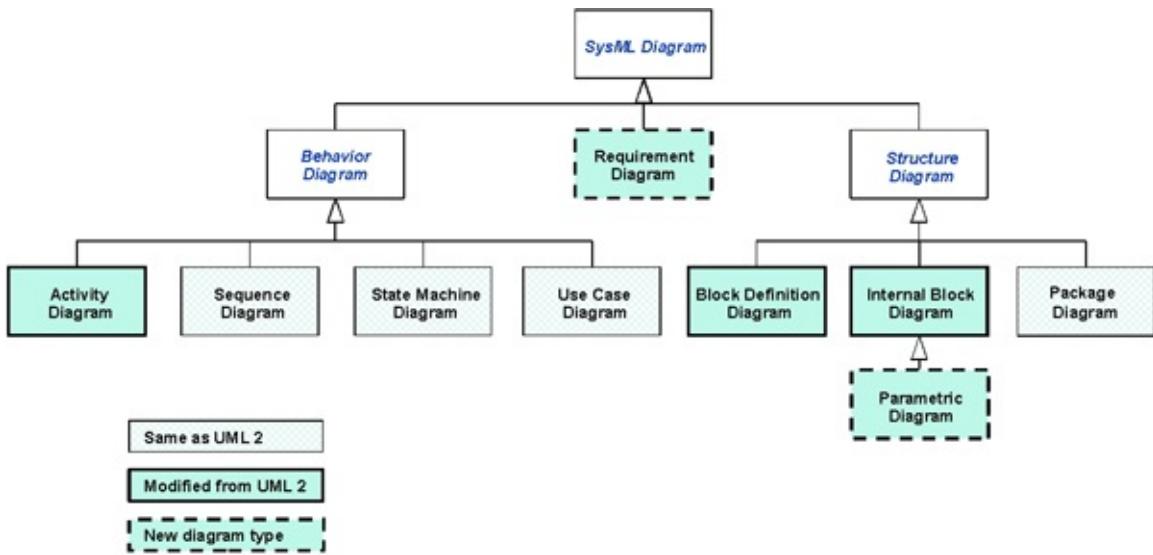


Abbildung 2: SysML und UML Teilmengen [OMG17](#)

SysML-Diagramme

Die SysML umfasst insgesamt neun Diagramme. Das Use Case-, Sequence-, State Machine- und Package Diagramm wurden von der UML übernommen. Das Activity-, Block Definition- und Internal Block Diagramm wurden auch von der UML übernommen, enthalten jedoch Modifikationen. Das Requirement Diagramm und Parametric Diagramm sind neu in der SysML. Abbildung 3 zeigt die Hierarchie der Diagramme.

Abbildung 3: SysML Diagrammtypen [OMG17](#)

Requirement Diagramm

Das Requirement Diagramm wird zur Visualisierung von text-basierten Anforderungen und deren Beziehung zu anderen Modelementen verwendet. Es können Beziehungen zwischen den Elementen hergestellt werden, um z.B. zu zeigen, dass eine Anforderung durch ein anderes Element (z.B. ein Systemelement) erfüllt wird oder das ein Testfall eine Anforderung verifiziert. Zusätzlich können Anforderungshierarchien oder Anforderungsableitungen dargestellt werden. [OMG17](#)

Block Definition Diagramm

Das Block Definition Diagramm wird verwendet um Systemhierarchien zu definieren. Das Basiselement ist der Block, welcher als Repräsentant von Hardware, Software, Hilfsmittel, Personal oder anderen Systemelementen dient. [OMG17](#)

Internal Block Diagramm

Das Internal Block Diagramm beschreibt die interne Struktur eines Blocks mit seinen Teilen (parts), Schnittstellen(ports) und den Verbindungen (connectors) in einem bestimmten Kontext. [OMG17](#)

Package Diagramm

Das Package Diagramm dient zur Darstellung der Organisation / Struktur eines Models. Es eignet sich sehr gut, um eine Navigation durch das Modell herzustellen. [OMG17](#)

Parametric Diagramm

Das Parametric Diagramm beschreibt die Beschränkungen (engl. constraints) der Systemeigenschaften wie z.B. Performanz, Zuverlässigkeit oder zulässige Eigenschaften und ist ein Hilfsmittel für die Analyse eines Systems. [OMG17](#)

Use Case Diagramm

Das Use-Case Diagramm liefert eine High-Level Beschreibung der Funktionalität, die das System seiner Umwelt bereitstellt.[OMG17](#)

Activity Diagramm

Das Activity Diagramm zeigt den Daten- und Kontrollfluss von verschiedenen Abläufen im System.[OMG17](#)

Sequence Diagramm

Das Sequence Diagramm zeigt die Interaktion zwischen dem System und seiner Umwelt oder zwischen verschiedenen Systemelementen.[OMG17](#)

State Machine Diagram

Das State Machine Diagramm beschreibt Zustandsübergänge und Aktionen, die ein System oder Systemelement eine Antwort auf Events durchführt. [OMG17](#)

Cross-cutting

Im Bereich des Systems Engineering wird oft der Begriff „Allokation“ verwendet. Dieser meint die Zuweisung von Funktion zu Komponenten, logische zu physikalische Komponenten oder Software zu Hardware. Die SysML stellt dafür eine spezielle Relationship „allocate“ bereit. [OMG17](#)

Metamodeling in SysML / UML

Die Modellierungssprachen wie UML, SysML oder die BPMN, welche durch die Object Management Group (OMG) definiert wurden, basieren auf dem Konzept von sprachbasierten Metamodellen. Ein sprachbasiertes Meta-Modell stellt die Elemente einer Modellierungssprache und ihre Beziehungen in einem Modell dar. [WIKI17c](#)

Den Meta-Modellen der OMG liegt die Meta Object Facility (MOF) zu Grunde. Die MOF beschreibt eine Metadaten-Architektur, dessen Kernbestandteil das Meta-Meta-Modell ist. Sie wurde mit dem Ziel definiert, eine gemeinsame Grundlage für verschiedene Meta-Modelle zu definieren. Dieser Ansatz bietet einige Vorteile, da verschiedene Meta-Modelle dadurch gleichzeitig persistiert und verarbeitet werden können. Die Modelltransformation ist eine Art der Verarbeitung von Modellen. [WIKI17d](#)

Abbildung 4 zeigt beispielhaft den Zusammenhang der verschiedenen Ebenen der (Meta-)Modelle anhand der UML.

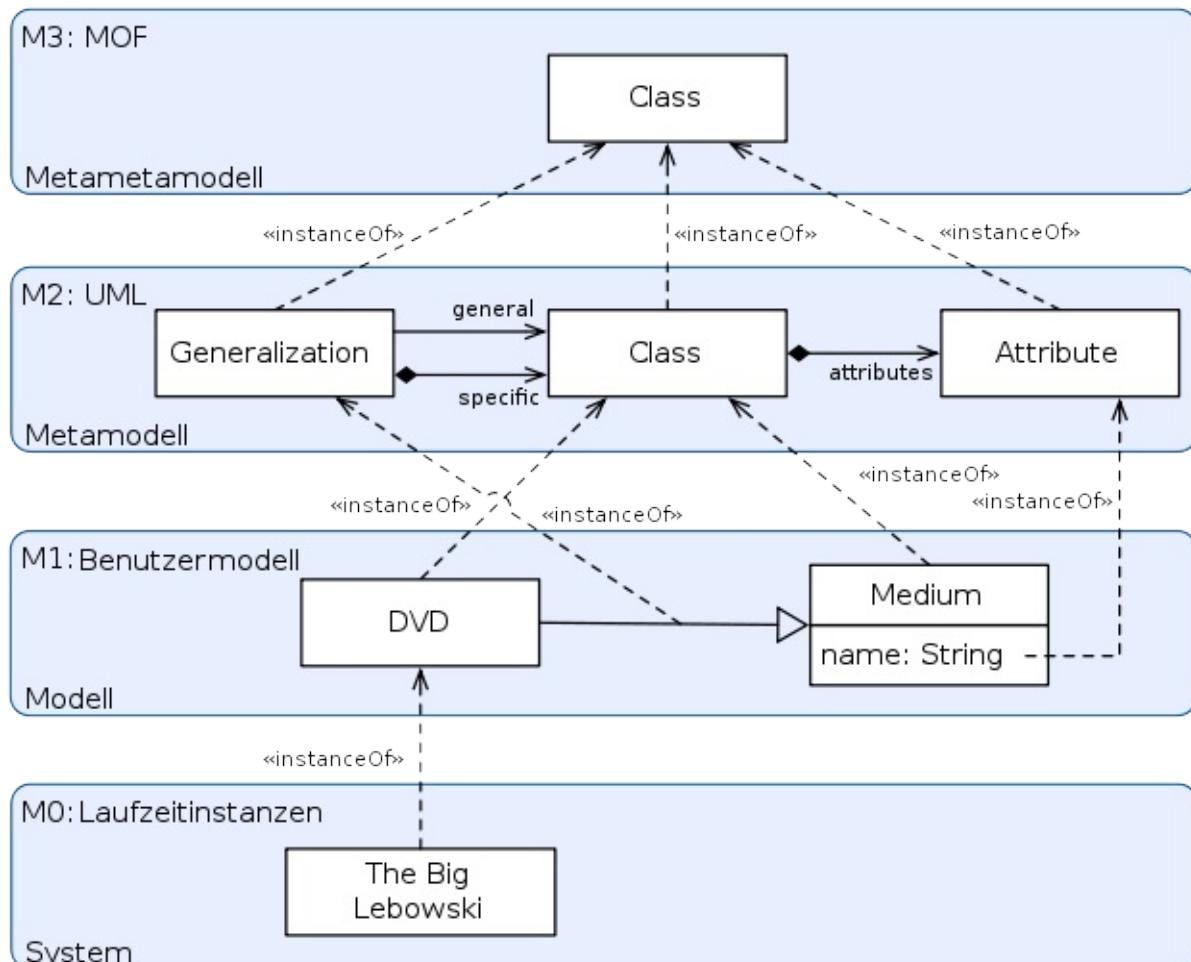


Abbildung 4: Ebenen der Meta-Modelle [WIKI17f](#)

Profil-Mechanismus

Um eine Modellierungssprache wie UML oder SysML um domänenspezifische Sprachelemente zu erweitern, stellt die OMG einen Profil-Mechanismus bereit. Ein Profil ist eine konkrete Erweiterung eines Metamodells. Ein leichtgewichtiger Mechanismus für die Erweiterung eines Metamodells sind Stereotypen. Ein Stereotyp spezialisiert eine Metaklasse für ein spezifisches Einsatzgebiet.

Für jeden Stereotypen können Stereotypeigenschaften definiert werden, in denen domänenspezifische Informationen gespeichert werden können. Der Profil-Mechanismus wird im Folgenden anhand von Beispielen erläutert.

Die folgende Abbildung erweitert die Metaklasse *Actor* um den Stereotypen *externalSystem*. Diese Erweiterung kann genutzt werden, um bei einem Use-Case-Diagramm die Akteure stärker zu differenzieren.

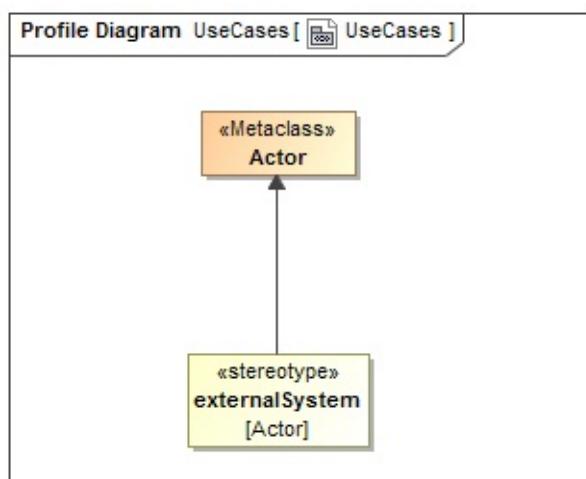


Abbildung 5: Erweiterung der Meta-Klasse Akteur

In Abbildung 6 wird die konkrete Anwendung eines *externalSystems* in einem Use-Case-Diagramm gezeigt. Das Diagramm zeigt Use-Cases für ein Smart Home System. In dem Use-Case „Jalousien mit Smartphone steuern“ ist der Akteur „Bewohner“ und das *externalSystem* „Jalousie“ beteiligt.

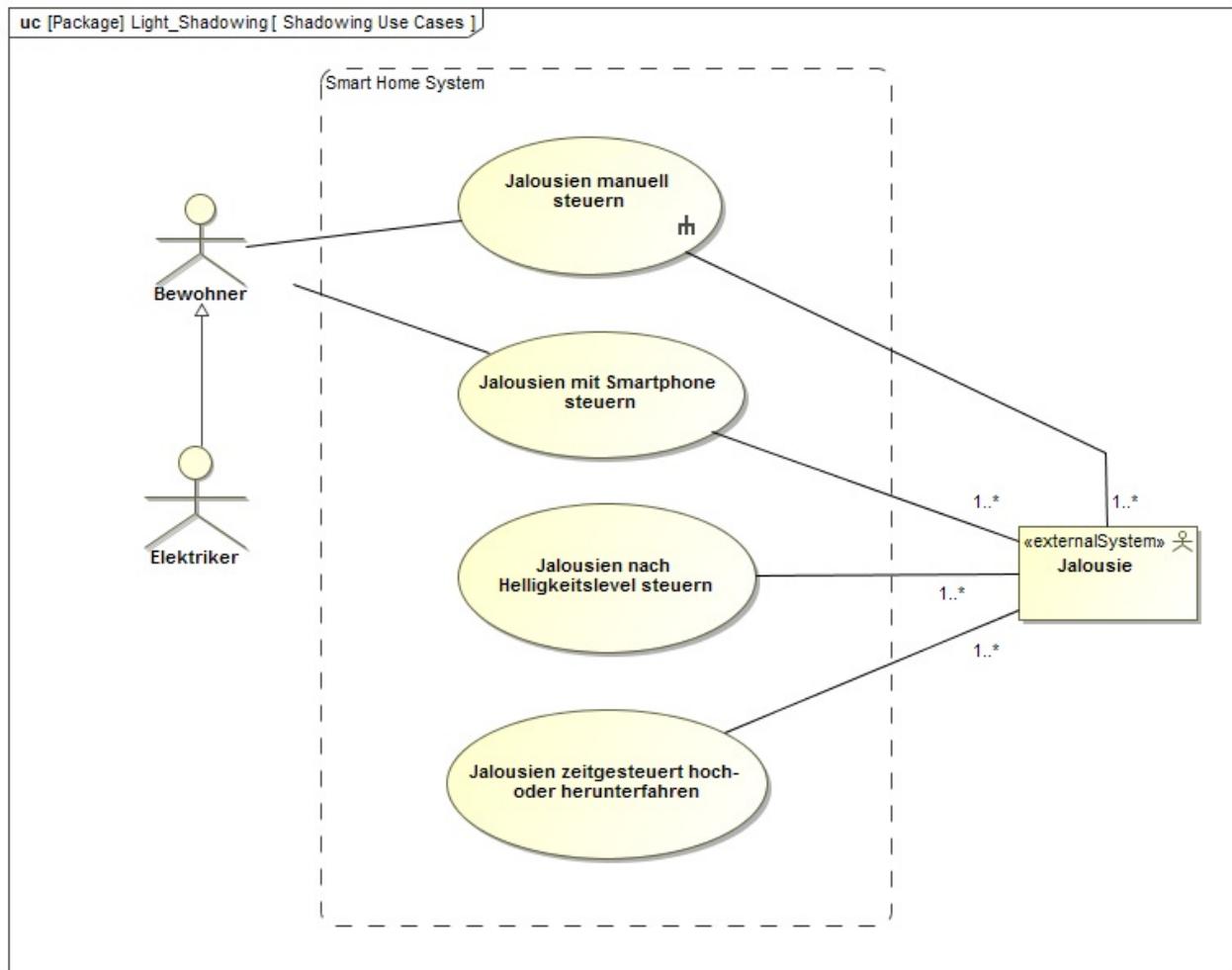


Abbildung 6: Konkrete Anwendung der Aktuer-Erweiterung

Ein Stereotyp kann von anderen Stereotypen mit Hilfe einer „Generalisierung“ spezialisiert werden. In Abbildung 7 ist der Stereotyp „Requirement“ von der Metaklasse *Class* erweitert worden. Der Stereotyp besitzt zusätzliche Eigenschaften wie z.B. *Id* und *Text*. Von „Requirement“ wurde ein spezialisierter Stereotyp „ExtendedRequirement“ erzeugt, der alle Eigenschaften von „Requirement“ besitzt und zusätzlich die Attribute *obligation* und *priority* enthält. Für die bessere Differenzierung von Anforderungen wurden von „ExtendedRequirement“ die drei Anforderungstypen „Business Requirement“, „System Requirement“ und „Design Requirement“ abgeleitet.

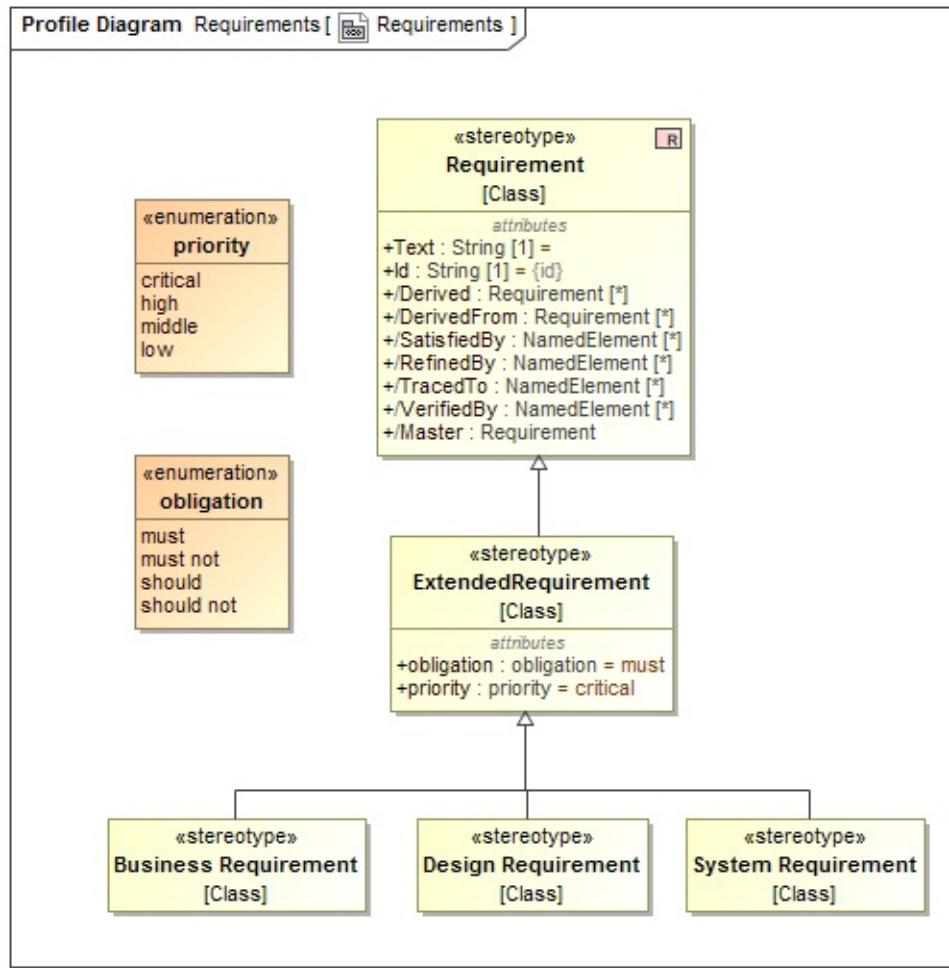


Abbildung 7: Erweiterung einer Anforderung

Abbildung 8 zeigt die Visualisierung des Stereotypen „Business Requirement“ in einem Requirement Diagramm der SysML.

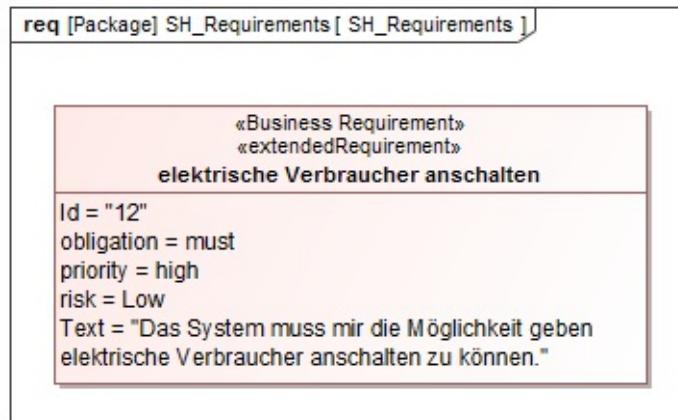


Abbildung 8: Anwendung des Stereotypens "Business Requirement"

Product Line Engineering

Unter dem Begriff Product Line Engineering (PLE) wird die Entwicklung eines Portfolios von verwandten Produkten durch die Verwendung gemeinsamer Entwicklungsartefakten und effizienten Produktionsmitteln verstanden. Anstatt verschiedene Produkte individuell zu entwickeln, steht beim Product Line Engineering die Entwicklung eines gesamten Produktportfolios in den frühen Phasen des Entwicklungsprozesses im Vordergrund. Durch diesen Ansatz können Synergien besser ausgeschöpft und die Wiederverwendung verbessert werden.

Beim Product Line Engineering werden die Produkte aufbauend auf einer Plattform entwickelt, die die Gemeinsamkeiten („Core Assets“) mehrerer Produkte zusammenfasst. Zu diesen Core Assets gehören z.B. Architekturen, Softwarekomponenten, Domain Models, Use-Cases oder Test Cases. Da diese Core-Assets bereitgestellt werden müssen, bevor ein konkretes Produkt entwickelt werden kann, ist die Zeit bis zur Markteinführung des ersten Produktes (time-to-market) und das Entwicklungsrisiko deutlich höher als bei der produktzentrischen Entwicklung. Jedoch können weitere Produkte später durch die Wiederverwendung deutlich schneller entwickelt werden. Die Zeit bis zur Markteinführung eines Produktes in Abhängigkeit der Anzahl der verschiedenen Produkte ist in Abbildung 9 dargestellt.

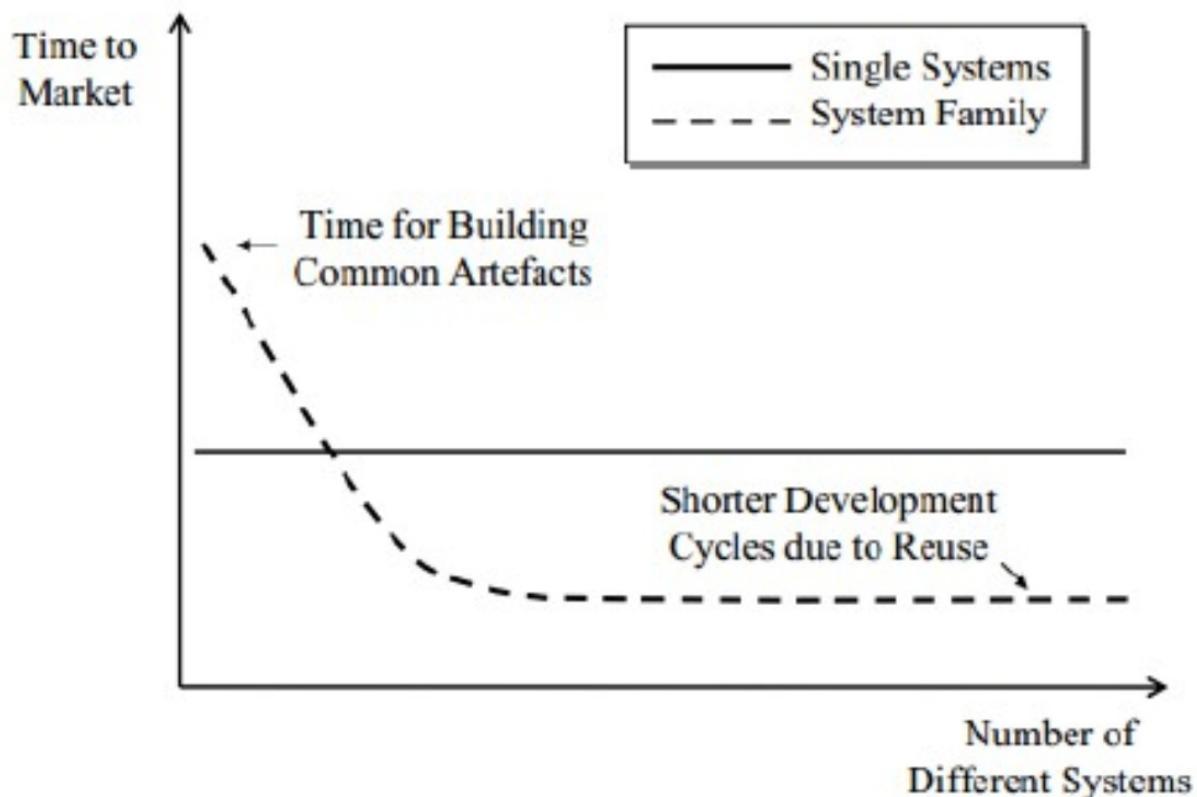


Abbildung 9: Time-to-Market in Abhängigkeit der Anzahl der Produkte KUBE15

Im Product Line Engineering wird zwischen zwei Bereichen unterschieden:

Im **Domain Engineering** werden die „Core-Assets“ herausgearbeitet und auf dieser Basis eine gemeinsame Systemarchitektur entwickelt.

Im **Application Engineering** werden auf Grundlage der Ergebnisse des Domain Engineerings die Produkte der Produktlinie entwickelt.

In Abbildung 10 ist die grundlegende Struktur des *Product Line Engineerings* dargestellt. Die beiden Bereiche werden im Folgenden genauer erläutert.

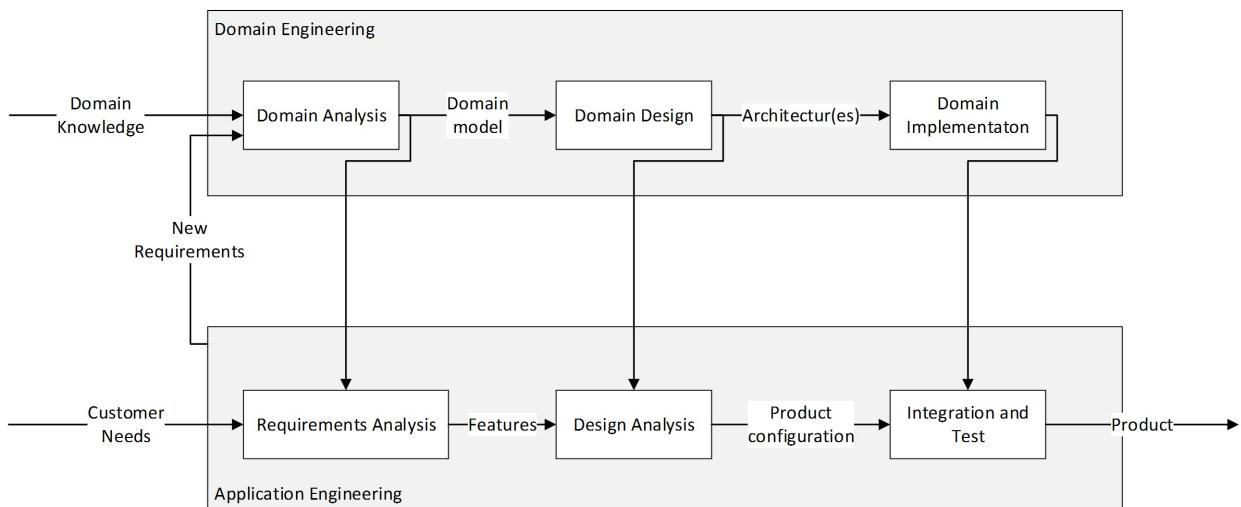


Abbildung 10: Struktur des Product-Line-Engineerings

Domain Engineering

Ziel des *Domain Engineering* ist es, bereits vorhandenes Wissen über eine Domäne zu erfassen und dieses Wissen bei der Entwicklung neuer Produkte in dieser Domäne zu nutzen [CZAR00](#).

„Domain Engineering is the activity of collecting, organizing, and storing past experience in building systems or parts of systems in a particular domain in the form of reusable assets (i.e., reusable work products), as well as providing an adequate means of reusing these assets (i.e., retrieval, qualification, dissemination, adaptation, assembly, and so on) when building new systems.“ [CZAR00](#)

Der *Domain Engineering* Ansatz ist hierbei in die drei Schritte *Domain Analysis*, *Domain Design* und *Domain Implementation* (siehe Abbildung 10) aufgeteilt.

In der **Domain Analysis** wird zu Beginn die „domain of focus“ definiert und anschließend Informationen über diese Domäne gesammelt. Aus diesen Informationen wird das Domain Model (siehe Absatz "Domain Model") erstellt. Als Informationsquelle dienen Domänen-Experten, Anforderungen von dedizierten Produkten der Produktlinie, sowie vorhandene (Mitbewerber-)Produkte. Das Domain Model dient als Eingangsartefakt für den nachfolgenden Schritt „Domain Design“. Die Ergebnisse gehen außerdem in die Anforderungsanalyse (Requirements Analysis) der Produkte der Produktlinie mit ein. Zusätzlich wird in der Domänen-Analyse über Feature-Modeling (siehe Absatz "Feature Modeling") die notwendigen Features der Produkte und deren Abhängigkeiten dargestellt. Die Feinheiten und Methoden des Feature-Modelings werden im späteren Verlauf des Kapitels erläutert.

Im **Domain Design** wird aus dem in der Domain-Analysis erstellten Domain-Model eine generische Systemarchitektur erstellt, die sich in allen Produkten der Produktlinie vereinigen lässt. (Vgl. [SCHR03](#))

In der **Domain Implementation** wird die generische Architektur implementiert. Diese kann in den diversen Produkten der Produktlinie Wiederverwendung finden. (Vgl. [SCHR03](#))

Application Engineering

Das *Application Engineering* beschreibt den Prozess der Entwicklung von Systemen, basierend auf den Ergebnissen des Domain Engineerings. Auch hierbei findet eine Auftrennung in drei Teilschritte statt (*Requirements Analysis*, *Design Analysis* und *Integration and Test* – siehe Abbildung 10).

Domain Model

Das Domain Model enthält alle Aspekte einer Domäne die zur Entwicklung einer Produktlinie innerhalb dieser Domäne von Relevanz sind. Das Domain Model ist hierbei eine abstrakte Beschreibung von Problemstellungen, die durch das System zu lösen sind. Ein einfaches Beispiel hierfür stellt eine Software zur Vertragsverwaltung dar. Das Domain Model enthält hierbei alle Informationen die zur Verwaltung der Verträge relevant sind - zum Beispiel die Beziehungen zwischen Verträgen und Kunden.

Vorgehensmodelle

Im Produktlinienansatz wird zwischen drei unterschiedlichen Vorgehensmodellen unterschieden (Vgl. [APEL10](#)):

Im **proaktiven Vorgehensmodell** wird eine Produktlinie komplett neu entwickelt. Hierbei wird zu Beginn der komplette Domain-Engineering-Prozess durchlaufen. Dies ist mit hohem initialen Aufwand und den damit einhergehenden Kosten verbunden. Der proaktive Ansatz ist sinnvoll, wenn die Anforderungen gut definiert und konstant sind. (Vgl. [APEL10](#))

Das **reaktive Vorgehensmodell** lässt sich mit agilen Vorgehensmodellen in der Software-Entwicklung vergleichen. Zu Beginn erfolgt eine Implementierung einer kleinen Basis. Nachfolgend werden in kleinen Schritten die Domänen-Analyse mit anschließender weiterer Implementierung durchgeführt. Hierdurch erhält man ein zyklisches Arbeiten. Die Vorteile dieses Vorgehensmodell sind wie beim agilen Ansatz die geringen initialen Kosten, wie auch das Ausliefern schneller erster Ergebnisse. Es können allerdings auch Probleme entstehen, die zu einer teuren Umstrukturierung der Arbeitsweise im späteren Verlauf des Projektes führen. Das reaktive Vorgehensmodell ist vor allem geeignet, wenn die Anforderungen zu Beginn noch sehr unklar sind.

Beim **extraktiven Vorgehensmodell** dient eine bereits entwickelte Produktlinie als Grundlage. Aus dieser werden Informationen extrahiert, die in die neue zu entwickelnde Produktlinie einfließen. Da eine existente und funktionsfähige Produktlinie als Basis dient, halten sich die Risiken und Kosten in Grenzen. Allerdings steigt auch der Anspruch im Domain Engineering, da Informationen aus einem System extrahiert werden, das nicht im Produktlinienansatz entwickelt wurde.

Das extractive Vorgehensmodell ist geeignet, wenn man einen schnellen Wechsel vom traditionellen zum Produktlinien-Ansatz realisieren möchte.

Feature Modeling

Im Folgenden wird gezeigt wie mit Feature Modeling die Features bzw. Varianten eines Produktes dargestellt werden können. Die hierbei verwendete Sprache ist die Feature Modeling Notation, die nicht in der UML bzw. SysML enthalten ist.

Ein *Feature* wird immer aus Anwendersicht betrachtet:

„A prominent or distinctive user-visible aspect, quality, or characteristic of a software system or systems“ [FODA90](#)

Zu Beginn erfolgt eine kurze Einleitung in die verwendete Notation, die abschließend in einem Beispiel angewendet wird.

Mit der **Konjunktion** (UND-Verknüpfung) wird beschrieben, dass alle verbundenen Elemente (Features) im Produkt umgesetzt werden müssen. In Abbildung 11 müssten also alle Elemente (A, B und C) vom Produkt implementiert werden:

$$A \wedge B \wedge C$$

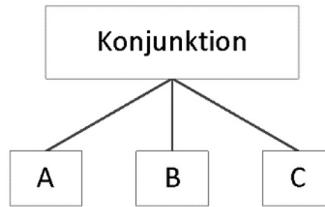


Abbildung 11: Konjunktion

Mit der **Disjunktion** (ODER-Verknüpfung) wird beschrieben, dass mindestens eines der verbundenen Elemente umgesetzt werden muss. Es besteht keine Abhängigkeit zwischen den Elementen. Es können also beliebige Kombinationen der Features umgesetzt werden. Die Disjunktion wird durch einen ausgefüllten Halbkreis am ausgehenden Feature-Strang markiert.

Im Beispiel dürften also beliebige Kombinationen aus den Features A, B und C umgesetzt werden. Jedoch mindestens eines von Ihnen:

$$A \vee B \vee C$$

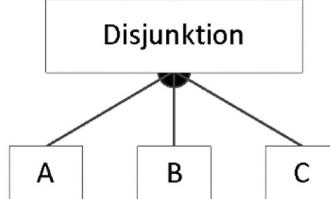


Abbildung 12: Disjunktion

Die **Kontravalenz** (Exklusiv-ODER-Verknüpfung) beschreibt, dass genau eines der verbundenen Elemente umgesetzt werden muss. Es darf kein weiteres Element umgesetzt werden. Die Kontravalenz wird durch einen Halbkreis am ausgehenden Feature-Strang markiert. Im Folgenden müsste also entweder Feature A, B oder C umgesetzt werden:

$$A \oplus B \oplus C$$

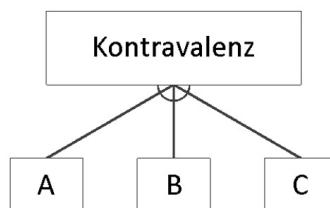


Abbildung 13: Kontravalenz

Mit **Implikationen** können Abhängigkeiten zwischen Features dargestellt werden. Wird ein Feature umgesetzt, kann dies bedeuten, dass auch ein anderes Feature umgesetzt werden muss. In Abbildung 14 können Feature A oder C oder beide umgesetzt werden. Wird C umgesetzt so muss auch B umgesetzt werden. Falls A umgesetzt wird darf B ebenfalls umgesetzt werden, muss es allerdings nicht. Es existiert keine Abhängigkeit zwischen A und B.

$$A \vee (C \wedge B)$$

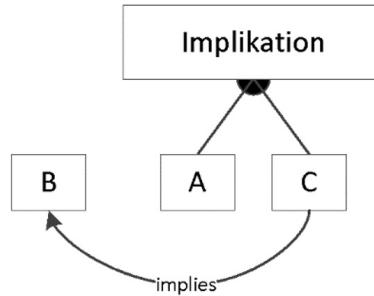


Abbildung 14: Implikation

Obligationen und **Optionen** werden eingesetzt um darzustellen, dass Features umgesetzt werden müssen (Obligation) bzw. umgesetzt werden können (Option). Die Obligation wird dabei durch einen ausgefüllten Kreis am entsprechenden Feature dargestellt, die Option durch einen nicht ausgefüllten. Im Beispiel müssen sowohl Feature A als auch C umgesetzt werden. Feature B kann optional umgesetzt werden:

$$(A \wedge \bar{B} \wedge C) \vee (A \wedge B \wedge C) = A \wedge C$$

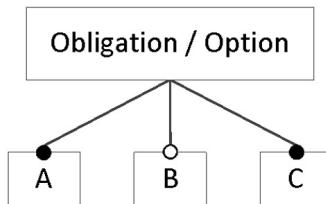


Abbildung 15: Obligation / Option

Im Folgenden wird eine exemplarisches Feature-Model eines Smartphones gezeigt. Dieses Model zeigt die Features, die ein Kunde bei der Auswahl seines Gerätes direkt konfigurieren kann.

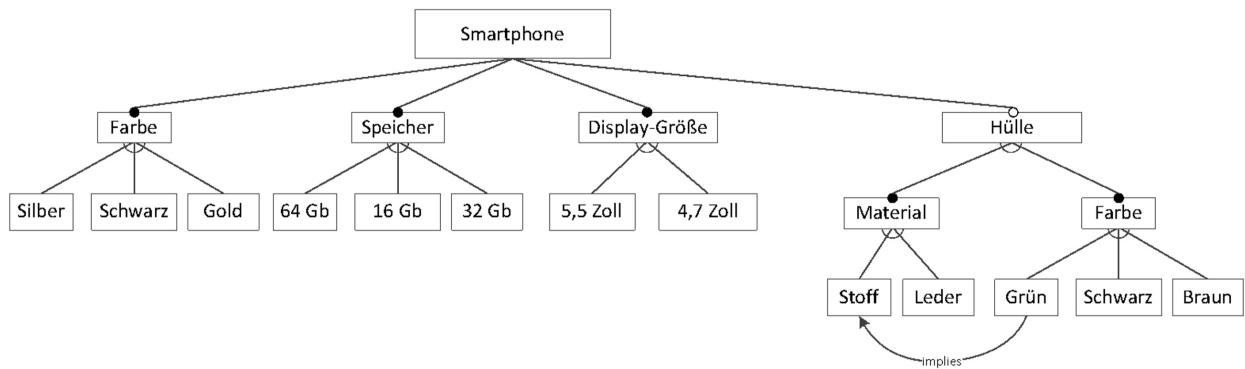


Abbildung 16: Smartphone-Feature-Modell

Durch die zuvor erläuterten Notationselemente ist zu erkennen, dass der Kunde in jedem Fall eine Auswahl der Features *Farbe*, *Speicher* und *Display-Größe* treffen muss. Des Weiteren hat er die Möglichkeit eine *Hülle* zum Gerät hinzuzufügen. Hierbei hat er die Auswahl zwischen zwei *Materialien* (*Leder* und *Stoff*) und verschiedenen *Farben* (*Grün*, *Schwarz* und *Braun*). Entscheidet sich der Kunde für die Farbe *Grün* ist er gezwungen ebenfalls das Material *Stoff* zu nehmen, da diese Farbe für das Material *Leder* nicht zur Verfügung steht.

Aus diesem relativ übersichtlichen Feature-Modell lassen sich bereits 108 unterschiedliche Varianten bilden (Berechnung siehe [Anhang](#)). Es ist also zu erkennen, dass auch eine geringe Variabilität von Features in einem Produkt zu einem schnellen Anstieg der Anzahl der Varianten führen kann.

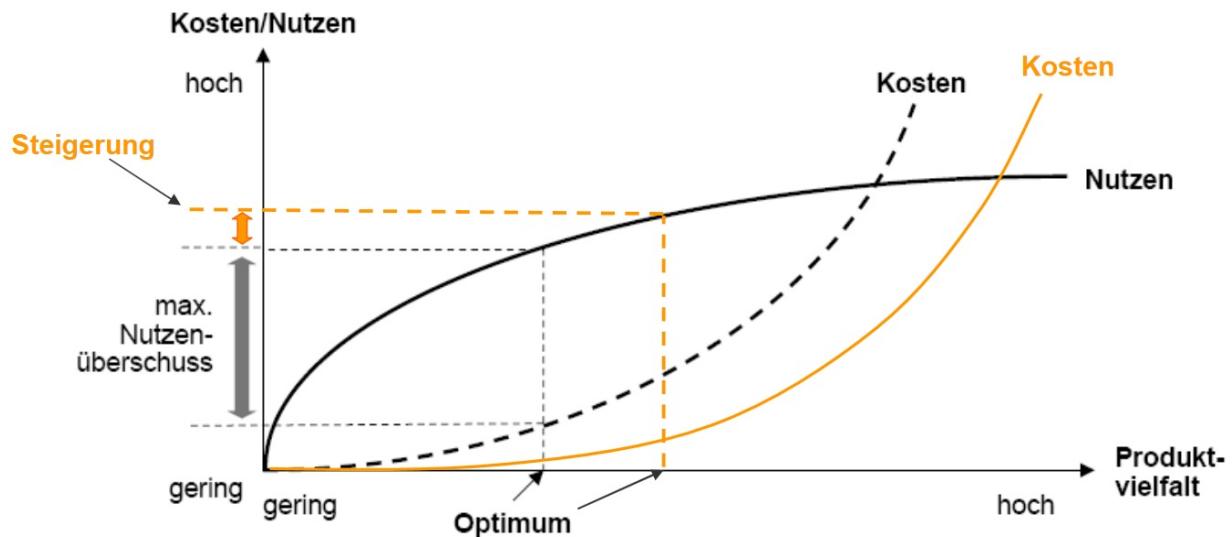


Abbildung 17: Produktvielfalt im Verhältnis zum Kundennutzen [ROCK09](#)

Die Varianten und die Anzahl der Features sind in einem durchschnittlichen Software-System um ein Vielfaches höher, als bei dem hier dargestellten Produkt. Umso wichtiger ist es die Variabilität so gering wie möglich zu gestalten, um auch die Komplexität auf einem

kontrollierbaren Level zu halten. Die Variantenbildung ist oft eine Gratwanderung zwischen Kundennutzen und Komplexität. Dieser Sachverhalt wird in Abbildung 17 gezeigt.

Model Transformation

Eine zentrale Technik bei der modellbasierten Entwicklung ist die Modelltransformation. Die Modelltransformation wird benötigt, um Informationen eines Datenmodells in ein anderes Datenmodell, welches eine andere Struktur besitzt, zu überführen. [OSTF17](#)

Die Bandbreite der Anwendungsgebiete reicht von der Dokumentengenerierung über die Variantengenerierung bis hin zur Erstellung neuer Modelle und der Codegenerierung. Dabei ist die Vorgehensweise der Transformation immer gleich. Es werden Informationen aus dem Quellmodell abgefragt und diese nach definierten Regeln in eine andere Form gebracht. Diese Regeln werden auch als Transformationsregeln bezeichnet und werden auf der Ebene der Metamodelle definiert.

Bei der Modelltransformation wird zwischen Modell-zu-Modell-Transformation (M2M-Transformation) und Model-zu-Text-Transformation (M2T-Transformation) unterschieden. [WIKI17b](#)

Modell-zu-Modell-Transformation

Innerhalb der M2M-Transformation kann erneut in den zwei Varianten Inplace-Transformation und Outplace-Transformation unterschieden werden. Bei der Inplace-Transformation wird durch die Transformation kein neues Modell erzeugt, sondern das bestehende Modell modifiziert. Die Outplace-Transformation generiert dagegen ein neues Modell und modifiziert das Quellmodell nicht. [WIKI17b](#)

Im Folgenden sollen verschiedene Transformationssysteme im Überblick dargestellt werden.

Query View Transformation (QVT)

Die Query View Transformation (QVT) ist eine formale Sprache für M2M-Transformation und wurde von der OMG standardisiert. Sie ist Teil der Meta Object Facility (MOF). Unter einer Query werden Anfragen auf ein MOF-Modell verstanden, um bestimmte Modellelemente zu finden, die transformiert werden sollen. Unter einer View versteht die OMG ein Modell, welches aus einem Quellmodell abgeleitet wurde. Die Transformation ist die Überführung von ein oder mehrere Quellelementen in ein oder mehrere Zielelemente. Für die Transformation stehen innerhalb der QVT zwei Sprachen zur Verfügung:

- die QVT Operational Mappings Language, welche eine imperitative Sprache ist.
- die QVT Relations Language, die eine deklarative Sprache ist und neben einer textuellen auch eine grafische Syntax definiert. [ITWI17a](#)

Atlas Transformation Language (ATL)

Die Atlas Transformation Language ist eine Sprache zur Modelltransformation und wurde auf Grund einer Ausschreibung der OMG für eine allgemeine Programmiersprache zur Modell- und Codetransformation entwickelt. Die ATL ist eine hybride Sprache. Dies bedeutet, dass sie Konzepte imperativer und deklarativer Programmierung vereint. [ITWI17b](#)

MOF Model to Text Transformation Language (MOFM2T)

Die MOF Model to Text Transformation Language ist ein weiterer Standard der OMG zur metamodeltbasierten Modell-zu-Text-Transformation. Sie kann genutzt werden, um z.B. Code auf einem UML oder SysML-Modell zu generieren. [WIKI17e](#)

[INC17a] INCOSE, What is Systems Engineering?, aufgerufen am 28.05.2017,

URL: <http://www.incose.org/AboutSE/WhatIsSE>

[INC17b] INCOSE, About Systems Engineering, aufgerufen am 28.05.2017,

URL: <http://www.incose.org/AboutSE>

[INC15] INCOSE, Systems Engineering Handbook,

A Guide for System Life Cycle Processes and Activities, Fourth Edition, 2015

[THM17] Technische Hochschule Mittelhessen (THM), Systemmodell, aufgerufen am 28.05.2017,

URL: <https://wiki.thm.de/Systemmodell>

[WIKI17a] Wikipedia, Repository, aufgerufen am 29.05.2017,

URL: <https://de.wikipedia.org/wiki/Repository>

[OMG17] ObjectManagement Group, What is SysML, aufgerufen am 18.06.2017,

URL: <http://www.omg.sysml.org/what-is-sysml.htm>

[WIKI17c] Wikipedia, Metamodell, aufgerufen am 18.06.2017,

URL: <https://de.wikipedia.org/wiki/Metamodell>

[WIKI17d] Wikipedia, Meta Object Facility, aufgerufen am 18.06.2017,

URL: https://de.wikipedia.org/wiki/Meta_Object_Facility

[ITWI17a] IT Wissen, QVT (query view transformation), aufgerufen am 18.06.2017,

URL: <http://www.itwissen.info/QVT-query-view-transformation.html>

[ITWI17b] ITWissen, ATL (ATLAS transformation language), aufgerufen am 18.06.2017,

URL: <http://www.itwissen.info/ATL-ATLAS-transformation-language.html>

[WIKI17e] Wikipedia, MOF Model to Text Transformation Language, aufgerufen am 18.06.2017,

URL: https://de.wikipedia.org/wiki/MOF_Model_to_Text_Transformation_Language

[WIKI17f] Wikipedia, Unified Modeling Language, aufgerufen am 18.06.2017,

URL: https://de.wikipedia.org/wiki/Unified_Modeling_Language

[WIKI17b] Wikipedia, Modelltransformation, aufgerufen am 29.05.2017,

URL: <https://de.wikipedia.org/wiki/Modelltransformation>

[OSTF17] Ostfalia,Hochschule für angewandte Wissenschaften, Modelltransformation,
aufgerufen am 02.06.2017,

URL: www.ostfalia.de/cms/de/ivs/ease/Merapi/Modelltransformation.html

[APEL10] Apel, Sven; Christian, Kästner; Saake, Gunter; Softwareproduktlinien,
Entwicklungsprozess und

Variabilitätsmodellierung, Universität Marburg, 2010

[FODA90] Kyo C. Kang; Sholom G. Cohen; James A.Hess; William E. Novak; A. Spencer
Peterson;

Feature-Oriented Domain Analysis(FODA) Feasibility Study; Software Engineering Institute,
Carnegie Mellon University, Pittsburgh 1990

[CZAR00] K. Czarnecki and U. W.Eisenecker; Generative Programming|Methods, Tools, and
Applications; Addison-Wesley, 2000. ISBN 0-201-30977-7

[SCHR03] Prof. Dr.-Ing. Wolfgang Schröder-Preikschat, Dr.-Ing. Olaf Spinczyk;
Vorlesung Betriebssystemtechnik (OSE) – Domain Engineering, 2003

[ROCK09] Dr. Georg Rock; Variantenmanagement –Forschung und industrieller Einsatz; TU
Darmstadt, 2009

[KUBE15] Kuberczyk, Christoph; Product Line Engineering (PLE) - Produktlinienentwicklung;
Universität Hamburg, 2015

Anhang

Berechnung der Anzahl der Varianten:

$$V = n_{Farbe} * n_{Speicher} * n_{DisplayGröße} * (1 + (n_{Material} * (n_{HüllenFarbe} - 1)) + 1)$$

Mit:

V := Anzahl aller möglichen Varianten

$n_{Farbe} = 3$:= Anzahl der möglichen Farbvarianten des Geräts

$n_{Speicher} = 3$:= Anzahl der möglichen Speichervarianten des Geräts

$n_{DisplayGröße} = 2$:= Anzahl der möglichen Displayvarianten des Geräts

$n_{HüllenFarbe} = 3$:= Anzahl der möglichen Farbvarianten der Hülle

$n_{Material} = 2$:= Anzahl der möglichen Materialvarianten der Hülle

Continuous Software Engineering

Ursprünglich verstand man unter dem Begriff der kontinuierlichen Softwareentwicklung die Pflege und Weiterentwicklung von bestehenden Softwarekomponenten. Im Zuge der zunehmenden Komplexität von Software zählen heute zur kontinuierlichen Softwareentwicklung Methodiken und Prozesse, die bereits während der Entwicklung zum Einsatz kommen. Dazu gehören agile Vorgehensmodelle wie Scrum und Kanban, mit denen umfangreichen Produktanforderungen iterativ erarbeitet werden. Zur Erreichung von Qualitätszielen sind unterschiedliche Workflow-Ansätze wie DevOps, Continuous Integration und Test Driven Development Bestandteile der Vorgehensmodelle. Zur technischen Umsetzung solcher Workflows bedarf es zudem Hardware- und Softwarekomponenten, die entwicklungsbegleitend und automatisiert bei der Anforderungsumsetzung und Qualitätssicherung unterstützen. [10](#)

Agile Vorgehensmodelle

Agile Vorgehensmodelle sind eine Methode des Projekt Managements. Sie zeichnen sich üblicherweise durch iteratives Vorgehen und kurze Feedback-Zyklen aus. Sie sollen nach häufiger Auffassung schwergewichtige Vorgehensmodelle ablösen, die mit viel Management-Aufwand verbunden sind, ablösen.

