

Software Architectures

WS 2018 / 2019

Bernhard Bauer



- Basiert auf
 - › Foliensätzen von FIM WS 2005/2009
 - › Foliensätze Seminar Software-Architekturen SS 2009
 - › ...

- Einführung in Software-Architekturen und Organisation
 - › Grundlagen
 - » Warum?
 - » Was?
 - » Bedeutung von Software
 - › Software-Architekturen in der Organisationsstruktur
 - » Wechselwirkung
 - » Rolle des Software Architekten
 - » Zusammenspiel Softwarearchitektur und PM
 - › Fallbeispiel

- Entwurf von SW-Architekturen
 - › Vorgehensmodelle
 - › Entwurfsumfeld
 - › Einflussfaktoren
 - › Qualitätsaspekte
 - › Software-Kategorien
 - › Entwurfsprinzipien
 - › Softwarearchitekturen und ihr Design
 - › Fallbeispiele

- Dokumentation von Software-Architekturen
 - › Bedeutung
 - › Anforderungen
 - › Bestandteile
 - › Architektursichten
 - › Software Architecture Viewtypes and Styles
 - › UML 2 & andere Möglichkeiten
 - » Architektursichten
 - » Klassen, Schnittstellen, Komponenten
 - » Fehler und Ausnahmen
 - » Spezifikation von Schnittstellen

- Evaluation / Bewertung von SW-Architekturen
 - › Grundlagen der Architekturbewertung
 - › Bewertungsmethoden
 - › ATAM (Architecture Tradeoff Analysis Method)
 - › SAAM (Software Architecture Analysis Method)
 - › ARID (Active Reviews for Intermediate Designs)
 - › CBAM (Cost Benefit Analysis Method)
 - › Vergleich unterschiedlicher Methoden

- **Toolbox des Softwarearchitekten**
 - › Einführung
 - › Lösungsvorlagen und Methoden
 - › Technologien und Werkzeuge
 - › Bibliotheken, Komponenten
 - › Modellierung und Generierung

- Produktlinien und mehr
 - › Produktlinien
 - » Was sind Produktlinien
 - » Aktivitäten und Vorgehen
 - » Architektur und SE
 - › Entwicklung von System aus Off-The-Shelf Components
 - › J2EE und EJB

- Enterprise Architecture Management
 - › Motivation: Was ist EAM und wieso wird es immer wichtiger?
 - › Elemente einer EA (EA-Prozess, Ebenenmodell, Governance)
 - › Frameworks (z.B. TOGAF)
 - › Modellierung (z.B. mit Archimate)
 - › Tooling

- Einführung in Software-Architekturen und Organisation
- Entwurf von SW-Architekturen
- Dokumentation von Software-Architekturen
- Evaluation / Bewertung von SW-Architekturen
- Toolbox des Softwarearchitekten
- Produktlinien für Software
- Enterprise Architecture Management

- **Einführung in Software-Architekturen und Organisation**
- Entwurf von SW-Architekturen
- Dokumentation von Software-Architekturen
- Evaluation / Bewertung von SW-Architekturen
- Toolbox des Softwarearchitekten
- Produktlinien für Software
- Enterprise Architecture Management

- Grundlagen der Software-Architektur
 - › Warum?
 - › Was?
 - › Bedeutung von SW-Architekturen
- Software-Architekturen in der Organisationsstruktur
 - › Wechselwirkung zwischen Architektur und Unternehmen
 - › Rolle des SW-Architekten
 - › Zusammenspiel SW-Architektur und Projektmanagement
- Fallbeispiel
 - › Software-Architektur des Airbus A380

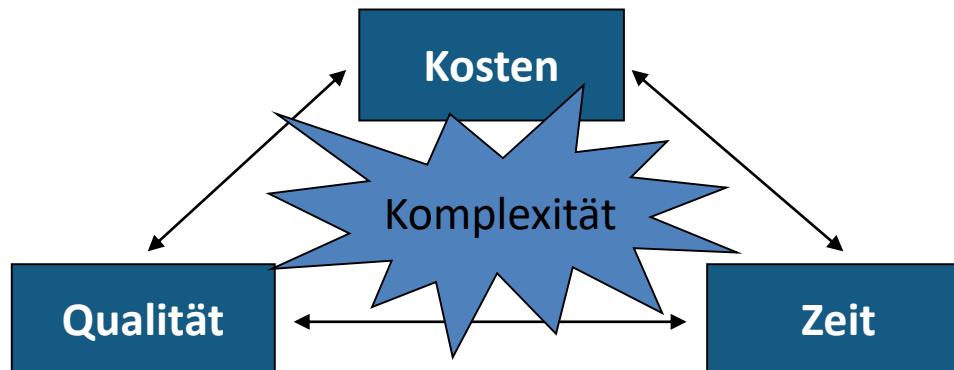
- **Grundlagen der Software-Architektur**
 - › Warum?
 - › Was?
 - › Bedeutung von SW-Architekturen
- Software-Architekturen in der Organisationsstruktur
 - › Wechselwirkung zwischen Architektur und Unternehmen
 - › Rolle des SW-Architekten
 - › Zusammenspiel SW-Architektur und Projektmanagement
- Fallbeispiel
 - › Software-Architektur des Airbus A380

- Warum benötigen wir Software-Architekturen?
 - › Bedeutung von Software
 - › Anforderungen an Software
 - › Rolle der Software-Architektur

- In unserer heutigen Gesellschaft spielt Software in praktisch allen Bereichen des Lebens eine entscheidende Rolle:
 - › Business-Domain (Office, ERP, B2B usw.)
 - › Heimanwendungen (Home-Office, Multimedia, Appliances usw.)
 - › Medienzugang und –nutzung (Internet, Handys, PDAs usw.)
 - › Transport (Autos, Flugzeuge usw.)
 - › Gesundheit (Medizintechnik, e-Health usw.)
 - › Versorgung (Anlagen, Kraftwerke usw.)
 - › Produktion (Mass-Customized Produkte usw.)
 - › ...