Robert Martin beschreibt hingegen, dass in den Anfangszeiten der Software-Entwicklung in den 60er Jahren bereits agil gearbeitet wurde. Er begründet das damit, dass damals hauptsächlich disziplinierte, professionelle Mathematiker und Fachanwender als Softwareentwickler tätig waren. Erst das auftauchen großer Mengen junger, undisziplinierter Entwickler machte es nötig, ihnen feste Strukturen vorzugeben. (vgl. [6], ab ca. 48m)

Bekannte, aktuelle Vertreter agiler Vorgehensmodelle sind Scrum, Kanban und Extreme Programming (XP). Die ersten beiden sollen nachfolgend kurz erläutert werden.

Scrum

Das zur Zeit bekannteste agile Modell ist Scrum, es soll hier nur in seinen Grundzügen erläutert werden. Es stammt zwar aus der Softwareentwicklung, ist aber prinzipiell unabhängig davon.

Einer der Hauptaspekte von Scrum ist die Einteilung der Projektlaufzeit in sogenannte Sprints. Diese dauern üblicherweise 2-4 Wochen und am Ende jedes Sprints sollte eine lauffähige, inkrementell verbesserte Software bereit stehen. Während eines Sprints werden einzelne Aufgaben in Form von Backlog-Items abgearbeitet. Die abzuarbeitenden Items werden jeweils vor Beginn eines Sprints festgelegt und dem Product-Backlog entnommen. Die Backlog-Items, die während eines Sprints abgearbeitet werden sollen, werden im Sprint-Backlog abgelegt.

Der Product-Backlog beinhaltet alle noch zu erledigenden Backlog-Items und wird vom Product-Owner gefüllt. Das geschieht initial beim Projektbeginn, sowie während der Projektlaufzeit (z.B. Änderungswünsche nach einem Sprint).

Neben dem Product Owner gibt es außerdem noch das Entwicklerteam als beteiligte, welche die eigentliche Entwicklungsarbeit erledigen. Außerdem gibt es den Scrum Master, der die Durchführung überwacht. Er stellt eine "dienende Führungskraft" dar, gibt also keine Arbeitsanweisungen, sondern steht beratend zur Seite.

Kanban

Kanban ist ein weiteres agiles Vorgehensmodell, sein zentraler Punkt ist das sogenannte Kanban-Board. Der Begriff "Kanban" stammt aus dem japanischen und bedeutet "Signalkarte", dort wurde es als Produktionsprozess ursprünglich von Toyota entwickelt. In der IT wurde es zuerst 2007 von David J. Anderson vorgestellt.

Pull-Prinzip

Kanban arbeitet nach dem sogenannten Pull-Prinzip. Das bedeutet, dass keine Arbeit auf Vorrat erledigt wird, sondern immer erst dann ausgeführt wird, wenn sie wirklich benötigt wird. Das führt in der Anwendung in Produktionsumgebungen zu deutlich geringeren Lagerkapazitäten und einer Reduktion von Abfall, falls auf Vorrat produziertes am Ende nicht mehr gebraucht werden kann. In der Software-Entwicklung ist Lagergröße zwar kein entscheidender Faktor, aber die Reduktion von Abfall, also Code der am Ende nicht gebraucht wird, ist mindestens genau so wichtig, wenn nicht wichtiger, da die Arbeitszeit der Entwickler die hauptsächlich begrenzende Ressource ist.

Kanban-Board

Im Kanban-Board werden alle einzelnen Aufgaben in Form von Zetteln aufgeklebt. Dabei gibt es eine Unterteilung in Spalten, die jeweils mit dem Zustand der darin enthaltenen Karten beschriftet sind (z.B. "To Do", "In Arbeit", "Testing", "Erlledigt").



Abbildung: Beispielhaftes Kanbanboard, Quelle: [Wikipedia](#), Jeff.lasovski

Das Board darf natürlich nicht nach belieben mit Aufgaben versehen werden. Es unterliegt einigen Regeln und Gesetzmäßigkeiten, diese werden in den sechs Kernpraktiken genauer beschrieben.

Kernpraktiken

Die sechs Kernpraktiken von Kanban lauten im einzelnen[7]:

1. Visualisiere den Fluss der Arbeit

Auf dem Kanban Board finden sich alle Aufgaben, die noch ausstehen, gerade erledigt werden, oder schon abgeschlossen sind. Das bringt eine gute Übersicht über den aktuellen Status einzelner Arbeitseinheiten sowie den Fortschritt des Projekts.

2. Begrenze die Menge angefangener Arbeit

Es darf jeweils nur eine begrenzte Menge an Karten in den Spalten vorhanden sein. Es kann allerdings sinnvoll sein, nicht für jede Spalte ein Limit zu setzen. So kann zum Beispiel eine Spalte "Fertig" alle bereits erledigten Karten enthalten. Diese zu beschränken würde schlussendlich dazu führen, dass keine Arbeit mehr verrichtet werden darf. Dadurch wird effektiv die angefangene Arbeit auf die Anzahl der Karten limitiert. Das Ziel ist es dabei, dass einzelne Aufgaben schneller erledigt werden, wenn nicht zwischendurch andere angefangen werden. Dabei wird die benötigte Zeit für die einzelnen Aufgaben zwar nicht signifigant gesenkt (höchstens durch weniger Wechsel zwischen Aufgaben), aber der Fortschritt ist schneller sichtbar. Außerdem können eventuell abhängige Aufgaben so eher von anderen Entwicklern begonnen werden.

3. Miss und steuere den Fluss

Typische Werte, die in einem Kanbanprozess gemessen werden, sind die Wartezeit, Zykluszeit und der Durchsatz. Diese erlauben es, Rückschlüsse auf die Organisation der Arbeit zu ziehen und sie zu verbessern.

4. Mache die Regeln für den Prozess explizit

Um unnötige Unsicherheiten und Ablaufprobleme zu vermeiden, sollen alle Beteiligten die genauen Regeln des Prozesses kennen. Dazu gehören unter anderem die exakte Bedeutung der Kanban-Board-Spalten, was genau fertig heißt (Definition of Done), und der genaue Ablauf, wer wann welche Karten auf dem Kanban-Board verschiebt.

5. Implementiere Feedback-Mechanismen

Um den Prozess kontinuierlich zu verbessern, ist es wichtig, Schwächen aufzudecken und Verbesserungsmöglichkeiten für diese zu finden. Damit diese kontinuierliche Verbesserung funktioniert, muss sie auf allen Ebenen der Organisation stattfinden. Dazu gehört, dass auch Entwickler konkrete Verbesserungsvorschläge einbringen.

Darüber hinaus kann aber auch Feedback von Außenstehenden wertvoll sein, da es neue Blickwinkel ermöglicht. Nützliche Mittel, um regelmäßiges Feedback zu ermöglichen, sind Daily-Standups und insbesondere (z.B. monatliche) Retrospektiven.

6. **Führe Verbesserungen auf Basis von Modellen durch**

Schon lange gibt es wissenschaftliche Modelle, die sich mit Engpässen und Problemen in Betriebsabläufen beschäftigen. Es ist sinnvoll, diese für die Verbesserung des Prozesses einzusetzen. Die Auswahl der Modelle ist nicht weiter eingeschränkt, damit die Modelle verwendet werden können, die für den konkreten Einzelfall am geeignesten sind.

Kanban und andere Vorgehensmodelle

Da Kanban im Gegensatz zu anderen Vorgehensmodellen einige Dinge nicht festlegt, ist es prinzipiell Möglich, Kanban gleichzeitig mit anderen Modellen einzusetzen. So ist zum Beispiel keine zeitliche Einteilung in Sprints gegeben, wie es in Scrum der Fall ist. Außerdem gibt es keine feste Rollenbeschreibung der Beteiligten. Kanban und Scrum können so zum Beispiel in einer Kombination durchgeführt werden. So können die Vorteile des Kanban-Boards und dessen Regeln mit den Rollen- und Zeiteinteilung von Scrum vereint werden.

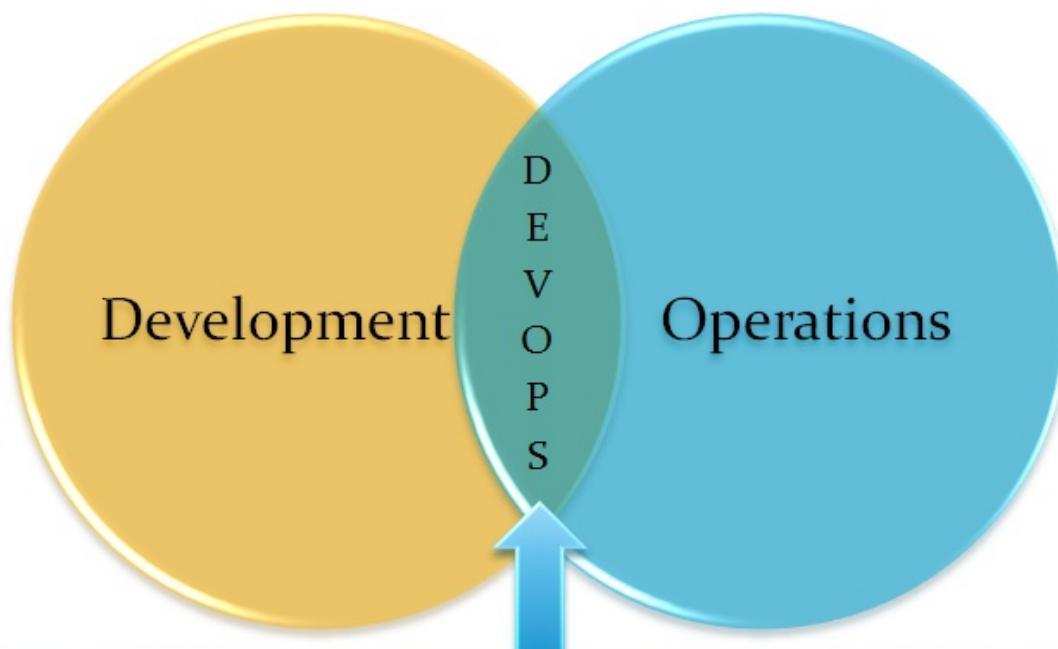
Es ist auch möglich, Kanban mit nicht agilen Vorgehensmodellen zusammen zu benutzen. Fall es in irgendeiner Art Kollisionen gibt, ist auch ein "Kanban-Light" vorstellbar, dass je nach Anwendungsfall eine oder mehrere Kernpraktiken ändert oder auslässt. In jedem Fall kann dabei der Vorteil des Kanban-Boards und dessen dokumentierender bzw. visualisierender Funktion genutzt werden.

Unterstützende Workflows

DevOps

Wichtig für das Verständnis ist, dass DevOps kein Framework, Tool oder Technologie ist. Bei DevOps geht es um die Kultur eines Unternehmens und um das Vorgehen, wie die Menschen in einem Unternehmen arbeiten. Unterstützende Faktoren sind definierte Prozesse und angepasste Tools um durch automatisierte Abläufe eine erhöhte Effektivität zu erreichen.

Durch die Anpassungen und Veränderungen in den Bereichen Kultur, Prozesse und Technologie soll die Kommunikation, sowie die Zusammenarbeit der Entwicklung (development) und "IT operations" verbessert werden [20](#).



Dies hat Auswirkungen auf die Effektivität des "application life cycles". Durch diese Auswirkungen sind die Vorteile, welche durch den Einsatz von DevOps erzielt werden können, sehr vielseitig.



Die Kultur DevOps kann als innovatives Bündel gesehen werden um "Dev" und "Ops" Teams effektiv zu integrieren. Dies beinhaltet Themen wie continuous build integration, continuous testing, continuous delivery, continuous improvement, etc., mit dem Ziel die Auslieferung von Software zu beschleunigen.

Die Schwierigkeit liegt darin, dass DevOps häufig nicht verstanden wird. Hier geht es nicht um einzelne Bereiche, wie bereits aufgezählt bspw. continuous integration. Es geht um die Kultur! Und eine Kultur zu wandeln, dauert sehr lange. Ein Beispiel soll das Missverständnis rund um DevOps verdeutlichen. Hier geht es um fünf blinde Männer und einen Elefanten. Jeder der Männer berührt einen spezifischen Teil des Elefanten und jeder nimmt an das dies der Elephant sei [20](#).

Selbstverständlich bezieht sich der Kulturwandel nicht nur auf das "development" und "operations" Team. An diesem Wandeln müssen sich alle Teams (testing team, cloud team, etc.) anschließen.

Für ein tieferes Verständnis werden wir im Folgenden auf zwei Fragestellungen eingehen: 1) Warum DevOps? 2) Wie kann eine DevOps Kultur entstehen?

Warum DevOps?

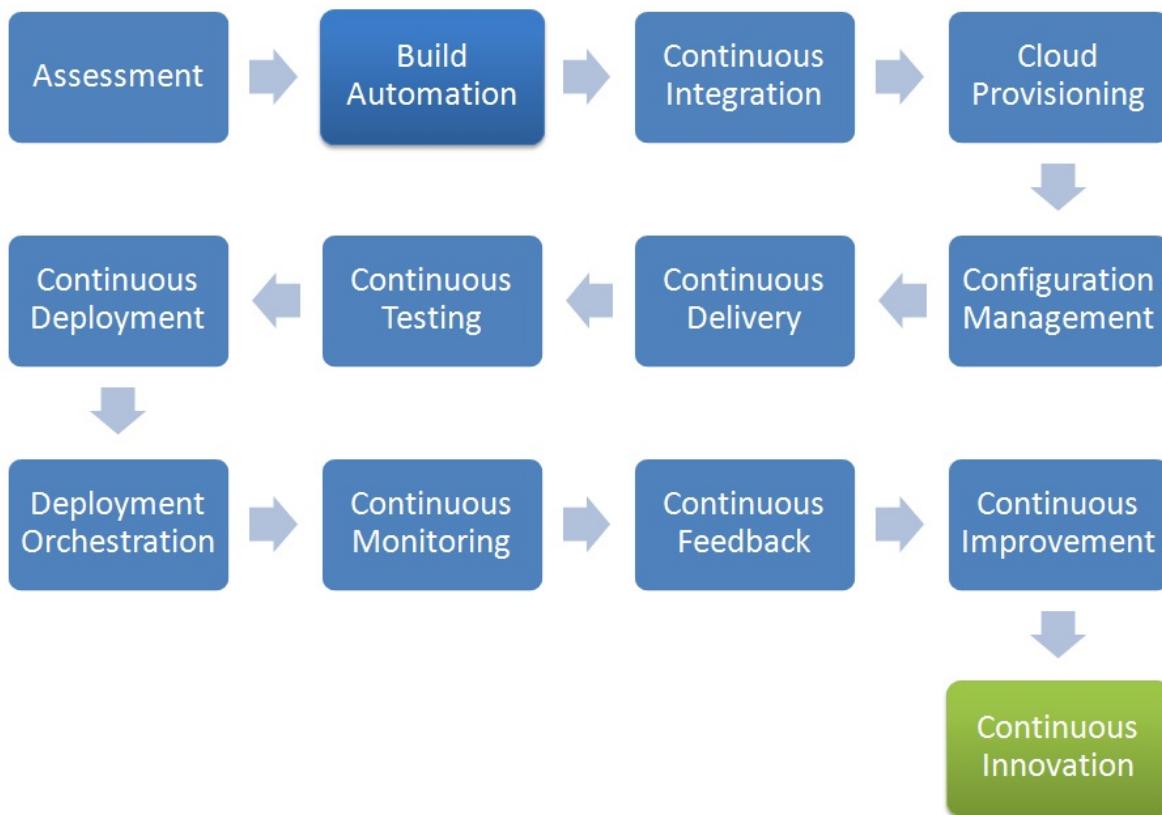
Veränderung ist eine essentieller Bestandteil des Lebens und auch von Organisationen. Schaut man lediglich auf bestehende Methodiken, Kulturen und Praktiken wird man zukünftige "best practices" nie erreichen. Vor allem in der sich schnell ändernden IT-Welt hat die Veränderung eine enorme Bedeutung.

Progress is impossible without change, and those who cannot change their minds cannot change anything. [20](#)

Jede Änderung muss abgewogen werden nach dem Nutzen und den durch die Änderung entstehenden Schwierigkeiten.

Wie kann eine DevOps Kultur entstehen?

Häufig führen Probleme und Ineffizienz zu Veränderungen. Hierdurch hat sich aus einem Wasserfallmodell die Agile Methodik entwickelt. Eine Kultur zu verändern geht nicht über Nacht und benötigt sehr viel Zeit. Hieraus resultiert, dass DevOps als Kultur schrittweise eingeführt und akzeptiert werden muss. Jeder Schritt ist unabhängig von einander zu implementieren und sollte keine Abhängigkeiten zu anderen Schritten haben [20](#).



Bspw. kann "Continuous Testing" ohne jede andere DevOps Praktik implementiert werden. Hierdurch entsteht bereits ein Mehrwert, welcher dazu beiträgt Akzeptanz zu schaffen und die Kultur der Menschen zu verändern [20](#).

Continuous Integration

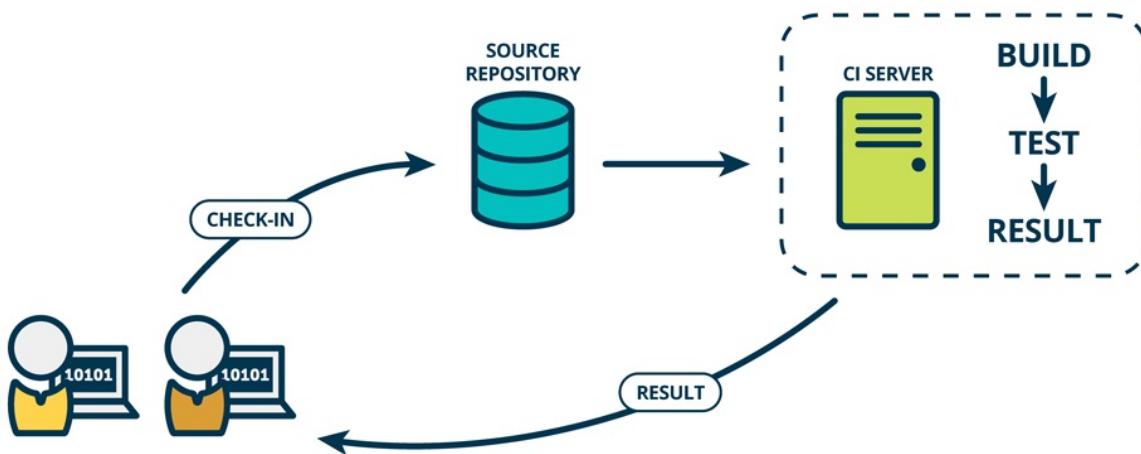
Traditionell wird bei der Softwareentwicklung die Integration am Ende eines Projektes stattfinden. Je nach Projektgröße und Komplexität liegt die Dauer der Integrationsphase im Bereich Wochen oder Monate.

Continuous Integration (CI) bedeutet, dass zwei Entwickler unabhängig von einander Software schreiben können, jedoch für das selbe Produkt. Dies geschieht durch ständige und regelmäßige Integration in ein "source repository".

In dem "source repository" werden die einzelnen Incremente mehrere Entwickler zusammengeführt und können durch einen CI Server integriert werden.

Grundvoraussetzung hierfür, ist das jeder Entwickler neben seinem Produktiv-Code auch Tests entwickelt. Diese Tests werden durch den CI Server ausgeführt um sicher zu stellen, dass die einzelnen Incremente funktionsfähig bleiben. Das Ergebnis des CI Servers wird an die jeweiligen Entwickler zurück gegeben. Hierdurch können Fehler sofort behoben werden.

Das folgende Bild veranschaulicht den geschilderten Prozess.



Der integrierte Code, ist durch Continuous Integration nicht bereit in Produktion zu gehen, da zwar die Komponenten mit einander funktionieren, allerdings das Produkt (die Software) noch nicht in einer "production-like environment" getestet und verifiziert wurde.

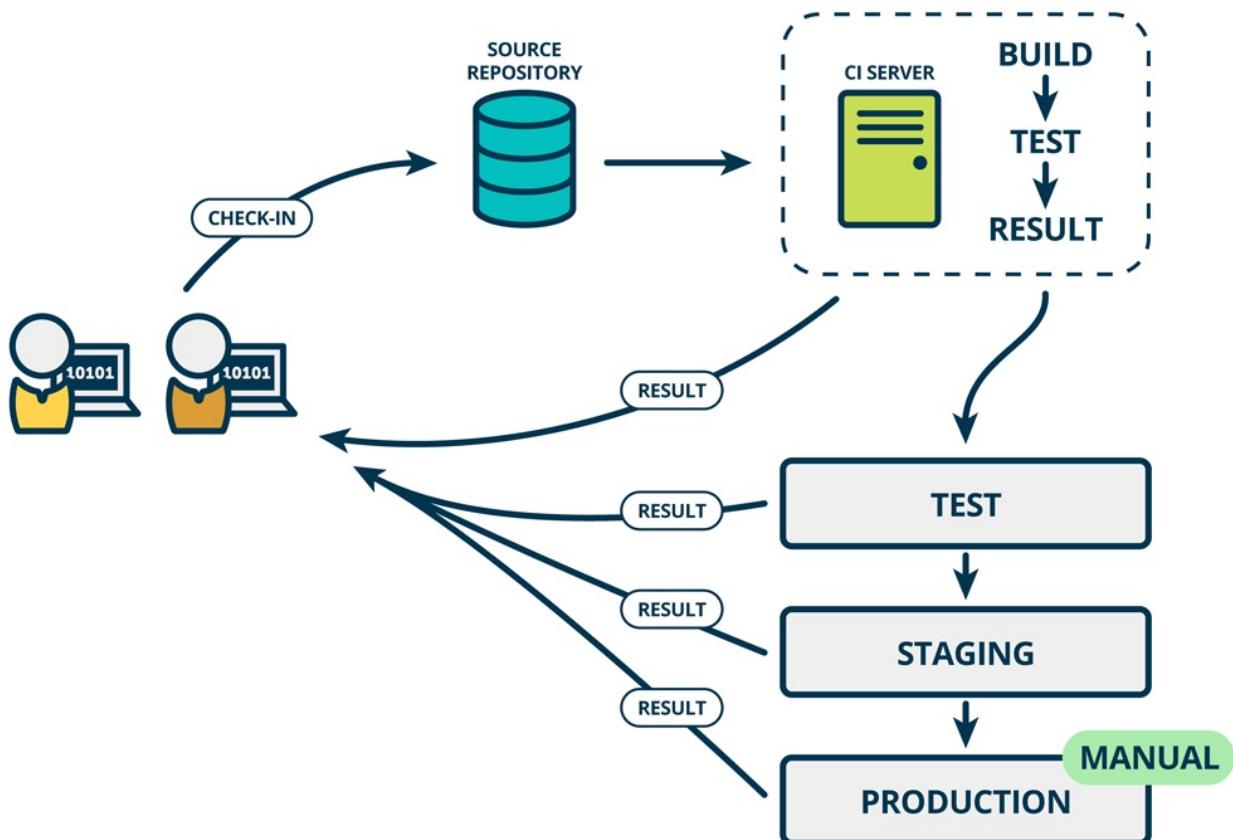
Der Vorteil von CI ist, dass die Integration zum täglichen Geschäft wird. Eine "Big-Bang" Integration am Ende einer Entwicklung wird vermieden.

CI ist zwingend notwendig um Continuous Delivery durchführen zu können [4](#).

Continuous Delivery

Unter Continuous Delivery versteht man, neben allen beschriebenen Schritten von Continuous Integration, das der integrierte Quelltext automatisiert auf verschiedenen Umgebungen getestet wird. Typischerweise sind diese Umgebungen sehr ähnlich zu einer potentiellen Produktiv-Umgebung.

In diesem Zusammenhang wird das deployen und testen auf den verschiedenen Umgebungen als "deployment pipeline" bezeichnet. Je nach Projekt, Team oder Organisation können unterschiedlich viele Umgebungen existieren. Das folgende Bild veranschaulicht diesen Prozess:

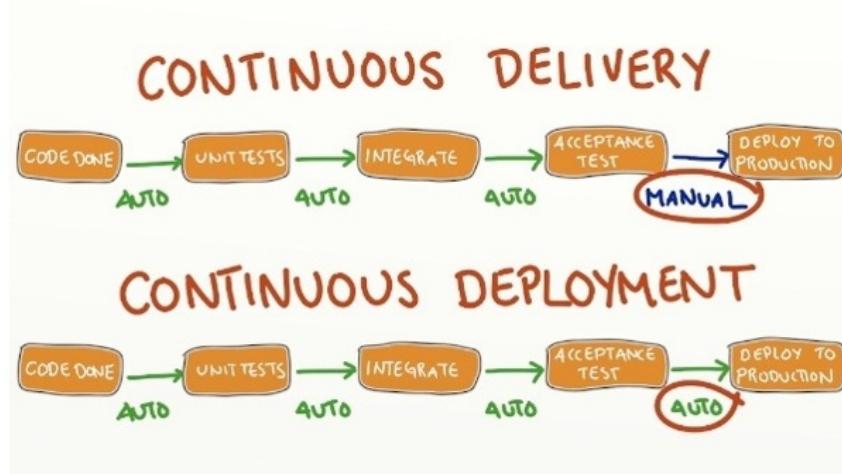


In diesem Beispiel gibt es drei Umgebungen (Development, Test, Staging), welche durchlaufen werden, bevor es in die Produktiv-Umgebung geht.

In jeder Umgebung wird der Quelltext unterschiedlich getestet. Es ergibt sich der Vorteil, dass mit jeder erfolgreich durchlaufenen Umgebung die Wahrscheinlichkeit wächst, dass der Quelltext auch in der Produktiv-Umgebung lauffähig sein wird [4](#).

Continuous Deployment

Continuous Deployment geht noch einen Schritt weiter und automatisiert ebenfalls die Übergabe in die Produktion.



5

Jeder einzelne Commit eines Entwickler kann potentiell automatisiert in der Produktion ankommen. Voraussetzung hierfür ist der erfolgreiche Durchlauf aller bisher aufgeführten Schritte aus den Kapiteln Continuous Integration und Delivery. Zusammengefasst muss der Commit, bzw. der Mehrwert oder das Increment erfolgreich integriert worden sein und alle Tests auf allen implementierten Umgebungen bestehen. Anschließend muss nicht mehr entschieden werden, ob der Mehrwert in der Produktion verwendet werden kann. Dies geschieht mit Continuous Deployment ebenfalls automatisiert [4](#).

TDD, BDD, ATDD

Bei dem Test-driven development (TDD) werden bestehenden Anforderungen zu Beginn in diverse Testfälle gegossen. Anschließend wird die bestehende Software angepasst, sodass die bestehenden Testfälle erfüllt werden. Durch ein solches Vorgehen soll die Implementierung von überflüssigem Quelltext reduziert werden.

Im Kontext des Test-driven developments wird häufig von sehr kurzen Zyklen gesprochen. Diese bestehen aus dem Hinzufügen von Tests, der Prüfung ob die neuen Tests fehlschlagen, dem Implementieren, der erneuten Testausführung und einer abschließenden Phase der Quelltextrestrukturierung.

Grundlegende Prinzipien, welche sich der Entwickler gewahr sein sollte sind bspw. „keep it simple, stupid“ (KISS) oder „You aren't gonna need it“ (YAGNI). Unter Beachtung solcher Prinzipien entsteht überschaubarer, „sauberer“ Quelltext.

Nachteile sind unter anderen das zeitintensive Testen und die Tatsache, dass in der Regel die Tests und der Quelltext von demselben Entwickler verfasst werden. Da dieses Verfahren nur so gut ist wie die Qualität der erstellten Tests, liegt hierin das größte Gefahrenpotential.

Darüber hinaus stoßen test-first Tests dort an ihre Grenzen, wo bspw. komplett Funktionstests implementiert werden müssen (User Interfaces). Diese können nicht implementiert werden bevor die Funktion fertig gestellt wurde [1](#).

Acceptance test-driven development (ATDD) beruht auf demselben Prinzip, wie das Test-driven development. Ein entscheidender Unterschied ist, dass bei diesem Verfahren ebenfalls nach dem test-first Prinzip Akzeptanztests geschrieben werden. Diese Art von Tests sollen die drei Gruppen Entwickler, Tester und den nicht/weniger technischen Part (Kunde, Produktmanager, o.ä.) näher zusammenbringen.

Der Anlass hierfür sind die, sehr häufig drastisch, verschiedenen Verständnisse einer Sachlage und die daraus resultierenden Fehlentwicklungen, zusätzlichen Kosten, etc. Es wird hierbei versucht eine Sprache zu sprechen, indem niedergeschriebene Anforderungen in Akzeptanztests umgewandelt werden. Diese werden anschließend implementiert und nur diese Testfälle im Quelltext implementiert und behoben. Akzeptanztests, welche anschließend hinzugefügt werden, sind neue Anforderungen.

Solche Tests können/sollten von allen Beteiligten gelesen/verstanden werden können. Dies ist beim TDD nicht gegeben [2](#).

Eine Erweiterung des Test-driven developments ist das Behavior-driven development. Die Methode verwendet eine eigene Sprache (domain-specific scripting language = DSL) um Testfälle zu beschreiben. Die verwendete Sprache ist in der Regel strukturiert aufgebaut und sollte auch für „Nicht-Programmierer“ verständlich sein. Hierdurch soll ein noch stärkerer Bezug zu den Akzeptanzkriterien hergestellt werden.

```
Story: Returns go to stock

In order to keep track of stock
As a store owner
I want to add items back to stock when they're returned.

Scenario 1: Refunded items should be returned to stock
Given that a customer previously bought a black sweater from me
And I have three black sweaters in stock.
When he returns the black sweater for a refund
Then I should have four black sweaters in stock.

Scenario 2: Replaced items should be returned to stock
Given that a customer previously bought a blue garment from me
And I have two blue garments in stock
And three black garments in stock.
When he returns the blue garment for a replacement in black
Then I should have three blue garments in stock
And two black garments in stock.
```

Eine Unit wird durch ihr Verhalten beschrieben. Dieses Verhalten sind die Akzeptanzkriterien, aus welchen durch test-first Tests erstellt werden. Diese Tests gehen direkt aus den gegebenen User Stories hervor und dessen Akzeptanzkriterien. Die obige

Abbildung zeigt ein Beispiel für einen solchen Behavior-Test 3.

Unterstützende Tools

Die in den vorangegangenen Kapiteln beschriebenen agilen Vorgehensmodelle und deren Workflows für die Entwicklung komplexer Softwareprodukte benötigen für die Sicherstellung der Softwarequalität kontinuierliche Tests. Dedizierte Server führen die Prozessschritte der vorgestellten Workflows (Continuous Integration, Deployment und Delivery) in sogenannten Pipelines durch. Dazu gehören neben den Modul- und Integrationstests, die der Entwickler auch in seiner lokalen Entwicklungsumgebung durchführen kann, umfangreichere Systemtests bzw. Ende-zu-Ende-Tests. Diese Systemtests benötigen definierte Produktivumgebungen, die sich mit Virtualisierungstechnologien oder neuerdings auch Containerisierungs-Technologien etablieren lassen.

Virtualisierung versus Containerisierung

Bis Anfang der 2010er fand fast ausschließlich die Virtualisierung von Betriebssystemen auf Personal Computern und Servern Anwendung; so auch im Kontext der CI- und CD-Workflows. In den letzten Jahren gewinnt die Technologie der Containerisierung immer mehr an Bedeutung und hat das Potenzial Virtualisierungslösungen teilweise abzulösen. In diesem Kapitel wird zuerst der Bedarf zur Virtualisierung im Kontext von Continuous Software Engineering hergeleitet. Danach wird die jeweilige Funktionsweise der konkurrierenden Technologien grundlegend erklärt, um sie daraufhin vergleichend gegenüberzustellen. Abschließend kommt ein Überblick über etablierte Frameworks.

Bedarf zur Virtualisierung

Im Kapitel zu unterstützenden Workflows wurde der Bedarf von Tests auf allen Integrationsebenen einer Software zur Qualitätssicherung hergeleitet. Bei diesen Tests stehen am Ende die Systemtests, die eine Produktivumgebung für die Applikation voraussetzen. Eine solche Umgebung kann auf dem Hauptbetriebssystem eines Servers konfiguriert werden. Dieses Vorgehen beansprucht einen Server in der Regel nur wenig, da eine Applikation typischerweise ein System nicht voll auslastet. Er ergibt sich somit ein hohes Maß an ungenutzten Ressourcen, was sich negativ auf die Wirtschaftlichkeit solcher Server auswirkt.

Das Konzept der Virtualisierung bietet an dieser Stelle die Möglichkeit die Ressourcen eines Servers deutlich besser auszunutzen, indem entsprechend der Leistungsfähigkeit eines Servers mehrere virtuelle Systeminstanzen erzeugt und parallel ausgeführt werden. Ein weiterer Vorteil der Virtualisierung ergibt sich, wenn eine Applikation auf verschiedenen

Betriebssystemen lauffähig sein soll. Dazu kann für alle zu unterstützenden Betriebssysteme eine separate virtuelle Produktivumgebung bereitgestellt werden. Für verschiedene und parallele Tests genau einer Produktivumgebung kann zudem ein virtuelles Systemabbild beliebig vervielfacht und betrieben werden.

Arten der Virtualisierung

Eine triviale und einfache Form, eine relativ isolierte Umgebung zu schaffen, ist das Partitionieren eines Systems. Diese Vorgehen ermöglicht aber kein wirklich paralleles Ausführen von Applikationen, da es stets zu Kollisionen bei der Verwendung von Hardware-Ressourcen kommen kann.

Tatsächlich brauchbare Möglichkeiten sind die vollständige Virtualisierung, die Paravirtualisierung und die Hardware-Virtualisierung. Wird in der Literatur von Virtualisierung gesprochen ist typischerweise implizit die vollständige Virtualisierung gemeint, da diese Variante meist zur Anwendung kommt. Es beschreibt die Bereitstellung, respektive Simulation eines gesamten Betriebssystems mit Zugriff auf alle Hardware-Ressourcen. Auf die anderen beiden Varianten soll nicht weiter eingegangen werden, da sie aufgrund von bestimmten Nachteilen gegenüber der vollständigen Virtualisierung in der Praxis kaum Anwendungsfälle haben.

Virtuelle Maschinen

Das Konzept der virtuellen Maschine ist die geläufige Technologie, um virtuelle Umgebungen auf einem Hardwaresystem zu betreiben. Die Basis ist das reale Hardwaresystem (Personal Computer oder Server) mit Prozessor, Hauptspeicher, Massenspeicher und diverser weiterer Peripherie. Je nach Zugriffsrechten einer Applikation können von dieser in bestimmten Adressräumen Daten ausgetauscht und Ereignisnachrichten übermittelt werden.

Auf dem Hardwaresystem wird das Gastgeber- bzw. Hauptbetriebssystem ausgeführt, das direkten Zugriff auf die Hardware hat. Im Hauptbetriebssystem wird eine virtualisierende Software als Benutzerprogramm ausgeführt, die man als Virtual Machine Monitor (VMM) oder Hypervisor bezeichnet. Bei einem solchen Hypervisor handelt es sich um einen Typ-2-Hypervisor bzw. Hosted Hypervisor, da er auf dem Hauptbetriebssystem aufsetzt und über dessen Gerätetreiber auf die Hardware-Ressourcen zugreift; das Gegenstück ist der Typ-1-Hypervisor bzw. Native Hypervisor, der als Metabetriebssystem mit eigenen Gerätetreibern direkt auf der Hardware aufsetzt und diese virtuell dupliziert, um sie mehreren Gastbetriebssystemen zur Verfügung zu stellen (vgl. Abbildung). [11](#)

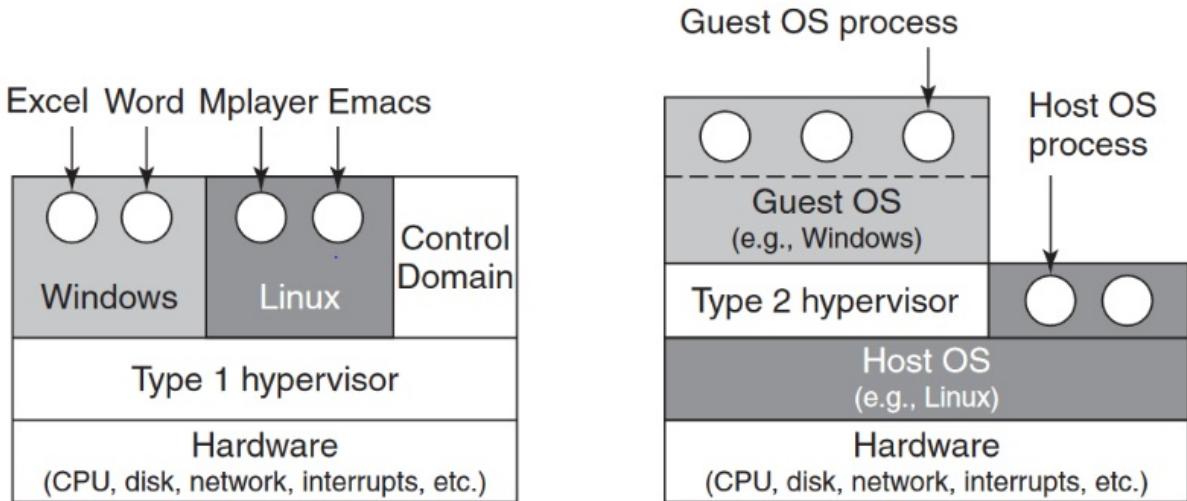


Abbildung: Unterschied zwischen einem Typ-1- und einem Typ-2-Hypervisor [11](#)

Gebräuchlich ist die Virtualisierung mittels Typ-2-Hypervisor (im Folgenden Hypervisor), da dieser betriebssystemspezifisch ist und dadurch eine Virtualisierung auf verschiedenen Hardware-Systemen möglich ist, sofern das Hauptbetriebssystem über die Gerätetreiber verfügt (was bei etablierten Betriebssystemen gegeben ist).

Durch den Hypervisor wird ein Gastbetriebssystem als Benutzerprogramm gestartet, für das ein vollständiges Hardwaresystem simuliert wird. Ein zu lösendes Problem dabei ist das Ausführen von sensitiven Befehlen durch das Gastbetriebssystem, was letzteres als Benutzerprogramm prinzipiell nicht kann. Zur Lösung des Problems verfügen moderne Prozessoren jedoch über Virtualisierungstechnologien mit speziellen Berechtigungsmodi (Intel Virtualization Technology und AMD Secure Virtual Machine). Neben dem Ring-Konzept als Berechtigungskonzept (Ring-0 für Kernmodule mit vollen Zugriffsrechten, Ring-3 für Benutzerprogramme mit eingeschränkten Privilegien) implementieren die Virtualisierungstechnologien das "trap-and-emulate"-Verfahren, das die zwei Modi "Root Operation" und "Non Root Operation" definiert. Der Hypervisor wird im Modus "Root Operation" und das Gastbetriebssystem im Modus "Non Root Operation" ausgeführt. Mit beiden Modi können Ring-3- als auch Ring-0-Befehle ausgeführt werden. Führt jedoch das Gastbetriebssystem Ring-0-Befehle im "Non Root Operation"-Modus aus, können diese durch den Hypervisor im "Root Operation"-Modus abgefangen und verarbeitet werden, was den sensiblen Bereich schützt. [12](#)

Auf einem Continuous Integration Server können auf diese Art mehrere virtuelle Maschinen ausgeführt werden. Auf dem Hauptbetriebssystem läuft die Server-Applikation als Master, während die Gastbetriebssysteme eine Slave-Applikation ausführen und Kontakt zum Master halten. Der Master kann so Bau- und Testaufgaben auslastungsregelnd an seine Slaves delegieren.

Containerisierung

Eine andere Art der Virtualisierung ist die sogenannte Virtualisierung auf Betriebssystemebene bzw. Containerisierung. Bei dieser Virtualisierungsart wird das Prinzip der Speicheraufteilung in Kernel und User Space genutzt, worüber moderne Betriebssysteme für Personal Computer und Server die Zugriffsverwaltung regeln.

Bei dieser Virtualisierungsart werden mehrere Instanzen des User Space erzeugt, die man Container nennt. Die Trennung ergibt sich durch die Verwendung einzigartiger Namespaces für jeden Container. Innerhalb eines Containers wird eine Applikation zusammen mit allen benötigten Bibliotheken und Binaries ausgeführt. Durch die verschiedenen Namespaces sind die Container voneinander unabhängig und isoliert, wodurch es wiederum für die jeweilig eingebetteten Applikationen den Schein eines eigenen Systems macht. Verwaltet werden die Container im Hauptbetriebssystem durch ein Container-Subsystem.

Gemeinsame Systemgrundlage der Container ist der Kernel des Hauptbetriebssystems. Das bringt die Einschränkung mit sich, dass bei Containerisierung kein anderes Betriebssystem als das des Hosts simuliert werden kann. Unter Linux können jedoch verschiedene Distributionen in Containern auf einem Hardwaresystem ausgeführt werden, sofern diese auf der gleichen Kernelversion beruhen.

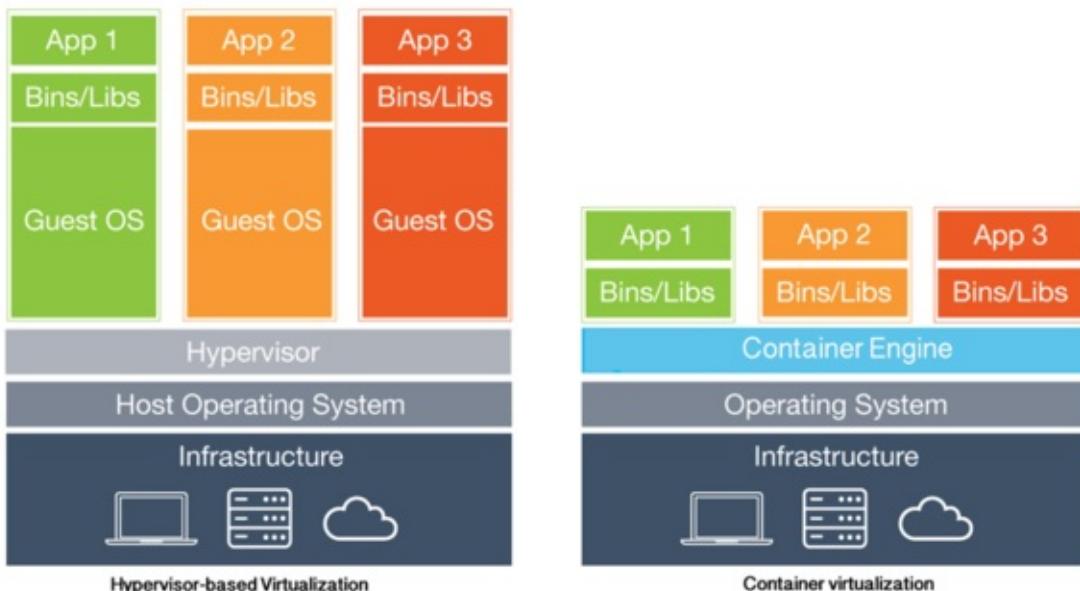


Abbildung: Architektonischer Unterschied zwischen Virtualisierung und Containerisierung [13](#)

Anders als bei einer virtuellen Maschine wird neben dem Hauptbetriebssystem kein weiteres Betriebssystem ausgeführt, weshalb es keinen Hypervisor braucht (vgl. Abbildung). Die verschiedenen Größen der einzelnen Komponenten in der Abbildung verdeutlichen den ungleich großen Ressourcenbedarf zwischen Virtualisierung- und Container-Frameworks.

Gegenüberstellung

Bei der Bewertung der beiden Technologien sind Konfigurierbarkeit, Simulationsfähigkeiten und Ressourcenbedarf maßgebend. Die Konfigurationsmechanismen bei der Containerisierung sind sehr dynamisch und flexibel durch dedizierte Orchestrierung verschiedener Container. Es können einzelne gekapselte Container entfernt und andere hinzugefügt werden, was am Ende ein neues lauffähiges System ergibt. Virtuelle Maschinen müssen hingegen stets als Ganzes rekonfiguriert und auch wieder verteilt und installiert werden. Hier ist die Containerisierung sehr attraktiv, was der Hauptgrund der wachsenden Anwendung in den letzten Jahren ist.

Hinsichtlich Simulationsfähigkeiten können mit virtuellen Maschinen verschiedene Gastbetriebssysteme beliebigen Typs auf einem Hauptbetriebssystem ausgeführt werden. Container sind hingegen an die systemische Umgebung des Hauptbetriebssystems und dessen Gerätetreiber und Kernelmodule gebunden. Andere Betriebssysteme oder andere Versionen des Hauptbetriebssystems können in Containern nicht simuliert werden.

Beim Ressourcenbedarf liegt der Vorteil bei der Containerisierung, da Container nativ Systembefehle ausführen können. Es braucht kein separates Betriebssystem, was einen Hypervisor erübriggt und einen effizienteren Umgang mit den verfügbaren Ressourcen mit sich bringt. Hinzu kommt, dass virtuelle Maschinen immer ein gesamtes Betriebssystem beinhalten und mindestens mehrere Hundert Megabyte groß sind, während Container teilweise nur wenige Megabyte umfassen. Das macht zusammen mit der dynamischen Konfigurierbarkeit die Verteilung besonders schnell und einfach.

Solange also nicht der Bedarf eines anderen Kernels als der des Hauptbetriebssystems benötigt wird, ist die Containerisierung im Vergleich zur klassischen Virtualisierung im Vorteil. Letztere ist bei der Simulation von verschiedenen Systemen auf einer Hardware weiterhin die einzige Möglichkeit.

Implementierungen und Standards

Im kommerziellen Umfeld sind die Virtualisierungsprodukte der Firma VMware mit leistungsfähigen Hypervisors für virtuelle Maschinen auf Personal Computern und Servern marktführend [14](#). Als einfache und zuverlässige Open Source-Alternative für Personal Computer erfreut sich der Hypervisor VirtualBox der Firma Oracle großer Beliebtheit [15](#).

Das erste Werkzeug, mit dem eine Art Containerisierung möglich war, ist das Linux-Programm chroot (change root), das durch das Ändern des Stammverzeichnis' eines Prozess' die Kapselung einer Applikation ermöglicht [16](#). Und obwohl andere Frameworks

wie Solaris Containers oder FreeBSD jail bereits seit Anfang der 2000er Jahre verfügbar sind [17](#) [18](#), wurde die Containerisierung erst 2013 mit dem Release des Frameworks Docker durch die gleichnamige Firma Docker Inc. populär.

Docker setzte sich im Laufe der letzten Jahre sowohl im Open Source als auch im industriellen bzw. kommerziellen Umfeld durch und ist mittlerweile der defacto-Standard (vgl. Abbildung) [19](#).

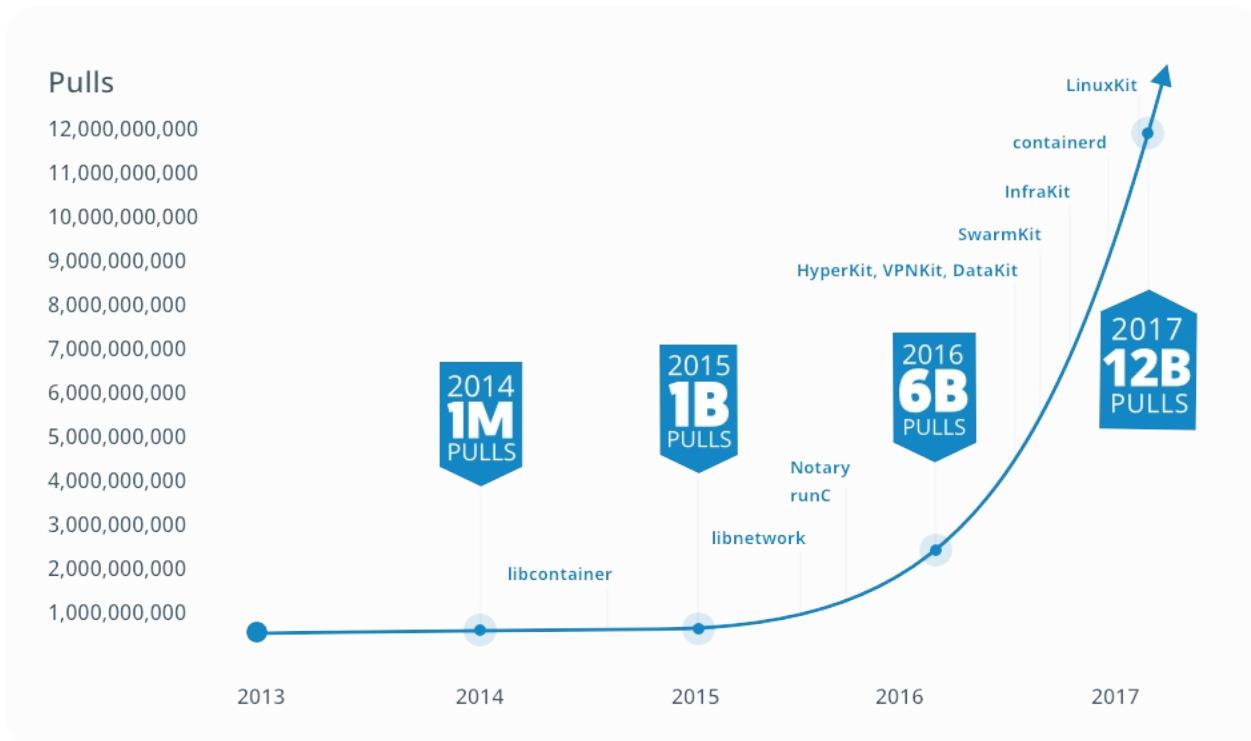
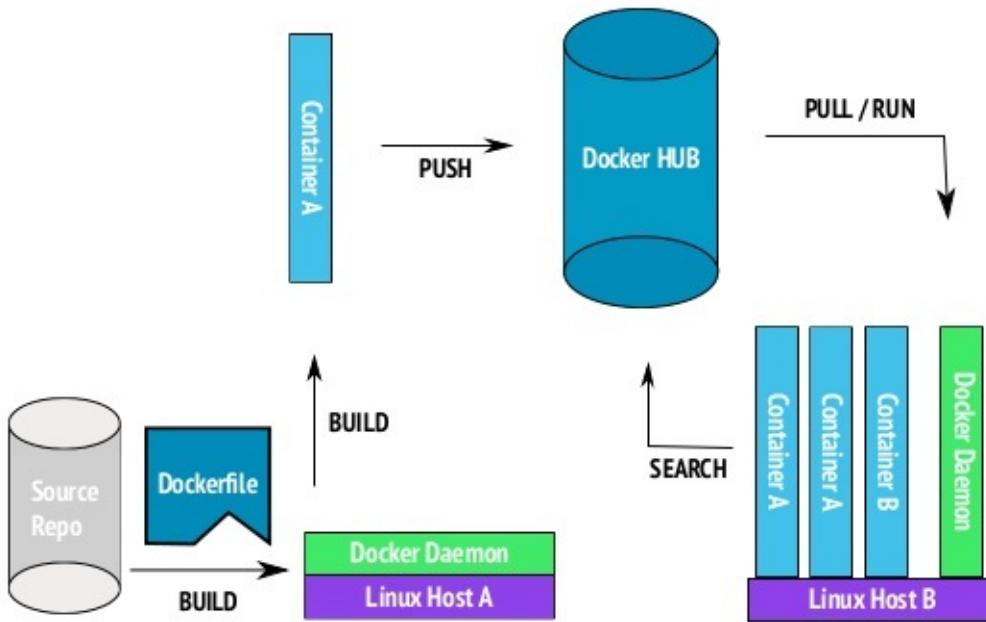


Abbildung: Veranschaulichung des Wachstums des Docker-Ökosystems anhand der steigenden Anzahl an Pulls auf DockHub [19](#)

Das Docker-Framework besitzt mit der Docker-Engine sein eigenes Container-Subsystem zur Ausführung von Containern. Die Implementierung basiert auf der Container-Runtime [containerd](#) [20](#). Die Stärke des Docker-Frameworks liegt in der Verfügbarkeit von etlichen standardisierten Images aller gebräuchlich Systemkomponenten (JRE, .NET, Apache Server, SQL Datenbank u.v.m.), die dynamisch konfiguriert und miteinander verknüpft werden können. Bereitgestellt werden alle Images über die Online-Plattform DockHub, von der die lokale Docker-Applikation bei Bedarf auch automatisch benötigte Images bezieht, um einen Container zusammenzubauen. Neue standardisierte Images können von jedermann implementiert und über DockHub angeboten werden. Die Konfiguration eines Containers erfolgt entweder aus der Konsole heraus über den parametrisierten Startbefehl oder mit einem Konfigurationsskript in YAML über das separate Werkzeug Docker-Compose (vgl. Abbildung). Letzteres macht die Verwendung im Kontext von CI-Servern sehr attraktiv.

Docker: Workflow



Milano - 2015 / 03 / 11

Abbildung: Erzeugung eines konfigurierten Containers mit Docker-Compose 21

Build Server

von Lukas Taake

Build Server sind ein zentraler Bestandteil im Continuous Software Engineering. Sie stellen einen zentralen Ort dar, an dem der aktuelle Stand der Software, z.B. in Form von Testergebnissen und kompilierten Artefakten, einsehbar ist.

Ein großer Vorteil ist, dass alle Prozesse innerhalb eines Build Server vollautomatisch ablaufen und somit Fehler durch falsche, ausgelassene oder vertauschte Prozess-Schritte ausgeschlossen werden können. Darüber hinaus stellt ein zentraler Build Server eine isolierte Umgebung dar, sodass es nicht dazu kommen kann, dass die Software aufgrund unterschiedlicher Konfiguration auf einem Entwickler-Rechner funktioniert (bzw. Tests erfolgreich sind) und auf einem anderen nicht. Wenn also der Build-Server erfolgreich ist und ein Entwickler-Rechner nicht, muss es sich um eine Fehlkonfiguration des Entwicklers handeln, da die des Build-Servers im Optimalfall möglichst nah an der eines Produktivsystems liegt.

Über das Kompilieren und Testen hinaus werden Build-Server auch zur Durchführung von Continuous Delivery und Deployment verwendet. Oftmals existieren bereits Build-Server-Plugins zur Unterstützung verbreiteter Deployment-Ziele.

Die konkrete Funktionsweise soll Beispielhaft am Beispiel von Jenkins und Travis CI dargestellt werden, wobei Jenkins ein Open Source Projekt ist und weitreichende Konfigurationsmöglichkeiten bietet, während es sich bei Travis um eine SaaS handelt.

Jenkins

Jenkins ist eins der populärsten CI-Tools, vor allem im Java-Umfeld. Üblicherweise wird es selbst gehostet und ist tiefgreifend konfigurierbar. Es unterstützt verschiedene Build-Tools, wie Ant, Maven, Gradle oder MSBuild, außerdem Versionskontrollsysteme, wie Git, Mercurial oder SVN. Darüber hinaus gibt es eine Vielzahl an Plugin, über die es um neue Funktionalität erweitert werden kann. (vgl. [9])

Gewöhnliche Build-Jobs können zwar aus mehreren auszuführenden Schritten bestehen, liefern als Ergebnis allerdings nur das Gesamtergebnis (und evtl. einen Test-Report) zurück. Im Gegensatz dazu gibt es auch Pipeline-Jobs, bei denen ein Job explizit in mehrere Stufen (Stages) unterteilt wird (z.B. Compile, Unit Test, Integration Test). Dann wird der Status jeder einzelnen Stufe ausgegeben, wenn also ein Integrationstest fehlschlägt, ist das sofort ersichtlich (vgl. Abbildung).

Stage View

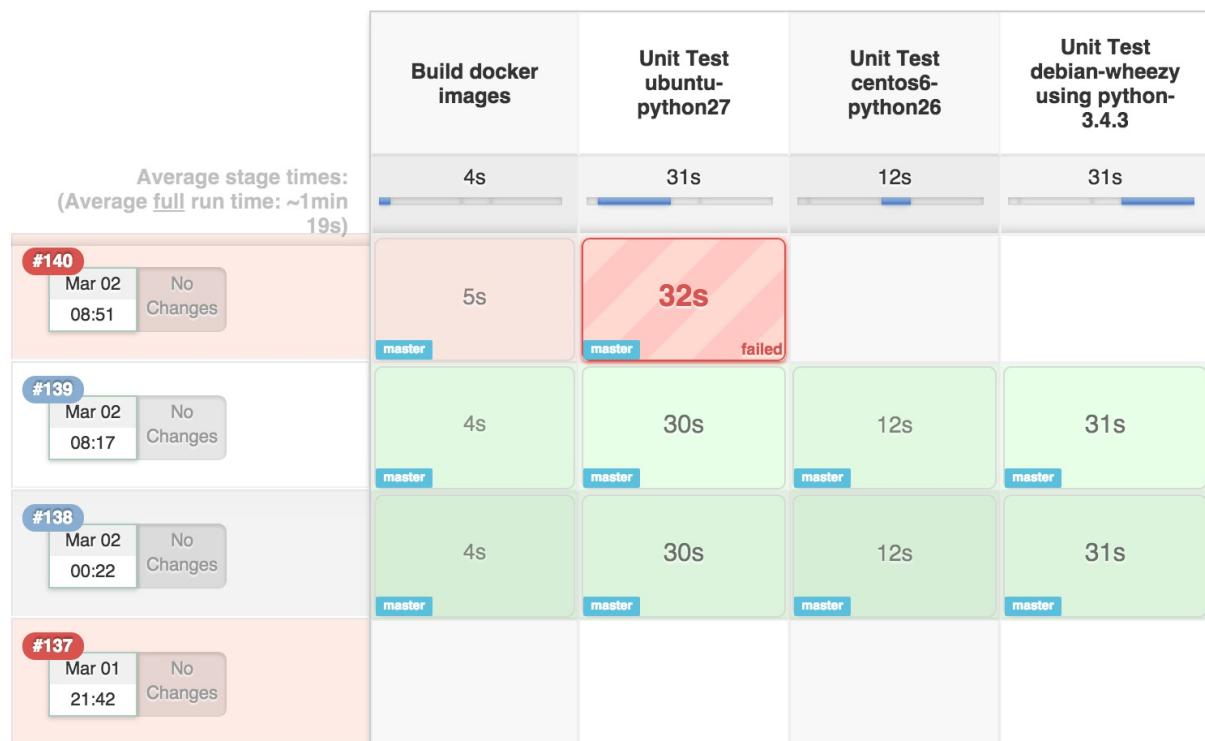


Abbildung: Übersicht der Build-Historie mit einzelnen Stages. Quelle: [Jenkins Wiki, Pipeline Stage View Plugin](#)

In komplexen Setups, oder wenn viele Projekte zentral verwaltet werden, ist es möglich, einzelne Build-Jobs auf Slave-Knoten zu verteilen. In Pipeline-Jobs ist es sogar möglich, verschiedene Stages auf verschiedenen Slaves ausführen zu lassen, um die Geschwindigkeit zu steigern.

Travis CI

Travis CI verfolgt einen anderen Ansatz als Jenkins, indem es ausschließlich als Software as a Service angeboten wird und als Versionskontrollsystem ausschließlich Github-Repositories akzeptiert. Dafür ist als Ausgleich zur dieser Einschränkung die Integration zu Github sehr komfortabel. So können Build Jobs für neue Repositories mit einem Maus-Klick erstellt werden und die Konfiguration geschieht über eine YAML-Datei (siehe Listing).

```
language: java

jdk: oraclejdk8

services:
- mysql

before_install:
- mysql -e 'CREATE DATABASE newsboard'
- mysql newsboard < src/main/resources/sql/create_script.sql
- mysql newsboard < src/main/resources/sql/example_data.sql

install: mvn install -DskipTests=true -Dmaven.javadoc.skip=true
script: mvn verify -Dspring.profiles.active=travis
```

Listing: Beispielhafte Travis CI Build Definition

Jede Ausführung eines Build-Jobs geschieht auf einer neu erstellten virtuellen Maschine um absolute Isolation sicherzustellen. Über die Angabe der Programmiersprache in der Konfiguration wird bestimmt, wie die Maschine vorkonfiguriert ist. Darüber hinaus können mit `services` auch Abhängigkeiten, z.B. Datenbanken, installiert werden und mit `before_install` initialisiert werden. Mit `install` und `script` werden die Befehle zum Kompilieren und Ausführen der Tests angegeben.

Auf den ersten Blick ähnlich dem Konzept der Pipeline bei Jenkins sind die Build Stages bei Travis. Die Verwendung ist allerdings im Gegensatz zu Jenkins Pipelines ausschließlich sinnvoll, um unterschiedliche zeitaufwändige Teststufen parallel ablaufen zu lassen. Es wird nämlich für jede einzelne Stage eine eigene virtuelle Maschine erstellt und die Software dort kompiliert. Um also z.B. Unit- und Integrationstests zu parallelisieren, sind sie nicht unbedingt geeignet. Darüber hinaus ist es nicht möglich, ohne zuhilfenahme externer Dienste, Daten (z.B. Test Reports) zwischen den einzelnen Stages auszutauschen. Eine Gesamt-Coverage unter Berücksichtigung aller Teststufen wäre so nicht mehr möglich. (vgl. [8])

Quellen

- [1] Test-driven development; https://en.wikipedia.org/wiki/Test-driven_development
- [2] Acceptance test-driven development;
https://en.wikipedia.org/wiki/Acceptance_test%20%93driven_development
- [3] Behavior-driven development; https://en.wikipedia.org/wiki/Behavior-driven_development
- [4] The Product Managers' Guide to Continuous Delivery and DevOps;
<http://www.mindtheproduct.com/2016/02/what-the-hell-are-ci-cd-and-devops-a-cheatsheet-for-the-rest-of-us/>
- [5] What is the difference between Continuous Integration, Continuous Deployment & Continuous Delivery?; <https://codeship.com/continuous-integration-essentials>
- [6] The Future of Programming, Robert C. Martin; <https://youtu.be/eclWPzGEbFc>
- [7] Kanban (Software Entwicklung);
[https://de.wikipedia.org/wiki/Kanban_\(Softwareentwicklung\)](https://de.wikipedia.org/wiki/Kanban_(Softwareentwicklung))
- [8] Build Stages - Travis CI; <https://docs.travis-ci.com/user/build-stages/>
- [9] Jenkins (Software); [https://de.wikipedia.org/wiki/Jenkins_\(Software\)](https://de.wikipedia.org/wiki/Jenkins_(Software)) [10] Jan Bosch, "Continuous Software Engineering", Springer International Publishing, Switzerland, 2014, Link: <http://www.springer.com/us/book/9783319112824> (Zugriff am 24.06.2017) [11] Andrew S. Tanenbaum, "Modern operating systems", Boston, 2015 [12] Gerald J. Popek, "Formal Requirements for Virtualizable Third Generation Architectures", University of California, Los Angeles, 1974 [13] Chenxi Wang, InfoWorld (IDG Communications, Inc.), 2017, Link: <http://www.infoworld.com/article/3204171/linux/what-is-docker-linux-containers-explained.html> (Zugriff am 26.06.2017) [14] Contel Bradford, "Virtualization Wars: VMware vs. Hyper V: Which is Right For Your Virtual Environment?", StorageCraft Technology Corporation, 2015, Link: <http://www.storagecraft.com/blog/virtualization-wars-vmware-vs-hyper-v-which-is-right-for-your-virtual-environment/> (Zugriff am 02.17.2017) [15] Heise Medien GmbH & Co. KG, 2017, Link: <https://www.heise.de/download/product/virtualbox-40385> (Zugriff am 02.07.2017) [16] Scott Hogg, "Software Containers: Used More Frequently than Most Realize", Network World (IDG Communications, Inc.), 2014, Link: <http://www.networkworld.com/article/2226996/cisco-subnet/software-containers--used-more-frequently-than-most-realize.html> (Zugriff am 28.06.2017) [17] Mike DeGraw-Bertsch, "FreeBSD jails", BSD DecCenter (O'Reilly Media, Inc.), 2003, Link: <http://www.onlamp.com/pub/a/bsd/2003/09/04/jails.html> (Zugriff am 28.06.2017) [18] Tucker Andrew G. et al., "Global visibility controls for operating system partitions", European Patent Office, 2005, Link: https://worldwide.espacenet.com/publicationDetails/biblio?FT=D&date=20050127&DB=&locale=en_EP&CC=US&NR=2005021788A1&KC=A1&ND=1 (Zugriff am 28.06.2017) [19] "What is a Container", Docker Inc., 2017, Link: <https://www.docker.com/what-container> (Zugriff am 01.07.2017) [20] "About Containerd",

Docker Inc., 2016, Link: <https://containerd.io> (Zugriff am 02.07.2017) [21] "Docker",
f2informatica, Milano, 2015, Link: <https://www.slideshare.net/Pokerone/docker-45628208>
(Zugriff am 01.07.2017)

AI

Teilbereich(e): Data Mining

Projektleiter:

Projektteam: Daniel Beneker, Sven Schirmer, Yannick Kloss

Github-Repo: [Twitter Miner](#)

Übersicht Data Mining Algorithmen

Die Datamining Algorithmen lassen sich grob in vier Kategorien untergliedern (vgl. Runkler S. 3):

- Klassifikation
- Clusteranalyse
- Korellationsanalyse
- Regressionsanalyse

Klassifikation

Klassifikation setzt man grundsätzlich ein, wenn man einen gelabelten Datensatz zur Verfügung hat. Gelabelt heißt in diesem Kontext, dass jedem Tupel eine Klasse zugeordnet wurde und das Modell anhand dieser Information trainieren kann. In der englischsprachigen Literatur ist dies unter supervised learning bekannt. Die Klassifikation bietet den großen Vorteil, dass eine Überprüfung des Algorithmus anhand der gelabelten Testdaten stattfinden kann. Hierfür wird der Quelldatensatz aufgeteilt; zum einen in einen Lerndatensatz für das Training des Modells und zum anderen in einen Testdatensatz für die spätere Evaluierung des Algorithmus.

Einige bekannte Algorithmen: (teilweise entnommen aus Tan)

- Naive Bayes
- Decision Tree
- Decision Forest
- Rule-Based
- Support vector machine
- Neural network

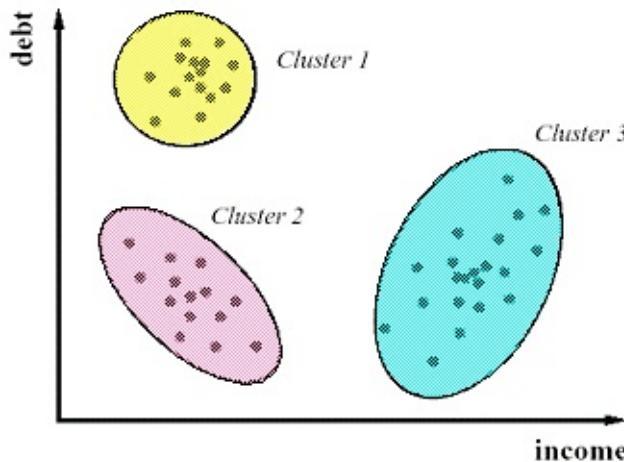
Die Algorithmen Naive Bayes, Decisiontree und Support Vector Machine sind in den folgenden Kapiteln detailliert beschrieben. In unserem Softwareprojekt der Twitter-Text-Analyse ist ein gelabelter Datensatz vorhanden, sodass das Projektteam die Klassifikation mit den drei genannten Algorithmen vornehmen kann.

Clusteranalyse

Die Clusteranalyse ist ein unsupervised learner. Beim unsupervised learning besitzen die Quelldaten kein Label. In den Quelldaten werden Ähnlichkeiten und Muster zwischen den Tupeln gesucht und die Tupel in Cluster aufgeteilt. Die Tupel sollen in ihrem Cluster

möglichst ähnlich sein und verschieden zu den Tupeln in den anderen Clustern. Somit ist auch eine Evaluierung möglich: denn ein gutes Clustering hat eine hohe Separation zwischen den einzelnen Clustern. (vgl. Cichosz S. 15)

Eine Beispielvisualisierung mit den Attributen *debt* und *income*.



Entnommen aus <https://www.analyticsvidhya.com/wp-content/uploads/2013/11/Clust1.gif>.

Einige bekannte Algorithmen:

- k-means
- k-medoids
- fuzzy c-Mean

Korrelationsanalyse

Die Korrelationsanalyse untersucht die Korrelation (Abhängigkeit) zwischen den Attributen über alle Daten. Die Stärke wird durch den Korrelationskoeffizienten ausgedrückt. Das Ergebnis der Korrelationsanalyse ist meistens eine Korrelationsmatrix, welche alle Korrelationskoeffizienten der Attribute zueinander enthält.

Attributes	Passen...	Name	Sex	Age	No of Si...	No of P...	Ticket N...	Passen...	Cabin	Port of ...	Life Boat	Survived
Passenger Class	1	0.898	0.125	-0.408	0.061	0.018	0.884	-0.559	0.546	0.038	0.431	0.312
Name	0.898	1	0.125	-0.344	0.066	0.004	0.989	-0.481	0.732	0.054	0.400	0.294
Sex	0.125	0.125	1	0.064	-0.110	-0.213	0.122	-0.186	0.060	-0.122	0.055	0.529
Age	-0.408	-0.344	0.064	1	-0.244	-0.151	-0.336	0.179	-0.145	0.049	-0.120	0.056
No of Siblings or Spouses on Board	0.061	0.066	-0.110	-0.244	1	0.374	0.073	0.160	-0.065	-0.074	-0.088	0.028
No of Parents or Children on Board	0.018	0.004	-0.213	-0.151	0.374	1	0.002	0.222	-0.002	-0.096	-0.112	-0.083
Ticket Number	0.884	0.989	0.122	-0.336	0.073	0.002	1	-0.497	0.638	0.059	0.410	0.291
Passenger Fare	-0.559	-0.481	-0.186	0.179	0.160	0.222	-0.497	1	-0.224	0.062	-0.370	-0.244
Cabin	0.546	0.732	0.060	-0.145	-0.065	-0.002	0.638	-0.224	1	0.031	0.090	0.095
Port of Embarkation	0.038	0.054	-0.122	0.049	-0.074	-0.096	0.059	0.062	0.031	1	0.170	-0.100
Life Boat	0.431	0.400	0.055	-0.120	-0.088	-0.112	0.410	-0.370	0.090	0.170	1	0.013
Survived	0.312	0.294	0.529	0.056	0.028	-0.083	0.291	-0.244	0.095	-0.100	0.013	1

Diese Korrelationsmatrix wurde mithilfe von Rapidminer erstellt und zeigt die Koeffizienten des "Titanic-Datensatzes". Der Datensatz ist zu finden unter <https://www.kaggle.com/c/titanic/data>. Auffällig ist die Korrelation zwischen "Sex" und "Survived".

Regressionsanalyse

Die Regressionsanalyse lässt sich am ehesten mit der Klassifizierung vergleichen. Doch hier wird keine diskrete Klasse vorhergesagt, sondern ein numerischer Wert. Dieser Wert wird durch die Regression bestimmt. (vgl. Cichosz S. 14)

Quellen

- Thomas A. Runkler: Data Mining - Modelle und Algorithmen intelligenter Datenanalyse; 2. Auflage; Springer; 2015
 - Tan, Steinbach, Kumar: Data Mining - Classification: Basic Concepts, Decision Trees, and Model Evaluation; 2004; https://www-users.cs.umn.edu/~kumar/dmbook/dmslides/chap4_basic_classification.pdf Aufgerufen am 01.07.2017.
 - Pawet Cichosz: Data Mining Algorithms - Explained Using R; Wiley; 2015; <http://pdf.th7.cn/down/files/1502/Data%20Mining%20Algorithms.pdf> Aufgerufen am 01.07.2017.
-

Author: Sven Schirmer

Knowledge Discovery in Databases (KDD)

Das Volumen der jährlich generierten digitalen Datenmenge ist in den letzten Jahren massiv angewachsen. Doch nur eine geringe Menge dieser Daten wird für Analysen genutzt.^[5] *Knowledge Discovery in Databases* beschreibt das generelle Vorgehen, um genau diese Daten zu nutzen, dass heißt Wissen zu generieren.

Definition

Der Begriff *Knowledge Discovery in Databases* wird in der Literatur unterschiedlich verwendet. Es kommt vor, dass es als Synonym zu dem Begriff *Data Mining* genutzt.^[4] Üblicherweise beschreibt *Knowledge Discovery in Databases* aber einen Prozess in dem *Data Mining* lediglich einen Teilschritt darstellt.^[2]

Das *Gabler Wirtschaftslexikon* definiert KDD wie folgt:

Knowledge-Discovery-in-Databases (KDD)-Prozess; umfassender Datenanalyseprozess, in dessen Kern Verfahren des Data Mining zur Anwendung kommen. Der-Knowledge-Discovery-in-Databases (KDD)-Prozess umfasst folgende Phasen:[...]^[3]

Hung (2009) beschreibt KDD als einen:

[...] nichttrivialer mehrstufiger Prozess der Wissensfindung aus vorhandenen Informationen. KDD-Prozess umfasst alle Schritte, von woher die Daten abgeholt werden, über Vorverarbeitung und eigentliche Verarbeitung zur Informationsgewinnung (Data-Mining-Schritt), bis hin wie die Endinformation interpretiert und dargestellt wird.^[2]

KKD-Prozess

Fayyad et al. beschrieben 1996 erstmals die einzelnen Schritte, die bei der *Knowledge Discovery in Databases* genutzt werden. Dazu entwickelten sie ein Modell, welches heutzutage oft unter den Begriffen "KDD Prozess" oder "Schritte des KDD" zu finden sind.

Nach Fayyad et al. besteht KDD aus den folgenden neun Schritten:^[1]

1. Problemabgrenzung
2. Auswahl der Daten
3. Datenvorverarbeitung
4. Datenreduktion und Kodierung
5. Auswahl der Data Mining Methode

6. Auswahl des Data Mining Algorithmus
7. Data Mining
8. Interpretation der Ergebnisse
9. Anwendung des gefundenen Wissens

Das *Fayyad Modell des KDD Prozesses* ist in Abbildung 1 dargestellt. Es handelt sich im ein iteratives Modell, dass bedeutet, die Schritte können mehrfach durchlaufen werden. Nach der Evaluation der Ergebnisse müssen eventuell einige Schritte neu durchlaufen werden, um das Ergebnisse in der Evaluation zu verbessern. So kann es beispielsweise sein, dass in der Evaluation auffällt, dass bei der Vorverarbeitung nicht alle unerwünschten Daten herausgefiltert wurden. [1][2]

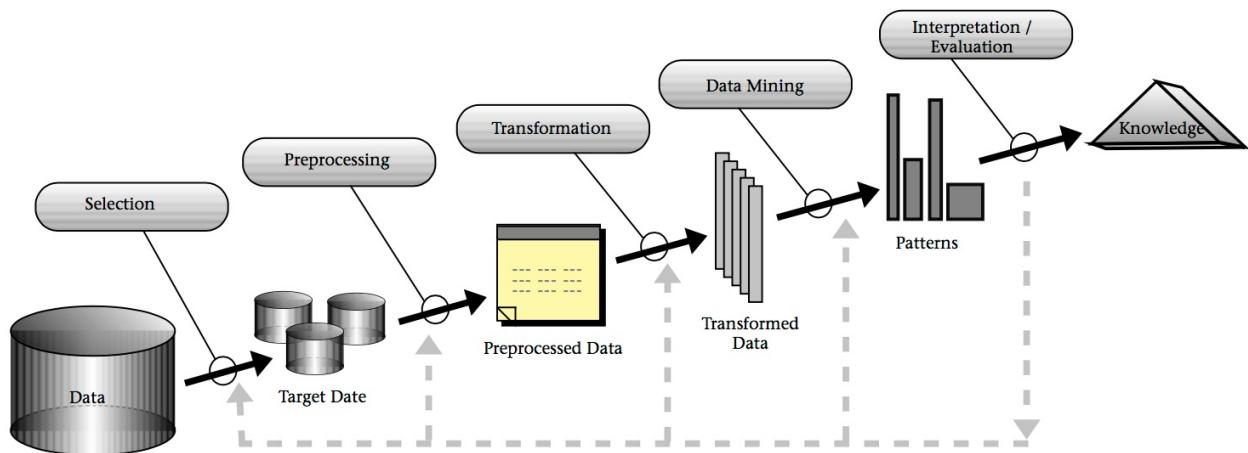


Abbildung 1: Fayyad Modell des KDD Prozesses [1]

Die Schritte im Detail

1. Problemabgrenzung: Im ersten Schritt muss zunächst das Ziel aus der Sicht des Kunden identifiziert werden. Außerdem ist es wichtig sich mit dem jeweiligen Fachbereich vertraut zu machen. [1]

2. Auswahl der Daten: Im zweiten Schritt wird ein Datensatz ausgewählt. Das Ziel aus Schritt eins sollte dabei beachtet werden, denn das erreichen des Ziels hängt von der Qualität des Datensatzes ab. [1]

3. Datenvorverarbeitung: Im dritten Schritt werden die Daten vorverarbeitet. Unerwünschte Informationen, die die Ergebnisse des Data Minings negativ beeinflussen würden, sollten herausgefiltert werden. In diesem Schritt sollte auch entschieden werden, wie mit fehlenden/unvollständigen Datenfeldern umgegangen wird. [1]

4. Datenreduktion und Kodierung: Im vierten Schritt werden die Daten in eine andere Form überführt, damit Data Mining Algorithmen sie verstehen und verarbeiten können. Dieses Verfahren wird oft auch "Kodierung", "Datenprojektion" oder "Datentransformation"

genannt. Bei der Datentransformation wird i.d.R. auch eine Datenreduktion durchgeführt, bei der Daten mit geringer Aussagekraft entfernt werden. [1]

5. Auswahl der Data Mining Methode: Im fünften Schritt wird mit Hilfe des Ziels aus Schritt 1 eine bestimmte Data Mining Methode ausgewählt. Bekannte Methoden sind beispielsweise die Klassifikation oder die Clusteranalyse. [1]

6. Auswahl des Data Mining Algorithmus: Im sechsten Schritt wird innerhalb der zuvor gewählten Data Mining Methode ein Algorithmus ausgewählt. Zudem wird entschieden mit welchen Parametern dieser Algorithmus am besten aufgerufen wird. [1]

7. Data Mining: Der siebte Schritt ist das eigentliche Data Mining. [1]

8. Interpretation der Ergebnisse: Im achten Schritt werden die Ergebnisse des Data Minings interpretiert. Dies kann auch eine Visualisierung umfassen. [1]

9. Anwendung des gefundenen Wissens: Im neunten Schritt werden die Ergebnisse angewendet. Das bedeutet beispielsweise das Wissen in ein anderes System weiter zu geben, oder auch einfach eine Dokumentation der Ergebnisse anzufertigen. Weiter sollte auf Konflikte zu dem bisherigen Wissensstand geprüft werden. [1]

Einordnung dieses Projektes in den KDD Prozess

Dieses Projekt "Twitter-Miner" durchläuft alle Schritte des KDD Prozesses. In diesem Kapitel soll ein kurzer Überblick gegeben werden, was in diesem Projekt in den jeweiligen Schritten gemacht wurde und wo weitere Informationen zu finden sind.

1. Problemabgrenzung: Die Problemabgrenzung und Definition des Ziels wurde u.a. im Pflichtenheft festgelegt. Ziel ist die Stimmungsanalyse von Tweets (=Sentiment Analysis).

2. Auswahl der Daten: Die Datenbasis dieses Projektes bilden Tweets. Zu einem bestimmten Hashtag werden über eine Schnittstelle Tweets geladen.

3. Datenvorverarbeitung: Tweets können viele unerwünschte Informationen enthalten wie beispielsweise Links, die in diesem Schritt entfernt werden. Weiteres zum Thema Datenvorverarbeitung ist [hier](#) zu finden.

4. Datenreduktion und Kodierung: Um die Tweets als Eingabe für unsere Data-Mining-Algorithmen zu nutzen wurden sie in entsprechende Zahlenrepräsentationen umgewandelt. Innerhalb dieses Projektes wurden dafür sogenannte *tf-idf Vektoren* genutzt. Dieses Thema wird ebenfalls in [diesem Kapitel](#) beschrieben.

5. Auswahl der Data Mining Methode: Das Ziel aus Schritt 1 (Sentimentanalyse) lässt sich mit der Data Mining Methode "Klassifikation" erreichen. Klassifikationsalgorithmen können bestimmen, ob ein Merkmal zu einer bestimmten Klasse gehört oder nicht. Das heißt bei der

Sentimentanalyse gibt der Klassifikationsalgorithmus aus, ob es sich um einen positiven Tweet oder einen negativen Tweet handelt.

Eine generelle Übersicht der Data Mining Methoden ist [hier](#) aufgelistet.

6. Auswahl des Data Mining Algorithmus: Innerhalb dieses Projektes werden die drei folgenden Klassifikationsalgorithmen genutzt.

- Bayes
- Support Vector Machine
- Decision Tree

7. Data Mining: In diesem Schritt wird ein Datensatz auf mehrere Tweets mit den Algorithmen aus Schritt 6 klassifiziert.

8. Interpretation der Ergebnisse: Die Ausgabe der Algorithmen wird in diesem Schritt interpretiert. Ist die Ausgabe beispielsweise [0,1], so ist der entsprechende Tweet positiv, ist die Ausgabe [1,0], so ist er negativ.

9. Anwendung des gefundenen Wissens: Die Anwendung des gefundenen Wissens erfolgt in einer Präsentation durch eine Weboberfläche.

1. Fayyad, U., Piatetsky-Shapiro, G., & Smyth, P. (1996). From Data Mining to Knowledge Discovery in Databases. *AI Magazine*, 17(3), 37. [↔↔↔](#)
 2. Hung, P. T. (2009). Data-Mining und Knowledge Discovery in Databases (KDD) Ein Überblick. Dresden. Retrieved from https://www.inf.tu-dresden.de/content/institutes/iai/tis-neu/lehre/archiv/folien.ws_2008/Vortrag_Hung.pdf [↔↔↔](#)
 3. Springer Gabler Verlag (Herausgeber), Gabler Wirtschaftslexikon, Stichwort: Knowledge Discovery in Databases (KDD), online im Internet:
<http://wirtschaftslexikon.gabler.de/Archiv/75635/knowledge-discovery-in-databases-v10.html>
↔
 4. Alpaydın, E. (2014). Introduction to machine learning. Methods in Molecular Biology (Second Edi, Vol. 1107). The MIT Press. [↔](#)
 5. Gantz, John und David Reinsel (2012). IDC IVIEW: THE DIGITAL UNIVERSE IN 2020: Big Data, Bigger Digital Shadows, and Biggest Growth in the Far East. Techn. Ber. URL: <http://www.emc.com/collateral/analyst-reports/idc-the-digital-universe-in-2020.pdf> [↔](#)
-

Embedded Computing

Teilbereich(e): Smart Home, IoT

Projektleiter: Gamze Söylev Öktem

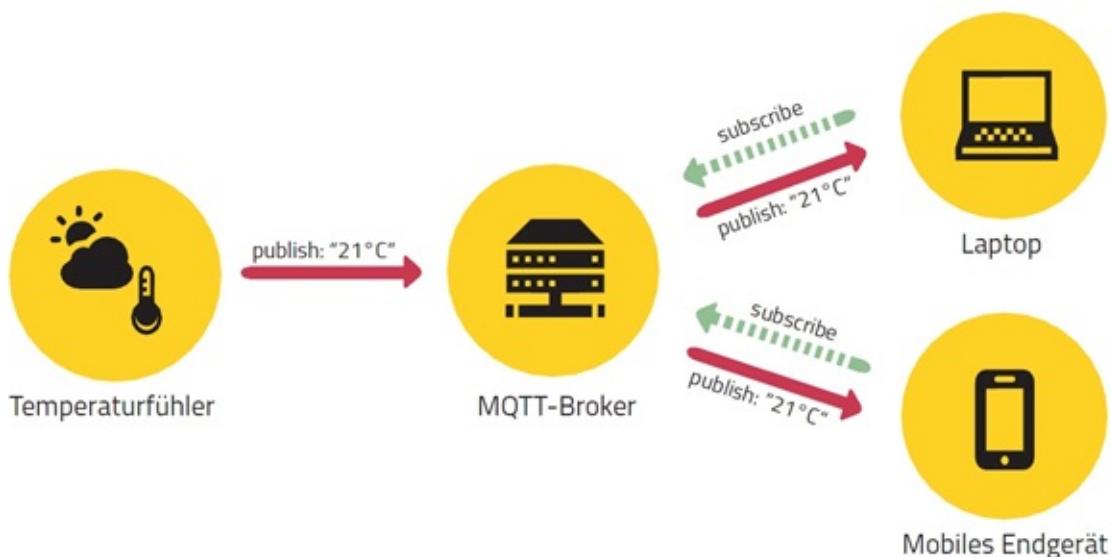
Projektteam: Nils Kohlmeier, Wladimir Streck, Gamze Söylev Öktem, Jonas Wiese, Justin Jagieniak

Github-Repo: <https://github.com/nkohlmeier/Spezielle-Gebiete-zum-Softwareengineering.git>

MQTT

Struktur & Nutzen

Das Message Queue Telemetry Transport Protokoll (MQTT) ist ein schlankes und leichtgewichtiges Kommunikationsprotokoll, dass auf dem Publish-Subscribe Prinzip basiert. Es wurde 1999 als Kommunikationsprotokoll zwischen Maschinen entwickelt. Besonders wichtig war den Entwicklern der geringe Protokoll-Overhead, sodass das Protokoll auch von Low-Energy Geräten ohne viel Prozessorzeit verarbeitet werden kann. Die „Organization for the Advancement of Structured Information Standards“ (OASIS) standardisiert seit 2013 MQTT als das Protokoll des Internets der Dinge [5](#). Zur Kommunikation zwischen zwei Clients ist ein Broker (Server) nötig. Beide Clients melden sich beim Broker an und teilen diesem mit, welche „Topics“ sie abonnieren möchten. Ein Topic wird mit einer Zeichenkette beschrieben und definiert den Kommunikationskanal. Der Broker sorgt dafür, dass die von Client A auf einem Topic gesendeten Daten, an alle Abonnenten dieses Topics zugestellt werden. Hierbei ist besonders zu beachten, dass auch viele Konsumenten und Produzenten das gleiche Topic verwenden können. MQTT eignet sich deshalb besonders als „Many-to-Many“ Kommunikationsprotokoll [6](#). Meistens wird zum Nachrichtenaustausch MQTT über TCP übertragen. Dies ist aber nicht vorgeschrieben, sodass auch andere Protokolle genutzt werden können [4](#). Das untenstehende Bild beschreibt den beschriebenen Ablauf mit einem Beispiel. Ein Temperaturfühler misst die Temperatur und sendet diesen Wert über ein Topic (z.B. „/temperatur“) an den Broker. Der Broker verteilt die Nachricht an alle Clients, die das Topic abonniert haben.

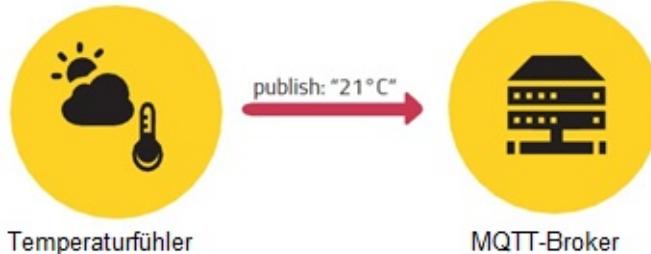


Die Konsumenten und Produzenten sind aber nicht fest an ihre Rollen gebunden, sondern können auch die jeweils andere Rolle übernehmen.

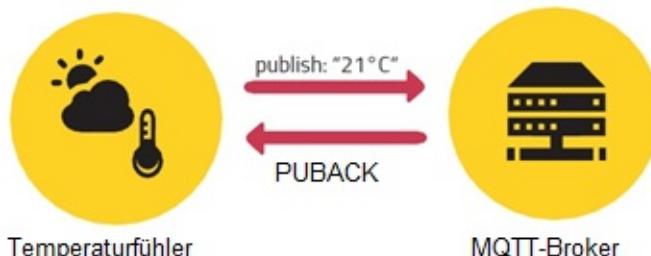
QoS

Da MQTT nicht zwangsläufig über TCP übertragen werden muss, das unterliegende Protokoll also nicht ein Erhalt einer Nachricht garantieren muss, definiert MQTT drei verschiedene Quality of Service (QoS) Klassen.

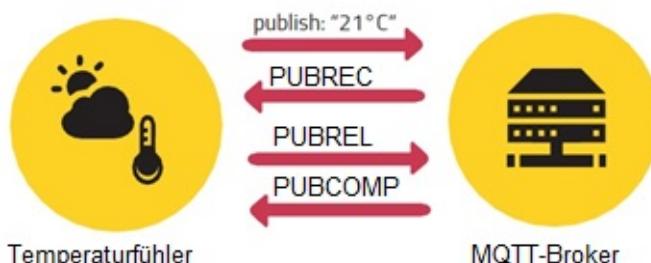
QoS Level 0 ist die niedrigste Qualitätsstufe. Bei dieser wird nicht garantiert, dass die Nachricht überhaupt irgendwo ankommt – „fire'n'forget“ [2](#).



QoS Level 1 garantiert, dass die Nachricht mindestens einmal beim Ziel ankommt. Der Broker muss das Erhalten der Nachricht einmal bestätigen. Bleibt die Bestätigung aus, oder kommt erst nach Ablauf eines Timeouts beim Client an, sendet er die Nachricht erneut. Deshalb kann es bei diesem Level zu mehrfach Sendungen kommen [2](#). ankommt – „fire'n'forget“ [2](#).



QoS Level 2 garantiert, dass die Nachricht genau einmal beim Ziel ankommt. Hierzu müssen vier Nachrichten zwischen Client und Broker ausgetauscht werden. Der Broker bestätigt das erhalten der Publish-Nachricht mit einer „PUBREC“ („Publish received“) Paket. Zu diesem Zeitpunkt ist dem Client klar, dass seine Nachricht erfolgreich beim Broker angekommen ist. Das bestätigt er wiederum mit einer „PUBREL“ („Publish release“) Paket, was der Broker wiederrum mit dem „PUBCOMP“ („Publish complete“) Paket bestätigt [2](#). ankommt – „fire'n'forget“ [2](#).



Diese QoS Klasse muss an zwei Stellen definiert werden. Zum einen muss der Sender jeder seiner Nachrichten ein QoS Level mit geben um zu definieren mit welcher Garantie die Nachricht beim Broker ankommt, zum anderen muss der Abonnent eines Topics definieren mit welcher Wahrscheinlichkeit die Nachrichten bei ihm ankommen sollen. Das gewählte QoS Level hängt von vielen Faktoren ab, die Wichtigkeit der Nachricht, die Zuverlässigkeit des Transportwegs oder die Kosten einer Nachricht können in die Entscheidung einfließen [2](#).

Last Will

Ein Client kann beim Broker festlegen, dass eine bestimmte Nachricht an alle Interessierten geschickt wird, sobald der Broker merkt, dass der entsprechende Client die Verbindung verloren hat. Hiermit kann der Client indirekt allen Abonnenten mitteilen, dass er nicht mehr erreichbar ist und auch keine Nachrichten mehr Senden kann. Auch bei LastWill Nachrichten kann ein QoS festgelegt werden [3](#).

Retained Messages

Ein Produzent kann pro Topic eine Retained Message festlegen, diese wird beim Broker gespeichert. Wenn sich nun ein Konsument dieses Topic abonniert, bekommt er sofort diese Nachricht. Der neue Abonnent bekommt also sofort eine Nachricht, ohne erst auf eine neue Nachricht warten zu müssen. Setzt nun der Producer jede seiner Nachrichten als Retained, bekommt jeder neue Abonnent sofort die letzte Nachricht [1](#).

Sicherheit

MQTT verfügt über einen einfachen Login Mechanismus. Ist dieser aktiviert, muss jeder Client, egal ob Producer oder Consumer einen Username und ein Passwort übermitteln. Zusätzlich kann im Broker konfiguriert werden, welcher User welche Rechte auf welches Topic hat. Ohne weiteren Schutz werden die Logininformationen allerdings im Klartext übertragen. Um die übertragenden Daten zu schützen, können TLS und SSL genutzt werden. Diese Verfahren garantieren eine Verschlüsselung der Daten und die hinterlegten Zertifikate garantieren sowohl Broker als auch Client seine Identität. Zusätzlich können die eigentlichen „Userdaten“ von MQTT, also z.B. die gemessenen Werte noch weiter verschlüsselt werden. Sowohl TLS / SSL als auch allgemeine Verschlüsselung führen aber zu Overhead, sodass die übertragenden Pakete größer werden und mehr Zeit zum verarbeiten benötigt wird [7](#) [8](#).

IoT Verwendung

Das Protokoll eignet sich sehr gut für den Einsatz im IoT. Grade die äußerst einfache Verwendung und der kleine Protokoll Overhead fallen positiv auf.

Smart Home

Gamze Söylev Öktem

Einführung

„Smart“ bedeutet „Intelligent“ auf Deutsch. Das Smart-Haus ist vernetzt und denkt selber. Damit ist Zeitsparen möglich und es senkt Energiekosten. Knapp ein Drittel der Deutschen benutzt schon Smart Home-Komponenten in ihrem Zuhause (Schiller 2016). Manche sehen im Smart Home sogar „die zeitgemäße Wohnform für die Anforderungen des 21. Jahrhunderts“ (Projektgruppe Smart Home 2015: 6). Doch wie genau funktioniert eigentlich das Smart Home?

In einem intelligenten Zuhause sind alle Geräte miteinander verbunden und man kann sämtliche dieser Geräte mit dem Smartphone, dem Computer oder einem Tablet überwachen und steuern (Internet of Things). Smart Homes erhöhen die Wohn- und Lebensqualität, sowie die Sicherheit der Bewohner. Man kann, aber muss nicht alles fernsteuern, es gibt auch automatisierbare Abläufe. Man kann viele verschiedene Geräte vernetzen, zum Beispiel Lampen, Jalousien, Heizung, aber auch Herd, Kühlschrank und Waschmaschine. Außerdem kann man auch Video- und Audio Eingaben benutzen, um die Geräte zu kontrollieren. Das Hauptziel ist, den Alltag komfortabler zu gestalten.

Die Idee des Smart Homes hat eine lange Geschichte. Der Begriff „smart house“ wurde zuerst in den 1980er Jahren von der American Association of House Builders verwendet. Doch bereits Jahrzehnte vorher wurden die ersten Smart Homes in 1960ern von Privatpersonen gebaut. Obwohl die Idee des Smart Homes nicht neu ist, war die Entwicklung von Smart Homes eher langsam. Die Hauptgründe für die langsame Entwicklung des Smart Homes sind folgende:

- Mangelnde Motivation zur Produktivitätssteigerung in der Hausarbeit
- Beschränkte Beteiligung der Nutzer im Entwurfsprozess
- Die weit verbreitete Annahme, dass Technologie im Haushalt langweilig ist
- Der Fokus auf Einzelgeräte in der Entwicklung der neuen Technologien (Harper 2003: 1-2)

Die effiziente Entwicklung von Smart Homes ist immer noch schwierig. Ein Problem ist, dass ein Haushalt im Gegensatz zum Arbeitsplatz keine Bereiche wie den technischen Support hat. Ein anderes Problem ist, dass die Benutzergruppe sehr heterogen ist. Sie kann Kinder, Personen mittleren Alters, sowie alte Menschen enthalten. Außerdem ist es schwer, ein Haus zu studieren, weil niemand möchte, dass er zuhause Tag und Nacht überwacht wird (Harper 2003: 1-2). Das ist nicht nur ein Problem beim Erforschen und Entwickeln des Smart Homes: Viele Menschen denken, dass die „intelligenten“ Geräte zuhause alles speichern werden und ihre Privatsphäre verletzt wird. Daher haben Sie Bedenken gegen die Nutzung von Smart Home-Technologien.

Trotz dieser zahlreichen Probleme ist die Entwicklung von Smart Homes in den letzten Jahren vorangeschritten. Heutzutage sind Smart Homes in aller Munde. Immer mehr Häuser benutzen Smart Home-Komponenten, auch wenn sie nicht immer ganz ‚smart‘ sind. In dieser Hausarbeit werde ich einige dieser Smart Home Entwicklungen analysieren. Ich werde zuerst erklären, was genau ein Smart Home ist. Danach werde ich drei Aspekte des Smart Home näher beschreiben: Energieeffizienz, Ambient Assisted Living und Sicherheit. Im vierten Kapitel, werde ich auf verschiedene Möglichkeiten für die Technische Umsetzung eingehen, bevor ich im abschließenden Kapitel ein Fazit ziehe.

Smart Home: Definitionen

Eigentlich gibt es bisher keinen allgemeinen Begriff für Smart Home. Die Begriffe Connected Home, Elektronisches Haus, Intelligentes Wohnen, Smart House, Smart Environment, Home of the Future, Smart Living, Aware Home können als Synonyme von Smart Home verstanden werden. Genauso wie es verschiedene Begriffe für das Smart Home gibt, gibt es auch verschiedene Definitionen des Konzeptes (Strese et al. 2010). Das Verständnis von Smart Home hängt dabei stark davon ab, in welcher Branche man ist. Im Gesundheitsbereich zum Beispiel wird unter Smart Home eine Wohnung verstanden, die Möglichkeiten zur Krankheitsverhütung, zum Gesundheitsmonitoring und zur Unterstützung bei Gesundheitsproblemen der Bewohner bietet (Solaimani et al. 2015: 371).

Wir können folgende Definition benutzen:

„Das Smart Home ist ein privat genutztes Heim (z. B. Eigenheim, Mietwohnung), in dem die zahlreichen Geräte der Hausautomation (wie Heizung, Beleuchtung, Belüftung), Haushaltstechnik (wie z. B. Kühlschrank, Waschmaschine), Konsumelektronik und Kommunikationseinrichtungen zu intelligenten Gegenständen werden, die sich an den Bedürfnissen der Bewohner orientieren. Durch Vernetzung dieser Gegenstände untereinander können neue Assistenzfunktionen und Dienste zum Nutzen des Bewohners bereitgestellt werden und einen Mehrwert generieren, der über den einzelnen Nutzen der im Haus vorhandenen Anwendungen hinausgeht.“ (Strese et al. 2010: 8)

Damit eine Wohnung ein Smart Home ist, sollte sie folgende Eigenschaften haben (Projektgruppe Smart Home 2015):

- Viele Geräte sind vernetzt oder vernetzbar
- Verbindung zum Internet ist vorhanden, bzw. Remote Kontrolle ist möglich
- Energieeffizienz und Nachhaltigkeit
- Offene Schnittstellen sind vorhanden (APIs)
- Zumindest Media-Anwendungen verfügen über einen schnellen Breitbandzugriff
- Mindestens AAL-Funktionen (Ambient Assisted Living) sind barrierefrei
- Steuerung von außen ist möglich
- Eingebundene Geräte, Sensoren/ Aktoren sind updatefähig

Lobaccaro und Kollegen geben eine etwas abstrakttere Liste von Eigenschaften des Smart Home (Lobaccaro et al. 2016: 2):

- Automation: Die Fähigkeit Funktionen automatisch erfüllen
- Multifunktionalität: Die Fähigkeit verschiedene Aufgaben zu erfüllen

- Anpassungsfähigkeit: Die Fähigkeit die Bedürfnisse der Bewohner zu lernen, sie vorherzusagen und sie zu erfüllen
- Interaktivität: Die Fähigkeit mit Bewohnern zu interagieren
- Effizienz: Die Fähigkeit Aufgaben so zu erfüllen, dass Zeit und Geld gespart wird

Das Thema Smart Home ist also ein sehr komplexes Thema, mit vielen verschiedenen Unterbereichen. Im folgenden Kapitel werde ich drei Aspekte des Smart Home näher beschreiben: Energieeffizienz, Ambient Assisted Living.

Verschiedene Aspekte des Smart Home

Energieeffizienz

Heutzutage ist Energieeffizienz ein sehr wichtiges Thema. Eine von den Motivationen der potentiellen Kunden des Smart Home ist, Energie zu sparen. Es gibt verschiedene Möglichkeiten mit dem Smart Home die Energieeffizienz zu verbessern. Smart Home kann sich unter anderem an Wetterprognosen orientieren. An einem heißen Tag kann zum Beispiel die Heizung automatisch runterschaltet werden. Ein anderes Beispiel kann sein, dass Jalousien an einem sonnigen Sommertag runtergefahren werden, damit die Wohnung kühler bleiben kann.

Außerdem kann das Smart Home die Anwesenheit der Bewohner bemerken und Heizung und Stromverbrauch daran orientieren. Beispielsweise kann das Licht in einem Raum ausgeschaltet werden, wenn niemand da ist (Projektgruppe Smart Home 2015).

Das Smart Home kann das Verhalten der Bewohner lernen und Strom- und Heizverbrauch daran orientieren. Beispielsweise kann die Heizung angemacht werden, wenn das System erwartet, dass ein Bewohner in die Wohnung kommen wird (Lobaccaro et al. 2016). Überdies, kann selbst die Visualisierung vom Energieverbrauch dabei helfen, dass die Bewohner weniger verbrauchen.

Ambient Assisted Living

Unter Ambient Assisted Living (AAL) versteht man altengerechte und behindertengerechte Assistenzsysteme für ein gesundes und unabhängiges Leben. Im Kontext einer alternden Gesellschaft ist AAL ein wichtiges Thema, das für Smart Home viel Potential bietet.

Alten und physisch behinderten Menschen fällt es oft schwer sich zu bewegen. Deswegen sind sie meistens abhängig von anderen Menschen. Mit Smart Home Anwendungen können diese Menschen alleine oder unabhängiger leben und dadurch kann ihre Lebensqualitäten erhöht werden. Beispielsweise kann man mit Hilfe von Kameras und Mikrofonen Besucher an der Tür sehen und mit ihnen kommunizieren. Ebenfalls kann sich das System Besucher merken und bestimmte Besucher automatisch zulassen.

In Deutschland gibt es viele AAL Labors. Die Ziele der Labors enthalten Realisierung, Demonstration, Evaluation und Entwicklung des Smart Home. In manchen dieser Labors leben Bewohner (Strese et al. 2010: 16-34).

Sicherheit

Sicherheit ist ein sehr wichtiges Thema im Bereich Smart Home. Auf der einen Seite, kann man Smart Home Technologien einsetzen, um die Sicherheit im Haus zu erhöhen. Auf der anderen Seite, stellen viele Smart Home Anwendungen potentielle Sicherheitsprobleme dar, die zum Beispiel von Hackern ausgenutzt können. Hier beschreibe ich nur den ersten Aspekt.

In einem Smart Home ist es möglich, viele Sicherheitsmaßnahmen zu treffen. Das System überwacht das Haus ganze Zeit und kontrolliert den Kohlenmonoxid-Gehalt. Bei einem Feuer, kann das System es sehr schnell bemerken und die Bewohner und die Feuerwehr informieren. Das System kann sogar den genauen Ort des Feuers bestimmen und die Feuerwehr Bescheid sagen.

Das System kann überwachen wer das Haus betritt und verlässt. Es kann mit Hilfe von einem Gesichtserkennungssystem unterscheiden, wer Bewohner, Besucher oder Eindringling ist. Bei unbekannten Besuchern, kann den Bewohnern ein Video geschickt werden oder die Polizei informiert werden (Robles und Kim 2010b: 18-19).

Fazit

In diesem Artikel habe ich den Bereich Smart Home beschrieben. Das Smart Home und das Internet of Things gewinnt immer mehr an Bedeutung. Von manchen wird das Smart Home sogar als „die zeitgemäße Wohnform für die Anforderungen des 21. Jahrhunderts“ (Projektgruppe Smart Home 2015: 6) gesehen. In dieser Arbeit habe ich zuerst die Bereiche Energieeffizienz, Ambient Assisted Living und Sicherheit vorgestellt. Danach habe ich verschiedene Möglichkeiten der technischen Umsetzung näher beschrieben.

Bei der technischen Umsetzung habe ich mich vor allem auf Z-Wave Systeme und das MQTT Protokoll konzentriert. Ein Problem des Smart Home ist, dass verschiedene Systeme existieren, die häufig nicht miteinander kompatibel sind. Daher wird von manchen gefordert, dass die „Vernetzbarkeit von Geräten unterschiedlicher Hersteller [...] weiter vereinfacht werden [muss]“ (Projektgruppe Smart Home 2015: 10). Es wird sich zeigen, welche Standards und Protokolle sich in Zukunft durchsetzen werden.

Smart Homes machen das Leben einfacher und komfortabler. Besonders für die alten und behinderten Menschen sind sie sehr hilfreich und wichtig. Sie haben auch viele Sicherheitsvorteile, zum Beispiel, weil sie Überwachungsmöglichkeiten bieten. Andererseits bringen sie eigene Sicherheitsprobleme mit sich. Dies wurde im Oktober 2016 deutlich als ein Hackerangriff das Internet of Things benutzte um Webseiten wie Twitter und PayPal anzugreifen. Man befürchtet, dass diese Angriffe in den nächsten Jahren häufiger und stärker werden, wenn immer mehr Menschen Smart-Home-Geräte nutzen (Thielman 2016).

Bibliographie

Aldrich, Francis K. (2003) Smart Homes: Past, Present and Future. In: Harper, Richard (ed.) (2003) Inside the Smart Home, pp.17-40. London: Springer.

Harper, Richard (2003) Inside the Smart Home: Ideas, Possibilities and Methods. In: Harper, Richard (ed.) (2003) Inside the Smart Home, pp.1-14. London: Springer.

HIVEMQ (2017) MQTT 101 – How to Get Started with the lightweight IoT Protocol.

Aufgerufen am 01. Juli 2017 unter: <http://www.hivemq.com/blog/how-to-get-started-with-mqtt>

IBM Knowledge Center (2017) IBM MQ, Version 9.0. Publish/subscribe messaging.

Aufgerufen am 01. Juli 2017 unter:

https://www.ibm.com/support/knowledgecenter/en/SSFKSJ_9.0.0/com.ibm.mq.pro.doc/q004870_.htm

Lobaccaro, Gabriele; Carlucci, Salvatore und Löfström Erica (2016) Review of Systems and Technologies for Smart Homes and Smart Grids. Energies, Vol. 9(5).

Nationaler IT-Gipfel Berlin 2015 (2015) Smart Home. Ergebnisdokument der Projektgruppe Smart Home. Konvergenz der Netze. Nationaler IT-Gipfel Berlin 2015. Aufgerufen am 30. Juni 2017 unter: http://www.bmvi.de/SharedDocs/DE/Anlage/Digitales/it-gipfel-fg-konvergenz-pg-smarthome.pdf?__blob=publicationFile

Pätz, Christian (2011) Z-Wave Grundlagen. Funksteuerung im Smart Home. Norderstedt: Books on Demand.

Robles, Rosslin John und Kim, Tai-hoon (2010a) Applications, Systems and Methods in Smart Home Technology: A Review. International Journal of Advanced Science and Technology, Vol. 15, February 2010.

Robles, Rosslin John und Kim, Tai-hoon (2010b) A review on security in smart home development. International Journal of Advanced Science and Technology, Vol. 15, February 2010.

Rouse, Margaret (2015) MQTT (MQ Telemetry Transport). Aufgerufen am 01. Juli 2017 unter: <http://internetofthingsagenda.techtarget.com/definition/MQTT-MQ-Telemetry-Transport>

Saad al-sumaiti, Ameena; Ahmed, Mohammed Hassan und M. A. Salama, Magdy (2014) Smart Home Activities: A Literature Review. Electric Power Components and Systems, 42(3-4), 294-305.

Schiller, Kai (2016) Was ist ein Smart Home? Geräte und Systeme. Aufgerufen am 30. Juni 2017 unter: <https://www.homeandsmart.de/was-ist-ein-smart-home>

Solaimani, Sam; Keijzer-Broers, Wally und Bouwman, Harry (2015) What we do – and don't – know about the Smart Home: An analysis of the Smart Home literature. Indoor and Built Environment, Vol. 24(3), 370-383.

Strese, Hartmut; Seidel, Uwe; Knappe, Thorsten und Botthof, Alfons (2010) Smart Home in Deutschland. Untersuchung im Rahmen der wissenschaftlichen Begleitung zum Programm Next Generation Media (NGM) des Bundesministeriums für Wirtschaft und Technologie. Berlin: Institut für Innovation und Technik (iit) in der VDI/VDE-IT. Aufgerufen am 30. Juni 2017 unter: <http://www.iit-berlin.de/de/publikationen/smart-home-in-deutschland>

Thielman, Sam (2016) Can we secure the internet of things in time to prevent another cyber-attack? The Guardian, 25. Oktober 2016. Aufgerufen am 30. Juni 2017 unter: <http://www.iit-berlin.de/de/publikationen/smart-home-in-deutschland>

Thielman, Sam und Hunt, Elle (2016) Cyber attack: hackers 'weaponised' everyday devices with malware. The Guardian, 22. Oktober 2016. Aufgerufen am 30. Juni 2017 unter: <https://www.theguardian.com/technology/2016/oct/22/cyber-attack-hackers-weaponised-everyday-devices-with-malware-to-mount-assault>

Fullstack Development

Teilbereich(e): Realtime DBs, SPAs, Responsive Design, NoSQL, Mobile Computing

Projektleiter:

Projektteam: Timo Rolfsmeier, Niklas Harting, Alexander Schwietert, Tolga Aydemir, Malte

Berg, Lutz Winkelmann, Fabian Lorenz, Benjamin Schmidt

Pflichtenheft: [PDF](#)

GitLab-Group: [ShareBase](#)

NoSQL

Autor: Fabian Lorenz

Einführung

Einleitung (vgl. [1 Kap. 1.4],[2 Kap. 1-2],[12 Kap. 1],[13 Kap. 1],[14 Kap. 2.18,5],[66-69])

Mit dem Einzug des Internets in das allgemeine Leben entstanden auch neue Anforderungen bezüglich Datenbanken. Das Arbeiten mit enorm großen Datenmengen brachte viele Probleme mit sich, die meist von klassischen relationalen Datenbanken nicht gelöst werden konnten. Es entstanden daher viele neue Datenbanken, die sich in ihren Eigenschaften stark von den herkömmlichen relationalen Datenbanken abheben, um diese Probleme zu lösen.

Diese Datenbanken werden heutzutage unter dem Begriff NoSQL Datenbanken zusammengefasst. Während dieser Begriff zwar schon 1998 genutzt wurde, um eine relationale Datenbank ohne SQL Zugriffsmöglichkeiten zu beschreiben ("No SQL"), wird der Begriff seit 2009 generell als "Not only SQL" verstanden. NoSQL Datenbanken zeichnen sich häufig durch folgende Eigenschaften aus:

- Das zugrundeliegende Datenmodell ist nicht relational.
- Sie verteilen ihre Daten auf viele verschiedene Serverknoten und Knoten lassen sich zwecks Skalierung leicht hinzufügen/entfernen.
- Es gibt nur eine schwache oder gar keine Schema-Restriktion.
- Sehr einfache Replikation von Daten.
- Es gibt eine einfache API.
- Es wird nicht das klassische ACID Transaktionskonzept genutzt, sondern beispielsweise BASE.

NoSQL Datenbanken bieten meist eine höhere Leistungsfähigkeit als relationale Datenbanken, sind durch schemafreie Datenstrukturen flexibler und dank verteilter und redundanter Datenhaltung ausfallsicherer.

In diesem Kapitel werden zunächst die Grundlegenden Begriff und Technologien beschrieben, die von NoSQL Datenbanken meist verwendet werden. Anschließend werden die wichtigsten verschiedenen NoSQL Arten beschrieben.

ACID, BASE und das CAP-Theorem

ACID (vgl. [1 Kap. 4.2.1],[2 Kap. 2.2.1],[12 Kap. 1.3],[13 Kap. 1],[14 Kap. 2.6],[15-23])

Ein wichtiger Aspekt einer jeden Datenbank ist ihr Transaktionskonzept. Viele der altbekannten relationalen Datenbanken setzen hier auf das ACID Prinzip um die Integrität der Daten zu gewährleisten. Eine Transaktion ist dabei beliebige Folge zusammenhängender Verarbeitungsschritte aus den Grundoperationen Create, Read, Update und Delete (CRUD). Damit mehrere Nutzer mit einer Datenbank arbeiten können ohne dabei Konflikte in den Daten hervorzurufen, müssen die Transaktionen bestimmten Regeln folgen. Im Falle von ACID müssen sie daher folgende Eigenschaften einhalten:

- Atomarität (A=Atomicity): Eine Transaktion ist atomar, wenn sie in ihrer Gesamtheit entweder ganz oder gar nicht ausgeführt wird (Alles oder nichts Prinzip). Folglich müssen die einzelnen Operationen einer Transaktion komplett und ohne Fehler durchlaufen um einen neuen Ergebniszustand zu erreichen. Tritt jedoch ein Fehler während der Transaktion auf, wird der Zustand der Datenbank auf den Anfangszustand zurückgesetzt, den sie zu Beginn der Transaktion hatte. Alle Operationen der fehlerhaften Transaktion werden somit ungültig und bereits durchgeführte rückgängig gemacht.
- Konsistenz (C=Consistency): Eine Transaktion muss die Datenbank von einem konsistenten Anfangszustand in einen konsistenten Endzustand überführen. Um Konsistent zu sein, muss die Integrität und Plausibilität der Daten gewährleistet sein, beispielsweise müssen Beziehungen über Fremd- und Primärschlüssel korrekt sein. Kann kein konsistenter Zustand nach dem Ausführen der Transaktion hergestellt werden, so wird die gesamte Transaktion als ungültig deklariert. Die Datenbank wird dann auf den Zustand vor Beginn der Transaktion zurückgesetzt.
- Isolation (I=Isolation): Isolation sagt aus, dass Transaktionen vollkommen getrennt voneinander ablaufen. Gleichzeitig ablaufende Transaktionen liefern somit dieselben Resultate, so als würden sie sequentiell hintereinander ausgeführt werden. Es soll damit verhindert werden, dass Transaktionen sich gegenseitig beeinflussen und beispielsweise nicht zwei gleichzeitig denselben Datenwert ändern.
- Dauerhaftigkeit (D=Durability): Jede erfolgreiche Transaktion setzt die Datenbank in einen dauerhaften Zustand, der sich erst ändert, wenn eine neue Transaktion durchgeführt wurde. Diese Dauerhaftigkeit muss auch bei Systemabstürzen gewährleistet sein.

Es ist ersichtlich, dass die Einhaltung des ACID Prinzips vor allem die Konsistenz der Daten zum Ziel hat. In der heutigen Zeit spielen bei verteilten Datenbanksystemen allerdings noch zwei weitere Aspekte eine Rolle: Verfügbarkeit (Antwortzeiten) und Ausfalltoleranz.

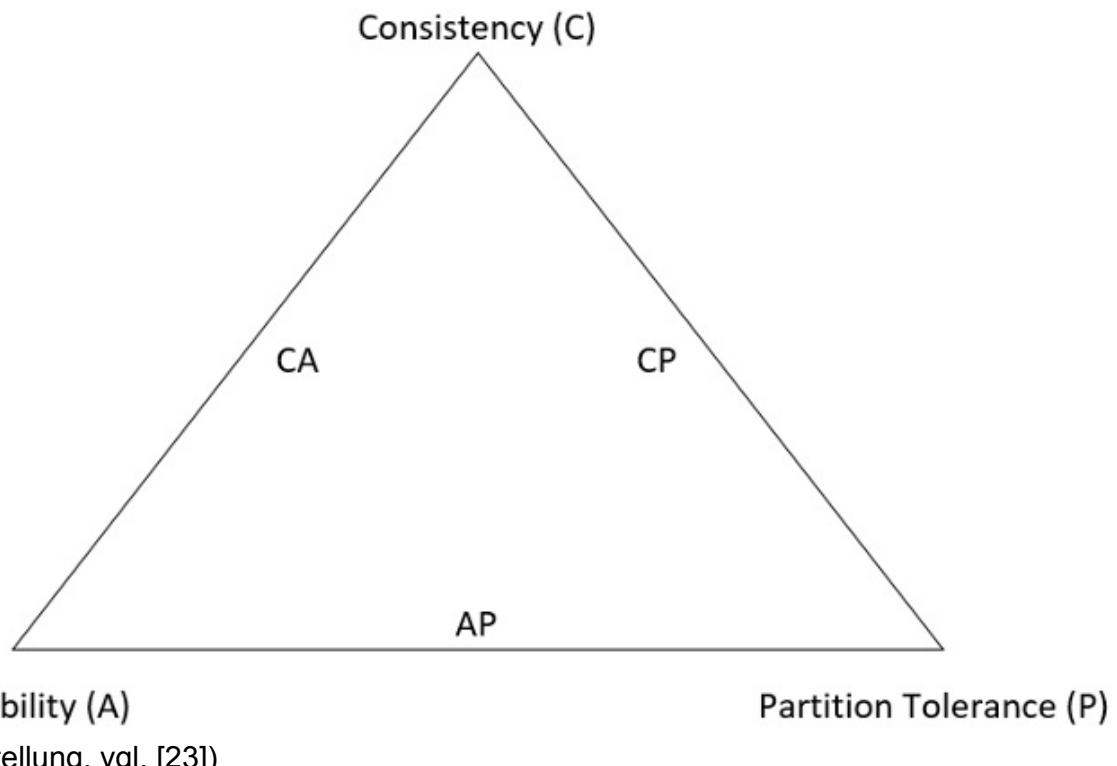
CAP (vgl. [1 Kap. 4.3.1],[2 Kap. 2.2.2],[12 Kap. 1.5],[13 Kap. 1],[14 Kap. 5.3],[23-28])

- Konsistenz (C=Consistency): Bei verteilten Datenbanksystemen mit replizierten Knoten

muss für die Konsistenz sichergestellt werden, dass bei Änderung der Daten auf einem Knoten alle folgenden Transaktionen auch mit diesen veränderten Werten arbeiten, selbst wenn sie mit einem anderen Knoten arbeiten.

- Verfügbarkeit (A=Availability): Verfügbarkeit bezeichnet die Fähigkeit ununterbrochen auf die Datenbank in angemessener Zeit zugreifen zu können.
- Ausfalltoleranz (P=Partition Tolerance): Ausfalltoleranz besagt, dass der Ausfall eines (oder mehrerer) Knoten eines verteilten Datenbanksystems, oder der Ausfall von Verbindungen zwischen den Knoten nicht den Ausfall des Gesamtsystems zur Folge hat.

Optimal wäre es, wenn alle drei Aspekte, Konsistenz, Verfügbarkeit und Ausfalltoleranz gleichzeitig im vollen Umfang erreicht werden könnten. Es hat sich jedoch gezeigt, dass dies innerhalb von verteilten Datenbanksystemen nicht möglich ist und sich immer nur zwei der drei Aspekte umsetzen lassen. Dieses Theorem wird dabei als CAP-Theorem bezeichnet. In der folgenden Abbildung ist das CAP-Theorem und die Tatsache, dass immer nur zwei Aspekte umgesetzt werden können, als Dreieck dargestellt.



Je nachdem welche zwei der drei Anforderungen an die Datenbank gestellt werden, werden diese Systeme als CA (Konsistenz und Verfügbarkeit), CP (Konsistenz und Ausfalltoleranz) oder AP (Verfügbarkeit und Ausfalltoleranz) Systeme bezeichnet. Da traditionelle relationale Datenbanksysteme wie bereits erwähnt meist die Konsistenz als oberstes Ziel haben, lassen sie sich als CA/CP Systeme bezeichnen. Viele NoSQL Datenbanksystem haben jedoch

einen viel stärkeren Fokus auf die Verfügbarkeit oder Ausfalltoleranz, die Konsistenz spielt dabei keine so große Rolle. Für diese Systeme ist es ausreichend, wenn die Konsistenz der Daten nach einer Änderung erst im Laufe der Zeit wiederhergestellt ist.

BASE (vgl. [1 Kap. 4],[2 Kap. 2.2.3],[12 Kap. 1.6],[13 Kap. 1], [14 Kap. 5.4],[23,25,29-30])

Diese Eigenschaft wird dabei auch häufig als BASE Modell (Basically Available, Soft State, Eventually Consistent) bezeichnet. Bei BASE wird die Verfügbarkeit über die Konsistenz gestellt und es gilt des Öfteren als Gegenstück zu ACID.

Die Verfügbarkeit wird dabei beispielsweise durch eine große Anzahl an Knoten erreicht (Basically Available).

Eine Änderung der Daten in einem Knoten hat dabei nicht zur Folge, dass die Änderungen direkt allen anderen Knoten mitgeteilt wird, sondern nur schrittweise. Es kann somit passieren, dass zwei Nutzer beim Lesen derselben Daten zwei unterschiedliche Datenwerte erhalten (Soft State).

Irgendwann haben jedoch alle Knoten die Änderungen verarbeitet und die Datensätze sind wieder konsistent (Eventually Consistent).

Dieses Konsistenzmodell wird von vielen NoSQL Datenbanksystemen genutzt.

Skalierung (vgl. [2 Kap. 2.2],[3-7])

Unter Skalieren versteht man die Eigenschaft eines Systems, sich an verändernde (meist wachsende) Ansprüche anzupassen. Im Bereich der Datenbanken kann beispielsweise der verfügbare Speicherplatz oder die Zugriffsgeschwindigkeit auf Daten eine Rolle spielen.

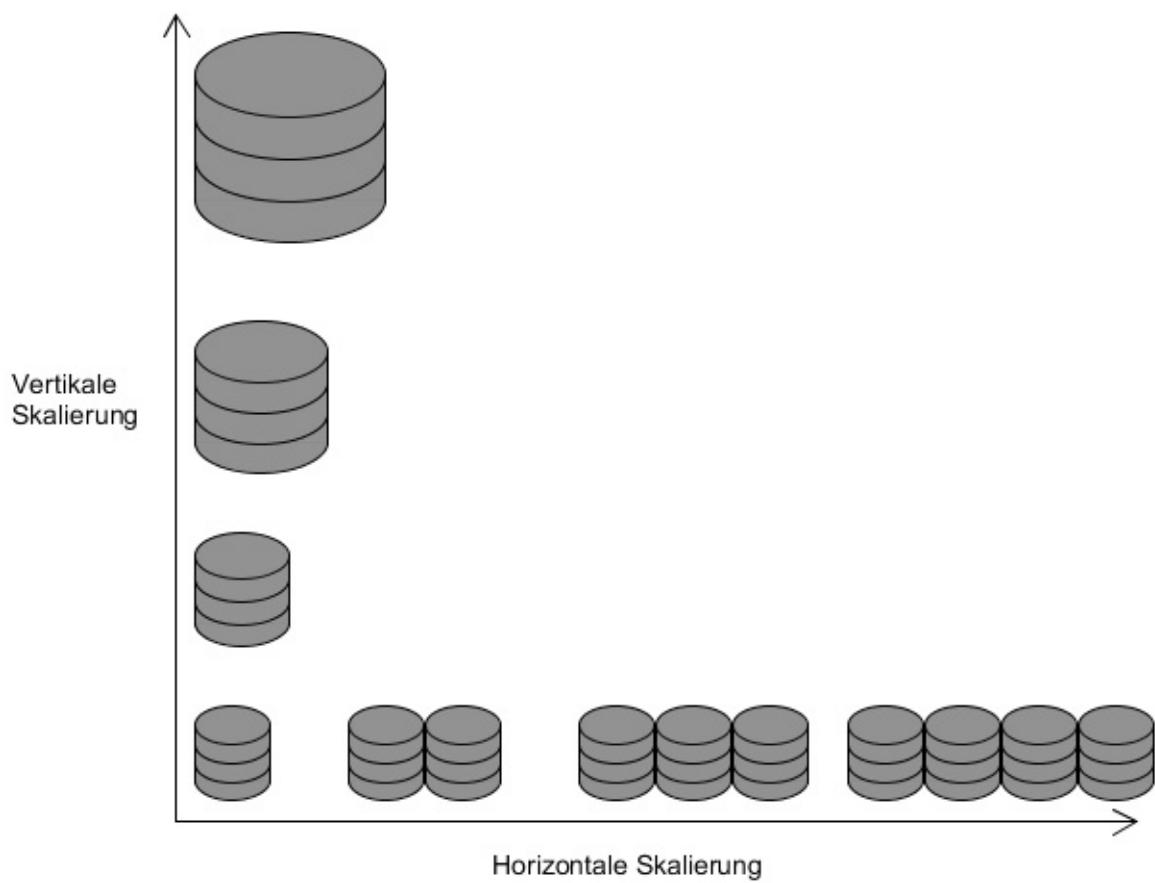
Eine Verbesserung eines Systems kann dabei auf zwei verschiedene Arten erfolgen:

- Vertikale Skalierung (Scale-Up)

Von einer vertikalen Skalierung spricht man, wenn das vorhandene System durch bessere Hardware/Komponenten aufgerüstet wird. Dies kann beispielsweise durch Hinzufügen einer CPU oder das Austauschen der alten durch eine neue erfolgen, um die Leistung des Systems zu erhöhen. Vergrößerung des Speicherplatzes kann durch Hinzufügen einer weiteren Festplatte erfolgen.

- Horizontale Skalierung (Scale-out)

Von horizontaler Skalierung ist die Rede, wenn das vorhandene System durch Hinzufügen neuer Ressourcen Einheiten (Rechner/Server) erweitert wird.



(Eigenerstellung, vgl. [6])

In der Abbildung ist das Prinzip der vertikalen und horizontalen Skalierung anhand des Beispiels eines Datenbanksystems dargestellt. Zu Beginn existiert ein Server, bei der vertikalen Skalierung wird dieser durch bessere Komponenten aufgerüstet, bei der horizontalen Skalierung werden weitere Server hinzugefügt.

Der Nachteil bei einer vertikalen Skalierung besteht darin, dass sie irgendwann an ihre Grenzen stößt. Wenn bereits die besten Komponenten, die auf dem Markt zu finden sind, verwendet werden, kann keine Verbesserung mehr erfolgen. Der Vorteil ist jedoch, dass meist keine Änderungen beispielsweise an der Anwendungssoftware durchgeführt werden müssen, da die Struktur des Systems gleich bleibt. Bei horizontaler Skalierung können Änderungen jedoch von Nöten sein, wenn Software stark an die bisherige Struktur angepasst ist. Viele NoSQL-Datenbanken haben dieses Problem jedoch nicht, da sie auf diese Art der Skalierung ausgelegt sind und Mechanismen dafür bieten.

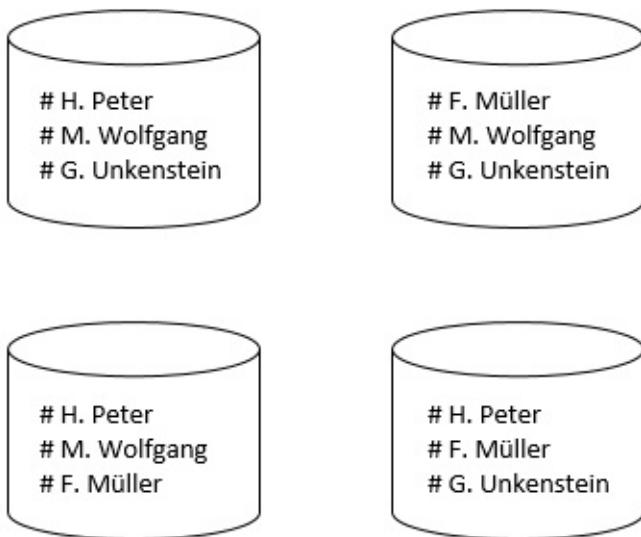
Clustering, Replikation, Sharding und Hashing

Clustering (vgl. [1 Kap. 7.2],[2 Kap. 2],[12 Kap. 1],[44-49])

Das Zusammenfassen mehrerer einzelnen Rechner/Server zu einem System wird im Allgemeinen als Clustering bezeichnet. Je nach Einsatzgebiet und genutzter Technologie kann dieses System dabei einen unterschiedlichen Aufbau haben. Beispielsweise kann es einen Master und mehrere Slaves geben, oder es können mehrere gleichberechtigte Knoten existieren. Im Allgemeinen ist der Vorteil eines Clusters jedoch, dass eine Lastverteilung stattfinden kann. Ist ein Knoten stark ausgelastet, können seine Aufgaben an andere Knoten im System verteilt werden. Auch können Aufgaben, die an das System gestellt werden, direkt an den Knoten geleitet werden, der zurzeit am wenigsten ausgelastet ist. Ein weiterer Vorteil besteht meist auch in einer verbesserten Ausfallsicherheit. Fällt ein Knoten aus, können die anderen Knoten des Systems diesen Ausfall kompensieren.

Replikation (vgl. [1 Kap. 6.2],[2 Kap. 2],[13 Kap. 3],[14 Kap. 3.11],[44-47,50-52])

Als Replikation wird das Speichern gleicher Datenbestände auf mehrere Knoten im System bezeichnet. Auch hier kann es je nach Datenbankmanagementsystem unterschiedliche Anwendungen dieses Prinzips geben. Alle Knoten des Systems können beispielsweise dieselben Daten besitzen und jede Änderung untereinander aktualisieren. Auch ist es möglich, dass es einen Master gibt, der für alle Schreibzugriffe zuständig ist und seine Datenbestände an die Slaves weitergibt. Leseanfragen können dabei über die Slaves abgearbeitet werden. Eine weitere Möglichkeit ist, dass nicht jeder Knoten dieselben Daten hält, sondern jeder hält immer nur einen bestimmten Teil der Daten, sodass am Ende ein bestimmter Datensatz zwar des Öfteren im Gesamtsystem vorkommt, aber nicht auf jedem einzelnen Knoten. Häufig wird dabei mit einer Replikationsrate von drei oder fünf gearbeitet, soll heißen ein bestimmter Datensatz ist auf drei, bzw. fünf verschiedenen Knoten im Clustersystem vorhanden. In der folgenden Abbildung ist ein kleines Beispiel mit einer Replikationsrate von drei dargestellt.

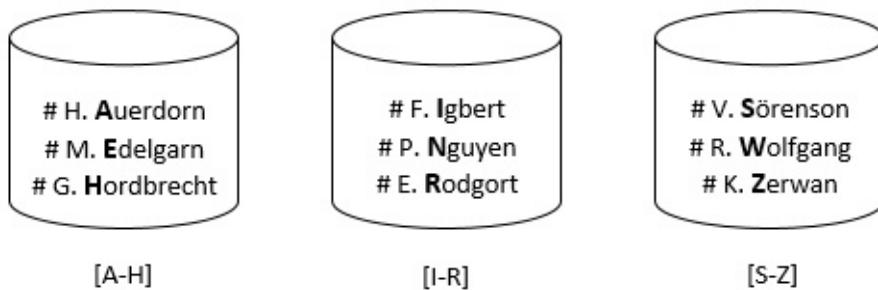


(Eigenerstellung, vgl. [45])

Vorteile der Replikation bestehen in der verbesserten Ausfallsicherheit und kürzeren Antwortzeiten. Fällt ein Knoten aus kann dennoch auf die betroffenen Datensätze zugegriffen werden, da diese auch von anderen Knoten gespeichert werden. Durch die Verteilung der Daten an verschiedene Orte kann eine bessere Lastverteilung stattfinden und dementsprechend die Antwortzeiten verringert werden. Probleme bestehen jedoch immer bei der Konsistenzbewahrung. Änderungen auf einem Knoten müssen auf alle anderen betroffenen Knoten übertragen werden.

Sharding (vgl. [1 Kap. 6.2],[14 Kap. 5.8],[44-47,53-55])

Beim Sharding werden Datenbestände möglichst gleichmäßig aufgeteilt und auf verschiedene Knoten im Cluster verteilt. Ein Stück eines solchen geteilten Datenbestands wird dabei als Shard (Bruchstück/Splitter) bezeichnet. Ein einfaches Beispiel dafür ist in der folgenden Abbildung dargestellt. Es werden die Namen anhand der alphabetischen Reihenfolge sortiert und auf drei Knoten aufgeteilt.



(Eigenerstellung, vgl. [45])

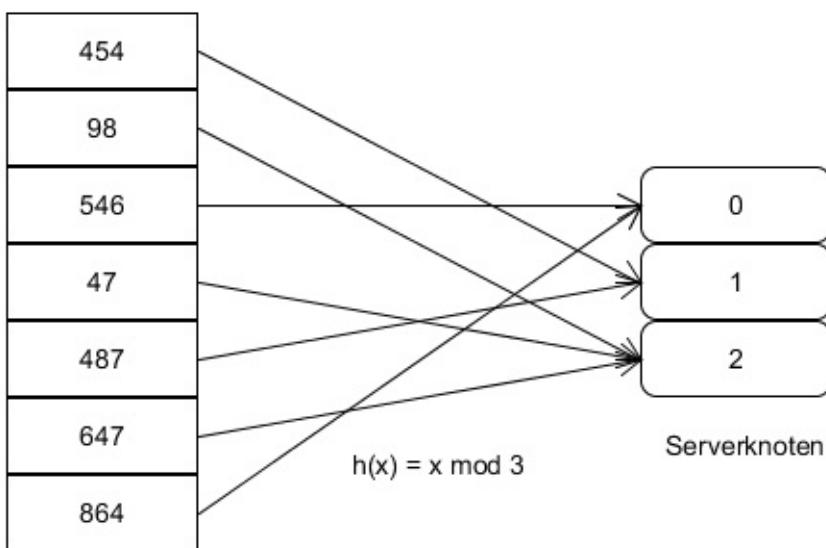
Durch Sharding ist es möglich auch sehr große und umfangreiche Datenmengen zu verwalten. Datenmengen, die die Kapazität eines Servers übersteigen würden, können auf mehrere aufgeteilt werden. Dadurch kann auch die Last verteilt werden und sich somit die Antwortzeiten verbessern. Der große Nachteil ist jedoch, dass beim Ausfall eines Shards auch nicht mehr auf die von ihm verwalteten Daten zugegriffen werden kann.

Dementsprechend wird Sharding sehr häufig mit Replikation zusammen verwendet. Gleiche Shards werden dabei wieder auf verschiedene Knoten verteilt, sodass beim Ausfall eines Knoten auf das Shard auf einem anderen Knoten zurückgegriffen werden kann. Sehr viele NoSQL Datenbanksysteme bieten automatische Mechanismen an, die die Verwendung von Sharding und Replikation stark vereinfachen. Datensätze werden bei Bedarf automatisch umverteilt, beispielsweise wenn ein neuer Knoten hinzugefügt wird oder die Datensätze nicht mehr gleichmäßig aufgeteilt sind.

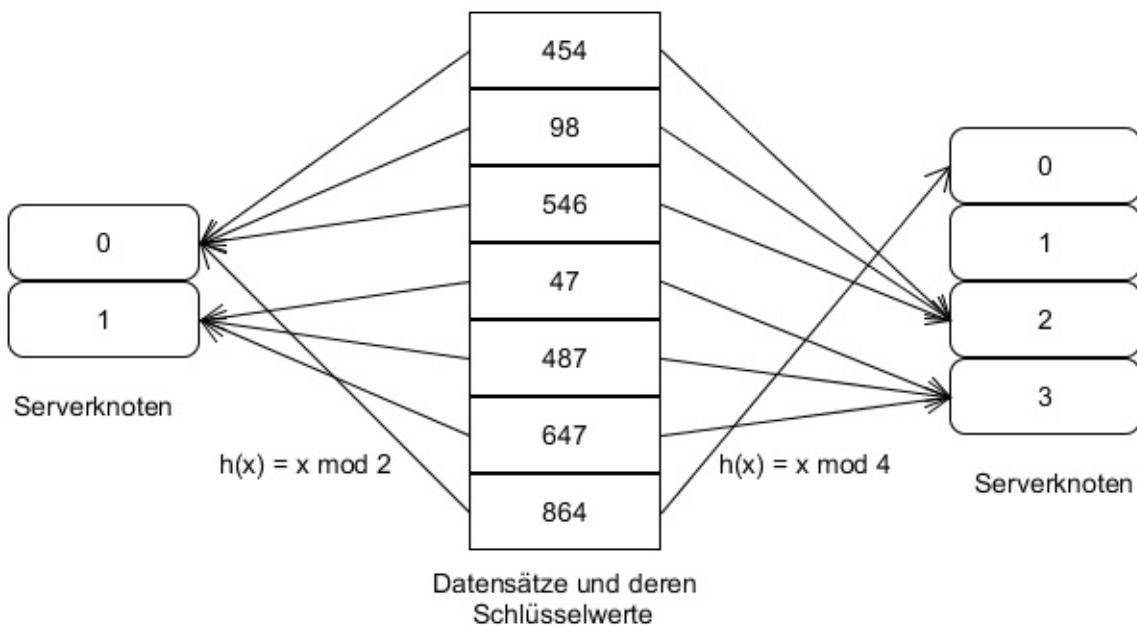
Hashing (vgl. [1 Kap. 5.2.2,5.2.3],[2 Kap. 2.3],[12 Kap. 2],[44-47,56-65])

Die verwendete Methode, um Datensätze in verteilten Datenbanksystemen auf mehrere Knoten aufzuteilen, bzw. zu speichern, ist dabei das Hashing. Beim Hashing wird durch eine Funktion ein Wert aus einem großen Wertebereich auf einen Wert aus einem kleineren Wertebereich abgebildet. Dabei wird meist eine möglichst gleichmäßige Verteilung angestrebt.

Eine einfache Hashfunktion, die für die Verteilung von Daten auf mehrere Datenbankknoten verwendet werden kann, ist dabei die Modulo Funktion $H(x) = x \bmod n$. Der Wert x ist dabei der Datensatz der auf einen der verfügbaren Knoten verteilt werden soll, bzw. ein entsprechender Schlüsselwert der den Datensatz identifiziert. Der Wert n ist die Anzahl der verfügbaren Knoten. Durch diese einfache Hashfunktion können alle Datensätze auf die zur Verfügung stehenden Knoten aufgeteilt werden. Problematisch ist es jedoch wenn ein Serverknoten ausfällt oder ein neuer hinzugefügt werden soll. Es muss für jeden Datensatz neu berechnet werden, auf welchen Knoten im Cluster er gespeichert werden soll und häufig müssen sehr viele Datensätze auf einen anderen Knoten als vorher verschoben werden. Dies wird in den folgenden Abbildungen deutlich. Die erste zeigt die Verteilung bei drei Serverknoten, die zweite die Verteilung wenn ein Knoten ausfällt oder dazukommt.



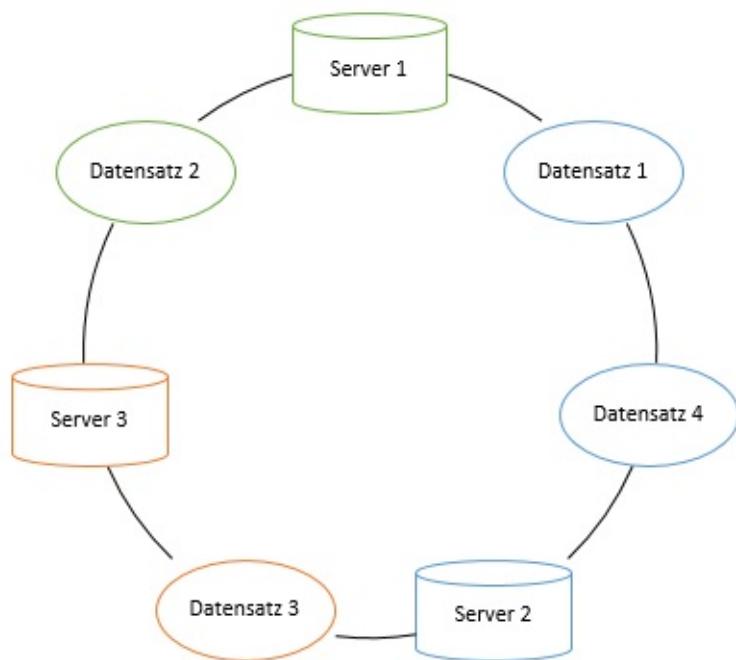
Datensätze und deren
Schlüsselwerte



(Eigenerstellungen, vgl. [2 Kap. 2.3])

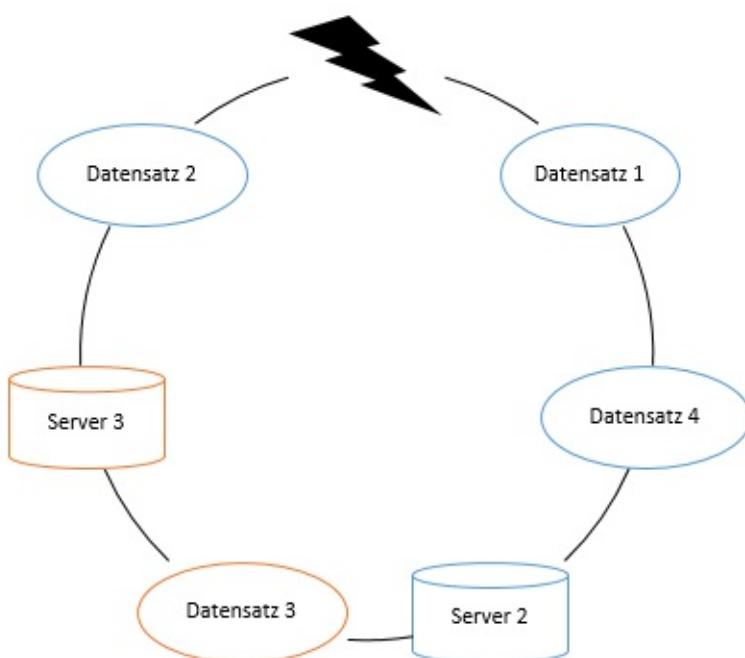
Der erste Datensatz wird bei drei Knoten auf dem zweiten gespeichert. Fällt der dritte aus muss der Datensatz auf den ersten verschoben werden, kommt ein vierter dazu muss er auf den dritten kopiert werden. Dies zeigt deutlich den Aufwand der Neuverteilung bei Änderungen der verfügbaren Serverknoten.

Um dieses Problem zu umgehen wird das Consistent Hashing verwendet. Dabei wird ein fester Adressbereich definiert, der von einer Hashfunktion abgedeckt ist. Jeder Serverknoten wird dann mithilfe der Hashfunktion in diesem Adressbereich platziert. Jeder Datensatz wird nun über die Hashfunktion in genau den Knoten gespeichert, der dem Hashwert am nächsten ist. Bildlich lässt sich dies am besten als Ring darstellen. Die Knoten werden auf dem Ring platziert und halten alle Datensätze die beispielsweise entgegengesetzt des Uhrzeigersinns vor ihnen liegen. Dieses Verfahren ist in der folgenden Abbildung dargestellt.



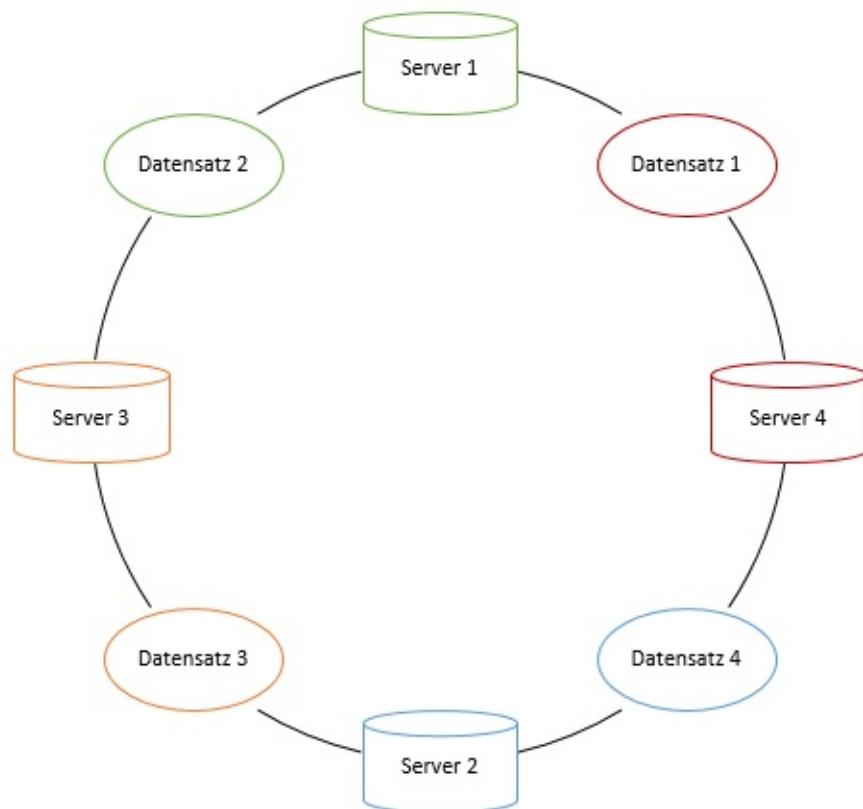
(Eigenerstellung, vgl. [2 Kap. 2.3])

Fällt nun ein Serverknoten aus, so müssen nur die Datensätze verschoben werden, die bisher von ihm gehalten wurden. Diese werden dann vom nächsten Knoten im Uhrzeigersinn gehalten. Alle anderen Datensätze müssen nicht neuverteilt werden und können weiterhin auf den bisherigen Knoten bleiben.



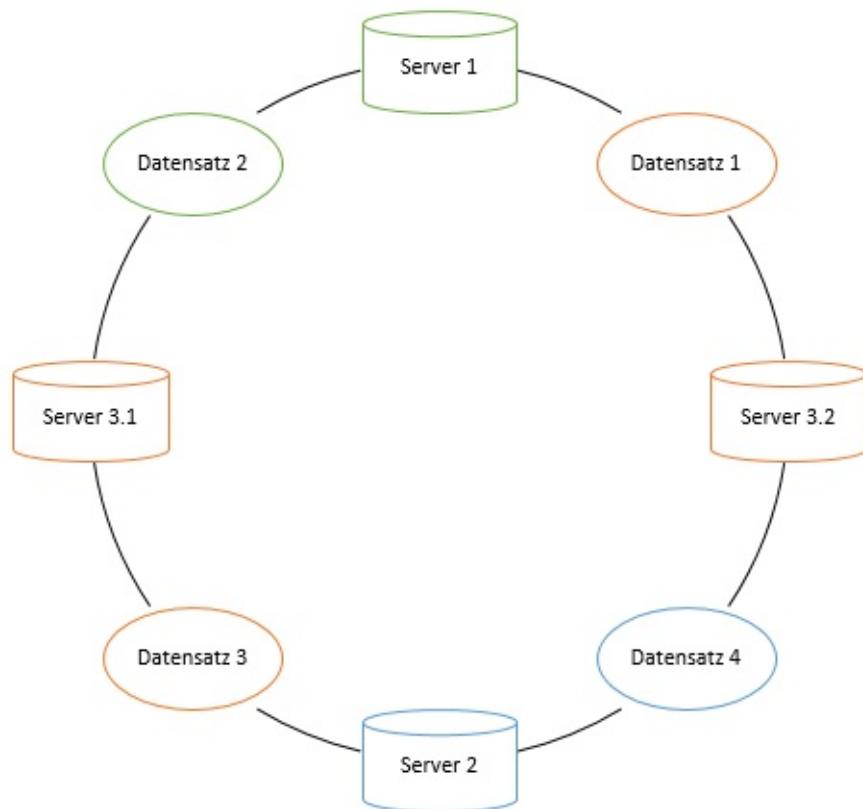
(Eigenerstellung, vgl. [2 Kap. 2.3])

Genauso verhält es sich, wenn ein neuer Knoten hinzugefügt wird. Der neue bekommt dann alle Datensätze, die entgegengesetzt des Uhrzeigersinns vor ihm liegen.



(Eigenerstellung, vgl. [2 Kap. 2.3])

Des Weiteren ist es möglich, virtuelle Serverknoten einzufügen um eine bessere Lastverteilung zu erreichen. Gibt es beispielsweise einen Knoten, der leistungsstärker ist als die anderen, oder mehr Speicherplatz besitzt, so wird für ihn ein virtueller Serverknoten auf dem Ring platziert werden. Jeder Datensatz, der nun diesem virtuellen Serverknoten zugeordnet ist, wird auf dem Knoten gespeichert, zu dem der virtuelle gehört. In der folgenden Abbildung besitzt beispielsweise Server drei mehr Speicherplatz als die anderen. Dementsprechend wird ein virtueller Knoten für ihn auf dem Ring platziert, sodass nun mehr Datensätze auf dem dritten Server gespeichert werden.



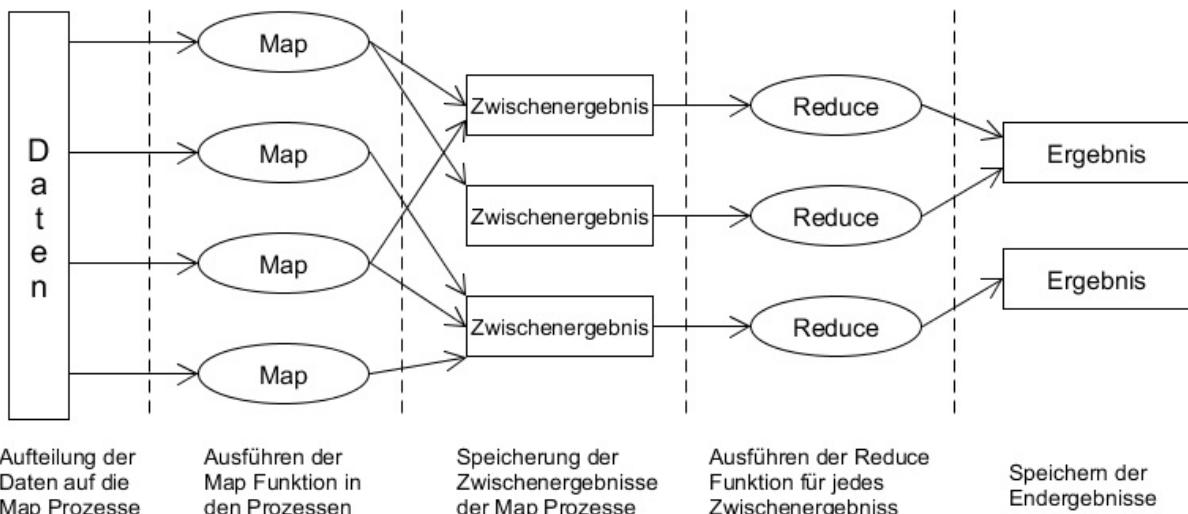
(Eigenerstellung, vgl. [2 Kap. 2.3])

Consistent Hashing wird von vielen NoSQL Datenbanksystemen verwendet. Im Zusammenspiel mit verschiedenen Formen der Replikation und Sharding lassen sich dadurch robuste Datenbankservercluster aufbauen, die durch einfaches hinzufügen oder entfernen von Knoten sich leicht an ändernde Begebenheiten anpassen lassen.

Map-Reduce (vgl. [1 Kap. 5.4],[2 Kap. 2.1],[12 Kap. 4],[14 Kap. 6.4.1,6.5],[31-43])

Durch den Bereich Big-Data wird es immer wichtiger, sehr große verteilte Datenmengen verarbeiten zu können. Da dies auch in angemessener Zeit erfolgen soll, ist es von Vorteil wenn die Datenverarbeitung parallel ausgeführt wird. Um dies umzusetzen wurde von Google das Map-Reduce Framework entwickelt.

Die Funktionsweise von Map-Reduce lässt sich in verschiedene Phasen untergliedern und ist in der folgenden Abbildung dargestellt.



(Eigenerstellung, vgl. [2 S. 17])

- Im ersten Schritt werden die Daten auf die verschiedenen Knoten im Netz verteilt, die auch die Map-Funktionen ausführen.
- Anschließend werden in den verschiedenen Prozessen parallel die vom Nutzer spezifizierte Map-Funktion ausgeführt.
- Die Ergebnisse der Map-Prozesse werden dann in unterschiedlichen Zwischenergebnissen gespeichert.
- Daraufhin wird für jedes Zwischenergebnis ein Reduce Prozess gestartet, die dann die vom Nutzer definierte Reduce-Funktion ausführen.
- Im letzten Schritt werden dann alle Ergebnisse der Reduce-Prozesse gespeichert.

Da das Framework die Aufteilung und Parallelisierung der Daten und Prozesse übernimmt, muss der Nutzer selbst nur die Map und Reduce Funktionen angeben. Diese spezifizieren dann die gewünschte Logik. Dabei wird mit Key-Value Wertpaaren gearbeitet. In der Map Phase können beispielsweise die Vorkommnisse einzelner Wörter gesammelt werden. In der Reduce Phase können dann Aggregationen durchgeführt werden, beispielsweise um Durchschnittswerte zu berechnen.

Es gibt einige NoSQL Datenbanksysteme, die das Map-Reduce Prinzip nutzen, um mit verteilten Datensätzen zu arbeiten. Häufig muss dabei keine Verteilung der Daten mehr stattfinden, da diese bereits auf verschiedene Knoten verteilt sind. Jeder Serverknoten kann daher für seine eigenen Datensätze die Map-Funktion ausführen.

In Memory Datenbanken (vgl. [8-11],[14 Kap. 8])

Klassischerweise werden Daten jeglicher Art meist auf Festplatten gespeichert. Dies ist auch generell bei Relationalen Datenbankmanagementsystemen der Fall, die Datenbank, bzw. die entsprechenden Daten werden auf Festplattenlaufwerke gespeichert. Es gibt jedoch auch eine andere Möglichkeit Daten zu speichern: Die Speicherung im

Arbeitsspeicher (RAM). Datenbanken die ihre Daten innerhalb des RAM speichern werden als In-Memory Datenbanken bezeichnet. Es gibt einige NoSQL Systeme, die diese Art der Datenhaltung nutzen.

Die Speicherung der Daten im Arbeitsspeicher hat den Vorteil, dass Zugriffsgeschwindigkeiten deutlich besser sind als im Vergleich zum Festplattenspeicher. Lese- und Schreiboperationen auf den Arbeitsspeicher sind deutlich schneller als auf Festplattenspeicher. Häufig finden In-Memory Datenbanken daher Anwendung im Big-Data Bereich, da dort mit sehr großen Mengen an Daten gearbeitet wird, die analysiert und ausgewertet werden müssen. Ein Nachteil ist im Allgemeinen jedoch die Tatsache, dass Arbeitsspeicher nur flüchtig ist. Ein Absturz des Systems führt daher dazu, dass alle Daten, die im Arbeitsspeicher gehalten wurden, verloren gehen. Eine dauerhafte Speicherung der Daten (Persistenz) ist somit nicht möglich. Es gibt jedoch Möglichkeiten, wie eine Persistente Datenhaltung trotzdem erreicht werden kann:

- Es können Schnappschuss-Dateien erstellt werden, die den Zustand der Datenbank zum Zeitpunkt des Schnappschuss vollständig erfassen. Diese Dateien können dann auf einem persistenten Speichermedium gespeichert werden. Der Nachteil bei dieser Variante ist, dass bei einem Systemabsturz alle Änderungen der Daten zwischen dem letzten Schnappschuss und dem Ausfall verloren gehen.
- Zusätzlich zum Anlegen von Schnappschüssen kann ein Protokoll geführt werden. Dieses speichert alle Änderungen, die an den Daten der Datenbank vollzogen werden. Somit kann bei einem Absturz mithilfe des Schnappschuss und des Protokolls der Zustand der Datenbank wiederhergestellt werden.
- Eine weitere Möglichkeit ist das Nutzen von nichtflüchtigem RAM Speicher (NVRAM). Die Datenhaltung im RAM wird dabei durch einen Energiespeicher, beispielsweise eine Batterie, sichergestellt, bis das System wieder am Laufen ist.

Da Arbeitsspeicher jedoch deutlich teurer ist im Vergleich zu Festplattenspeicher werden häufig auch Hybride Ansätze genutzt. Die Daten der Datenbank können dabei sowohl im Arbeitsspeicher, als auch auf der Festplatte gespeichert werden.

Spaltenorientierte Datenbanken (vgl. [1 Kap. 7.3],[2 Kap. 3],[12 Kap. 2],[13 Kap. 3], [14 Kap. 5.5,5.6],[72,98-106])

Spaltenorientierte Datenbanken (auch Wide-Column-Stores genannt) speichern ihre Daten, vereinfacht gesagt, nicht wie relationale Datenbanken in Zeilenform, sondern in Spaltenform. Eine relationale Datenbank würde Datensätze beispielsweise auf folgende Weise speichern:

- 1, Peter, 40000
- 2, Maria, 45000
- 3, Dieter, 35000

Der erste Wert stellt dabei eine ID dar, der zweite den Namen der Person und der dritte das Gehalt.

Eine spaltenorientierte Datenbank würde die gleichen Datensätze auf folgende Weise speichern:

- 1, 2, 3
- Peter, Maria, Dieter
- 40000, 45000, 35000

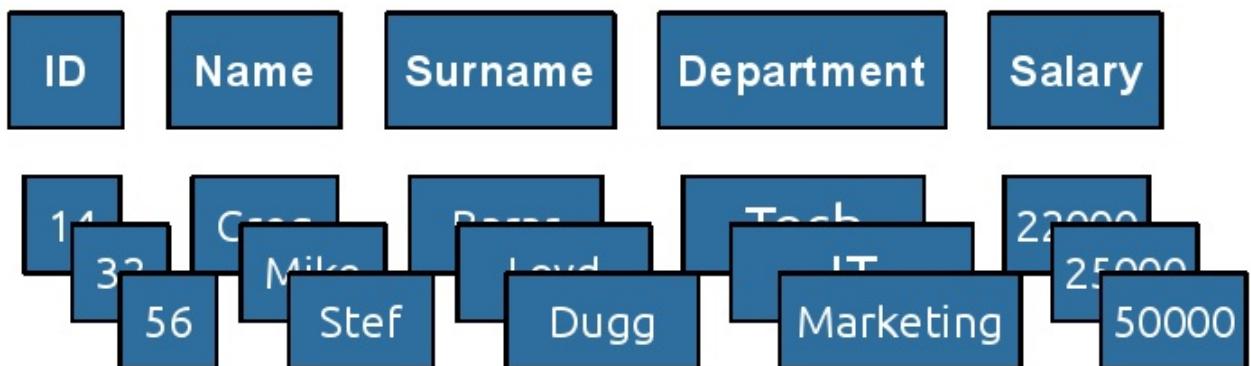
Dies ist zwar nur ein sehr vereinfachtes Beispiel, da die tatsächliche Art der Speicherung von spaltenorientierten Datenbanken anders umgesetzt ist, es zeigt jedoch direkt den Nutzen einer solchen Form der Datenstrukturierung. Würde man beispielsweise das Durchschnittsgehalt aller Personen berechnen wollen, so wird bei relationalen Datenbanken zuerst der erste Datensatz ausgewählt werden und in diesem dann der dritte Wert, der das Gehalt beschreibt. Dasselbe müsste für den zweiten und dritten Datensatz geschehen. Bei spaltenorientierten Datenbanken müsste nur der dritte Datensatz ausgewählt und all seine Werte gelesen werden. Statt auf drei muss also nur auf ein Datensatz zugegriffen werden, was deutlich schneller ist.

Funktionsweise

Die tatsächliche Art der Speicherung der Daten unterscheidet sich teilweise bei den verschiedenen Anbietern. Im Allgemeinen gibt es jedoch Spalten, die einen Namen besitzen, Daten halten und einen Zeitstempel haben. Die Struktur einer Spalte ist in folgender Abbildung dargestellt [99].



Spalten die ähnliche oder verwandte Inhalte speichern können in einer Column Family zusammengefasst werden. Beispielsweise könnten alle Spalten, die eine Person beschreiben, Teil einer Column Family werden, was in folgender Abbildung abgebildet ist [99].



[99]

Eine solche Column Family hat dabei keine logische Struktur und kann sogar aus Millionen Spalten bestehen.

Vorteile

Wie bereits durch das Beispiel deutlich wurde, sind spaltenorientierte Datenbanken besonders dafür geeignet Aggregate von Daten zu bilden, wie beispielsweise der Durchschnittswert oder Minima/Maxima. Dadurch, dass die Daten einer Spalte zusammen gespeichert werden, ist der Zugriff darauf bei spaltenorientierten Datenbanken deutlich schneller als bei relationalen Datenbanken. Auch das Ändern von allen Daten einer Spalte, beispielsweise bei einer allgemeinen Gehaltserhöhung ist deutlich effizienter. Des Weiteren lassen sich Daten in einer spaltenorientierten Datenbank leichter komprimieren. Allgemein sind sie also gut für Einsatzzwecke wie analytische Informationssysteme und Finanzberichte.

Nachteile

Der Nachteil dieser Datenbanken ist jedoch, dass Zugriffe auf alle Attribute eines Objekts, also eine Zeile, dementsprechend deutlich langsamer sind. Das Hinzufügen neuer Datensätze, beispielsweise ein neuer Mitarbeiter, ist bei spaltenorientierten Datenbanken deutlich weniger effizient als bei relationalen Datenbanken.

Anbieter [107]

Die am häufigsten genutzten Spaltenorientierten Datenbanken laut db-engines.com (Stand Juli 2017)

- Cassandra: 124,12 Punkte
- HBase: 63,62 Punkte
- Microsoft Azure Cosmos DB: 7,71 Punkte

Dokumentenorientierte Datenbanken (vgl. [1 Kap. 7.4],[2 Kap. 4],[13 Kap. 3],[14 Kap. 5.8],[72,108-114])

Diese Art der NoSQL Datenbanken speichern ihre Daten in Dokumenten ab, beispielsweise in Form von JSON, XML aber auch in beliebig anderer Form. Dabei sind diese Dokumente Schemafrei, das heißt es existieren keine Regeln nach denen der Inhalt dieser Dokumente aufgebaut werden muss. Jedes einzelne Dokument ist also in der Lage, einen völlig anderen Inhalt zu speichern. Häufig werden diese Datenbanken im Bereich der Web-Applikationen verwendet und bei unstrukturierten Daten.

Funktionsweise

In den meisten Fällen bestehen die Dokumente aus einer Sammlung von Key-Value Paaren. Die entsprechenden Values müssen dabei nicht atomar sein, sondern können auch aus Arrays, Listen oder ganzen weiteren Objekten bzw. Datensätzen bestehen. Das folgende Beispiel zeigt ein Dokument im JSON Format.

```
{  
  "Vorname": "Dieter",  
  "Nachname": "Skolmsund",  
  "Gehalt": "40000",  
  "Bisherige Positionen": ["Gärtner", "Entwickler", "Abteilungsleiter"],  
  "Kinder":  
    [  
      {"Name": "Bjorn"},  
      {"Name": "Freya"}  
    ]  
}
```

In vielen Datenbanken wird zusätzlich eine ID für das Dokument gespeichert, diese kann manuell angegeben werden oder wird vom Datenbankmanagementsystem selbst angelegt.

Vorteile

Aufgrund der Schemafreiheit bieten dokumentorientierte Datenbanken eine große Freiheit bei der Gestaltung der Datenstrukturen. Änderungen können meist sehr einfach umgesetzt werden, gerade in der Entwicklungsphase sind sie daher äußerst praktisch. Des Weiteren können zusammenhängende Objekte bzw. Datensätze innerhalb eines einzelnen Dokuments gespeichert werden. Operationen die auf all diese Daten zugreifen müssen, sind daher schneller als bei relationalen Datenbanken, bei denen viele Join Befehle dafür nötig wären.

Nachteile

Zwischen den einzelnen dokumentorientierten Datenbankmanagementsystemen existieren viele Unterschiede, sodass ein Wechsel von einem Anbieter zu einem anderen nicht immer ganz einfach ist. Beispielsweise erlauben manche Anbieter das Speichern eines Dokuments in XML Format, andere wiederum nicht. Auch in den entsprechend genutzten Abfragesprachen gibt es häufig Unterschiede. Die Eigenschaft der Schemafreiheit kann unter Umständen auch als Nachteil angesehen werden. Anwendungsfälle, bei denen es auf eine Konsistente und strukturierte Speicherung der Daten ankommt, können mit dokumentorientierten Datenbanken nur deutlich schwerer gelöst werden als mit klassischen relationalen Datenbanken. Während bei relationalen Datenbanken ein Anwendungsprogramm keine Daten speichern kann, die nicht zum Schema passen, muss bei dokumentorientierten Datenbanken eine Überprüfung auf Programm Ebene erfolgen. Bei Fehlern in der Programmierung kann es also schnell dazu kommen, dass Daten in der Datenbank gespeichert werden, die an anderer Stelle nicht mehr korrekt interpretiert werden können.

Anbieter [115]

Die am häufigsten genutzten dokumentenorientierten Datenbanken laut db-engines.com (Stand Juli 2017)

- MongoDB: 332,77 Punkte
- Amazon DynamoDB: 36,46 Punkte
- Couchbase: 33,02 Punkte

Key-Value Datenbanken (vgl. [1 Kap. 7.2],[2 Kap. 5],[12 Kap. 6],[13 Kap. 3],[72,116-123])

Key-Value Datenbanken existieren schon seit den 70er Jahren und gehören somit zu den ältesten NoSQL Datenbanken. Durch das Internet und der damit einhergehenden Notwendigkeit mit extrem großen Datenmengen arbeiten zu können, haben diese Datenbanken einen neuen Aufschwung erlebt. Sie speichern ihre Daten in Form von Schlüssel-Wert Paaren, eine sehr einfache, aber dadurch auch sehr effiziente Form der Datenverwaltung.

Funktionsweise

Je nach Anbieter unterscheiden sich die Freiheiten beim Setzen der Schlüssel und Werte. Im Allgemeinen sind sie jedoch Schemafrei, die Schlüssel und die Werte können daher vollkommen beliebig aufgebaut sein. Häufig können Werte daher nicht nur einfache Strings halten, sondern auch Listen und Sets, also mehrere Werte. Auch Hashs können gespeichert werden, welche aus weiteren Key-Value Paaren aufgebaut sind.

Vorteile

Der größten Vorteile von Key-Value Datenbanken sind ihre Skalierbarkeit und ihre Geschwindigkeit. Die Daten lassen sich sehr leicht verteilen und der Zugriff auf Werte über ihre Schlüssel erfolgt sehr schnell. Durch die Schemafreiheit können beliebige Strukturen aufgebaut werden, daher sind sie flexibel und lassen sich leicht anpassen. Oft werden diese Datenbanken auch als In-Memory Datenbanken verwendet, wodurch sie noch schneller werden.

Nachteile

Da der Zugriff auf Werte nur über deren Schlüssel erfolgen kann, sind die Abfragemöglichkeiten bei diesen Datenbanken deutlich eingeschränkt. Komplexe Abfragen müssen somit von den Entwicklern in ihrem Programmcode selbst erstellt werden. Des Weiteren lassen sich Objektbeziehungen nur sehr schwer darstellen. Integritätsbedingungen

dieser Beziehungen müssen immer im Programmcode definiert werden. Key-Value Datenbanken sind daher nicht dafür geeignet, Datensätze, mit vielen komplexen Beziehungen untereinander, zu speichern.

Anbieter [123]

Die am häufigsten genutzten Key-Value Datenbanken laut db-engines.com (Stand Juli 2017)

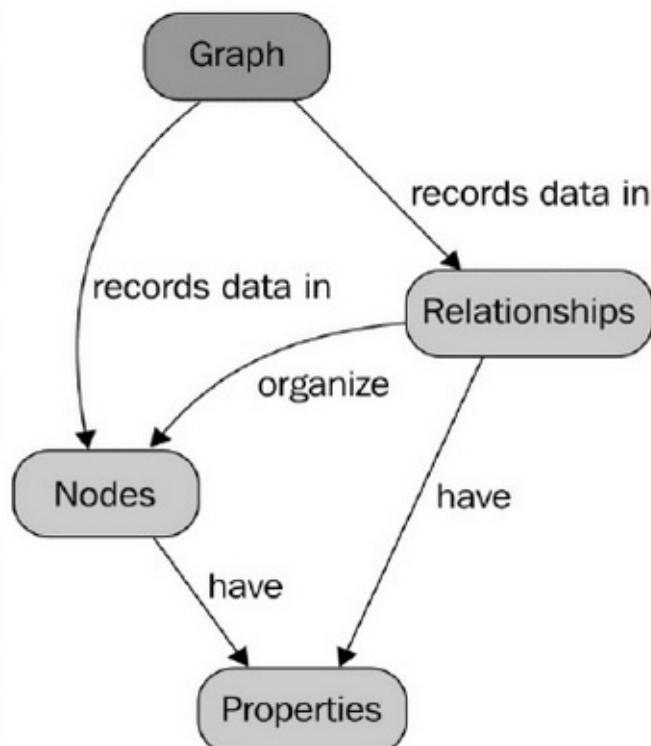
- Redis: 121,51 Punkte
- Memcached: 28,53 Punkte
- Hazelcast: 8,91 Punkte

Graphen Datenbanken (vgl. [1 Kap. 1.4,2.4,7.6],[2 Kap. 6],[12 Kap. 3],[13 Kap. 3],[72,89-96])

Graphen Datenbanken eignen sich dafür, Objekte und ihre komplexen Beziehungen untereinander zu speichern und zu verwalten. Die Daten werden dabei als Graphen gespeichert, Objekte werden als Knoten realisiert, Beziehungen über Kanten dargestellt. Sie eignen sich daher besonders gut für Daten mit sehr vielen Beziehungen und bei Graphen theoretischen Problemen. Anwendungsfälle sind beispielsweise Soziale Netzwerke, Empfehlungssysteme und Geoinformationssysteme.

Funktionsweise

Die meisten Graphen Datenbanken speichern ihre Daten in einem Property-Graphen. Dies ist ein Graph, der zusätzlich zu Knoten und Kanten (Beziehungen) auch beliebige Eigenschaften zu diesen Elementen speichern kann. Knoten und Beziehungen können somit durch Attribute weiter spezifiziert werden. Die nachfolgende Abbildung stellt grafisch dar, wie eine Graphen Datenbank aufgebaut ist [13].



[13]

Eine weitere wichtige Eigenschaft ist, dass es zwischen zwei Knoten auch mehrere Kanten geben kann. So lassen sich auch mehrere komplexe Verbindungen zwischen beliebigen Objekten, bzw. Knoten darstellen.

Die Funktionsweise von Graphen Datenbanken basiert dabei auf den mathematischen Eigenschaften der Graphentheorie.

Vorteile

Der größte Vorteil dieser Datenbanken besteht in ihrem speziellen Einsatzgebiet: Alles was sich als Graph abbilden lässt, kann mit deutlich besserer Performance in Graphen Datenbanken gespeichert und verwaltet werden, als in anderen Datenbanken. Alle Probleme die auf der Graphentheorie basieren, lassen sich leicht lösen, beispielsweise die Suche nach den kürzesten Wegen. Während bei relationalen Datenbanken dafür komplexe Joins verwendet werden müssen, können in Graphen Datenbanken durch einfaches Traversieren zwischen den Knoten diese Probleme schnell und einfach gelöst werden. Graphen Datenbanken eignen sich daher am besten für Datensätze, die sehr viele Beziehungen untereinander haben.

Nachteile

Ihre Spezialisierung auf Graphen ist allerdings auch ihr größter Nachteil. Sie eignen sich nicht dafür große Datensätze mit keinen oder wenigen Beziehungen untereinander zu verwalten, relationale Datenbanken können mit diesen Daten meist schneller und effizienter umgehen. Des Weiteren existieren noch keine standardisierte Abfragesprache, jeder Anbieter hat hier meist eigene Lösungen, was es nicht einfach macht, einen Anbieter zu wechseln.

Anbieter [97]

Die am häufigsten genutzten Graphen Datenbanken laut db-engines.com (Stand Juli 2017)

- Neo4j: 38,52 Punkte
- Microsoft Azure Cosmos DB: 7,71 Punkte
- OrientDB: 5,57 Punkte

Objektorientierte Datenbanken (vgl. [2 Kap. 8.5],[14 Kap. 2.11,7],[72-86])

Heutzutage werden sehr viele Programme mit dem Konzept der Objektorientierung programmiert, bekannte Sprachen sind dabei beispielsweise Java und C#. Daten werden dabei als Objekte betrachtet, mit denen gearbeitet werden kann. Relationale Datenbanken haben jedoch keine Möglichkeit diese Objekte einfach so zu speichern, da diese immer mit Tabellen arbeiten. Dieses Problem wird als Impedance Mismatch bezeichnet.

Impedance Mismatch (vgl. [70-71])

Impedance Mismatch (zu Deutsch: Objektrelationale Unverträglichkeit) beschreibt die Unverträglichkeit zwischen dem objektorientierten und dem relationalen Modell. Die Probleme sind dabei einerseits Unterschiede zwischen den Datentypen einer Datenbank und einer Programmiersprache. Der Datentyp String kann beispielsweise in einer Datenbank anders aufgebaut sein als in einer Programmiersprache und eventuell Beschränkungen in der Länge haben. Andererseits sind es vor allem die Unterschiede in den Datenstrukturen an sich. Objekte sind anders aufgebaut als Tabellen. Häufig müssen Daten von Objekten in verschiedene Tabellen aufgeteilt werden. Des Weiteren ist es nicht möglich objektorientierte Eigenschaften wie Vererbung, Objektidentität und das Objektverhalten in einer Relation zu speichern

Funktionsweise

Objektorientierte Datenbanken sind dabei eine Antwort auf das Problem des Impedance Mismatch. Diese Datenbanken speichern die Daten als Objekte gemäß der Objektorientierung. Objekte können somit direkt gespeichert werden, ohne dass ihre Daten erst aufgeteilt werden müssen. Dazu gehört auch die Objektidentifikation, es müssen für Objekte also keine zusätzlichen Primärschlüssel generiert werden, wie es bei relationalen Datenbanken der Fall ist. Dadurch, dass Objekte auch komplexe Datentypen halten können, sprich andere Objekte, lassen sich somit auch direkte Objektreferenzen speichern. Beziehungen über Primär- und Fremdschlüssele sind daher nicht notwendig. Auch lassen sich Vererbungshierarchien speichern und abbilden. Zusätzlich kann das Objektverhalten definiert und gespeichert werden, eine Eigenschaft, für die es im klassischen relationalen Modell kein Gegenstück gibt.

In vielen Objektorientierten Datenbanken wird als Abfragesprache OQL (Object Query Language) verwendet, eine Abfragesprache angelehnt an SQL. Häufig werden Anbindungen für die wichtigsten Objektorientierten Programmiersprachen angeboten.

Vorteile

Durch die komplette Speicherung der Daten als Objekte gemäß der Objektorientierung ist es deutlich einfacher Anwendungen zu programmieren, da keine Möglichkeit mehr gefunden werden muss, wie sich die Objekte am besten in Relationen abbilden lassen. Beim Arbeiten mit wenigen, aber sehr komplexen Objekten mit Verknüpfungen, sind Objektdatenbanken oft schneller als relationale, da keine großen Join Abfragen benötigt werden.

Nachteile

Der größte Nachteil objektorientierter Datenbanken ist ihre Verbreitung, bzw. das Fehlen davon. Obwohl die Entwicklung dieser Datenbanken bereits seit den 80er Jahren stattfindet, werden sie nicht sehr häufig genutzt und es gibt nur wenige Anbieter. Bedingt dadurch sind sie auch meist nicht so ausgereift, wie die älteren relationalen Datenbanken. Ihre Performance beim Arbeiten mit großen aber einfachen Datenmengen liegt deutlich unter der von relationalen Datenbanken.

Anbieter [87]

Die am häufigsten genutzten Objektorientierten Datenbanken laut db-engines.com (Stand Juli 2017)

- Caché: 2,72 Punkte
- Db40: 1,42 Punkte
- ObjectStore: 1,10 Punkte

Objektrelationale Datenbanken (vgl. [1 Kap. 6.6],[14 Kap. 7],[72-86])

Neben den Objektorientierten Datenbanken gibt es auch die sogenannten objektrelationalen Datenbanken. Auch diese haben zum Ziel das Problem des Impedance Mismatch anzugehen, allerdings streben sie eher eine Verminderung statt vollständiger Beseitigung des Problems an. objektrelationale Datenbanken versuchen die Vorteile von relationalen und objektorientierten Datenbanken zu vereinen. Dazu verwenden sie das relationale Grundgerüst und erweitern es um objektorientierte Features. Als Abfragesprache wird weiterhin das bekannte SQL genutzt. Die Forschung an objektrelationalen Datenbanken begann Anfang der 90er Jahren, erste kommerzielle Produkte gab es gegen Ende der 90er. Ihre Entwicklung wurde durch den SQL:1999 Standard und seine Structured Types vorangetrieben, welche es ermöglichen eigene komplexe Datentypen in Datenbanken zu definieren.

Funktionsweise

Objektrelationale Datenbanksysteme sind zum Großteil bestehende relationale Datenbanksysteme, die zusätzlich manche der folgenden objektorientierten Eigenschaften bieten:

- Komplexe Datentypen:

Neben den einfachen Datentypen wie Int und String sind auch komplexe wie Geldbeträge, Geometrie Daten und Dokumenttypen wie JSON und XML verwendbar.

- Benutzerdefinierte Datentypen:

Benutzer können eigene Datentypen spezifizieren und nutzen. Diese können aus einfachen, komplexen oder auch aus benutzerdefinierten Datentypen bestehen.

- Objekttabellen/Typisierte Tabellen:

Es lassen sich Tabellen anhand eines Datentyps erstellen. Die Tabelle hält dann Objekte/Datensätze des spezifizierten Typs. Die Attribute eines Datentyps werden dabei zu den Spalten der Tabelle.

- Objektidentifikation:

Objekte lassen sich durch einen Objektidentifikator identifizieren.

- Objektreferenzen:

Mithilfe der Objektidentifikatoren lassen sich Objekte einfach referenzieren.

Beziehungen über Primär- und Fremdschlüssel werden dadurch überflüssig.

- Methoden:

Es lassen sich Objektmethoden erstellen.

- Vererbung:

Es lassen sich Vererbungshierarchien zwischen Objekten herstellen.

Vorteile

Der Vorteil objektrelationaler Datenbanken ist, dass das Impedance Mismatch verringert wird und dennoch das relationale Modell verwendet werden kann. Es ist daher sehr einfach, bestehende relationale Systeme in objektrelationale Datenbanken zu migrieren, um die Vorteile der Objektorientierung zu nutzen.

Nachteile

Ein großer Nachteil der objektrelationalen Datenbanken ist der große Unterschied zwischen den einzelnen Anbietern. Nicht alle objektorientierten Features werden immer umgesetzt und auch generell gibt es große Unterschiede in der Umsetzung. Während Oracle beispielsweise fast alle objektorientierten Features umsetzt, bietet der Microsoft SQL Server nur benutzerdefinierte Datentypen an. Das Wechseln auf ein anderes objektrelationales Datenbankmanagementsystem ist daher in vielen Fällen nur schwer oder gar nicht möglich.

Anbieter [88]

Die am häufigsten genutzten relationalen Datenbanken mit objektorientierten Features laut db-engines.com (Stand Juli 2017)

- Oracle: 1374,88 Punkte
- Microsoft SQL Server: 1226,00 Punkte
- PostgreSQL: 369,44 Punkte

Fazit

NoSQL Datenbanken zeichnen sich durch große Vielfalt und Unterschiede aus. Es wurden die grundlegenden Theorien und Verfahren erläutert, die von NoSQL Datenbanken genutzt werden. Statt auf das ACID Transaktionskonzept setzen NoSQL Datenbanken meist auf BASE. Statt also die Konsistenz im CAP-Theorem als oberstes Ziel zu nehmen, begnügen sie sich meist mit eventueller, bzw. verzögerter Konsistenz. Dadurch, dass sie meist als verteilte Datenbanken im Cluster realisiert werden, sind sie optimiert für eine vertikale Skalierung. Mithilfe konsistenter Hashverfahren können sie ihre Daten optimal auf Shards aufteilen und auch Replikationen können damit effizient verteilt werden. Aufgrund der Verteilung setzen einige NoSQL Datenbanken auch auf Map-Reduce Verfahren, um Daten in angemessener Zeit verarbeiten zu können. Andere wiederum nutzen das In-Memory Verfahren und speichern ihre Daten im Arbeitsspeicher, um einen Geschwindigkeits-Gewinn beim Arbeiten mit den Daten zu erlangen.

Außerdem wurden die wichtigsten NoSQL-Arten erklärt und es wurde beschrieben, wie sie ihre Daten speichern. Neben den vier Kern NoSQL-Arten, Dokument-, Spalten-, Graph- und Key-Value-basiert wurden dabei auch Objekt und Objektrelationale Datenbanken erwähnt. Es hat sich gezeigt, dass alle Varianten sowohl Vorteile, als auch Nachteile besitzen und meist für bestimmte Anwendungsfälle besser geeignet sind als für andere. Des Weiteren wurde deutlich, dass auch innerhalb der Varianten Unterschiede zwischen den verschiedenen Anbietern existieren. Bei der Auswahl einer Datenbank sollte dementsprechend geprüft werden, welcher Anwendungsfall vorliegt und welche Besonderheiten die einzelnen Datenbanken der Anbieter haben, um eine entsprechend bestmögliche Datenbank auszuwählen.

Quellen NoSQL

- [1] Meier, Andreas; Kaufmann, Michael. "SQL- & NoSQL-Datenbanken", Springer Vieweg. (2016)
- [2] Edlich, Stefan; Friedland, Achim; Hampe, Jens; Brauer, Benjamin; Brückner, Markus. "NoSQL Einstieg in die Welt nichtrelationaler Web 2.0 Datenbanken", Carl Hanser Verlag GmbH & Co. KG. (2011)
- [3] http://lwibs01.gm.fh-koeln.de/wikis/wiki_db/index.php?n=Datenbanken.Skalierbarkeit (abgerufen am 22.06.2017)
- [4] <http://www.se.uni-hannover.de/pub/File/kurz-und-gut/ws2011-labor-restlab/RESTLab-Skalierbarkeit-Oliver-Beren-Kaul-kurz-und-gut.pdf> (abgerufen am 22.06.2017)
- [5] <https://de.wikipedia.org/wiki/Skalierbarkeit> (abgerufen am 22.06.2017)
- [6] http://www.it-administrator.de/themen/server_client/fachartikel/204189.html (abgerufen am 22.06.2017)
- [7] <http://habacht.blogspot.de/2007/11/das-uneindeutig-unklare-duo-scale-out.html> (abgerufen am 22.06.2017)
- [8] http://trends-in-der-it.de/?Fachartikel/In-Memory_-Technologie (abgerufen am 23.06.2017)
- [9] <http://www.datenbanken-verstehen.de/lexikon/in-memory-datenbank/> (abgerufen am 23.06.2017)
- [10] http://winfwiki.wi-fom.de/index.php/Beschreibung_und_Einsatz_von_In-Memory-Computing#In-Memory-Datenbanken (abgerufen am 23.06.2017)
- [11] <https://de.wikipedia.org/wiki/In-Memory-Datenbank> (abgerufen am 23.06.2017)
- [12] Celko, Joe. "Joe Celko's Complete Guide to NoSQL", Elsevier Science. (2013)
- [13] Vaish, Gaurav. "NoSQL Starter", Packt Publishing. (2013)
- [14] Lake, Peter; Crowther, Paul. "Concise Guide to Databases", Springer Verlag. (2013)
- [15] <https://www.bg.bib.de/portale/dab/Grundlagen/acid.html> (abgerufen am 23.06.2017)
- [16] <http://www.itwissen.info/ACID-atomicity-consistency-isolation-durability-ACID-Paradigmen.html> (abgerufen am 23.06.2017)
- [17] http://www.dbs.ethz.ch/education/timi/WS_03_04/Vorlesungsunterlagen/1-Einfuehrung.pdf (abgerufen am 23.06.2017)
- [18] <https://de.wikipedia.org/wiki/ACID> (abgerufen am 23.06.2017)
- [19] <http://datenbanken-verstehen.de/blog/datenbankentwicklung/datenbanktransaktionen-acid-prinzip/> (abgerufen am 23.06.2017)
- [20] <https://dbs.uni-leipzig.de/buecher/DBSI-Buch/HTML/kap13-2.html> (abgerufen am 23.06.2017)

- 23.06.2017)
- [21] <http://www.searchenterprisesoftware.de/definition/AKID-ACID> (abgerufen am 23.06.2017)
 - [22] <https://db-engines.com/de/article/ACID> (abgerufen am 23.06.2017)
 - [23] <http://www.norcom.de/de/fachartikel/datenbanktechnologien-acid-base-cap-und-google-spanner> (abgerufen am 23.06.2017)
 - [24] http://wikis.gm.fh-koeln.de/wiki_db/Datenbanken/CAP (abgerufen am 23.06.2017)
 - [25] <https://blog.codecentric.de/2011/08/grundlagen-cloud-computing-cap-theorem/> (abgerufen am 23.06.2017)
 - [26] <https://dzone.com/articles/better-explaining-cap-theorem> (abgerufen am 23.06.2017)
 - [27] <https://db-engines.com/de/article/CAP+Theorem> (abgerufen am 23.06.2017)
 - [28] <http://blog.nahurst.com/visual-guide-to-nosql-systems> (abgerufen am 23.06.2017)
 - [29] http://lwibs01.gm.fh-koeln.de/wikis/wiki_db/index.php?n=Datenbanken.BASE (abgerufen am 23.06.2017)
 - [30] <https://db-engines.com/de/article/BASE> (abgerufen am 23.06.2017)
 - [31] http://wikis.gm.fh-koeln.de/wiki_db/Datenbanken/MapReduce (abgerufen am 23.06.2017)
 - [32] <https://en.wikipedia.org/wiki/MapReduce> (abgerufen am 23.06.2017)
 - [33] <https://de.wikipedia.org/wiki/MapReduce> (abgerufen am 23.06.2017)
 - [34] <http://www.pal-blog.de/entwicklung/perl/einfach-erklaert-mapreduce-tutorial.html> (abgerufen am 23.06.2017)
 - [35] https://dbs.uni-leipzig.de/file/seminar_0910_koenig_ausarbeitung.pdf (abgerufen am 23.06.2017)
 - [36] https://dbs.uni-leipzig.de/file/seminar_0910_findling_K%C3%B6nig.pdf (abgerufen am 23.06.2017)
 - [37] <https://docs.mongodb.com/manual/core/map-reduce/> (abgerufen am 23.06.2017)
 - [38] <http://scienceblogs.de/diаксs-rake/2012/11/18/big-data-1-einfaches-word-count-beispiel-in-python/?all=1> (abgerufen am 23.06.2017)
 - [39] <http://www.searchenterprisesoftware.de/definition/MapReduce> (abgerufen am 23.06.2017)
 - [40] <https://nirajrules.wordpress.com/2013/05/31/big-data-nosql-and-mapreduce/> (abgerufen am 23.06.2017)
 - [41] https://de.slideshare.net/j_singh/nosql-and-mapreduce (abgerufen am 23.06.2017)
 - [42] Dean, Jeffrey; Ghemawat, Sanjay. "MapReduce: Simplified Data Processing on Large Clusters", Google, Inc. (2004)
 - [43] <https://db-engines.com/de/article/MapReduce> (abgerufen am 23.06.2017)
 - [44] <https://dba.stackexchange.com/questions/52632/difference-between-sharding-and-replication-on-mongodb> (abgerufen am 24.06.2017)
 - [45] <http://blog.zuehlke.com/skalierung-von-datenbanken/> (abgerufen am 24.06.2017)

- [46] <https://www.quora.com/What-is-the-difference-between-replication-partitioning-clustering-and-sharding> (abgerufen am 24.06.2017)
- [47] <https://www.gi.de/service/informatiklexikon/detailansicht/article/partitionierung-von-datenbanktabellen.html> (abgerufen am 24.06.2017)
- [48] <https://www.techopedia.com/definition/17/clustering-databases> (abgerufen am 24.06.2017)
- [49] <https://de.wikipedia.org/wiki/Rechnerverbund> (abgerufen am 24.06.2017)
- [50] http://lwibs01.gm.fh-koeln.de/wikis/wiki_db/index.php?n=Datenbanken.Replikation (abgerufen am 24.06.2017)
- [51] <http://www.itwissen.info/Replikation-replication.html> (abgerufen am 24.06.2017)
- [52] <http://www.datenbanken-verstehen.de/lexikon/replication-server/> (abgerufen am 24.06.2017)
- [53] http://lwibs01.gm.fh-koeln.de/wikis/wiki_db/index.php?n=Datenbanken.Sharding (abgerufen am 24.06.2017)
- [54] <https://db-engines.com/de/article/Sharding> (abgerufen am 24.06.2017)
- [55] <https://www.it-visions.de/glossar/alle/6301/Sharding.aspx> (abgerufen am 24.06.2017)
- [56] <https://www.toptal.com/big-data/consistent-hashing> (abgerufen am 25.06.2017)
- [57] <http://theory.stanford.edu/~tim/s16/l/l1.pdf> (abgerufen am 25.06.2017)
- [58] https://en.wikipedia.org/wiki/Consistent_hashing (abgerufen am 25.06.2017)
- [59] https://de.wikipedia.org/wiki/Konsistente_Hashfunktion (abgerufen am 25.06.2017)
- [60] <http://www.paperplanes.de/2011/12/9/the-magic-of-consistent-hashing.html> (abgerufen am 25.06.2017)
- [61] <http://www.mikeperham.com/2009/01/14/consistent-hashing-in-memcache-client/> (abgerufen am 25.06.2017)
- [62] <http://www.tom-e-white.com/2007/11/consistent-hashing.html> (abgerufen am 25.06.2017)
- [63] <http://www.tomkleinpeter.com/2008/03/17/programmers-toolbox-part-3-consistent-hashing/> (abgerufen am 25.06.2017)
- [64] <http://michaelnielsen.org/blog/consistent-hashing/> (abgerufen am 25.06.2017)
- [65] http://wikis.gm.fh-koeln.de/wiki_db/Datenbanken/ConsistentHashing (abgerufen am 25.06.2017)
- [66] http://lwibs01.gm.fh-koeln.de/wikis/wiki_db/index.php?n=Datenbanken.NoSQL (abgerufen am 27.06.2017)
- [67] <http://www.datenbanken-verstehen.de/lexikon/nosql/>
- [68] <http://nosql-database.org/> (abgerufen am 27.06.2017)
- [69] <https://db-engines.com/de/article/NoSQL> (abgerufen am 27.06.2017)
- [70] <http://www.agiledata.org/essays/impedanceMismatch.html> (abgerufen am 28.06.2017)
- [71] <http://www.odbms.org/wp-content/uploads/2013/11/023.01-Shusman-The->

- [Impedance-Mismatch-2002.pdf](#) (abgerufen am 28.06.2017)
- [72] https://dbs.uni-leipzig.de/file/seminar_1112_tran_ausarbeitung.pdf (abgerufen am 28.06.2017)
 - [73] <http://www.sts.tu-harburg.de/teaching/ws-98.99/DBIS/6-dbis.pdf> (abgerufen am 28.06.2017)
 - [74] http://pi.informatik.uni-siegen.de/lehre/1999w/1999w_proseminar/ausarbeitungen/thema7/Objektrelationale_Datenbanken.pdf (abgerufen am 28.06.2017)
 - [75] <http://wwwbayer.in.tum.de/lehre/WS2001/DBS-kossmann/folienD.pdf> (abgerufen am 28.06.2017)
 - [76] <http://www.itwissen.info/Objektrelationale-Datenbank-object-relational-database-ORD.html> (abgerufen am 28.06.2017)
 - [77] <http://www.itwissen.info/Objektorientierte-Datenbank-object-oriented-database-OODB.html> (abgerufen am 28.06.2017)
 - [78] http://lwibs01.gm.fh-koeln.de/wikis/wiki_db/index.php?n=Datenbanken.Objektdatenbank (abgerufen am 28.06.2017)
 - [79] http://lwibs01.gm.fh-koeln.de/wikis/wiki_db/index.php?n=Datenbanken.Objektrelationale-Datenbank (abgerufen am 28.06.2017)
 - [80] <http://www.datenbanken-verstehen.de/datenbank-grundlagen/datenbankmodell/objektorientiertes-datenbankmodell/> (abgerufen am 28.06.2017)
 - [81] <https://de.wikipedia.org/wiki/Objektdatenbank> (abgerufen am 28.06.2017)
 - [82] <https://db.in.tum.de/~grust/teaching/ss06/DB2/db2-02.pdf> (abgerufen am 28.06.2017)
 - [83] <http://www-lehre.inf.uos.de/~dbs/2005/skript/node154.html> (abgerufen am 28.06.2017)
 - [84] https://de.wikipedia.org/wiki/Objektrelationale_Datenbank (abgerufen am 28.06.2017)
 - [85] https://en.wikipedia.org/wiki/Object-relational_database (abgerufen am 28.06.2017)
 - [86] https://en.wikipedia.org/wiki/Object_database (abgerufen am 28.06.2017)
 - [87] <https://db-engines.com/de/ranking/object+oriented+dbms> (abgerufen am 01.07.2017)
 - [88] <https://db-engines.com/de/ranking/relational+dbms> (abgerufen am 01.07.2017)
 - [89] <https://db-engines.com/de/article/Graph+DBMS> (abgerufen am 29.06.2017)
 - [90] http://lwibs01.gm.fh-koeln.de/wikis/wiki_db/index.php?n=Datenbanken.Graphendatenbank (abgerufen am 29.06.2017)
 - [91] http://lwibs01.gm.fh-koeln.de/wikis/wiki_db/index.php?n=Datenbanken.Graphenmodell (abgerufen am 29.06.2017)
 - [92] https://en.wikipedia.org/wiki/Graph_database (abgerufen am 29.06.2017)
 - [93] <https://de.wikipedia.org/wiki/Graphdatenbank> (abgerufen am 29.06.2017)

- [94] <http://www.datenbanken-verstehen.de/lexikon/graphdatenbanken/> (abgerufen am 29.06.2017)
- [95] <http://www.itwissen.info/Graphen-Datenbank-graph-database.html> (abgerufen am 29.06.2017)
- [96] <https://neo4j.com/developer/graph-database/> (abgerufen am 29.06.2017)
- [97] <https://db-engines.com/de/ranking/graph+dbms> (abgerufen am 01.07.2017)
- [98] <http://www.datenbanken-verstehen.de/lexikon/spaltenorientierte-datenbanken/> (abgerufen am 30.06.2017)
- [99] http://wikis.gm.fh-koeln.de/wiki_db/Datenbanken/SpaltenorientierteDatenbank (abgerufen am 30.06.2017)
- [100] https://de.wikipedia.org/wiki/Spaltenorientierte_Datenbank (abgerufen am 30.06.2017)
- [101] https://en.wikipedia.org/wiki/Column-oriented_DBMS (abgerufen am 30.06.2017)
- [102] <http://www.itwissen.info/columnar-database-Spaltenorientierte-Datenbank.html> (abgerufen am 30.06.2017)
- [103] <http://eliteinformatiker.de/2011/08/07/zeilenorientierte-und-spaltenorientierte-datenbanken> (abgerufen am 30.06.2017)
- [104] <http://www.searchenterprisesoftware.de/definition/Spaltenorientierte-Datenbank> (abgerufen am 30.06.2017)
- [105] <https://www.gi.de/service/informatiklexikon/detailansicht/article/spaltenorientierte-datenbanken.html> (abgerufen am 30.06.2017)
- [106] <http://database.guide/what-is-a-column-store-database/> (abgerufen am 30.06.2017)
- [107] <https://db-engines.com/de/ranking/wide+column+store> (abgerufen am 01.07.2017)
- [108] http://wikis.gm.fh-koeln.de/wiki_db/Datenbanken/DokumentenorientierteDatenbank (abgerufen am 01.07.2017)
- [109] https://de.wikipedia.org/wiki/Dokumentenorientierte_Datenbank (abgerufen am 01.07.2017)
- [110] https://en.wikipedia.org/wiki/Document-oriented_database (abgerufen am 01.07.2017)
- [111] <http://eliteinformatiker.de/2011/05/18/nosql-document-store-couchdb-mongodb> (abgerufen am 01.07.2017)
- [112] <https://db-engines.com/de/article/Document+Stores> (abgerufen am 01.07.2017)
- [113] <http://database.guide/what-is-a-document-store-database/> (abgerufen am 01.07.2017)
- [114] <http://insights.dice.com/2012/06/04/nosql-document-storage-benefits-drawbacks/> (abgerufen am 01.07.2017)
- [115] <https://db-engines.com/de/ranking/document+store> (abgerufen am 01.07.2017)
- [116] http://wikis.gm.fh-koeln.de/wiki_db/Datenbanken/KeyValueSysteme (abgerufen am 01.07.2017)

am 02.07.2017)

- [117] <https://de.wikipedia.org/wiki/Schl%C3%BCssel-Werte-Datenbank> (abgerufen am 02.07.2017)
- [118] https://en.wikipedia.org/wiki/Key-value_database (abgerufen am 02.07.2017)
- [119] <http://www.datenbanken-verstehen.de/lexikon/key-value-datenbanksysteme/> (abgerufen am 02.07.2017)
- [120] <https://db-engines.com/de/article/Key-Value+Stores> (abgerufen am 02.07.2017)
- [121] <http://eliteinformatiker.de/2011/06/01/nosql-key-value-datenbanken-memcachedb-project-voldemort-redis> (abgerufen am 02.07.2017)
- [122] <http://database.guide/what-is-a-key-value-database/> (abgerufen am 02.07.2017)
- [123] <https://db-engines.com/de/ranking/key-value+store> (abgerufen am 01.07.2017)

Verwendete Programme für die Abbildungen:

- UMLet: <http://www.umlet.com/>
- Microsoft Word: <https://products.office.com/de-de/word>

Single Page Application

Autor: Alexander Schwietert

Single Page Application

Eine Single Page Application ist eine Web-Applikation, die lediglich aus einem einzigen HTML Dokument besteht und soll dadurch eine Nutzererfahrung ähnlich der einer Desktop-Applikation bieten. Der komplette Seiteninhalt (HTML, Javascript und CSS) wird dabei entweder beim initialen Laden der Seite übermittelt oder einzelne logische Seiten innerhalb der SPA dynamisch nachgeladen, in Abhängigkeit mit den Aktionen des Nutzers. Mit SPAs wird die Grundlage für eine Rich-Client bzw. Fat-Client-Architektur geboten, in der ein Großteil der Ausführung einer Webanwendung auf Clientseite geschieht und somit die Serverlast reduziert wird.

Eigenschaften

- Der Sitzungszustand wird bei dem Client gespeichert, der Server stellt hingegen lediglich noch die Nutzdaten zur Verfügung. Ein erneutes Laden der SPA ist mit einem Neustart einer Anwendung gleichzusetzen.
- Serverseitige Funktionalitäten daher Zustandslos
- "offline-friendly", Nutzungsdaten können im local storage zwischengespeichert werden, die SPA kann so auch bei Verbindungsverlust weiter laufen
- WebClient fungieren als unabhängige Einheit mit verschiedenen Services und kann somit selbstständig auf Benutzerinteraktionen reagieren. Dadurch wird der Datenverkehr zum Server verringert
- WebClient und Serverseite können separat entwickelt werden, lediglich die Schnittstellen für die Services müssen vorher definiert werden.

Entwurfsmuster

Entwurfsmuster beschreiben einen abstrakten Lösungsansatz für Problemstellungen in der Softwarearchitektur und -entwicklung, der für ein individuelles Problem entsprechend wiederverwendet werden kann.

Model-View-Controller

Das MVC Muster beschreibt die Trennung einer Software-Architektur in drei Komponenten. Das Datenmodell (Model), die Präsentation (View) und die Programmsteuerung / Logik (Controller). (*Lahres 2006*)

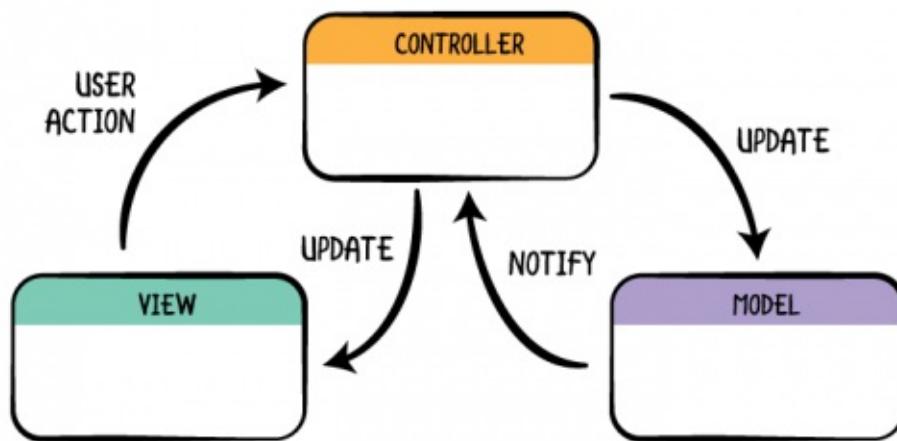


Abbildung 1: MVC (Peres 2016)

Model

Das Model enthält die darzustellenden Daten und ist von den anderen beiden Schichten unabhängig. Das Model publiziert entsprechende Änderungen an seinen Daten, welche von den lauschenden Komponenten registriert werden. Dies geschieht nach dem sogenannten Beobachter-Muster.

View

Die View Komponente subscribed auf die Daten des Models und updated seine Sichten entsprechend der Daten, sollten sie geändert werden. Mit der View Komponente werden die Benutzerinteraktion realisiert. Diese kennt ebenfalls den Controller, ist jedoch nicht für die Weiterverarbeitung der Daten zuständig.

Controller

Der Controller enthält in der Regel die Geschäftslogik (diese kann auch direkt im Model liegen, je nach Programmiersprache und Implementierung des MVC) und kann mehrere Views verwalten, wobei zu jeder View eine Steuerung existiert. Der Controller sorgt dafür, dass die Benutzerinteraktionen in der View ihre entsprechende Wirkung haben, verändert gegebenenfalls die View und gibt z.B. eingegebene Daten aus der View weiter an das Model.

Model-View-Presenter

Das MVP-Muster ist aus dem MVC-Muster heraus entstanden und setzt auf einen neuen Ansatz, um das Model komplett von der View zu trennen und über einen sogenannten Presenter zu verbinden. Dies soll eine bessere Testbarkeit als auch eine bessere Übersicht bei der Implementierung zur Folge haben. (Fowler 2006a)

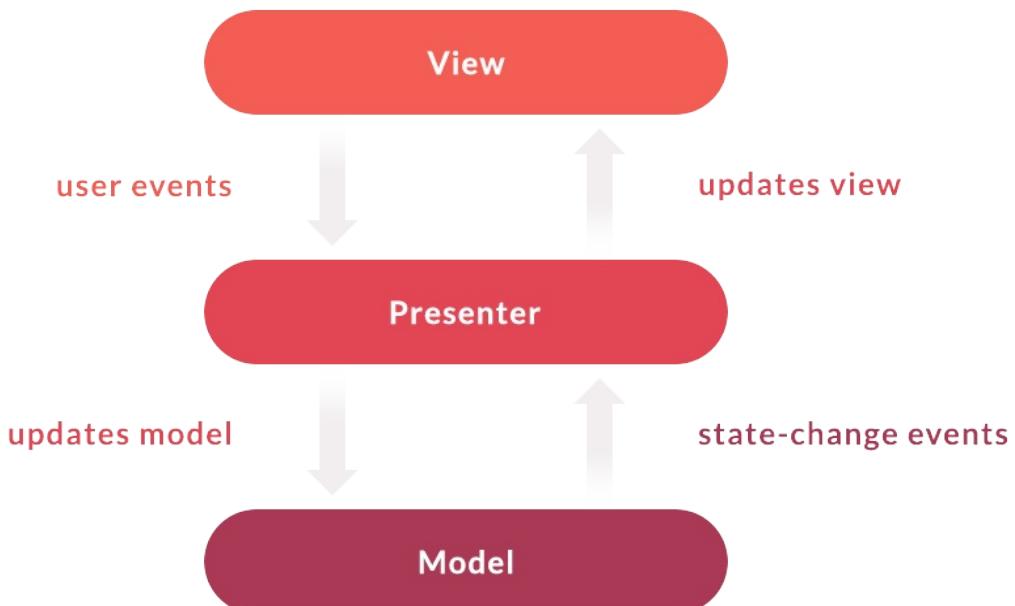


Abbildung 2: MVP (Ziomacki 2016)

Model

Das Modell stellt die Logik der Ansicht bzw. die Geschäftslogik dar und kennt dabei weder die View noch den Presenter. Es muss jedoch alle Funktionalitäten implementiert haben um die View betreiben zu können. Die Steuerung des Models erfolgt durch den Presenter.

View

Die View kennt ebenfalls weder das Model noch den Presenter und ist nur für die Ein- und Ausgabe von Daten zuständig, sowie die allgemeine Darstellung der Software. Die Steuerung der View erfolgt durch den Presenter.

Presenter

Der Presenter ist das Bindeglied zwischen View und Model, welche jeweils ihre Schnittstellen definieren, die anschließend vom Presenter miteinander verknüpft werden. Die Schnittstellen beschreiben dabei den genauen Aufbau der anderen zwei Komponenten. Dies resultiert in einer vollkommenen Austauschbarkeit von View und Model.

Supervising Controller

In einer Abwandlung des MVP-Musters kann die View zusätzlich für die Datensynchronisation zuständig sein, wodurch der Presenter sich lediglich um alle anderen Abläufe zwischen Model und View kümmert. Der Presenter kann zum data-binding hierbei Klassen zur Verfügung stellen, die View als auch Model bekannt sind, wodurch weiterer Synchronisationsaufwand entfällt und bei veränderten Daten die View direkt ihre Sicht aktualisiert. (Fowler 2006b)

Model-View-ViewModel

Das MVVM-Muster ist eine weitere abgewandelte Variante des MVC. Es ist eine Spezialisierung des MVP-Musters und kommt ebenfalls ohne Controller aus, da auch hier Zustand und Verhalten die Presenter bzw. ViewModel Komponenten regeln. Die Testbarkeit wird verbessert und die Arbeitsteilung zwischen UI-Designer und Programmierern gefördert, wodurch separate Teams gleichzeitig an einer Software entwickeln können. (Kühnel 2013)

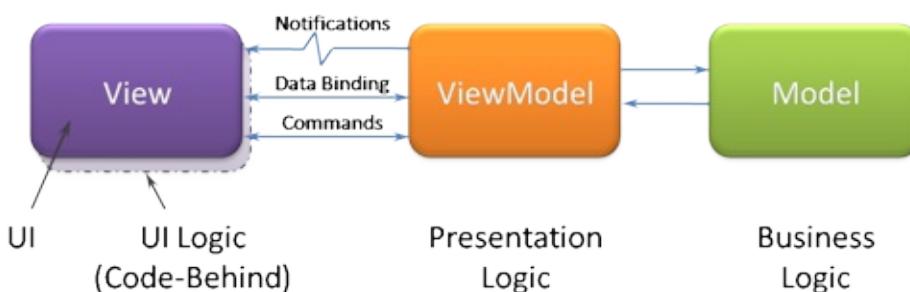


Abbildung 3: MVVM [Hill2009]

Model

Das Model hält die komplette Geschäftslogik sowie die dem Nutzer präsentierten Daten inne, welche nach Manipulation des Nutzers aktualisiert und validiert werden können.

View

Die View enthält alle Elemente zur Präsentation von Software und Daten. Sie bindet sich an Funktionen und Objekte des ViewModel um Daten darzustellen und Interaktionen der Nutzer weiterzugeben.

ViewModel

Das ViewModel dient als Bindeglied zwischen View und Model, wobei das ViewModel die View nicht kennt. Es stellt der View öffentliche Methoden und Eigenschaften zur Verfügung, über die die View Daten holen kann und Ereignisse in der View weiter an das ViewModel gibt. Das ViewModel wiederum ruft Methoden und Elemente des Models aus um Informationen auszutauschen.

Flux

Flux ist eine Architektur die erstmals 2013 von Facebook in Zusammenhang mit react.js vorgestellt wurde. Bei react bestehen Komponenten aus HTML, JavaScript Code als auch CSS und es herrscht somit keine Trennung von View und Model, wie man es aus den vorherigen Mustern kennt. (*Deutsch 2015*)

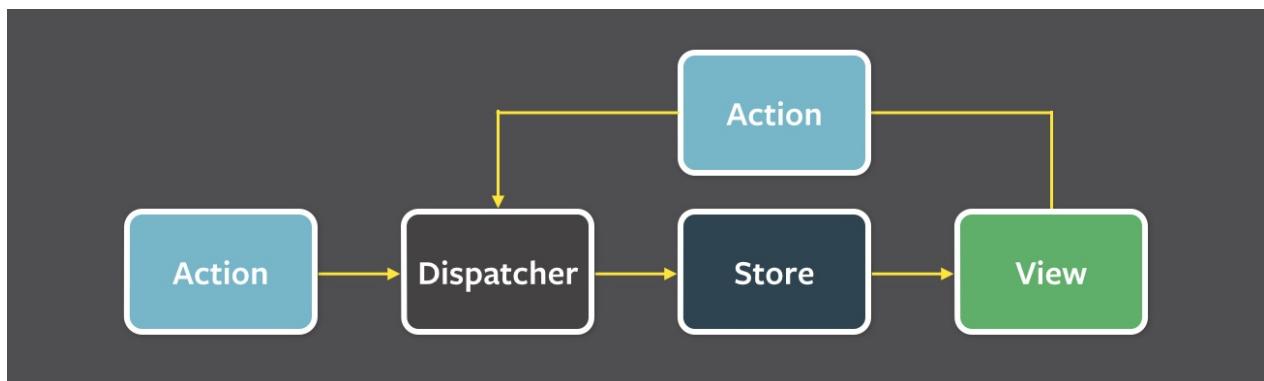


Abbildung 4: Flux (Zeigermann 2015)

Action

Eine Action ist ein normales JavaScript Objekt, welches aus seinem Payload (beliebig viele Attribute / Daten) sowie dem festen Attribut "type" besteht, welches in der Regel ein konstanter String ist. Über das type-Attribut können die Stores erkennen, ob es sich um eine für sie relevante Aktion handelt.

Sogenannte ActionCreators werden aus den Views (React Komponenten) heraus aufgerufen, um eine Action zu erstellen und diese zum Dispatcher weiter zu reichen, welches sie den jeweiligen Stores zuordnet. Die Views wiederum sind mit den Stores verbunden und werden bei Veränderungen der Daten im Store durch Actions neu gerendert.

Dispatcher

Der Dispatcher ordnet lediglich die Action Objekte einem Store zu, dabei behandelt er immer nur ein Objekt gleichzeitig.

Store

In Stores werden die Logik einer Applikation verwaltet und fungieren als Datenquellen für die Komponenten. Sie empfangen Actions, welche die Daten der Stores verändern können, woraufhin ein Store seine React Komponenten benachrichtigt.

View

Die React Komponenten subscriben auf ihre Stores und rendern sich bei Veränderungen gegebenenfalls neu. Es wird zwischen einfachen und smarten Komponenten unterschieden.

Einfache Komponenten sind abstrakter und leichter wieder zu verwenden, erhalten ihre Daten und selbst ausgelöste Actions als Properties.

Smarte Komponenten wiederum haben Funktionen um auf Stores zu subscriben und Actions zu dispatchen.

Da der Datenfluss von den Views über Actions und Dispatcher zu den Stores verläuft, spricht man auch von einem **Single-Directional-Dataflow**.

Quellen

1. Marlon Brüntje . Single-Page-Application - individuelle Web-Anwendung für Unternehmen. <http://www.flyacts.com/blog/single-page-application-individuelle-web-anwendung-fuer-unternehmen>, 2015. Aufgerufen: 25.06.2017 14:00
2. Rui Peres. Model View Controller in iOS: A modern approach. <https://www.raywenderlich.com/132662/mvc-in-ios-a-modern-approach>, 2016. Aufgerufen: 01.07.2017 13:00
3. Bernhard Lahres, Gregor R. Rheinwerk Computing :: Praxisbuch Objektorientierung – 8.2 Die Präsentationsschicht: Model, View, Controller (MVC). http://openbook.rheinwerk-verlag.de/oo/oo_06_moduleundarchitektur_001.htm, 2006. Aufgerufen: 01.07.2017 12:45
4. Martin Fowler. Supervising Controller. <https://martinfowler.com/eaaDev/SupervisingPresenter.html>, 2006a. Aufgerufen: 01.07.2017 13:20
5. Martin Fowler. GUI Architecture. <https://martinfowler.com/eaaDev/uiArchs.html>, 2006b. Aufgerufen: 01.07.2017 13:10
6. Piotr Ziomański. Model-View-Presenter Architecture in Android Applications | Macoscope Blog. <http://macoscope.com/blog/model-view-presenter-architecture-in-android-applications/>, 2016. Aufgerufen: 01.07.2017 13:40
7. Andreas Kühnel. Rheinwerk Computing :: Visual C# 2012 - 28 WPF-Commands. http://openbook.rheinwerk-verlag.de/visual_csharp_2012/1997_28_005.html, 2013. Aufgerufen: 01.07.2017 14:10
8. David Hill. The ViewModel Pattern. <https://blogs.msdn.microsoft.com/dphill/2009/01/31/the-viewmodel-pattern>, 2009. Aufgerufen: 01.07.2017 14:30
9. Sebastian Deutsch. Die Flux Architektur und React. <http://reactjs.de/posts/die-flux-architektur-und-react>, 2015. Aufgerufen: 01.07.2017 16:00
10. Oliver Zeigermann. Flux – Ein Versprechen an UI-Architekten. <http://www.embarc.de/flux-architektur-webanwendungen>, 2015. Aufgerufen: 01.07.2017 17:00
11. Angular. Angular - Architecture Overview. <https://angular.io/guide/architecture>, 2017a. Aufgerufen: 02.07.2017 12:00

12. Angular. Angular - Template Syntax. <https://angular.io/guide/template-syntax>, 2017b.
Aufgerufen: 02.07.2017 15.00
13. Angular. Angular - Lifecycle Hooks. <https://angular.io/guide/lifecycle-hooks>, 2017c.
Aufgerufen: 02.07.2017 16:23
14. Angular. Angular - Pipes. <https://angular.io/guide/pipes>, 2017d. Aufgerufen: 02.07.2017
17:15
15. Angular. Angular - HTTP. <https://angular.io/guide/http>, 2017e. Aufgerufen: 02.07.2017
18:00
16. Angular. Angular - Router. <https://angular.io/guide/router>, 2017f. Aufgerufen: 02.07.2017
20:00

Hybride_AppEntwicklung

Autor: Benjamin Schmidt

2. Unterschiede zwischen Web-Apps, Native Apps und Hybride Apps

2.1 Web-Apps

Web-Apps werden mit JavaScript, HTML und CSS entwickelt. Web-Apps laufen im Browser, da Web-Apps Mobile/Responsive Seiten sind. Sie passen sich jeder Größe genau an und sind plattformunabhängig, da jedes Smartphone einen eigenen Browser mit sich bringt. Sie können aber nicht im App Store veröffentlicht werden und unterstützen keine nativen Gerätefunktionen wie z.B. die Benutzung der Kamera oder das Auslesen der Kontakte. [1]

2.2 Native Apps

Native Apps werden mit plattformabhängigen Programmiersprachen wie Java für Android, Objective-C für IOS usw. entwickelt. Dadurch können die Apps im App Store der jeweiligen Plattform angeboten und heruntergeladen werden. Native Apps unterstützen die nativen Gerätefunktionen z.B. die Benutzung der Kamera im Vergleich zu Web-Apps. Sie sind aber nicht plattformunabhängig wie Web-Apps, da Sie für jede Plattform (Android, IOS usw.) neu entwickelt werden. Denn jede Plattform hat ihre eigene Programmiersprache. Die Wartung ist hoch, denn jede einzelne App muss getestet und gewartet werden und Updates müssen einzeln angepasst werden. Es gibt viele verschiedene Geräte z.B. Smartphones und Tablets für die auch einzelnen Apps realisiert werden. [2]

2.3 Hybride Apps

Hybride Apps kombinieren die Vorteile beider vorherigen App Varianten in einer. Hybride Apps werden genau wie Web-Apps mit JavaScript, HTML und CSS entwickelt. Sie können aber im App Store veröffentlicht und heruntergeladen werden wie Native Apps. Sie laufen in einem spezifischen Browser der jeweiligen Plattform, unterstützen aber auch native Gerätefunktionen der gewählten Plattform. Sie haben nicht so eine gute Performanz wie Native Apps, bei Spielen kann es daher zu Leistungseinbrüchen kommen, da empfiehlt sich lieber native Apps zu entwickeln. Ein großer Vorteil von hybriden Apps ist, wenn sie einmal entwickelt und getestet wurden, können Sie für jede Plattform erstellt werden. Man spart

sich die Zeit mehrere Apps für jedes System einzeln zu entwickeln. Dadurch ist die App leichter zu warten und auch Updates können besser eingespielt werden, da sie auf einer App basieren. [3]

Die folgende Tabelle zeigt die Unterschiede für die einzelnen App-Varianten im Vergleich zu den anderen.

App-Varianten	Web-Apps	Native Apps	Hybride Apps
Performanz der App	Sehr gut	Sehr gut	Beim Start dauert es, da die App erst geladen werden muss.
Plattformabhängigkeit	Nein	Ja	Ja
App-Store Veröffentlichung	Kann nicht im App-Store veröffentlicht werden.	Kann im App-Store veröffentlicht werden.	Kann im App-Store veröffentlicht werden.
Unterstützt Gerätefunktionen	Es werden nur einige Gerätefunktionen mit der Nutzung von Web-APIs unterstützt.	Es werden alle Gerätefunktionen unterstützt.	Es werden alle Gerätefunktionen unterstützt, durch eigene und externe Plug-Ins.

In der Tabelle sieht man in der Spalte für hybride Apps, dass es nur Leistungseinbrüche bei der Performanz der App gibt. Das merkt man aber auch beim direkt Start einer hybriden App sehr genau.

Hier sieht man eine Grafik, welche die internen Abläufe zeigen. Man sieht, dass eine Native App und eine Web-App sich darin unterscheiden, dass die Web-App in einem Browser ausgeführt wird. Der Browser wird vom Betriebssystem aufgerufen. Das ist auch der Grund, dass man keine Gerätefunktionen bei Web-Apps verwenden kann, da der Browser vom System abgekapselt ist. Wäre das nicht der Fall, dann könnte man mit einer einfachen Webseite auf alle Informationen eines PCs zugreifen und manipulieren. Das hat Sicherheitsgründe. Für einen Angreifer wäre das ein leichtes Spiel sich alle Kontakte und sonstige Informationen von einem Smartphone zu beschaffen. Bei nativen Apps ist das möglich Gerätefunktionen zu nutzen, da native Apps auf dem Betriebssystem installiert werden. Sie greifen direkt auf die System internen Prozesse und Daten zu. Eine hybride App muss beides kombinieren, dazu wird ein Browser simuliert, welches auf dem Betriebssystem installiert wird. Dieser spezifische Browser besteht aus einer Web View zum Anzeigen des HTML-Codes und einer nativen Schnittstelle. Die native Schnittstelle erlaubt native Zugriffe auf die Gerätefunktionen und stellt die native UI zur Verfügung, damit die hybride App wie eine native App aussieht.

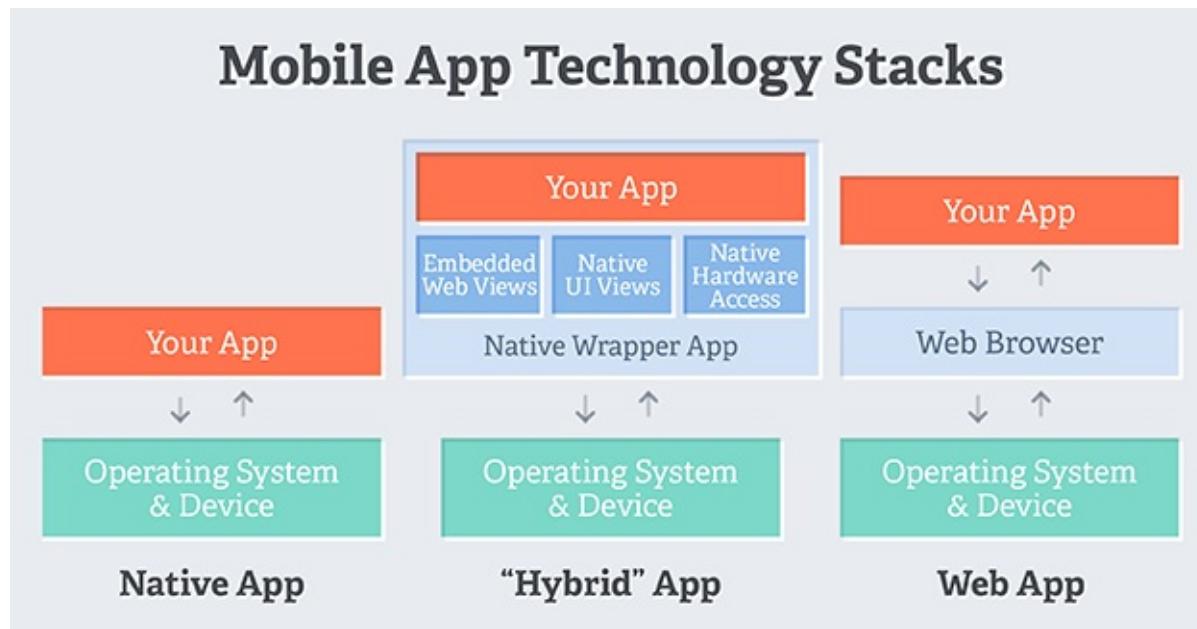


Abbildung 1: Der interne Ablauf einer App. [8]

Kommunikation in verteilten Systemen

Autor: Timo Rolfsmeier (Einführung, RPC, REST, GraphQL und Fazit) bzw. Tolga Aydemir (WebSocket)

Einführung

Ein verteiltes System ist als ein Verbund von Rechenknoten definiert, der zur Erreichung eines übergeordneten Ziels dienen soll. Die Instanzen innerhalb des Verbunds sind jeweils für einzelne Teifunktionen verantwortlich und kommunizieren zwecks Informationsaustausch über ein Netzwerk. Die Gründe, warum verteilte Systeme errichtet werden, sind vielfältig. Zum einen ist die Flexibilität zu nennen. Ein verteiltes System kann dynamisch um neue Rechenknoten erweitert bzw. reduziert werden, ohne dass die bestehenden Instanzen hiervon betroffen sind. Weiterhin erlauben verteilte Systeme die Trennung von Rechenknoten über Orts- und Organisationsgrenzen hinweg. Es kann somit dynamisch auf die Informationen von entfernten Hard- und Softwarekomponenten zugegriffen werden, ohne diese auf dem eigenen System redundant zu implementieren bzw. tiefere Kenntnisse von ihnen zu haben. Auch die Performanz spielt für diese Architekturform eine zentrale Rolle. Ein Rechenknoten kann beispielsweise mehrfach instanziert werden, um Aufgaben zu parallelisieren und somit für eine bessere Lastverteilung zu sorgen. Schließlich reduziert sich auch das Risiko für einen Dienstausfall bzw. Systemfehler, wenn Rechenknoten redundant und dezentralisiert vorliegen. (vgl. Schill und Springer 2012, S. 3-6)

Im Gegensatz zu einer monolithischen Architektur ist ein verteiltes System auf eine interne Kommunikation angewiesen. Da Rechenknoten sowie deren Speicher physikalisch voneinander getrennt sind, müssen Informationen innerhalb von Nachrichten übermittelt werden. Um nicht zu einem Flaschenhals für das Gesamtsystem zu werden, ist der Datenaustausch möglichst performant, sicher und flexibel zu gestalten. Innerhalb dieses Kapitels werden verschiedene Methoden und Ansätze vorgestellt, welche eine Lösung bezüglich dieser Problematik bieten. Konkret wird dabei auf die Technik RPC (Remote Procedure Call), das Netzwerkprotokoll WebSocket, den Architekturstil REST (Representational State Transfer) und die Abfragesprache GraphQL eingegangen.

RPC

RPC (Remote Procedure Call) ist eine Basistechnik zur Kommunikation innerhalb von heterogenen Anwendungslandschaften. Grundgedanke von RPC ist es, Teile eines Kontrollflusses an externe Rechenknoten mit anderen Adressräumen auszulagern (vgl. Schill 2012, S.46). Das Prinzip wurde 1976 veröffentlicht und befindet sich seit 1988 im RFC-Standard. Im gleichen Jahr erfolgte die erstmalige Implementierung von RPC durch das ONC RPC (Open Network Computing RPC), welches von Sun Microsystems entwickelt wurde.

Ablauf

Der Ablauf eines entfernten Methodenaufrufs gliedert sich in zwei verschiedene Abschnitte. Bevor der Client einen externen Server mit der Abarbeitung seines Kontrollflusses beauftragen kann, ist ein passender Service zu finden. Der Bindevorgang sowie der nachgelagerte Methodenaufruf werden im Folgenden nach (Schill 2012, S.47-51) beschrieben.

Bindevorgang

In komplexen Netzwerken hat ein Client typischerweise keine Kenntnis von der Gesamtheit der erreichbaren Serverknoten. Es liegen ihm dementsprechend keine Informationen darüber vor, welche Services von Servern angeboten werden bzw. mit welcher Syntax und

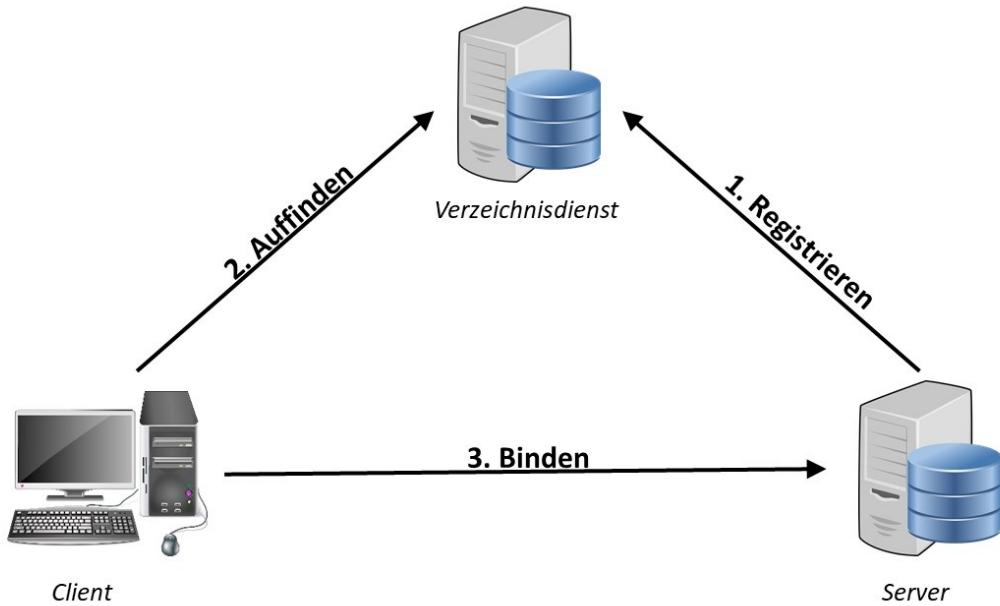


Abbildung 1: Bindevorgang mithilfe von Verzeichnisdienst

Methodenaufruf

Nachdem ein Client Kenntnis von einem Dienst hat, kann er einen entfernten Methodenaufruf auf diesem durchführen. Initial ruft der Client die benötigte Methode auf seinem lokalen System auf (siehe Abbildung 2). Dieser Aufruf wird nun an einen sogenannten **Stub** weitergeleitet, welcher ebenfalls auf dem Client befindlich ist. Ein Stub wird mithilfe der Schnittstellenbeschreibung des Servers erstellt und umfasst alle darin deklarierten Methoden. Das lokale Programm hat keine Kenntnis vom entfernten Server, sondern ausschließlich vom Stub. Dieser kann somit als ein clientseitiger Stellvertreter für die Serverfunktion angesehen werden. Der aufgerufene Stub wandelt nun alle an den Server zu übermittelnden Daten in ein für die Netzwerkübertragung geeignetes Format um, was als **Marshalling** bezeichnet wird. Hierunter befinden sich Angaben über die aufzurufende Methode sowie ggf. Methodenparameter. Anschließend erfolgt eine Übermittlung an den Server. Auf diesem befindet sich nun wiederum ein Stellvertreter, welcher **Skeleton** genannt wird. Dieser repräsentiert das aufrufende Objekt innerhalb des Servers. Nach dem Deserialisieren (**Unmarshalling**) der empfangenden Daten, ruft der Skeleton die definierte Funktion des lokalen Server-Programms auf. Das berechnete Ergebnis wird nun auf umgekehrtem Weg wieder an das lokale Programm des Clients übermittelt, wo die Prozessausführung fortzusetzen ist.

sogenannten **Stub** weitergeleitet, welcher ebenfalls auf dem Client befindlich ist. Ein Stub wird mithilfe der Schnittstellenbeschreibung des Servers erstellt und umfasst alle darin deklarierten Methoden. Das lokale Programm hat keine Kenntnis vom entfernten Server, sondern ausschließlich vom Stub. Dieser kann somit als ein clientseitiger Stellvertreter für die Serverfunktion angesehen werden. Der aufgerufene Stub wandelt nun alle an den Server zu übermittelnden Daten in ein für die Netzwerkübertragung geeignetes Format um, was als **Marshalling** bezeichnet wird. Hierunter befinden sich Angaben über die aufzurufende Methode sowie ggf. Methodenparameter. Anschließend erfolgt eine Übermittlung an den Server. Auf diesem befindet sich nun wiederum ein Stellvertreter, welcher **Skeleton** genannt wird. Dieser repräsentiert das aufrufende Objekt innerhalb des Servers. Nach dem Deserialisieren (**Unmarshalling**) der empfangenden Daten, ruft der Skeleton die definierte Funktion des lokalen Server-Programms auf. Das berechnete Ergebnis wird nun auf umgekehrtem Weg wieder an das lokale Programm des Clients übermittelt, wo die Prozessausführung fortzusetzen ist.

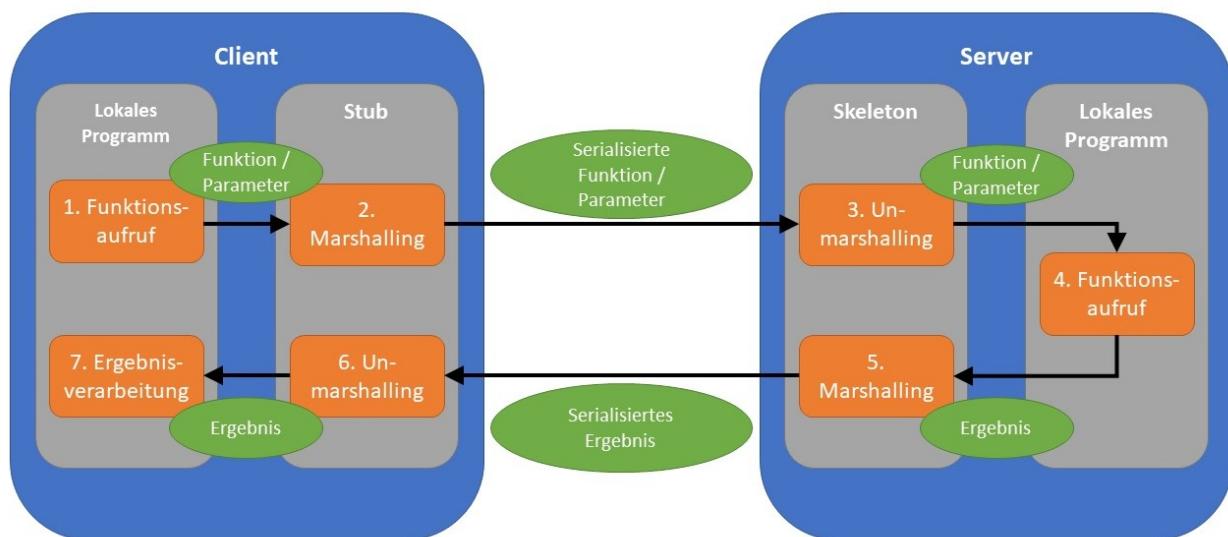


Abbildung 2: Schematische RPC-Kommunikation

Quelle: In Anlehnung an (Schill 2012, S. 47)

Wichtige Punkte:

Wichtige Punkte bezüglich der Verwendung von RPC werden im Folgenden nach (Schill 2012, S. 51-55) dargestellt.

Parameterübergabe: Client und Server werden in den meisten Fällen verschiedene Rechenknoten sein, weshalb sie auch über verschiedene Adressräume verfügen. Dies kann zu Problemen führen, wenn Funktionen per Call-by-Reference aufgerufen werden. Das

Client. Hat der Server seine Berechnung abgeschlossen, wird dieses Objekt auf Clientseite durch das tatsächliche Ergebnis ersetzt. Der Vorteil eines Future bzw. Promise ist, dass für den Client zu jedem Zeitpunkt ersichtlich ist, ob dieses bereits das gewünschte Resultat beinhaltet oder noch als Platzhalter dient. Der Client kann hierdurch seine Programmprozedur nach Aufruf einer entfernten Methode weiterführen und befindet sich nicht in einem blockierenden Zustand.

Fehlersemantik: Eine RPC-Kommunikation besitzt drei verschiedene Schwachpunkte. Sowohl der Client als auch der Server sind nicht vor Ausfällen geschützt. Weiterhin kann es zu Nachrichtenverlusten innerhalb des Netzwerks kommen. Diese Ausfallrisiken sind in Tabelle 1 verschiedenen klassifizierten RPC-Systemen gegenübergestellt. Jedes Paar von Systemklasse und Fehlerart sind zwei Zahlen zugeordnet. Die Ausführungen beschreiben, wie oft die vom Client angeforderte Operation auf dem Server ausgeführt wurde. Das Ergebnis definiert, wie oft der Client ein Resultat vom Server tatsächlich erhalten hat.

Systemklasse / Fehlerart	Fehlerfreier Ablauf	Nachrichtenverluste	Zus. Ausfall Server	Zus. Ausfall Client
Maybe	Ausführung: 1; Ergebnis: 1	Ausführung: 0/1; Ergebnis: 0/1	Ausführung: 0/1; Ergebnis: 0/1	Ausführung: 0/1; Ergebnis: 0/1
At-Least-Once	Ausführung: 1; Ergebnis: 1	Ausführung: >=1; Ergebnis: >=1	Ausführung: >=0; Ergebnis: >= 0	Ausführung: >=0; Ergebnis: 0
At-Most-Once	Ausführung: 1; Ergebnis: 1	Ausführung: 1; Ergebnis: 1	Ausführung: 0/1; Ergebnis: 0/1	Ausführung: 0/1; Ergebnis: 0
Exactly-Once	Ausführung: 1; Ergebnis: 1	Ausführung: 1; Ergebnis: 1	Ausführung: 1; Ergebnis: 1	Ausführung: 1; Ergebnis: 1

Tabelle 1: Verhalten von RPC-Systemklassen bei verschiedenen Fehlerarten

Quelle: In Anlehnung an (Schill 2012, S. 54)

Ein System mit der Fehlerklasse **Maybe** vollzieht keine Fehlerbetrachtung. Eine Anfrage wird einmalig versendet, unabhängig davon, ob der Client ein Resultat erhält oder nicht. Ein System mit der Fehlerklasse **At-Least-Once** reagiert auf ausbleibende Antworten seitens des Servers, indem der Client weitere Anfragen derselben Art stellt, bis schließlich eine Antwort erhalten wurde. Wurden Antworten aber beispielsweise lediglich aufgrund eines überlasteten Netzwerks noch nicht zugestellt, aber dennoch vom Server bearbeitet, kann die

Systemklasse / Fehlerart	Fehlerfreier Ablauf	Nachrichtenverluste	Zus. Ausfall Server	Zus. Ausfall Client
Maybe	Ausführung: 1; Ergebnis: 1	Ausführung: 0/1; Ergebnis: 0/1	Ausführung: 0/1; Ergebnis: 0/1	Ausführung: 0/1; Ergebnis: 0/1
At-Least-Once	Ausführung: 1; Ergebnis: 1	Ausführung: >=1; Ergebnis: >=1	Ausführung: >=0; Ergebnis: >= 0	Ausführung: >=0; Ergebnis: 0
At-Most-Once	Ausführung: 1; Ergebnis: 1	Ausführung: 1; Ergebnis: 1	Ausführung: 0/1; Ergebnis: 0/1	Ausführung: 0/1; Ergebnis: 0
Exactly-Once	Ausführung: 1; Ergebnis: 1	Ausführung: 1; Ergebnis: 1	Ausführung: 1; Ergebnis: 1	Ausführung: 1; Ergebnis: 1

Tabelle 1: Verhalten von RPC-Systemklassen bei verschiedenen Fehlerarten

Quelle: In Anlehnung an (Schill 2012, S. 54)

Ein System mit der Fehlerklasse **Maybe** vollzieht keine Fehlerbetrachtung. Eine Anfrage wird einmalig versendet, unabhängig davon, ob der Client ein Resultat erhält oder nicht. Ein System mit der Fehlerklasse **At-Least-Once** reagiert auf ausbleibende Antworten seitens des Servers, indem der Client weitere Anfragen derselben Art stellt, bis schließlich eine Antwort erhalten wurde. Wurden Antworten aber beispielsweise lediglich aufgrund eines überlasteten Netzwerks noch nicht zugestellt, aber dennoch vom Server bearbeitet, kann die wiederholte Anfrage für ungewollte Seiteneffekte sorgen. Dieses Problem vermeiden Systeme mit der Fehlerklasse **At-Most-Once**. Aufrufwiederholungen werden hier vom Server explizit erkannt und nicht behandelt. Die stärkste Klasse wird als **Exactly-Once** bezeichnet. Bei einem Server- bzw. Client-Ausfall werden beide Systeme auf den Zustand gebracht, der vor der Anfrage herrschte. Die Anfrage wird anschließend erneut durchgeführt. Somit ist die Konsistenz der Systeme zueinander gewährleistet.

gRPC

gRPC ist ein öffentlich zugängliches RPC-Framework, das 2015 von Google veröffentlicht wurde. Mithilfe von gRPC ist möglich, verschiedensprachige Microservices bzw. Endgeräte miteinander zu verknüpfen sowie Client-Bibliotheken zu erstellen. (Google 2017)

```
// Proto-Datei

service Greeter {
    // Berechne Preis von speziellem Auto
    rpc CalculatePrice (Auto) returns (int32) {}
}

// Datentyp Auto
message Auto {
    required Hersteller hersteller = 1;
    required int32 id = 2;
    optional string modell = 3;
    optional AntriebType type = 4 [default = VORDERRAD];
}

enum AntriebType {
    ALLRAD = 0;
    VORDERRAD = 1;
    HINTERRAD = 2;
}

message Hersteller {
    required string number = 1;
    optional string standort = 2;
}
```

Code-Beispiel 1: Protocol Buffer

Nachdem die Proto-Datei erstellt wurde, ist diese zu kompilieren. Anschließend erhält man einen nicht mehr lesbaren Protocol Buffer.

Verwendung

Mithilfe des Protocol Buffers generiert gRPC nun automatisch Quellcode. Es werden Getter- und Setter-Methoden für die Message Types erstellt sowie Funktionen zum (De-)Serialisieren. Weiterhin liegen nach Kompilierung Stub- sowie Skeleton-Objekte für Client und Server vor. Die Server-Entwickler können sich hierdurch maßgeblich auf die Geschäftslogik konzentrieren, anstatt auf Implementierungsdetails von RPC. Ein weiterer Vorteil von gRPC besteht darin, dass Services und Message Types generell sprachunabhängig sind und sich somit für verschiedene Systeme nicht unterscheiden. Lediglich die Art der Kompilierung gibt vor, in welcher Sprache die Methoden und Objekte erstellt werden. Hierdurch ist es ohne Aufwand möglich, zwischen heterogenen Systemen zu kommunizieren. Bisher liegen Bibliotheken für zehn verschiedene Programmiersprachen vor. Weiterhin ermöglicht gRPC die einfache Anbindung von vorgefertigten Modulen bezüglich Authorisierung, Lastenverteilung und Health Checking.

```
// Proto-Datei

service Greeter {
    // Berechne Preis von speziellem Auto
    rpc CalculatePrice (Auto) returns (int32) {}
}

// Datentyp Auto
message Auto {
    required Hersteller hersteller = 1;
    required int32 id = 2;
    optional string modell = 3;
    optional AntriebType type = 4 [default = VORDERRAD];
}

enum AntriebType {
    ALLRAD = 0;
    VORDERRAD = 1;
    HINTERRAD = 2;
}

message Hersteller {
    required string number = 1;
    optional string standort = 2;
}
```

Code-Beispiel 1: Protocol Buffer

Nachdem die Proto-Datei erstellt wurde, ist diese zu kompilieren. Anschließend erhält man einen nicht mehr lesbaren Protocol Buffer.

Verwendung

Mithilfe des Protocol Buffers generiert gRPC nun automatisch Quellcode. Es werden Getter- und Setter-Methoden für die Message Types erstellt sowie Funktionen zum (De-)Serialisieren. Weiterhin liegen nach Kompilierung Stub- sowie Skeleton-Objekte für Client und Server vor. Die Server-Entwickler können sich hierdurch maßgeblich auf die Geschäftslogik konzentrieren, anstatt auf Implementierungsdetails von RPC. Ein weiterer Vorteil von gRPC besteht darin, dass Services und Message Types generell sprachunabhängig sind und sich somit für verschiedene Systeme nicht unterscheiden. Lediglich die Art der Kompilierung gibt vor, in welcher Sprache die Methoden und Objekte erstellt werden. Hierdurch ist es ohne Aufwand möglich, zwischen heterogenen Systemen zu kommunizieren. Bisher liegen Bibliotheken für zehn verschiedene Programmiersprachen vor. Weiterhin ermöglicht gRPC die einfache Anbindung von vorgefertigten Modulen bezüglich Authorisierung, Lastenverteilung und Health Checking.

WebSocket

Das WebSocket-Protokoll wurde entwickelt, um bidirektionale Verbindungen zwischen einer Client-Anwendung und einer Host-Anwendung in einer möglichst effizienten Art und Weise zu ermöglichen.

Vor der Entwicklung dieses Protokolls musste das sog. *polling* (über HTTP) verwendet werden, um eine Art bidirektionaler Kommunikation zu ermöglichen, wobei die Kommunikation immer von der Client-Anwendung aus gestartet werden muss. Polling kommt aus dem Englischen und bedeutet übersetzt: befragen oder abfragen. Das heißt, die Client-Anwendung befragt immer wieder die Host-Anwendung nach Neuigkeiten und die Host-Anwendung gibt eine Antwort bzw. eine leere Antwort zurück. Mit dem sog. *long polling* gibt es eine etwas effizientere Möglichkeit des einfachen pollings.

Wieder startet die Client-Anwendung eine Anfrage. Die Host-Anwendung antwortet aber nur, wenn es auch eine Neuigkeit gibt. Ansonsten wird die HTTP-Verbindung so lange offen gehalten, bis eine Neuigkeit entsteht. Erst dann sendet die Host-Anwendung über die bereits etablierte HTTP-Verbindung die Nachricht an die Client-Anwendung und die Verbindung ist damit beendet. Die Client-Anwendung wird dann wieder eine neue HTTP-Verbindung zur Host-Anwendung öffnen und damit auf Neuigkeiten warten.

Die Nutzung des pollings bzw. long pollings erlaubt zwar eine Art der bidirektionalen Kommunikation, es ergeben sich jedoch einige Nachteile. In dem RFC 6455 Dokument der IETF zur Standardisierung des WebSocket-Protokolls werden die folgenden Punkte genannt:

- Die Host-Anwendung muss für jede Client-Anwendung mehrere TCP-Verbindungen aufrechterhalten. Dabei entfällt immer mindestens eine Verbindung auf die polling-Verbindung zur Client-Anwendung und für jede weitere Nachricht durch die Client-Anwendung entsteht eine neue Verbindung.
- Es besteht allgemein ein hoher Anteil an ungenutzten Daten beim Informationsaustausch (Overhead durch HTTP-Header), da die Header-Informationen immer hin und her gesendet werden. Insbesondere in einem Szenario, wie einer Chat-Anwendung, kann die Größe des Headers die Größe der zu transportierenden Nachricht übersteigen.
- Die Client-Anwendung muss die ausgehenden Verbindungen mappen, um die Antworten der Host-Anwendung richtig interpretieren zu können.

Das WebSocket-Protokoll kann diese Nachteile durch die Nutzung einer einzigen TCP-Verbindung umgehen und bricht dennoch nicht mit der HTTP-Infrastruktur, sodass auch die WebSocket-Kommunikation über die Ports 80 und 443 oder aber über HTTP-Proxy

Verbindungen erfolgen kann.

Nach einem sog. *handshake* der Teilnehmer (Client- und Host-Anwendung) kann der bidirektionale Nachrichtenaustausch beginnen.

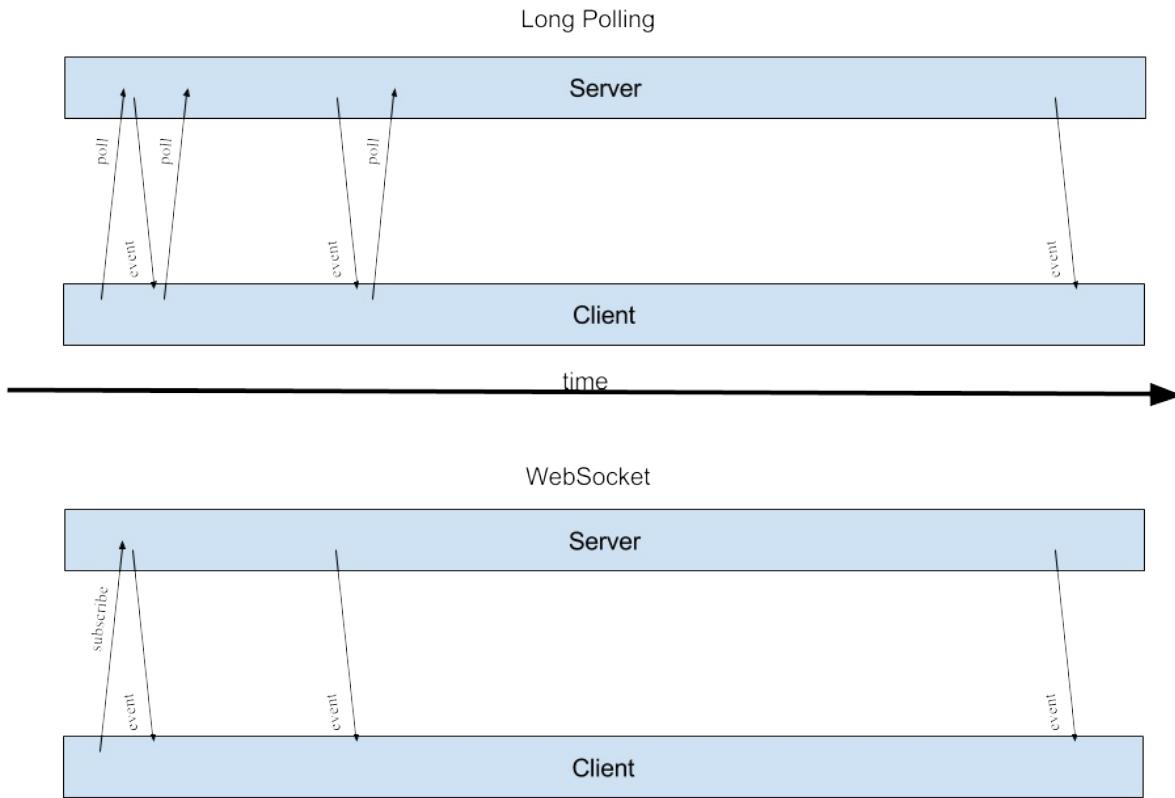


Abbildung A: Veranschaulichung Long Polling und WebSocket Kommunikation

REST

Representational State Transfer (REST) ist ein Architekturstil für verteilte Systeme, der von Roy Fielding innerhalb seiner im Jahr 2000 abgeschlossenen Dissertation "Architectural Styles and the Design of Network-based Software Architectures" entworfen wurde. REST beschreibt auf abstrakte Weise, wie das World Wide Web idealerweise aufzubauen ist. Die lose Kopplung einzelner Komponenten sowie ihre einheitliche Kommunikation untereinander sind nach Fielding die wesentlichen Voraussetzungen eines effizienten und dynamisch skalierbaren WWW. REST ist in seiner ursprünglichen Form dabei gänzlich unabhängig von Techniken sowie Werkzeugen und spezifiziert lediglich einen theoretischen Ansatz. [vgl. Pautasso et al. 2014, S. 1]

Konzepte

Constraints

Entgegen den meisten Behauptungen ist nicht jede Applikation, die Daten in Abhängigkeit des angefragten URI übermittelt, konform hinsichtlich des REST-Standards. Damit sich ein Service als RESTful bezeichnen darf, muss er Fielding zufolge eine Menge von Constraints erfüllen (vgl. Fielding 2000, S. 76-85). Diese werden im Folgenden erläutert. Zur groben Übersicht dient Abbildung 3.

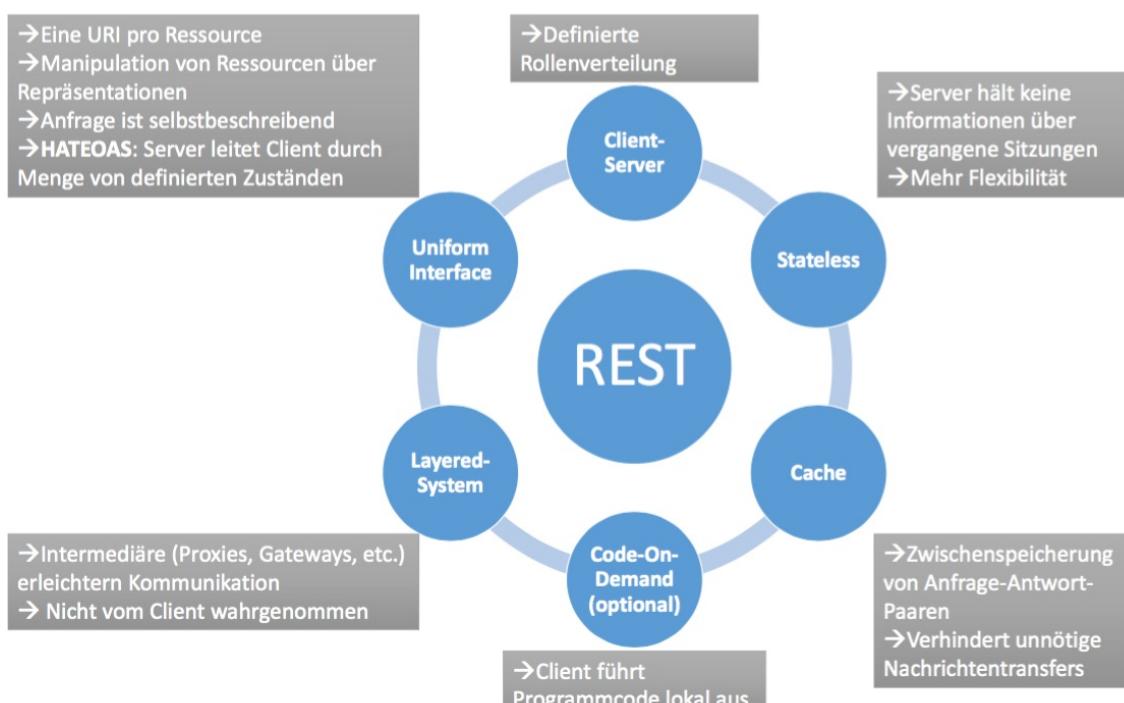


Abbildung 3: Veranschaulichung REST-Constraints

- **Client-Server:** REST kann als hybrider Architekturstil bezeichnet werden, da er auf bereits bestehende Paradigmen, wie beispielsweise das Client-Server-Modell zurückgreift. Für REST ist eine strikte Trennung zwischen der Geschäftslogik des Dienstes und der Schnittstelle des Benutzers fundamental. Die hierdurch erreichte lose Kopplung ermöglicht sowohl die Nutzung von Services unabhängig des verwendeten Endgeräts als auch die getrennte Weiterentwicklung von Client- bzw. Serverkomponenten.
- **Stateless:** RESTful Services müssen zustandslos sein. Die Server-Komponente darf demnach keine Informationen über ihre jeweiligen Clients halten. Dieses Constraint impliziert, dass jeder Aufruf eines Clients sämtliche für die Verarbeitung erforderlichen Informationen beinhalten muss. Der Client hat demnach die Verantwortung seine relevante Daten selber zu verwalten und dem Server bei jedem Aufruf geeignet mitzuteilen. Der Vorteil dieses Verfahrens liegt darin, dass die Performanz des Servers erheblich gesteigert wird. Dieser muss keine aufwändigen Lese- und Schreiboperationen zur Verwaltung der Benutzerdaten durchführen, sondern kann Anfragen ausschließlich anhand ihrer Semantik beantworten. Im Fall eines Serverausfalls bzw. bei unzureichender Lastverteilung ermöglicht das zustandslose Verfahren weiterhin die Umleitung von Clients an eine weitere Server-Instanz, ohne dass diese Kenntnisse bezüglich der Benutzerhistorien haben muss.
- **Cache:** Einen weiteren Beitrag zur Optimierung des World Wide Web leistet Fielding zufolge der Gebrauch von Caching. Ziel ist es, den Server zu entlasten und damit die Performanz von Datenaustauschen zu verbessern. Hierzu platziert sich in der Regel eine weitere Server-Instanz zwischen Client und dem eigentlichen Server bzw. der Client führt das Caching lokal durch. Jede Antwort, die vom Server zurück an den Client versendet wird, ist entweder als cacheable oder non-cacheable zu deklarieren. Eine Antwort, die cacheable ist, wird vom Intermediär inklusive der dazugehörigen Anfrage abgespeichert. Jede weitere Anfrage stellt die Client-Instanz nun an den Intermediär, der diese mit seinen abgespeicherten Anfragen vergleicht. Im Fall einer Übereinstimmung wird die dazugehörige Antwort direkt an den Client geliefert, ohne dass der Server in die Kommunikation involviert ist. Befindet sich die Anfrage nicht im Cache, wird sie zwecks normaler Verarbeitung an den Server weitergeleitet. Caching soll sicherstellen, dass der Server niemals die gleiche Antwort zweimal generieren bzw. übermitteln muss.
- **Uniform Interface:** Einheitliche Schnittstellen zwischen Client- und Server-Instanzen sind eine wesentliche Abgrenzung von REST zu anderen Architekturstilen. Sie verbessern die Nachvollziehbarkeit von Nachrichtenaustauschen für Außenstehende und ermöglichen den Aufruf von Services ohne genauere Kenntnisse der jeweiligen Implementierung. Aufgrund der Allgemeinheit des Constraints vollzieht Fielding eine Verfeinerung in vier Unter-Constraints.
 - *Identification of resources:* Jede Ressource eines Dienstes wird über einen

eindeutigen URI abgebildet.

- *Manipulation of resources through representations*: Wenn der Client Kenntnis von der Repräsentation einer Ressource besitzt, kann er diese nutzen, um die Ressource auf Serverseite zu manipulieren. Hält der Client beispielsweise eine JSON-Repräsentation einer Ressource Bestellung, kann er hiermit die Ressource serverseitig ändern (vorausgesetzt er besitzt die nötigen Zugriffsrechte). Weitere Benutzer, die sich eben diese Bestellung beispielsweise als XML-Objekt repräsentieren lassen, erhalten nun auch die manipulierte Version.
- *Self-descriptive messages*: Jede einzelne Nachricht enthält alle Informationen, die für ihre Verarbeitung nötig sind. Anfragen spezifizieren die Ressource, auf die sich beziehen, und die auszuführende Methode auf dieser Ressource. Im Fall, dass weitere Daten mitgesendet werden, muss zudem der Datentyp deklariert werden. Antworten hingegen sind als cacheable bzw. non-cacheable zu markieren.
- *HATEOAS*: Hypermedia as the engine of application state (HATEOAS) ist ein zentrales Prinzip von REST. Der Kerngedanke dahinter ist, dass der Client Ressourcen nicht selbstständig über URLs abfragt bzw. Kenntnisse von einer API hat, sondern für die gesamte Interaktionsdauer vom Server geführt wird. Nimmt der Client initial Kontakt mit dem Server auf, erhält er eine Antwort, in der spezifiziert ist, welche weiteren Anfragen möglich sind. Tätigt der Client eine dieser Anfragen, erhält er wiederum eine neue Menge von Anfragen, die aus seinem aktuellen Status getätigkt werden können. Diese Prozedur wiederholt sich bis zum Ende der Kommunikation. Eine Antwort seitens des Server enthält somit nicht nur das Ergebnis der eigentlichen Anfrage, sondern auch die nächsten potentiellen Interaktionsmöglichkeiten. Diese Möglichkeiten liefert der Server in der Regel über src- bzw. href-Attribute innerhalb des zurückgelieferten HTML-Objekts aus. Weiterhin kann er dem Client ein Formular senden, in dem dieser nur eingeschränkte Auswahlmöglichkeiten besitzt, was seine potentiellen Aktionen ebenfalls eingrenzt. HATEOAS baut somit auf dem Prinzip des endlichen Automatens auf. Ein Client gelangt in einen Initialzustand, für den eine Menge von Transitionen vom Server festgelegt sind. Mit jeder Transition erhält der Client die Informationen seines neuen Zustands sowie neue Transitionsmöglichkeiten vom Server vorgelegt. Diese Prozedur ist aus dem folgenden Beispiel (siehe Abb. 4) ersichtlich. Der Client gelangt initial auf die Homepage des RESTful Dienstes Versandhandel. Von diesem Zustand ermöglicht der Server ihm ausschließlich die Transition in den Status "Alle Produkte anzeigen". Gelangt er in diesen Status, kann er ein spezifisches Produkt anfragen bzw. eine Rücktransition vollziehen. Lässt er sich ein spezifisches Produkt anzeigen, kann er dieses erwerben bzw. eine Bestellung abschicken oder zurück auf die Startseite wechseln. Jeder Status beinhaltet demnach die tatsächlich angefragten Informationen, wie beispielsweise die Darstellung sämtlicher Produkte, als auch die möglichen Transitionen zu neuen

Status. Welche Status und Transitionsmöglichkeiten existieren, gibt alleine der Server vor. Hierdurch entstehen diverse Vorteile. Die Komplexität auf Seiten des Clients wird erheblich verringert, da dieser keine API verinnerlichen muss, sondern ausschließlich vom Server gelenkt wird. Hierdurch wird auch das Risiko von inkorrekt Anfragen gesenkt. Weiterhin kann der Server flexibler agieren. Er stellt dem Client ausschließlich die Ressourcen zur Verfügung, die im aktuellen Moment tatsächlich angefragt werden können. Ist ein Produkt nicht mehr im Sortiment, kann sich der Client dieses auch nicht mehr anzeigen lassen. Im Fall, dass der Server ausgelastet ist, kann er den Client dynamisch an eine weitere Server-Instanz verweisen, ohne dass dieser Kenntnis davon hat.

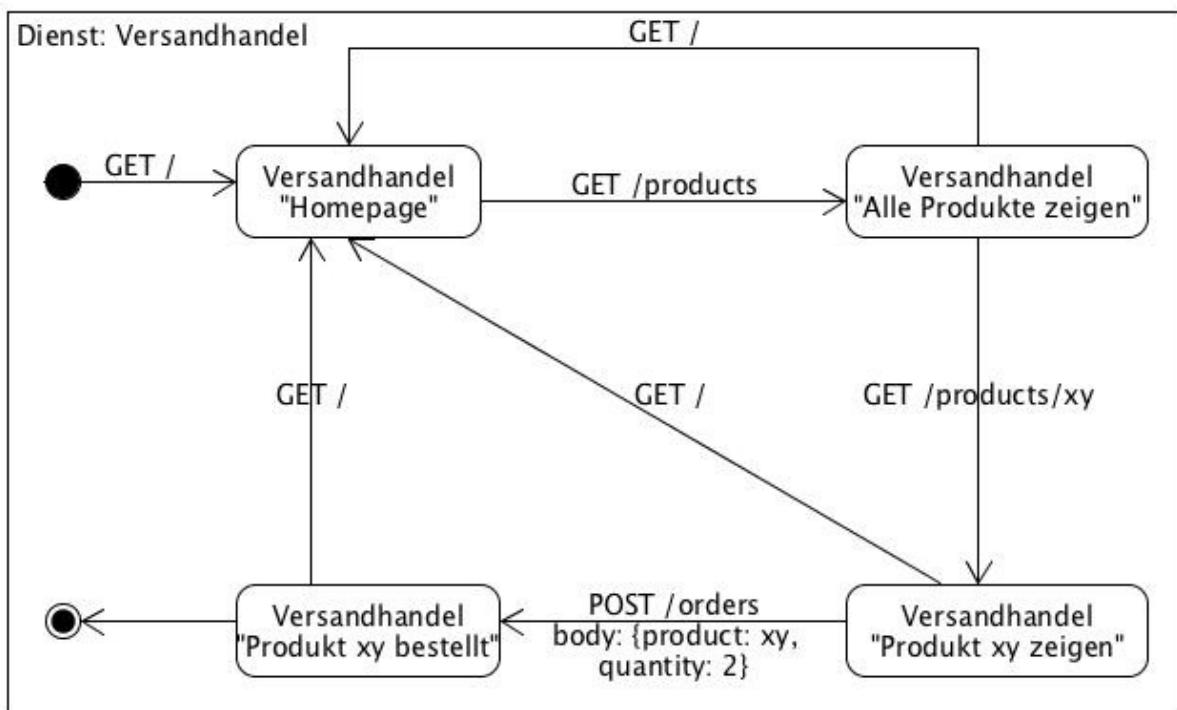


Abbildung 4: HATEOAS am Beispiel Versandhandel

- **Layered-System**: Ein geschichtetes System soll die lose Kopplung innerhalb des World Wide Web weiter vorantreiben. Das Ziel ist es, die Komplexität von Diensten zu verringern, ohne weitere Änderungen an ihnen vorzunehmen. Hierzu werden weitere Schichten zwischen Client und Server gesetzt. Hierunter können Proxy-, Gateway- oder andere Server-Instanzen verstanden werden, die diverse Funktionen, wie beispielsweise Caching, Vermittlung oder Übersetzung von Anfragen, übernehmen. Diese Intermediäre sind so einzusetzen, dass für den Client keine zusätzlichen Anstrengungen bezüglich des Kommunikationsaufbaus entstehen. Er weiß also zu keiner Zeit, ob er direkt mit dem Server oder mit einem Intermediär kommuniziert, da ihre Schnittstellen identisch sind.

- **Code-On-Demand:** Unter Code-On-Demand ist die Übermittlung von Programmcode vom Server an den Client zu verstehen. Sowohl Client als auch Server können hierdurch entlastet werden, da Operationen nun lokal beim Anwender aufgerufen werden können, ohne dass eine Übermittlung von Eingabe-Parametern bzw. Programmergebnissen stattfinden muss. Fielding sieht es jedoch als optionales Constraint an, da es die Nachvollziehbarkeit einer Kommunikation verringert, wenn der Client teilweise seine eigenen Berechnungen durchführt.

Ressourcen und Repräsentationen

Der Austausch von Informationen zwischen Client und Server ist die Grundlage von verteilten Systemen. REST führt hier eine weitere Differenzierung durch und teilt Informationen in Ressourcen und Repräsentationen ein. Ressourcen bilden die Ausgangsbasis und stellen in der Regel Objekte des realen Lebens dar. Beispielhaft für eine Ressource ist ein Mensch oder ein Auto. Auch nicht greifbare Objekte können Ressourcen sein, wie zum Beispiel eine Bestellung. Eine Ressource kann weiterhin durch eine beliebige Anzahl an Repräsentationen beschrieben werden. Eine Repräsentation der Ressource Auto ist beispielsweise ein JSON-Objekt, in dem alle technischen Daten des Autos aufgelistet sind. Auch ein Foto im JPEG-Format, auf dem das Auto abgebildet ist, kann als Repräsentation verstanden werden. Der Vorteil der Abgrenzung von Ressourcen und Repräsentationen ist die daraus resultierende Flexibilität. Der Server speichert alle Daten, die er einer Ressource zuordnen kann in einer Datenbank ab. In Abhängigkeit davon, welche Darstellung der Client bevorzugt, kann er aus diesen Daten dynamisch eine geeignete Repräsentation generieren und übermitteln. Unabhängig davon, welche Repräsentation angefordert wird, bleibt die Aufrufroutine für eine Ressource jedoch immer gleich. Lediglich die übermittelten Informationen ändern sich. HTTP erlaubt beispielsweise, den bevorzugten Typ einer angefragten Ressource in dem Headerfeld der Anfrage mitzusenden. (vgl. Tilkov et al. 2015, S. 35-36)

(Tilkov et al. 2015, S. 36-39) teilen Ressourcen weiterhin in verschiedene Unterkategorien ein, um die Übersichtlichkeit eines RESTful Services zu steigern:

- **Primärressourcen:** Unter Primärressourcen sind Ressourcen im eigentlichen Sinn zu verstehen. Jedes Objekt, welches ein elementarer Bestandteil des Dienstes ist, ist dieser Kategorie zuzuordnen. Ein Dienst, der dem Verkauf von Autos dient, wird somit höchstwahrscheinlich eine Primärressource Auto aufweisen. Jedoch kann auch eine Bestellung oder ein Kunde hier als Primärressource angesehen werden.
- **Subressourcen:** Subressourcen sind die elementaren Bestandteile übergelagelter Ressourcen. Die Primärressource Auto könnte beispielsweise die Subressourcen Motor und Lenkrad umfassen.
- **Listenressourcen:** Listenressourcen kommen immer dann zum Einsatz, wenn man

nicht auf eine einzelne Primär- bzw. Subressource zugreifen möchte, sondern auf die Gesamtheit aller Ressourcen des Typs. Die Listenressource Autos kann beispielsweise alle Primärressourcen Auto umfassen, welche aktuell zum Verkauf angeboten werden.

- **Filter:** Eine nach einem bestimmten Kriterium gefilterte Listenressource ist als Filterressource anzusehen. Beispielhaft dafür seien alle zu verkaufenden Autos mit mehr als 150 PS.
- **Paginierung:** Eine Paginierung teilt eine Listenressource auf einzelne Seiten auf, sodass eine übersichtliche Darstellung erfolgen kann.
- **Projektionen:** Werden nur bestimmte Informationen einer Ressource benötigt, kann eine Projektion erfolgen. Beispielsweise wird für die Listenressource Autos nur das Bild und das Modell des jeweiligen Autos für die Darstellung benötigt. Die Informationen bezüglich der Leistung müssen zwecks Performanz somit nicht übermittelt werden.
- **Aggregationen:** Sollen bestimmte Attribute verschiedenartiger Ressourcen in einer einzelnen Antwort übermittelt werden, ist eine Aggregation durchzuführen.
- **Aktivitäten:** Schritte innerhalb einer Verarbeitung sind den Aktivitäten zuzuordnen. Hierbei handelt es sich um nicht greifbare Tätigkeiten, die dennoch als Ressourcen abgelegt werden. Kauft man beispielsweise ein Auto bei einem Dienst, kann initial eine neue Ressource "SCHUFA-Prüfung" angelegt werden, die die Kreditwürdigkeit feststellen soll.

URLs

Ein Universal Resource Identifier (URI) definiert den Zugriffsort einer einzelnen Ressource in einem Netzwerk. Der Identifikator muss dabei nach einem standardisierten Schema (siehe Abb. 5) aufgebaut sein, welches fünf Teile spezifiziert (vgl. Berners-Lee 2005, S. 16-24).

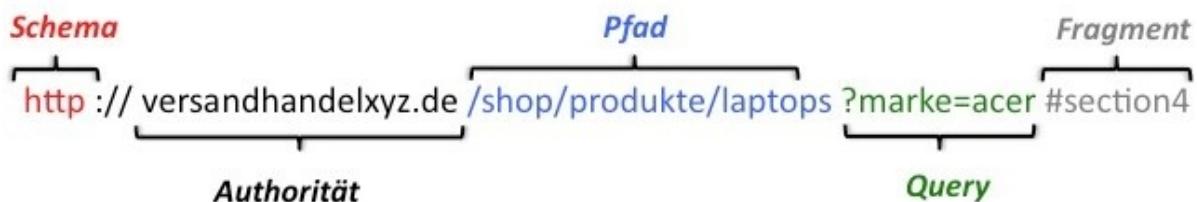


Abbildung 5: Beispielhafter Aufbau eines HTTP-URI

- **Schema:** Unter dem Schema kann das Protokoll verstanden werden, mit dem die jeweilige Ressource anzufragen ist. Hierzu sind neben Hypertext Transfer Protocol (HTTP) und File Transfer Protocol (FTP) auch beispielsweise Lightweight Directory Access Protocol (LDAP) und Teletype Network (Telnet) zu zählen. Für REST wird jedoch in der Regel HTTP verwendet.

- **Authorität:** Die Authorität bezeichnet die Organisation, der die angebotene Ressource unterliegt. In manchen Fällen ist dieser Authorität zusätzlich eine Portnummer nach- bzw. der die Ressource anfragende Benutzer vorgestellt.
- **Pfad:** Der Pfad beschreibt, an welcher Stelle sich die benötigte Ressource innerhalb der Organisation befindet und ist typischerweise hierarchisch aufgebaut. Einzelne Hierarchieebenen werden dabei mit einem Schrägstrich voneinander getrennt.
- **Query:** Optional ist der Pfad um eine Abfrage zu erweitern. Mit einem Kaufmanns-Und sind beliebig viele Schlüssel-Wert-Paare anzuhängen, die vom Server ausgelesen und in die Verarbeitung miteinbezogen werden.
- **Fragment:** Ein Fragment wird ausschließlich auf Seite des Clients aufgelöst und nicht per Anfrage an den Server übertragen. Die konkrete Verwendung des Fragments ist dabei abhängig vom verwendeten Protokoll bzw. von dem Medientyp der Antwort. Wird beispielsweise ein HTTP-URI mit Fragment aufgerufen, der ein HTML-Objekt liefert, scrollt der Browser nach Darstellung der kompletten Webseite automatisch zu dem von dem Fragment beschriebenen HTML-Tag.

Innerhalb von REST sind URLs unerlässlich. Jede Ressource muss über einen definierten URI erreichbar sein. Weiterhin darf sich maximal ein URI auf eine Ressource beziehen. Diese Auflagen garantieren einen eindeutigen und konsistenten Zugriff.

Verben

Die Semantik einer Anfrage an einen RESTful Service ergibt sich nicht ausschließlich aus der Lokalität der angeforderten Ressource, sondern auch aus der Methode, die darauf angewendet werden soll. HTTP bietet eine Vielzahl an Methoden, die potentiell für REST benutzt werden können. In der Praxis hat sich jedoch nur eine Teilmenge an Methoden etabliert, die auf die grundlegenden Datenbankoperationen (Create, Read, Update und Delete) abgebildet werden können. Die Methoden werden im Folgenden nach (Tilkov et al. 2015, S. 53-59) beschrieben.

- **GET:** Die grundlegendste HTTP-Methode ist GET. Ein mit GET angefragter Service liefert eine Repräsentation der durch den URI beschriebenen Ressource an den Client zurück. Der Typ der übermittelten Repräsentation kann bereits bei der HTTP-Anfrage festgelegt werden. Im Fall, dass der Server die deklarierte Repräsentation nicht liefern kann, übermittelt er eine andere Repräsentation seiner Wahl.
- **PUT:** Eine bestehende Ressource kann von einem Client auf dem Server manipuliert werden. Dafür muss der Client geeignete Zugriffsrechte und eine Repräsentation der Ressource besitzen. Die veränderte Repräsentation wird nun im Body einer PUT-Anfrage an den URI der Ressource versendet. Im Header wird der Typ der übermittelten Repräsentation spezifiziert, sodass der Server die Daten geeignet

verarbeiten kann.

- **POST:** Mit dem POST-Befehl teilt der Client dem Server eindeutig mit, dass eine neue Ressource angelegt werden soll. Äquivalent zum PUT-Befehl wird eine Repräsentation im Body der Anfrage mitgeliefert und im Header beschrieben. Ein Hauptunterschied zwischen PUT und POST ist, dass der URI einer PUT-Anfrage auf eine Primär- bzw. Subressource verweist, während der URI eines POST-Befehls eine Listenressource beschreibt, in die das neue Element einzufügen ist.
- **DELETE:** Soll eine Ressource gelöscht werden, ist der DELETE-Befehl mit dem dazugehörigen URI zu verwenden. In der Regel wird die Ressource jedoch nicht physikalisch aus dem Speicher des Servers entfernt, sondern lediglich als nicht mehr existent gekennzeichnet.

Weiterhin können die beschriebenen Methoden hinsichtlich diverser Faktoren kategorisiert werden. Ein Faktor ist die Sicherheit. Ist eine Methode als sicher einzustufen, kann sie ausgeführt werden, ohne dass dies Auswirkungen auf die gespeicherten Daten des Servers hat. Ein GET-Befehl manipuliert beispielsweise keine serverseitigen Informationen, weshalb er ohne Bedenken genutzt werden kann. Ein PUT-Befehl ist im Gegensatz dazu unsicher, da durch ihn potentiell weitreichende Manipulationen durchgeführt werden können. Idempotenz liegt dann vor, wenn eine Methode beliebig oft auf denselben URI angewendet werden kann, und lediglich die erste Anfrage Auswirkungen hat. Führt man beispielsweise den PUT-Befehl mit gleichem URI und gleichem Body zweimal durch, wirkt nur die erste Anfrage. Die zweite Anfrage wird zwar vom Server abgearbeitet, hat aber keine Auswirkungen, da das Update der Ressource bereits geschehen ist. Ein zweimaliges Posten derselben Anfrage führt im Gegensatz zur Anlage zwei neuer Ressourcen und ist somit nicht idempotent. Die Kenntnis über die Idempotenz einer Methode kann beispielsweise bei Verbindungsstörungen hilfreich sein. Wurde eine Anfrage abgeschickt, ohne dass eine Antwort eingegangen ist, können idempotente Anfragen ohne Sorge wiederholt ausgeführt werden. Selbst wenn der Server die erste Anfrage bereits verarbeitet hat und die Antwort lediglich aufgrund von Latenzproblemen noch nicht zugestellt wurde, hat das nochmalige Senden keine Auswirkungen. Wie schon angeschnitten, unterscheiden sich Methoden bezüglich ihrer Adressierung. Ein GET-Befehl wird immer direkt auf eine einzelne Ressource angewendet und identifiziert diese dementsprechend. Im Gegensatz dazu steht eine POST-Anfrage, die lediglich auf eine Liste angewendet wird und dementsprechend nicht auf eine konkrete Instanz einer Ressource. Letztlich ist eine Differenzierung der Methoden im Hinblick auf ihre Cache-Fähigkeit zu vollziehen. Ein GET-Befehl ist beispielsweise cache-fähig, da er ausschließlich Daten zurückliefert. Diese müssen nicht zwingend vom Server stammen, sondern können auch durch einen Intermediär bereitgestellt werden. Befehle, die Manipulationen auf dem Server einfordern, müssen zwangsweise an diesen zugestellt werden und sind dementsprechend nicht von einer zwischengeschalteten Instanz zu verarbeiten. Sie dienen jedoch der Invalidierung. Wird beispielsweise ein PUT auf eine

Ressource ausgeführt, die sich im Cache befindet, ist diese anschließend aufgrund der Manipulation aus dem Cache zu entfernen. Eine detaillierte Gegenüberstellung der HTTP-Methoden bezüglich der genannten Bewertungsfaktoren ist aus Tabelle 2 ersichtlich.

Methode	Sicher	Idempotent	Identifiziert einzige Ressource	Cache-fähig
GET	Ja	Ja	Ja	Ja
PUT	Nein	Ja	Ja	Nein
POST	Nein	Nein	Nein	Nein
DELETE	Nein	Ja	Ja	Nein

Tabelle 2: Vergleich der grundlegenden HTTP-Methoden

Quelle: In Anlehnung an (Tilkov et al. 2015, S. 59)

Caching

Wie sich im vorherigen Abschnitt bereits herausgestellt hat, ist Caching ein essentieller Bestandteil von REST. Die Zwischenspeicherung von Anfrage-Antwort-Paaren soll zum einen den Client unterstützen. Erhält dieser seine angeforderten Informationen, ohne dass sie auf Seiten des Servers aufwändig generiert werden mussten, kann er Zeit einsparen und seine Performanz steigern. Zum anderen wird auch der Server durch Caching entlastet, da ihn weniger Anfragen erreichen. Ohne zu tief in Implementierungsdetails zu gehen, kann ein einzelnes Caching-Verfahren für RESTful Services anhand von zwei Dimensionen charakterisiert werden, wobei auch hybride Ansätze möglich und praktikabel sind. (Tilkov et al. 2015, S. 127-133) definieren diese Dimensionen als Modell und Topologie.

Modell

Das Modell legt fest, von welcher Instanz das Caching initiiert wird (siehe Abb. 6). Beim **Expirationsmodell** erweitert der Server jede von ihm ausgelieferte Nachricht um den Cache-Control-Header. In diesem wird vermerkt, wie lange die übermittelte Repräsentation bzw. die dazugehörige Ressource aktuell ist. Der Server spezifiziert damit, dass sich die jeweilige Ressource in dem festgelegten Zeitraum (wahrscheinlich) nicht ändern wird und ermöglicht somit die zielführende Zwischenspeicherung des bereits ausgelieferten Ergebnisses. Nach Ablauf der Zeit sind Anfragen bezüglich der Ressource wieder konventionell an den Server zu richten, bis möglicherweise wieder ein Cache-Control-Header empfangen wurde. Der Vorteil dieses Modells liegt in seiner Effizienz. Es ist kein zusätzlicher Datenverkehr notwendig, um den Client auf die Gültigkeit von Repräsentationen bzw. Ressourcen aufmerksam zu machen. Hierunter leidet jedoch in manchen Fällen die Aktualität von zwischengespeicherten Antworten. Hat sich trotz Versprechen des Servers

eine Ressource während der Caching-Zeitspanne verändert, bleibt der Client davon uninformiert und greift weiterhin auf die veralteten Daten zu. In Abhängigkeit des Kontexts kann dies mehr oder weniger schwere Konsequenzen nach sich ziehen. Beim **Validierungsmodell** geht die Interaktion hingegen von Seiten des Clients aus. Die nochmalige Anfrage einer Ressource, von der er bereits eine Repräsentation besitzt, erweitert er um zusätzliche Informationen. Typischerweise ist dies ein Zeitstempel von der erstmaligen Anforderung. Der Server prüft nun, ob die Ressource seit dem angegebenen Zeitpunkt einer Änderung unterlag. Im Fall, dass die Ressource konstant blieb, ist die nochmalige Übertragung nicht nötig. Der Server teilt dem Client nun lediglich mit, dass er mit der früheren Repräsentation weiterarbeiten kann. Sollte jedoch eine Änderung aufgetreten sein, wird die neue Repräsentation im Body mitgeschickt. Anstatt eines Zeitstempels ist es weiterhin möglich, den Hashwert einer Repräsentation zu übergeben. Der Server kann auch hiermit feststellen, ob die clientseitige Repräsentation der Ressource noch immer mit der serverseiten Repräsentation übereinstimmt. Der Vorteil vom Validierungsmodell liegt darin, dass der Client stets mit aktuellen Repräsentationen arbeitet, auch wenn sie nicht erneut durch den Server übertragen wurden. Der Nachteil ergibt sich jedoch dadurch, dass die Einsparung der Kommunikationsaufwände im Vergleich zum vorherigen Modell geringer ausfällt. Der Server muss in jedem Fall die Anfrage des Clients verarbeiten. Effizienz wird lediglich dadurch gewonnen, dass die Antwort teilweise verschlankt wird bzw. keine Repräsentation auf Serverseite zu generieren ist.

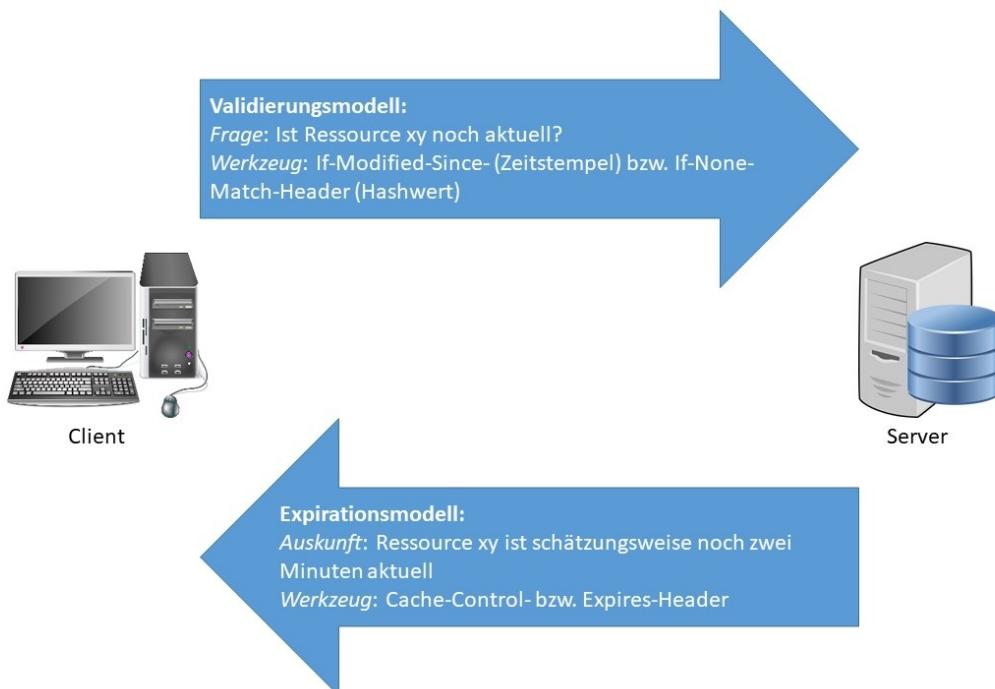


Abbildung 6: Veranschaulichung Caching-Modelle

Topologie

Die Topologie eines Caching-Verfahren definiert, an welcher Lokalität das Caching stattfindet bzw. wo der Zwischenspeicher angesiedelt ist (siehe Abb. 7). Beim **clientseitigen Caching** (typischerweise in jedem Webbrower implementiert) besitzt jeder Client, der mit einer Server-Instanz kommuniziert, seinen eigenen Cache. Der Vorteil liegt darin, dass auch verschlüsselte bzw. vertrauliche Anfragen und Antworten zu cachen sind, da keine Sicherheitsrisiken bezüglich weiterer Nutzer entstehen. Der **clientseitige Shared Cache** bzw. **Proxy-Cache** erweitert diese Topologie. Der Zwischenspeicher gilt nun für eine beliebige Anzahl an Benutzern und ist typischerweise auf einem zwischengelagerten Proxy-Server befindlich. Die Vielzahl der anfragenden Clients erhöht die Chance für das Caching einer später benötigten Repräsentation. Beim **Server-Caching** wird jede an den Server gestellte Anfrage zwischengespeichert und für alle nachkommenden Clients verfügbar gemacht. Zwar erhöht sich die Chance, dass eine Anfrage ohne Berechnung auf Serverseite beantwortet werden kann, jedoch steigen auch die Aufwände. Anfragen sind stets durch das gesamte Netzwerk hindurch zu leiten, bevor sie überprüfbar sind. In der Praxis werden alle Topologien miteinander verknüpft, um bestmögliche Caching-Resultate zu erzielen.

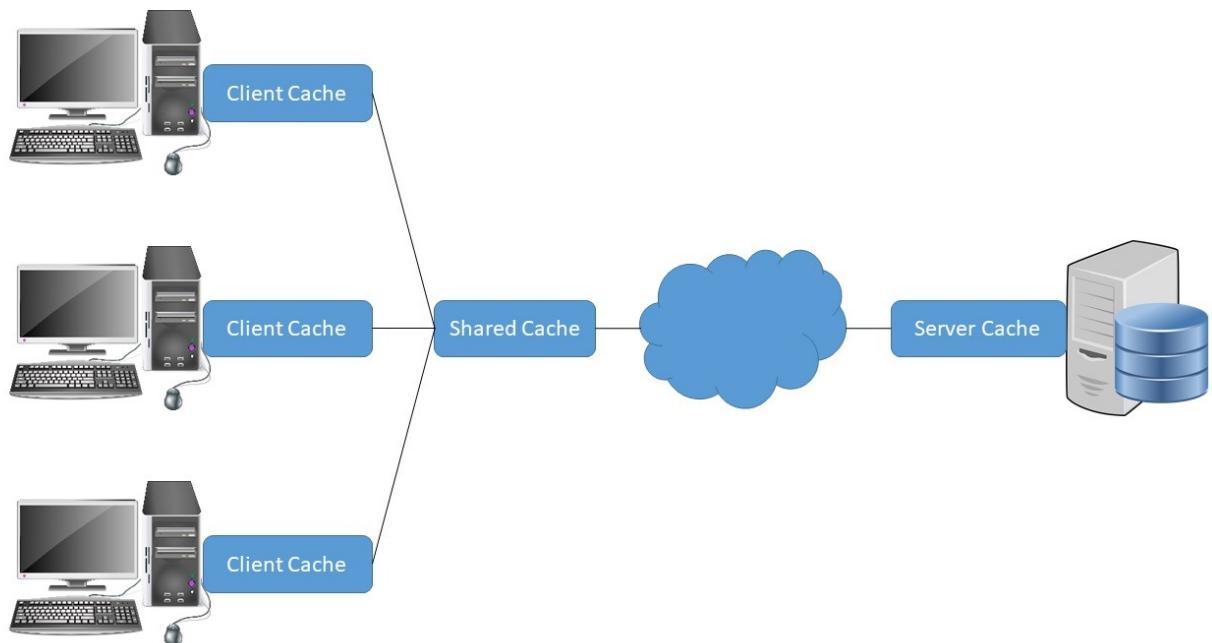


Abbildung 7: Veranschaulichung Caching-Topologien

Quelle: In Anlehnung an (Tilkov et al. 2015, S. 133)

Security

Sicherheit ist für verteilte Systeme und insbesondere RESTful Services ein zentrales Thema. Letztere bieten aufgrund ihrer meist öffentlich zugänglichen Schnittstelle eine breite Fläche für potentielle Angreifer und sind somit zu schützen. (Levin 2016) definiert dabei fünf verschiedene Maßnahmen, deren Umsetzung einen sicheren Dienst gewährleisten soll.

1. **Authorisierung:** Wie bereits erwähnt, sind RESTful Services oft öffentlich zugänglich. Dies sei gewollt, um einen Dienst möglichst vielen Nutzern zur Verfügung zu stellen und somit seine Popularität zu steigern. Mithilfe von HATEOAS leitet der Server den spezifischen Client durch eine Menge von für ihn verfügbaren Zuständen. Benutzer, die erweiterte Kenntnisse bezüglich der REST-Schnittstelle haben, können jedoch selbstständig Anfragen an Ressourcen stellen. Obwohl diese Herangehensweise vom Server ungewollt ist, kann er sie nicht unterbinden. Um trotzdem zu verhindern, dass jeder Benutzer potentiell dazu in der Lage ist, jede existierende Ressource einzusehen bzw. zu manipulieren, muss eine Authorisierung erfolgen. Der Client erweitert jede Anfrage um ein Authorisierungsobjekt (falls er ein solches besitzt), das in Abhängigkeit des angefragten Dienstes als API- bzw. Session-Key oder Benutzername-Passwort-Paar anzusehen ist. Der Server entscheidet anschließend, ob die beantragte Operation auf der Ressource zu bewilligen ist und führt diese ggf. aus. Welche Benutzergruppen für welche Aktionen befähigt sind, hängt hingegen allein von der Geschäftslogik ab.
2. **Eingabeverifikation:** Auch autorisierte Clients sind nicht zwingend vertrauenswürdig und können dem RESTful Service schaden. Insbesondere durch die Nutzung von POST- und PUT-Befehlen ergeben sich für sie diverse Angriffsmöglichkeiten. Eine simple HTTP-Anfrage, die per POST gestellt wurde, beinhaltet eine Vielzahl an Informationen. Der URI verrät, an welche Ressource die Operation gerichtet ist. Eine breite Anzahl von Headern erweitert diese Information um verschiedenartige Details, die sich sowohl auf die Syntax als auch auf die Semantik der Anfrage beziehen. Im Body kann weiterhin jede erdenkliche Zeichenfolge mitgeliefert werden. Schädlinge können diese Freiheit nutzen, um eine Anfrage derart zu formulieren, dass sie auf dem Server Schaden anrichtet. Cross-Site-Scripting und SQL-Injection sind hier nur zwei mögliche Angriffsmethoden. Um diesen entgegenzuwirken, sollte der Server initial den angefragten URI sowie die beigefügten Header validieren. Sind die diese nicht gültig bzw. entsprechen nicht dem typischen Schema, ist die gesamte Anfrage zu verwerfen. Weiterhin muss auch der Inhalt der Anfrage geeignet verarbeitet werden. Der Server sollte sich dabei nicht ausschließlich auf den vom Client angegebenen Content-Type verlassen, sondern den Body immer explizit mit diesem abgleichen. Bei fehlender Übereinstimmung ist die weitere Verarbeitung der Anfrage ggf. mit Gefahr verbunden. Für die weitere Nutzung ist der Body-Inhalt immer mit einem sicheren Parser zu verarbeiten, sodass weitere Risiken minimiert werden. Bei der Erstellung neuer oder der Manipulation von bestehenden Ressourcen sollte der Server schließlich stets strikte Wertebereiche für einzelne Attribute vorgeben, um Angriffsmöglichkeiten weiterhin einzuschränken.
3. **Ausgabeverifikation:** Angreifer können nicht nur dem Server schaden, sondern auch allen weiteren Clients, die mit diesem kommunizieren. Liefert der Server an diese eingeschleuste und schadhafte Inhalte aus, sind Sicherheitsrisiken die Folge. XML- oder JSON-Objekte sollten vom Server immer mit einem Serializer erzeugt werden, um

beispielsweise SQL-Injection zu verhindern. Der mitgelieferte Content-Type muss in jedem Fall mit dem Body-Inhalt der Antwort übereinstimmen, damit die Antwort auf Clientseite geeignet verarbeitet werden kann.

4. **Kryptographie:** Auch die ausgetauschten Nachrichten zwischen Client und Server sind zu schützen, sodass weder Lese- noch unbemerkte Schreiboperationen von Unbefugten während der Übermittlung möglich sind. Hierzu wird typischerweise auf das Protokoll TLS (Transport Layer Security) zurückgegriffen. TLS ist der Nachfolger von SSL (Secure Sockets Layer) und besteht aus den zwei Unterprotokollen TLS Handshake Protocol und TLS Record Protocol, welche den Verbindungsauflaufbau zwischen Client und Server bzw. deren Nachrichtenaustausch absichern sollen. Durch den Einsatz von TLS ist die Kommunikation für Dritte nicht einseh- oder manipulierbar. Weiterhin sind sensitive Daten, welche im Speicher des Servers befindlich sind, gegen Angriffe zu schützen. Kreditkartennummern, Adressen und andere benutzerspezifische Informationen dürfen ausschließlich verschlüsselt abgelegt werden. Dabei ist darauf zu achten, dass die zugehörigen Schlüssel nicht an der derselben Lokalität befindlich sind. Passwörter sind hingegen immer als Hashwerte ihrer selbst aufzubewahren.
5. **HTTP-Status-Codes:** HTTP-Status-Codes dienen dazu, den Client über den (Miss-)Erfolg seiner gesendeten Anfrage zu informieren. Wird bei der Verarbeitung differenziert auf Client-Anfragen eingegangen und spezifische HTTP-Status-Codes erzeugt, hilft dies nicht nur dem anfragenden Benutzer bei der Fehlersuche, sondern unterstützt auch die Identifizierung von möglichen Sicherheitsproblemen auf der Serverseite.

Richardson Maturity Model

Um die REST-Konformität eines Services zu bestimmen, dient das Richardson Maturity Model, welches von Leonard Richardson entwickelt wurde. Ein Service wird dabei einem spezifischen Level zugeordnet, wobei vier verschiedene Level existieren. Jedes Level beinhaltet dabei alle Forderungen der untergeordneten Level inklusive seiner eigenen Forderung. Tabelle 3 veranschaulicht die einzelnen Abstufungen nach (Fowler 2010).

Level	Bedeutung	Bemerkung
0	Der Service besitzt ausschließlich einen Endpunkt. Anfragen jeglicher Art werden an diesen URI gestellt.	Kein REST - mit RPC oder SOAP vergleichbar.
1 - Ressourcen	Der Service ordnet jeder existierenden Ressource einen eigenen URI zu.	Schwaches REST - kein Gebrauch der HTTP-Semantik. Lesende und manipulierende Operationen werden über denselben HTTP-Befehl abgebildet (beispielsweise POST zum Einsehen, Aktualisieren, Erstellen und Löschen von Ressourcen).
2 - Verben	Der Service versteht und unterscheidet die Semantik der verschiedenen HTTP-Verben und wendet diese dementsprechend auf seine Ressourcen an.	Mittelmäßiges REST - Clients müssen Kenntnis bezüglich der Schnittstelle besitzen, um Anfragen zu stellen.
3 - Hypermedia	HATEOAS - Server führt Client durch Menge von Zuständen, ohne dass dieser Wissen über die Schnittstelle besitzen muss.	REST

Tabelle 3: Stufen des Richardson Maturity Model

Anwendung

Best Practices

Fielding definiert mit dem Architekturstil REST Richtlinien, an denen sich Webservices idealerweise orientieren sollten. Diese Richtlinien sind jedoch absichtlich so grob gehalten, dass sie Raum für verschiedenartige Umsetzungen lassen. In der Vergangenheit hat sich jedoch nach (Sahni 2015) eine Menge von Ansätzen etabliert, mithilfe der RESTful Dienste ihr maximales Potential bezüglich Benutzer- und Entwicklerfreundlichkeit ausschöpfen können. Eine Auswahl dieser Ansätze wird im Folgenden vorgestellt.

Einheitliche Verwendung von URIs und HTTP-Methoden: Auch wenn der Nutzer nach HATEOAS keine direkte Kenntnis von der API des Dienstes haben sollte, sind die Routen nach einem konsistenten Schema zu bestimmen. Der Pfad eines URI ist immer hierarchisch

aufzubauen. Er beginnt mit der übergeordneten Ressource und wird immer weiter durch untergeordnete Ressourcen verfeinert. Weiterhin sind Ressourcen innerhalb des URI immer als Nomen zu formulieren, wobei es üblich ist, Listenressourcen im Plural anzugeben. Die Operationen, die auf den Ressourcen ausgeführt werden, sind ausschließlich über die HTTP-Methoden abzubilden (siehe Tabelle 4). Auch serverseitige Funktionen, welche nicht als eigene Ressource existieren, sind nach dem definierten Schema aufzurufen. Die Funktion *produktBewerten(int bewertung, int produktID)* könnte beispielsweise über eine POST-Anfrage an den URI <http://versandhandelxyz/produkte/{productID}/bewertungen> gestellt werden, wobei die neue Bewertung im Body befindlich ist. Ab einem gewissen Grad der Komplexität sollten dem URI Query-Parameter beigelegt werden. In Abhängigkeit davon, ob eine Paginierung, Sortierung, Filterung oder Projektion auf der anforderten Ressource durchzuführen ist, sind diese auszuwählen.

Ressource / Methode	GET	PUT	POST	DELETE
/produkte	Liefert Repräsentationen aller vorhandenen Produkte aus.	Aktualisiert alle Ressourcen in einer Anfrage (eher unüblich).	Erstellt ein neues Produkt, wobei Server die ID selbstständig vergibt.	Logisches Löschen aller Produkte.
/produkte/33	Liefert die Repräsentation eines speziellen Produktes aus.	Aktualisiert eine spezifische Ressource, falls diese existiert. Andernfalls tritt ein Fehler auf.	Fehler, da POST nicht auf existierende Ressourcen angewendet werden kann.	Logisches Löschen eines Produktes.

Tabelle 4: Typische Semantik von HTTP-Methoden bezüglich URLs

Benachrichtigung des Clients: Der Client sollte auf jede seiner Anfragen eine Antwort erhalten. Dabei ist es unwichtig, mit welcher HTTP-Methode die Anfrage verschickt wurde bzw. welche Ressource sie adressierte. Jede Antwort ist weiterhin mit einem geeigneten Statuscode zu versehen. Die HTTP-Statuscodes können grob in fünf verschiedene Kategorien eingeteilt werden (siehe Abb. 8). Diese Einteilung ist jedoch nicht ausreichend, um den Client geeignet über die Verarbeitung seiner Anfrage zu informieren. Beispielsweise macht es für ihn einen Unterschied, ob seine Anfrage fehlerhaft aufgebaut war (HTTP-Statuscode 400), die Ressource nicht gefunden wurde (404) oder die spezifizierte Methode nicht auf der Ressource ausgeführt werden darf (405). Dementsprechend ist der Statuscode möglichst differenziert auszuwählen, um dem Client ein Höchstmaß an Unterstützung zu bieten. Weiterhin ist es gängige Praxis, dem Client bei manipulierenden Anfragen (durch

POST und PUT) mit einer Repräsentation der erstellten bzw. aktualisierten Ressource zu antworten. Diese Repräsentation kann er direkt in sein System eingepflegen, ohne eine weitere Anfrage tätigen zu müssen.

1xx - Informational

2xx - Success

3xx - Redirection

4xx - Client Error

5xx - Server Error

Abbildung 8: Kategorisierung der HTTP-Statuscodes

Quelle: (Reich 2017)

Dokumentation der API: Insbesondere bei großen Webservices können APIs schnell komplex und unübersichtlich werden. Um die Entwickler zu unterstützen, ist eine einheitliche API-Dokumentation unerlässlich. Insbesonders die Kommunikation zwischen Frontend- und Backendentwicklern wird hierdurch vereinfacht. Idealerweise ist eine API-Dokumentation sowohl vom Menschen lesbar als auch vom Server zu interpretieren. Änderungen in dem zentralen Dokument haben dadurch nun gleichermaßen Auswirkungen auf die tatsächliche und die vom Benutzer wahrgenommene API.

API-Dokumentation mit Swagger

Swagger ist ein verbreitetes Open-Source-Framework, das zur Dokumentation und Generierung von REST-APIs genutzt wird. Das Framework beinhaltet dafür drei Grundfunktionalitäten, welche im Folgenden nach (Swagger 2017) erläutert werden.

Swagger Editor: Der Kern einer jeden Swagger-API ist eine Datei im YAML- bzw. JSON-Format, in denen die API nach einem von der OpenAPI Specification (OpenAPI Specification 2017) festgelegten Schema beschrieben ist. Die Generierung der Datei kann automatisch über Annotations im Quellcode oder manuell erfolgen. Für Letzteres stellt Swagger den sogenannten Swagger Editor bereit. Diese Applikation bietet diverse Hilfsfunktionalitäten bei der Erstellung der YAML- bzw. JSON-Dateien, wie beispielsweise Auto vervollständigung oder Live-Visualisierung.

Eine API (beispielsweise im YAML-Format) lässt sich dabei grob in drei verschiedene Abschnitte einteilen. Im Abschnitt *Tags* (siehe Code-Beispiel 2) werden die Ressourcen aufgeführt, auf die die API Zugriff gewährt. Beispielhaft seien hierfür Pet, Store und User.

```
tags:  
# 1. Ressource Pet  
- name: "pet"  
  description: "Everything about your Pets"  
  externalDocs:  
    description: "Find out more"  
    url: "http://swagger.io"  
# 2. Ressource Store  
- name: "store"  
  description: "Access to Petstore orders"  
# 3. Ressource User  
- name: "user"  
  description: "Operations about user"  
  externalDocs:  
    description: "Find out more about our store"  
    url: "http://swagger.io"
```

Code-Beispiel 2: Definition von Tags in YAML-API

Quelle: (Swagger 2017)

Im zweiten Abschnitt erfolgt die Definition der API-Routen (siehe Code-Beispiel 3). Für jede Route werden gültige HTTP-Operationen spezifiziert. Jedes Paar von URI und Methode ist weiterhin mit diversen Informationen anzureichern. Anfänglich wird beschrieben, auf welche Ressource bzw. Tag sich das Paar bezieht. Im Anschluss werden die Eingabe- bzw. Ausgabedaten der Route spezifiziert und mögliche Fehlermeldungen in Form von HTTP-Statuscodes aufgeführt und erläutert. Abschließend erfolgt die Definition von Nutzerrollen, denen eine Anfrage erlaubt ist.

```

paths:
  # Mit der Route "/pet" ...
  /pet:
    # ... und der Methode POST ...
    post:
      tags:
        # ... kann auf die Ressource Pet zugegriffen werden.
        - "pet"
      summary: "Add a new pet to the store"
      description: ""
      operationId: "addPet"
      consumes:
        - "application/json"
        - "application/xml"
      produces:
        - "application/xml"
        - "application/json"
      parameters:
        - in: "body"
          name: "body"
          description: "Pet object that needs to be added to the store"
          required: true
          schema:
            $ref: "#/definitions/Pet"
      responses:
        405:
          description: "Invalid input"
      security:
        - petstore_auth:
          - "write:pets"
          - "read:pets"
  # Mit der Route "/pet" und der Methode PUT ...
  put:
    tags:
      # ... kann auf die Ressource Pet zugegriffen werden
      - "pet"

  ...

```

Code-Beispiel 3: Definition von Routen in YAML-API

Quelle: (Swagger 2017)

Der letzte Teil des Dokuments wird durch die Definitionen gebildet (siehe Code-Beispiel 4). Die Repräsentationen, die vom Server entgegengenommen bzw. ausgeliefert werden, sind hier im Hinblick auf ihren Aufbau bzw. ihre Attribute zu spezifizieren.

```

definitions:
  Pet:
    type: "object"
    required:
      - "name"
      - "photoUrls"
    properties:
      id:
        type: "integer"
        format: "int64"
      category:
        $ref: "#/definitions/Category"
      name:
        type: "string"
        example: "doggie"
      photoUrls:
        type: "array"
        xml:
          name: "photoUrl"
          wrapped: true
          items:
            type: "string"
      tags:
        type: "array"
        xml:
          name: "tag"
          wrapped: true
          items:
            $ref: "#/definitions/Tag"
      status:
        type: "string"
        description: "pet status in the store"
        enum:
          - "available"
          - "pending"
          - "sold"
    xml:
      name: "Pet"

```

Code-Beispiel 4: Definition von Datenobjekten in YAML-API

Quelle: (Swagger 2017)

Swagger Codegen: Mithilfe von Swagger Codegen kann die bereits erstellte API-Spezifikation automatisch zur Generierung von Programmcode auf Server- sowie auf Clientseite genutzt werden. Für jedes Paar von Route und Methode wird auf dem Server eine prototypische Methode erstellt, die vom Entwickler lediglich mit der Programm- bzw.

Geschäftslogik zu befüllen ist. Gleichzeitig werden für Clients individuelle Programmhbibliotheken generiert, mit der sie unkompliziert auf die API zugreifen können. Bisher unterstützt Swagger Codegen über 40 unterschiedliche Programmiersprachen.

Swagger UI: Das Gegenstück zu Swagger Codegen bildet Swagger UI. Hiermit ist es möglich, die vorliegende YAML- bzw. JSON-Spezifikation für Benutzer geeignet zu visualisieren. Mithilfe einer grafischen Oberfläche sind Ressourcen, URLs, Methoden und weitere Informationen schnell einsehbar. Einzelne Elemente können zwecks übersichtlicherer Darstellung ausgeblendet werden. Weiterhin ermöglicht Swagger UI dem Benutzer das Testen von API-Routen. Durch das Betätigen von speziellen Schaltflächen können ohne programmatische Aufwände Anfragen an den Server gesendet sowie dessen Antworten eingesehen werden. Eine Fehlersuche ist somit einfach und effizient möglich. Letztlich stellen Swagger Codegen und Swagger UI bei gemeinsamer Nutzung sicher, dass die Implementierung einer API zu jedem Zeitpunkt konsistent zu ihrer Dokumentation bzw. Visualisierung ist. Sowohl die Entwickler als auch die Nutzer einer API profitieren von diesem Umstand.

Fazit

REST ermöglicht eine strukturierte Kommunikation innerhalb von verteilten Systemen, welche in den meisten Fällen über HTTP realisiert wird. Nach (Tilkov et al. 2015, S. 2-4) sind als Vorteile des Architekturstils die lose Kopplung und Interoperabilität von Rechenknoten anzusehen. Die Wiederverwendung von Diensten sowie ihre hohe Performanz und Skalierbarkeit sind ebenfalls als positiv aufzufassen. Ein Nachteil von REST ist die fehlende Standardisierung, sodass sich viele Dienste als RESTful bezeichnen, welche die zentralen Constraints nicht erfüllen. Auch die Starrheit bezüglich der Anfrage von Ressourcen bzw. Repräsentationen ist als negativ einzustufen. Die Vor- und Nachteile sind abschließend in Abbildung 9 veranschaulicht.



Abbildung 9: Vor- und Nachteile von REST

GraphQL

GraphQL ist eine Abfragesprache für strukturierte Daten, welche 2012 von Facebook entwickelt und 2015 anschließend veröffentlicht wurde (vgl. GraphQL 2017). Ziel von GraphQL ist es, die Datenübermittlung zwischen Server und Client einfacher und effizienter zu gestalten. Neben der Referenz-Implementierung in JavaScript existieren Bibliotheken für andere Programmiersprachen, wie beispielsweise C#, Java oder Python.

Probleme von REST

Dienste, die streng nach dem Architekturstil REST aufgebaut sind, können zum Flaschenhals für eine Client-Server-Interaktion werden. Dies beruht nach (Facebook 2015) auf drei Tatsachen.

Starrheit der Daten: Ein RESTful Dienst legt statisch fest, welche Ressourcen bzw. Repräsentationen angefragt werden können. Er spezifiziert dazu jeweils die URLs, auf die sich der Client beziehen muss, sowie die Operationen, die ausgeführt werden können. Folgendes Szenario verdeutlicht dabei ein mögliches Problem (siehe Abbildung 10). Der Client möchte beispielsweise wissen, mit welchem Wetter eine bestimmte Person morgen bei der Arbeit rechnen muss. Der Server bietet ihm dafür drei Endpunkte, mithilfe der er Personen, Arbeitgeber und Standorte abfragen kann. Um seine Ausgangsfrage zu beantworten, muss der Client nun drei verschiedene Anfragen absenden und jeweils auf die Ergebnisse warten. Dieser erhöhte Aufwand resultiert daher, dass der Server keinen Endpunkt anbietet, anhand dessen die Problemstellung innerhalb lediglich einer Anfrage bzw. Antwort gelöst werden kann. Eine REST-API zu konstruieren, die auf jede nur mögliche verschachtelte Anfrage innerhalb einer Nachricht antwortet, ist für mittlere bis große Dienste nicht machbar.

Frage des Clients: Mit welchem Wetter muss Person 123 morgen bei der Arbeit rechnen?

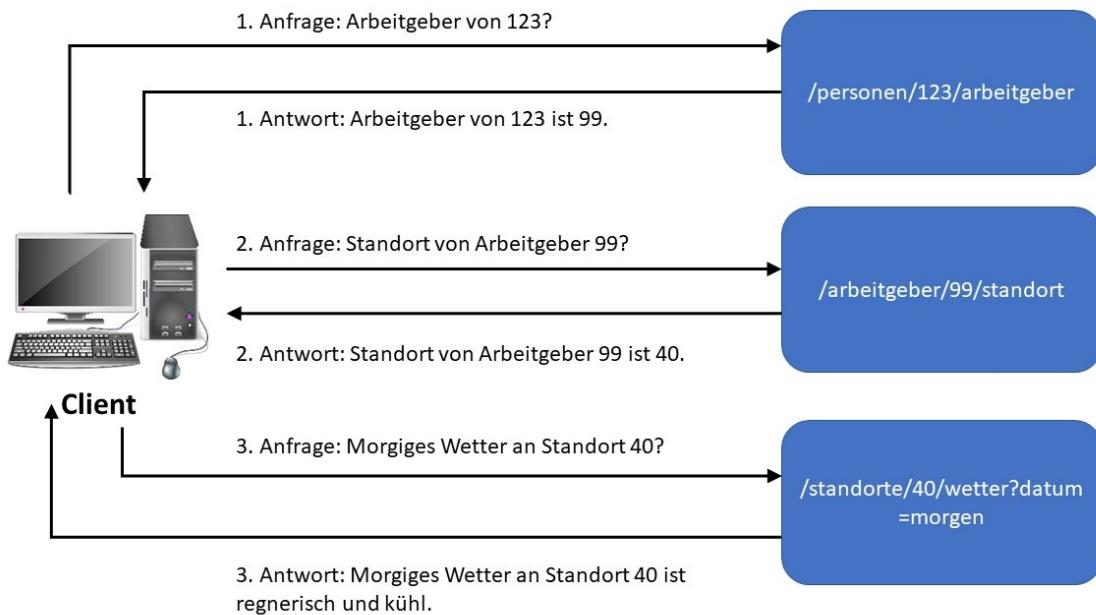


Abbildung 10: Mehrmaliges Anfragen als Nachteil von REST

Fehlende Versionierung: Die Server-Komponenten eines RESTful Dienstes entwickeln sich kontinuierlich weiter. Insbesondere die Ressourcen bzw. Repräsentationen unterliegen dabei häufig Veränderungen in Form von neuen Datenfeldern. In der Regel greifen jedoch heterogene Clients auf den Dienst zu, welche sich im Hinblick auf ihre Bedürfnisse voneinander unterscheiden. Manche von diesen Benutzern benötigen beispielsweise die hinzugefügten Datenfelder als Antwort auf ihre Anfrage, während die Datenfelder für ältere Clients keine Bedeutung haben könnten und nur zusätzlicher Ballast sind. Dadurch bedingt, dass der Server lediglich einen Endpunkt für jede Ressource anbietet, erhalten alle Clients dieselbe Version der Repräsentation. Auf individuelle Ansprüche wird somit keine Rücksicht genommen.

Schwache Typisierung: RESTful Services empfangen Informationen häufig in der Form von JSON- oder XML-Dokumenten. Diese müssen lediglich eine bestimmte Syntax erfüllen, um vom Server akzeptiert zu werden. Einzelne Datenfelder innerhalb der Dokumente sind jedoch nur schwach typisierbar. Für den Server bedeutet es daher einen erheblichen Mehraufwand, jede übermittelte Repräsentation auf ihre semantische Korrektheit zu überprüfen.

Abhilfe

Ein GraphQL-Server wird von jedem Client und unabhängig von den angefragten Daten über denselben Endpunkt erreicht (siehe Abb. 11). Weiterhin unterscheidet der Server nicht zwischen den Aufrufoperationen. Geschieht die Client-Server-Kommunikation über HTTP,

führt der Client beispielsweise nur POST-Anfragen an eine bestimmte URI durch. Die Semantik der Anfrage ist dabei ausschließlich innerhalb des Bodies definiert.

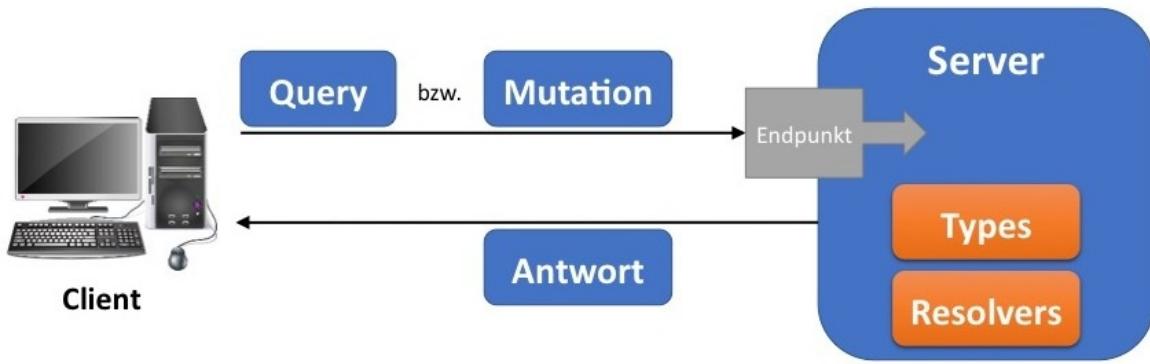


Abbildung 11: Schaubild GraphQL

Anfragen, die dem Client lediglich Daten übermitteln sollen, werden innerhalb einer sogenannten Query formuliert (analog zu GET-Anfrage). Ist eine Manipulation der serverseitigen Daten gewollt, muss eine Mutation übermittelt werden (analog zur POST-, PUT- und DELETE-Anfrage). Zur Verarbeitung von Queries und Mutations besitzt der Server Types und Resolvers. (vgl. GraphQL 2017)

Query

Eine Query veranlasst eine Datenübermittlung vom Server zum Client. Der Client bestimmt dabei dynamisch, welche Informationen er erhalten möchte. Er kann somit nicht nur das Objekt eingrenzen, das übermittelt werden soll, sondern zusätzlich die Datenfelder selektieren.

Besitzt ein Objekt Auto beispielsweise die Attribute Hersteller, Modell, Leistung, Hubraum, Sitzanzahl und Preis, kann der Client explizit mitteilen, welche Datenfelder er tatsächlich benötigt (siehe Code-Beispiel 5). Eine Query hat dementsprechend immer dieselbe Form, wie die zugehörige Antwort. Weiterhin ist es möglich, eine Query beliebig tief zu verschachteln. Ist das Datenfeld eines Objekts wiederum ein Objekt, können auch hier Datenfelder selektiert werden.

```

# Query an den Server
query AllgemeineInformationenZuAuto455{
  auto(id: "455") {
    hersteller{
      name
      standort
    }
    modell
    leistung
    hubraum
  }
}

# Antwort vom Server:
{
  "data": {
    "auto": {
      "hersteller": {
        "name": "vw",
        "standort": "wolfsburg"
      },
      "modell": "golf",
      "leistung": "150",
      "hubraum": "2"
    }
  }
}

```

Code-Beispiel 5: Statische GraphQL-Query mit passender Antwort

Variablen

Der bisherige Ansatz hat den Nachteil, dass Parameter fest mit der Query verknüpft sind. Sollte im vorherigen Beispiel ein Auto mit der Identifikationsnummer 499 abgefragt werden, ist der komplette Query-String zu verändern bzw. eine komplett neue Query zu spezifizieren. Um Queries generisch nutzen zu können, dienen Variablen (siehe Code-Beispiel 6). Innerhalb der Anfrage werden alle dynamischen Parameter durch Platzhalter ersetzt. Abhängig vom eingesetzten Protokoll erfolgt die Auflösung der Variablen in einem separaten Wörterbuch. Eine Anfrage muss somit lediglich einmalig erzeugt werden, während das Wörterbuch dynamisch anzupassen ist.

```

# Query an den Server
query HerstellerinformationenZuEinemBestimmtenAuto($id: ID){
  auto(id: $id) {
    hersteller
    modell
  }
}

# Wörterbuch an den Server
{
  "id": "499"
}

# Antwort vom Server
{
  "data": {
    "auto": {
      "hersteller": "fiat",
      "modell": "500"
    }
  }
}

```

Code-Beispiel 6: Variable GraphQL-Query in Verbindung mit Wörterbuch und passender Antwort

Direktiven

Um Queries noch generischer formulieren zu können, dienen Direktiven (siehe Code-Beispiel 7). Jedes angefragte Datenfeld kann um ein `@include(if: Boolean)` bzw. `@skip(if: Boolean)` ergänzt werden. Soll ein Datenfeld bei einer bestimmten Bedingung beispielsweise nicht angefragt werden, ist ein `@skip(if: $Bedingungsvariable)` anzuhängen. Dieselbe Query kann somit in Abhängigkeit des Wörterbuchs für verschiedene Zwecke genutzt werden.

```

# Query an den Server
query AllgemeineInformationenZuEinemBestimmtenAuto($id: ID, $preisangabe: Boolean){
  auto(id: $id) {
    hersteller
    modell
    preis @include(if: $preisangabe)
  }
}

# Wörterbuch an den Server
{
  "id": "1"
  "preisangabe": true
}

# Antwort vom Server
{
  "data": {
    "auto": {
      "hersteller": "mercedes",
      "modell": "a-klasse",
      "preis": 20000
    }
  }
}

```

Code-Beispiel 7: GraphQL-Query mit Variablen und Direktive

Mutation

Das Pendant zu der Query bildet die Mutation. Immer wenn Daten auf Seiten des Servers generiert, aktualisiert oder gelöscht werden sollen, ist diese Art der Anfrage zu übermitteln. Der Aufbau einer Mutation ähnelt dabei dem einer Query. Es kann ebenfalls eine beliebig tiefe Verschachtelung der Anfrage erfolgen. Auch Variablen und Direktiven sind nutzbar. Lediglich das Schlüsselwort *query* ist in *mutation* zu ändern.

Types

Types liegen im Server vor und bilden die Kommunikationsgrundlage zwischen Server und Client. Hierunter sind die logischen Datenmodelle zu verstehen, auf denen Queries und Mutations arbeiten. Jeder sogenannte GraphQL Object Type beinhaltet eine Menge von Attributen (siehe Code-Beispiel 8). Dies können zum einen Skalare sein, die keine weitere Auflösung erlauben. Von GraphQL bereitgestellte Skalare sind Int, Float, String, Boolean und ID. Auch GraphQL Object Types können als Attribute vorliegen, wenn sie an anderer

Stelle definiert werden. Weiterhin erlaubt GraphQL Listentypen sowie die Deklaration eigener Aufzählungen. Spezielle Typen sind hingegen der Query bzw. Mutation Type. Mithilfe dieser kann spezifiziert werden, welche Objekte seitens des Clients abgerufen bzw. manipuliert werden können. Sie bilden damit den Einstiegspunkt für jede passende Anfrage.

```
# Serverseitiges Datenschema

# Einstiegspunkte für Queries
type Query {
    # Client kann Auto-Objekt anhand von ID abrufen
    auto(id: ID!): Auto
}

type Mutation {
    ...
}

type Auto {
    # Hersteller darf nicht null sein
    Hersteller: Hersteller!
    Modell: String
    Hubraum: Int
    Antrieb: AntriebEnum
    # Liste von Vorbesitzern
    Vorbesitzer: [String]
}

type Hersteller {
    Name: String
    Standort: String
}

enum AntriebEnum {
    Allrad
    Hinterrad
    Vorderrad
}
```

Code-Beispiel 8: Serverseitige Definition von Object Types

Resolvers

Types definieren den Aufbau von einzelnen Datentypen. Durch Kenntnis dieser Struktur kann der Server entscheiden, ob eingehende Anfragen valide sind oder das Schema verletzen. Um Antworten zu generieren, sind Types jedoch nicht ausreichend. Es muss zwangsläufig ein Bezug zwischen der Struktur der Daten und ihrer tatsächlichen Lage hergestellt werden. Hierzu dienen sogenannte Resolvers (siehe Code-Beispiel 9). Ein

Resolver ist eine Funktion (in diversen Sprachen implementierbar), die der Server aufruft, wenn ein spezifisches Datenfeld zurückgeliefert werden muss. Der Inhalt der Funktion wird durch die Geschäftslogik des Serverdienstes definiert und ist demnach eigenständig zu implementieren. Typischerweise wird bei eingehender Anfrage der Root Resolver aufgerufen. Dieser liefert das Ausgangsobjekt (im aktuellen Beispiel ein Auto) zurück. Abhängig von den angefragten Datenfeldern werden auf diesem Objekt weitere Resolver aufgerufen, die zusätzliche Daten beschaffen. Die ermittelten Informationen werden nach Abschluss der Routine geeignet zusammengefügt und an den Client übermittelt.

```
# Resolver-Beispiel-Implementierung in JavaScript

# Root Resolver
Query: {
  auto(obj, args, context) {
    return context.meineDatenbank.ladeAutoAnhandID(args.id).then(
      data => new Auto(data)
    )
  }
}

# Aufgerufen, wenn Auto-Objekt bereits vorhanden
Auto: {
  hersteller(obj, args, context) {
    return context.meineDatenbank.ladeHerstellerAnhandID(obj.herstellerID).then(
      data => new Hersteller(data)
    )
  }

  modell(obj, args, context) {
    return obj.modell
  }

  ...
}

...
}
```

Code-Beispiel 9: Serverseitige Definition von Resolvers

Best Practices

GraphQL ist lediglich eine Abfragesprache und gibt keine Implementierungs- bzw. Nutzungsrichtlinien vor. Dennoch existiert nach (GraphQL 2017) eine Reihe von Best Practices, die im Folgenden skizziert werden.

- **HTTP:** GraphQL-Anfragen und -Antworten können theoretisch über jedes

Netzwerkprotokoll übermittelt werden. Aufgrund seiner Verbreitung ist jedoch das Hypertext Transfer Protocol zu bevorzugen. Der Dienst wartet dabei typischerweise unter dem Pfad `/graphql` auf Anfragen seitens des Clients. Queries und Mutations können weiterhin sowohl im Body eines POST-Requests als auch in der Query eines GET-Requests übergeben werden. Unabhängig davon, ob eine Query oder eine Mutation versendet wurde, antwortet der Server idealerweise mit dem angefragten bzw. dem manipulierten Datenobjekt. Auch Fehlermeldungen werden im Body der Antwort übergeben.

- **JSON:** Vom Server übermittelte Daten befinden sich zwecks Lesbarkeit stets in der JavaScript Object Notation.
- **Keine Versionierung:** Der Client spezifiziert bei jeder Anfrage, auf welche Datenfelder er sich bezieht. Dies macht eine Versionierung des GraphQL-Dienstes überflüssig.
- **Caching:** Im Gegensatz zu REST ist Caching bei GraphQL-Diensten nicht über die URI einer Anfrage möglich. Serverseitiges Caching kann dennoch betrieben werden, wenn jedes Objekt durch eine global eindeutige ID identifiziert wird. Häufige Datenbankzugriffe sind somit vermeidbar, was in einer Performancesteigerung des Dienstes resultiert.

Zusammenfassung

Die Grundprinzipien von GraphQL werden abschließend nach (Facebook 2015) zusammengefasst.

- **Hierarchisch:** Anfragen sowie Datenmodelle sind hierarchisch aufgebaut. Ein Kernelement wird dabei immer weiter in seine Grundbestandteile zerlegt, bis ausschließlich Skalare vorliegen. Dies unterstützt sowohl Entwickler als auch Anwender bei ihrer Arbeit.
- **Produktbezogen:** Im Mittelpunkt von GraphQL stehen die Daten bzw. deren Benutzer.
- **Clientspezifische Anfragen:** Der Client bestimmt dynamisch, welche Daten er benötigt bzw. welche Daten für ihn gegenstandslos sind. Der Server richtet sich vollkommen nach dem Client.
- **Rückwärtskompatibilität:** Jede Clientversion kommuniziert mit demselben Endpunkt und beschafft sich ihre benötigten Daten.
- **Beliebige Datenbeschaffung:** Der Server bestimmt für jedes einzelne Datenfeld seine eigene Beschaffungsroutine, ohne dabei von bestimmten Technologien abhängig zu sein.
- **Applikationsschicht:** GraphQL ist in der Applikationsschicht angesiedelt und muss sich daher nicht um den Transport von Daten kümmern, sondern lediglich um deren Verarbeitung.

- **Stark typisiert:** Queries und Mutations können vom Server effizient hinsichtlich Semantik und Syntax validiert werden.
- **Selbstbeschreibend:** GraphQL-Anfragen und -Datenmodelle werden in einem eigenen Format beschrieben. Hierdurch erhöht sich die Unabhängig zu anderen Systemen und Programmiersprachen.

Fazit

Ein direkter Vergleich der vorgestellten Kommunikationsmethoden ist aufgrund ihrer Heterogenität nur schwer möglich. Es wurden eine Basistechnik, ein Netzwerkprotokoll, ein Architekturstil sowie eine Abfragesprache vorgestellt, welche zwar im Allgemeinen dem Informationstausch dienen, diesen jedoch auf verschiedenen Ebenen unterstützen. Welche Methode Anwendung finden sollte, hängt immer von der jeweiligen Problemstellung ab und ist nicht generell vorherzusagen. Die nachfolgende Gegenüberstellung soll die jeweiligen Anwendungsgebiete trotzdem grob verdeutlichen und somit Klarheit bezüglich der Verwendung schaffen (siehe Tab. 5).

Kommunikationsmethode / Eignung:	Geeignet für:	Ungeeignet für:
RPC	Auslagerung von Methodenlogik	Datenlastige Nachrichtenaustausche
WebSocket	Echtzeitanwendungen	Statische bzw. unkritische Daten
REST	Öffentliche Dienste mit standardisierter Schnittstelle, Austausch von einheitlichen Datenobjekten	Methodenaufrufe, kontinuierlich angepasste Datentypen
GraphQL	Austausch von kontinuierlich angepassten Datentypen / -objekten	Methodenaufrufe

Tabelle 5: Gegenüberstellung Kommunikationsmethoden

Literatur- und Quellenverzeichnis

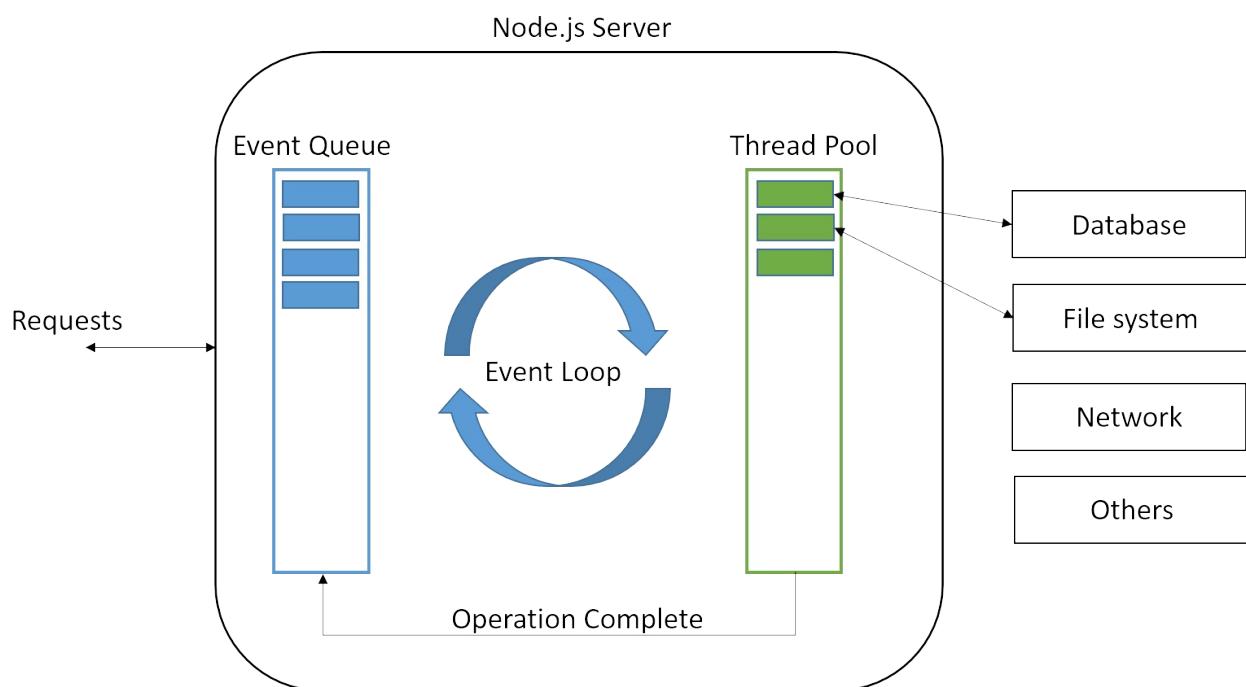
- Alexander Schill und Thomas Springer. Verteilte Systeme: Grundlagen und Basistechnologien. Springer Vieweg, Berlin; Heidelberg, 2. Auflage, 2012.
- Cesare Pautasso, Erik Wilde und Rose Alarcon. REST: Advanced Research Topics and Practical Applications. Springer, New York; Heidelberg; Dordrecht; London, 2014.
- Erich Reich. <https://blog.qmo.io/ultimate-guide-to-api-design/>, 2017. Aufgerufen: 22.06.2017
- Facebook. <https://facebook.github.io/react/blog/2015/05/01/graphql-introduction.html>, 2015. Aufgerufen: 28.06.2017
- Google. <http://www.grpc.io/>, 2017. Aufgerufen 29.06.2017
- GraphQL. <http://graphql.org>, 2017. Aufgerufen: 28.06.2017
- Guy Levin. <https://dzone.com/articles/top-5-rest-api-security-guidelines>, DZone, 2016. Aufgerufen: 20.06.2017
- Martin Fowler. <https://martinfowler.com/articles/richardsonMaturityModel.html>, 2010. Aufgerufen: 20.06.2017
- OpenAPI Specification. <https://github.com/OAI/OpenAPI-Specification/blob/master/versions/2.0.md>, 2017. Aufgerufen: 22.06.2017
- Roy Thomas Fielding. Architectural Styles and the Design of Network-based Software Architectures. University of California, Irvine, 2000.
- Stefan Tilkov, Martin Eigenbrodt, Silvia Schreier und Oliver Wolf. REST und HTTP. Entwicklung und Integration nach dem Architekturstil des Web. dpunkt.verlag, Heidelberg, 3., aktualisierte und erweiterte Auflage, 2015.
- Swagger. <https://swagger.io>, 2017. Aufgerufen: 22.06.2017
- Tim Berners-Lee, Roy Thomas Fielding und Larry Masinter. RFC 3986 - Uniform Resource Identifier (URI): Generic Syntax. 2005.
- Vinay Sahni. <http://www.vinaysahni.com/best-practices-for-a-pragmatic-restful-api>, 2015. Aufgerufen: 22.06.2017
- Fette, I. & Melnikov, A. <https://tools.ietf.org/html/rfc6455>

Backend Entwicklung

Autoren: Niklas Harting

Node.js Event-Loop

Node.js Anwendungen werden in nur einem Thread ausgeführt (Code des Programmierers). Alle anderen Operationen laufen parallel und sollte also eine Operation diesen "Anwendungs"- Thread blockieren, wird diese in einen separaten Thread aus einem Pool ausgelagert. Sobald ein Event auftritt (also z.B. Daten wurden aus einer Datei in einen RAM-Puffer geschrieben), wird dieses in die Event Queue eingereiht. Der Event-Loop iteriert nun über alle Einträge in der Event-Queue und ruft die passende Callback-Funktion zu diesem Event auf. [5 6](#)



Node.js Event-Loop 5

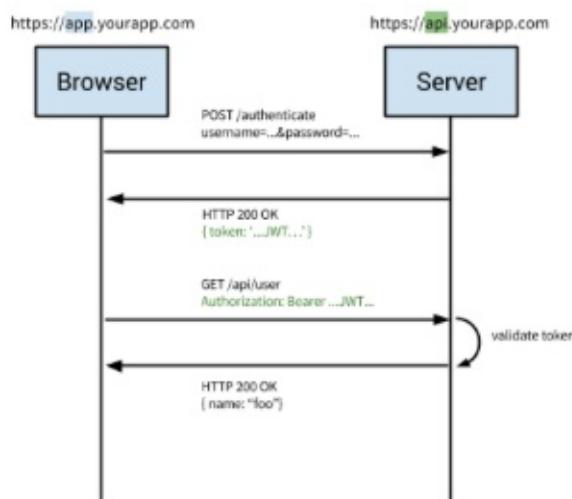
Bedingt durch das Modell der eventgesteuerten Programmierung, ist es im Allgemeinen nicht notwendig, sich innerhalb einer Anwendung Gedanken über die Verteilung von Aufgaben in verschiedene Threads zum Zwecke der Laufzeitoptimierung zu machen. Probleme, die nebenläufige Anwendungen für den Entwickler mit sich bringen, verschwinden so gänzlich. Ebenfalls erwächst aus einer Single-Thread Lösung der Vorteil, dass die aufwendige und Ressourcen intensive Verwaltung von verschiedenen Threads und Prozessen durch das Betriebssystem reduziert werden kann. Dadurch werden eventgesteuerte Anwendungen sehr performant und können hoch skaliert werden. Diesen Vorteil hat man aber nur, wenn das Betriebssystem diese Funktion unterstützt, da der Thread Pool vom Betriebssystem verwaltet wird.

JWT vs. Session Cookie

JSON Web Token (JWT)

Die Token-Based Authentication ist **Stateless**. Das bedeutet, dass auf dem Server keine Information gespeichert werden, sondern nur das JSON Web Token beim Client gespeichert wird. [9](#)

Beim einer Authentifizierung wird auf dem Server ein neues JWT erstellt und zurück an den Client geschickt. Dieser speichert das JWT im Local Storage. Bei jeder weiteren Anfrage sendet der Client nun das JWT mit, welches dann auf dem Server verifiziert wird. [9](#)



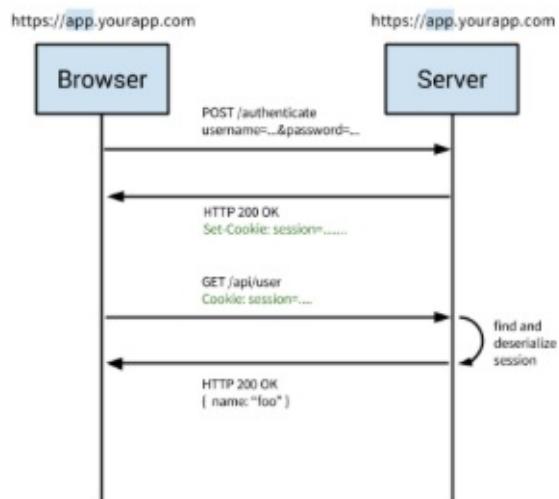
Token-Based Authentication [9](#)

Bei einem Logout des User, wird das JWT dann einfach beim Client gelöscht und es ist keine weitere Interaktion mit dem Server notwendig. [9](#)

Session Cookie

Die Cookie-Based Authentication ist im Gegensatz zur Token-Based Authentication **Stateful**. Es müssen also sowohl auf dem Server sowie beim Client Daten gespeichert werden. Der Server speichert in einer Datenbank alle aktiven Sessions und auf der Client Seite wird ein Cookie mit der Session ID gespeichert. [9](#)

Beim einer Authentifizierung wird dann also auf dem Server eine neue Session ID erstellt, welche in der Datenbank gespeichert wird und in einem Cookie zurück an der Client geschickt wird. Bei jeder weiteren Anfrage zum Server wird nun der Cookie mitgeschickt und es wird überprüft, ob die Session ID aus dem Cookie mit einer Session ID aus der Datenbank übereinstimmt. [9](#)



Cookie-Based Authentication 9

Wenn sich der User nun ausloggt, muss auf dem Client der Cookie und auf dem Server die Session ID aus der Datenbank gelöscht werden. 9

Design

Autor: Lutz Winkelmann

Einführung

Im Folgenden sollen Designentscheidungen, deren Auswirkungen und die Notwendigkeit dieser beschrieben werden. Der erste Eindruck ist entscheidend und wird durch das Aussehen und die Handhabe einer Anwendung geprägt. Das Konzipieren eines Designs bringt gewisse Herausforderungen mit sich. Einige Fragen die sich jeder Designer stellen sollte wären:

- Auf welchen Endgeräten wird die Anwendung genutzt?
- Mit welchen Browsern muss die Anwendung kompatibel sein? (Webanwendungen)
- Welche Displaygrößen müssen unterstützt werden?
- Sind alle Displays hochauflösend?
- Stellen die Displays eine ausreichende Farbpalette für ein ausgefallenes Design zur Verfügung?
- Müssen technische Voraussetzungen wie der Verbrauch von Datenvolumen, beispielsweise beim Laden von Bildern, betrachtet werden?
- Welche Zielgruppe wird adressiert?
- Wird die Anwendung womöglich von Nutzern mit Einschränkungen genutzt?
- Wenn ja, wie kann man dennoch eine gute Nutzbarkeit gewährleisten?

Die nachfolgenden Kapitel beschäftigen sich mit diesen Fragestellungen und sollen einen groben Überblick über Designentscheidungen und deren Folgen nahe bringen.

Konzepte und Designansätze

Bisher wurde die Designkonzeption nur sehr abstrakt geschildert. Im Folgenden sollen jetzt einige Konzepte und Ansätze erläutert werden, welche diese aufgreifen und ergänzen. Bei der Konzeptionierung eines Designs hat der Entwickler meist schon eine grobe Vorstellung des Aussehens einer Anwendung im Kopf. Nachdem die Zielgruppen analysiert wurden kann diese Vorstellung jetzt überarbeitet werden.

Skeumorphismus

Beim Skeumorphsimus wird versucht, das Aussehen von Dingen zu immitieren. Der Skeumorphismus taucht in der Softwareentwicklung bereits bei den Anfängen der Betriebssysteme mit grafischer Oberfläche auf. Icons wie der Papierkorb oder Ordner sollten hier ein schnelles Verständnis und einen leichten Einstieg in die Computerwelt bieten. Wenn eine Anwendung zur Verwaltung von Notizen geschrieben werden soll, so könnte man diese Notizen beispielsweise wie reale Notizzettel aussehen lassen. Hierbei wird das dritte, vierte, siebte und achte Gesetz der Einfachheit erfüllt. Der Anwender kennt, was er sieht und braucht somit keine Zeit für das Lernen der Handhabung. Es kann auf eine Nutzbarkeit wie bei normalen Notizzetteln vertraut werden und durch das reale Aussehen werden bei den Nutzern positive Emotionen ausgelöst. Aus den genannten Gründen ist der Skeumorphismus besonders bei Zielgruppen angebracht, welche sich nicht schnell an neue Technologien anpassen. In der Regel sind dies ältere Personen, welche technisch nicht sonderlich bewandt sind.

Ein weiterer sinnvoller Anwendungsfall ist gegeben, wenn die Anwendung der Nutzung des realen Gegenstands ähnelt. Ein DJ-Pult kann digital ähnlich dargestellt werden wie ein reales. Außerdem können die Schalter, Regler und Scheiben intuitiv genutzt werden, wie in der folgenden Abbildung dargestellt.

Oft bekommen Anwendungen einen Hintergrund aus der realen Welt, ohne dass dieser einen Zusammenhang zu der Anwendung darstellt. Beispielsweise könnte eine Kalenderanwendung einen Baumrinden- oder Leder-Hintergrund bekommen, um ein edleres Aussehen zu vermitteln. (vgl [3])



Mischpult im Skeumorphismus Design ([4])

Flat Design

Das Flat Design wurde mit dem Kachel-Design des ersten Windows Phones bekannt. Hier wird nicht versucht die Realität zu immitieren, sondern Elemente so einfach wie möglich ohne große Schnörkel darzustellen. Dies bringt die Möglichkeit mit sich, Elemente viel freier zu gestalten und völlig neue Ansätze für bereits vorhandene Dinge zu definieren.

Beispielsweise könnte ein Taschenrechner einen größeren Fokus auf Funktionalität bekommen, anstatt mit dem bekannten Layout nur die einfachsten Rechenfunktionen zur Verfügung zu stellen.

Die dargestellten Elemente sind meist in einer Farbe gestaltete, flache Objekte ohne 3D-Effekte wie einen Schlagschatten. Dies sorgt für eine schnellere Umsetzung der Design-Konzeption, da keine Grafiken pixelgenau platziert werden müssen. Die gesparte Zeit kann dann in die Logik der einzelnen Komponenten investiert werden.

Durch das Flat Design kann das erste, zweite, fünfte und sechste Gesetz der Einfachheit erfüllt werden. Die Darstellung der Elemente wird auf ein Minimum reduziert, wodurch die Anwendungen meist geordneter wirken. Außerdem wird die Anwendung durch einheitlich große Elemente, eine gleiche Farbgestaltung dieser oder eine Gruppierung dieser, organisiert. Zuerst mag das Flat Design aufgrund seiner geringeren Ausprägungen der gestalterischen Merkmale komplexer erscheinen, jedoch wird die Nutzung nach kurzer Einarbeitungszeit intuitiver, da es in vielen verschiedenen Anwendungen ähnlich eingesetzt

wird. Es ist sehr einfach den gewünschten Kontext mit den Elementen zu schaffen, da hierbei großer Wert auf die Farbgestaltung sowie die Positionierung gelegt werden kann. (vgl. [3])

Material Design

Im Jahr 2014 hat Google das Material Design vorgestellt. Auf den ersten Blick ähnelt es stark dem Flat Design aufgrund seiner einfachen Darstellung. Allerdings wird beim Material Design die Dreidimensionalität durch eine z-Koordinate ergänzt. Jede Komponente hat die gleiche Dicke von 1dp (Pixel in Abhängigkeit zur Bildschirmgröße). Es ist möglich, Komponenten übereinander darzustellen, wodurch diese einen Schatten auf die unterliegende Ebene werfen. Des Weiteren können die Komponenten durch dp Angaben skaliert und verschoben werden, was für eine bessere Darstellung auf verschiedenen Bildschirmgrößen sorgt. Der Inhalt von Komponenten kann ebenfalls frei auf diesen verschoben und skaliert werden. Die Form der Komponenten kann sich während der Laufzeit ändern was für neue Möglichkeiten des Designs sorgt. Google entwickelt das Material Design stetig weiter und stellt bereits viele Komponenten zur Verfügung, welche sich einfach in eigene Projekte einbauen und verändern lassen. Besonders hervorzuheben sind die vielseitigen Animationen, welche im Material Design eingesetzt werden können und somit, abgesehen von den im Flat Design besprochenen, besonders das siebte Gesetz der Einfachheit ansprechen. (vgl. [5])

Skeumorphismus vs Flat Design vs Material Design

Welches Design Konzept angewendet wird, ist weitestgehend geschmackssache. Allerdings sollte der Designer bei der Wahl des Konzepts die Zielgruppe beachten. Für Personen, welche sich nicht schnell an neue Technologien gewöhnen, beziehungsweise diese verweigern, ist der Skeumorphismus eine gute Wahl, da in diesem bekannte Dinge der realen Welt eingesetzt werden. Auch wenn die Anwendung lediglich einen realen Gegenstand imitieren soll, ist es sinnvoll, den Skeumorphismus anzuwenden, da der Nutzer bereits mit dem Umgang oder dem Aussehen des Gegenstands vertraut ist.

Steht die Funktionalität oder die Einfachheit einer Anwendung hingegen im Vordergrund, so ist das Flat Design oder das Material Design aufgrund der einfacheren Designerstellung vorteilhaft. Auch abstrakte Dinge können mit diesen Konzepten einfacher dargestellt werden. (vgl. [3])

Das Material Design wirkt zwar durch 3D-Ebenen und Animationen freundlicher als das Flat Design, allerdings stellen diese Eigenschaften auch einen gewissen Overhead dar. Bei Anwendungen, welche häufig genutzt werden, beispielsweise weil der Beruf die Nutzung erfordert, kann dies auf Dauer eher negativ als positiv wirken. Also sollten vor allem die Animationen mit Bedacht eingesetzt werden, da sie gegen das dritte Gesetz der Einfachheit, die Zeit, sprechen.

Dynamische Bildschirmgrößen

Häufig werden Anwendungen für verschiedene Plattformen oder Geräte zur Verfügung gestellt. So ist eine Webanwendung beispielsweise auf einem Desktop-PC, einem Tablet, einem Smartphone oder ähnlichem ansteuerbar. Hierbei ergibt sich das Problem, dass Komponenten der Anwendung entweder mit der Displaygröße skalieren oder austauschbar sein müssen. Für diese Problemstellung gibt es mehrere Ansätze, wovon einige im Folgenden beschrieben werden sollen. Diese arbeiten mit Media Queries, welche die verschiedenen Viewportgrößen des Displays an die Anwendung weitergeben. (vgl. [6])

Adaptive Design

Beim Adaptive Design werden mehrere Layouts erstellt, welche für verschiedene Bildschirmgrößen verwendet werden sollen. Meist wird ein Layout für ungefähr 19 Zoll und größer, 13 Zoll und größer und kleiner als 13 Zoll erstellt. Die Layouts werden in CSS mit dem Zusatz der Bildschirmgrößen formuliert und entsprechend der Viewportgröße des Endgeräts angezeigt. Das Adaptive Design ist also an die Endgeräte ausgerichtet, von welchen die Anwendung genutzt wird. (vgl. [6])

Responsive Design

Beim Responsive Design werden die Komponenten so gestaltet, dass sie sich dynamisch der vorhandenen Bildschirmgröße anpassen. Auch hier gibt es einige Layoutvorgaben, welche bei bestimmten Viewportgrößen angewendet werden. Diese beziehen sich allerdings meist auf eine Umstrukturierung des Layouts. Hierdurch können Elemente versteckt, anders angeordnet oder sichtbar gemacht werden. (vgl. [6])

Eine Navigationsleiste am linken Bildschirmrand wird bei großen Bildschirmen beispielsweise meist mit einem Icon und einem beschreibenden Namen pro Element versehen. Auf mittelgroßen Bildschirmen wird dann nur noch das entsprechende Icon angezeigt. Auf einem Smartphone verbirgt sich die Navigationsleiste dann oft hinter einem sogenannten Burger-Menü, welches ein Button ist, der durch Anklicken die Navigationsleiste erscheinen lässt.

Adaptive Design vs Responsive Design

Das Adaptive Design lässt sich schneller umsetzen, da es nur für eine bestimmte Anzahl an Bildschirmgrößen getestet werden muss. Außerdem ähneln die Resultate meist stärker den Mockups als beim Responsive Design, da hauptsächlich die Größe der Elemente angepasst wird. Beim Responsive Design können verschiedene Viewportgrößen zu unterschiedlichen Layouts führen, was den Nutzer verwirren kann, allerdings ist beim Responsive Design eine gute Darstellung für alle Viewportgrößen bis zu einem gewissen Maximum gegeben. Wenn jemand bei einer adaptiv gestalteten Anwendung auf einem 17 Zoll Monitor eine Anwendung startet, welche für 13 und 19 Zoll optimiert wurde, so gibt es viele Leerräume, welche aufgrund der statischen Darstellung entstehen. Somit kann man sagen, dass der Entwicklungsaufwand eines adaptiven Designs zwar deutlich geringer ist, jedoch das Resultat eines Responsive Designs meist besser aussieht und zukunftssicherer ist, was neue Bildschirmgrößen anbelangt. (vgl. [6])

Pagination vs Infinite Scrolling

Muss in einer Anwendung eine Liste mit vielen Einträgen dargestellt werden, so gibt es hierfür zwei Möglichkeiten mit Vor- und Nachteilen. Die bekannteste Variante ist wohl eine Tabelle, welche eine gewisse Anzahl an Elementen auf einer Seite darstellt. Durch einen Klick auf die nächste Seitenzahl werden neue Elemente geladen und anstatt der bisher angezeigten, dargestellt. Dieses Prinzip ist bei den meisten Onlineshops anzutreffen, da schnell durch die Elemente navigiert werden kann. Wird eine Liste beispielsweise nach dem Preis sortiert und man ist auf der Suche nach etwas aus dem mittleren Preissegment, so muss man nur auf die mittlere Seitenzahl gehen und kann von dort aus weiter navigieren.

Auch beim Infinite Scrolling gibt es häufig die Möglichkeit Suchergebnisse zu sortieren. Hier werden die einzelnen Elemente allerdings nicht auf mehreren Seiten angezeigt, sondern durch das Runterscrollen automatisch nachgeladen. Dieser Prozess wird "Lazy Loading" genannt. Ein Entwickler muss sich hierbei allerdings die Frage stellen, wie viele Elemente pro Ladezyklus geladen werden sollen und wann dieser Zyklus ausgelöst wird. Auf mobilen Endgeräten sollten beispielsweise maximal 15 bis 30 Elemente geladen werden, da das Display kleiner ist und das Datenaufkommen möglichst gering gehalten werden sollte. Teilweise wird das Infinite Scrolling auch mit einem "Elemente Nachladen" Button versehen, wodurch der Ladezyklus manuell ausgeführt werden muss. Dieser Ansatz ähnelt dann bereits wieder der Paginierung, allerdings werden die neuen Elemente zusätzlich, zu den bereits geladenen, angezeigt und nicht stattdessen.

Auch muss für das Infinite Scrolling eine Routine definiert werden, die bei der anwendungsinternen Navigation dafür sorgt, dass nach der Detailansicht eines Elements wieder die richtige Stelle der Liste präsentiert wird, damit der Nutzer die Liste nicht ein

weiteres Mal durchscrollen muss.

Somit kann man sagen, dass das Infinite Scrolling flüssiger abläuft, die Pagination allerdings Vorteile beim Wiederfinden von Elementen hat. (vgl. [7])

Quellen

- [1] John Maeda. The Laws of Simplicity. Design, Technology, Business, Life. The MIT Press, 2006.
- [2] W3C. Accessibility. URL:<https://www.w3.org/standards/webdesign/accessibility> (besucht am 27.06.2017).
- [3] Nadja Krakow. Flat Design vs. Skeuomorphismus – eine Frage des Geschmacks?.
1. Mai 2013.
URL:<https://www.mittwald.de/blog/webentwicklung-design/webdesign/flat-design-vs-skeuomorphismus-eine-frage-des-geschmacks> (besucht am 27.06.2017).
- [4] John Biggs. After Skeumorphism. 04. Aug. 2013.
URL:<https://techcrunch.com/2013/08/04/after-skeumorphism/>. (besucht am 27.06.2017).
- [5] Apache. Material design. URL:<https://material.io/guidelines/>. (besucht am 28.06.2017).
- [6] Jonas Hellwig. Adaptive Website vs. Responsive Website. 09. Aug. 2015.
URL:<https://blog.kulturbanause.de/2012/11/adaptive-website-vs-responsive-website/>. (besucht am 28.06.2017).
- [7] Christian Holst. Infinite Scrolling, Pagination Or “Load More” Buttons? Usability Findings In eCommerce. 01. März 2016.
URL:<https://www.smashingmagazine.com/2016/03/pagination-infinite-scrolling-load-more-buttons/>. (besucht am 28.06.2017).
- [8] Chris House. A Complete Guide to Grid. 03. Mai 2017. URL:<https://css-tricks.com/snippets/css/complete-guide-grid/#prop-grid-area>. (besucht am 28.06.2017).
- [9] Chris Coyier. A Complete Guide to Flexbox. 21. März 2017. URL:<https://css-tricks.com/snippets/css/a-guide-to-flexbox/>. (besucht am 28.06.2017).
- [10] Mario Janschitz. Sass vs. Less: So findest du den richtigen Präprozessor für dich. 09. Sep. 2015. URL:<http://t3n.de/news/sass-vs-less-636820/>. (besucht am 28.06.2017).
- [11] SASS. SASS Basics. URL:<http://sass-lang.com/guide>. (besucht am 28.06.2017).

Software-Architektur

Teilbereich(e): Serverless, Microservices

Projektleiter: Christian Holzberger

Projektteam: Christian Holzberger

Github-Repo: <https://github.com/cHolzberger/2017-Sw-Eng-Projekt.git>