- Einsatz von Software ermöglicht höhere Leistungsfähigkeit von Produkten und Prozessen bei geringeren Kosten!
- Immens hohe Abhängigkeit von Software und u. Ust. verheerende Folgen bei Eintritt von Fehlern!
 - ▶ Diese kritische Abhängigkeit erfordert hohes Maß an Verantwortung und Qualitätssicherung bei der Software-Entwicklung.

- Anforderungen und Ziele der Software-Entwicklung
(„Spannungsdreieck“)



- Trotz der vielen Software-Projekte und der gesammelten Erfahrungen, verfehlten viele von ihnen diese wesentlichen Ziele.

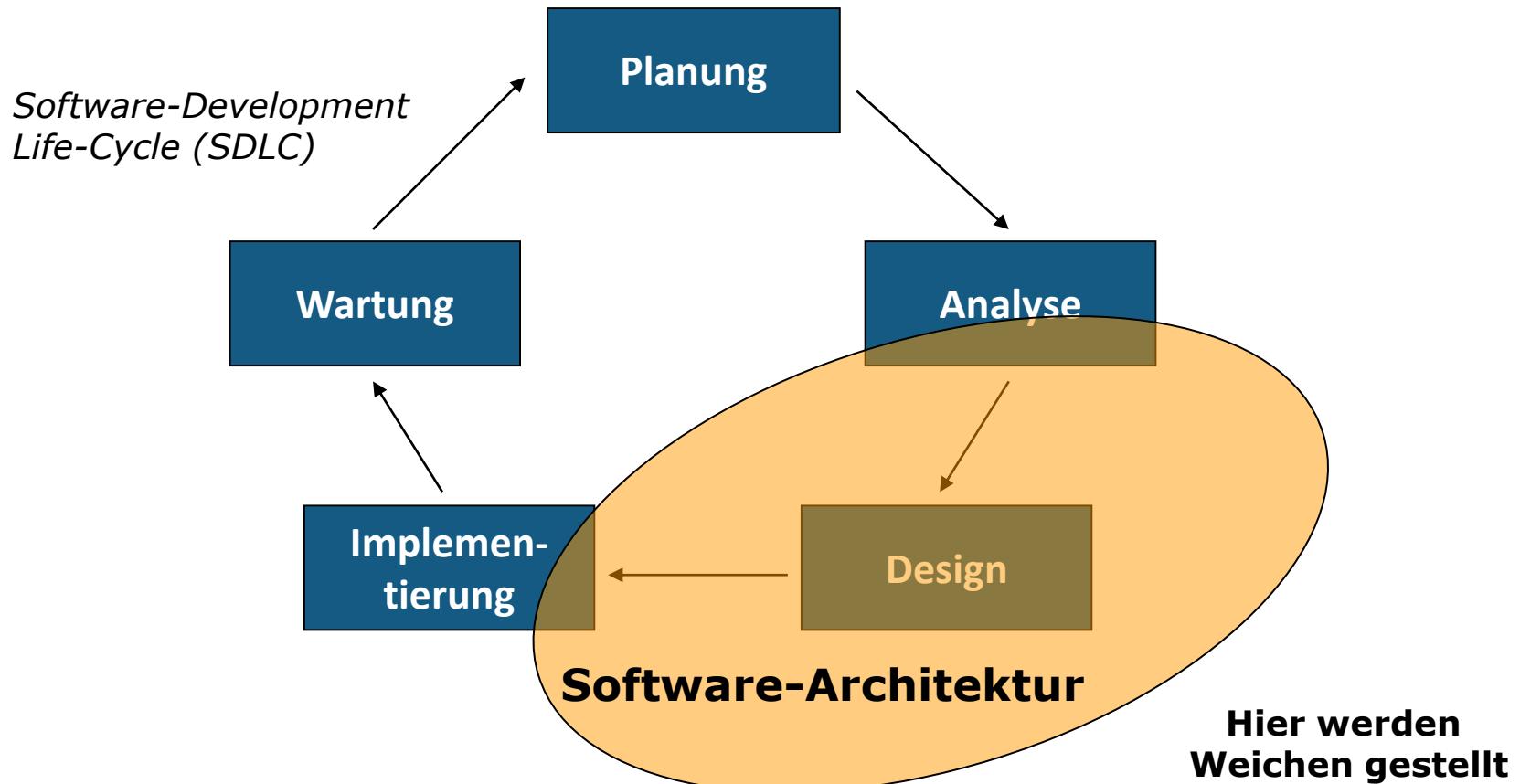
- **SW-Architektur hilft wesentlich bei der Erreichung der Ziele**
 - › Software-Architektur hilft bei der **Beherrschung von Komplexität** auf Grund von Größe und Technologie und spielt Schüsselrolle für **Wiederverwendbarkeit** von Software und **Prozessqualität** bei Software-Projekten.
- **Wie?**
- Software-Architektur **beschreibt**
 - › **Strukturen** eines Systems
 - › auf **hohem Abstraktionsniveau**
 - › in Form von **Bausteinen** und
 - › deren **Beziehungen**.

- SW-A ist eine junge Disziplin, jedoch erste Ansätze bereits in den 60er Jahren (Parnas, Brooks, Dijkstra u.a. beschäftigten sich zwischen 1960-1980 mit der Partitionierung und Strukturierung von SW.)
- Obwohl keine allgemein akzeptierte Definition existiert hat sich in den letzten Jahren ein Konsens gebildet
- Software-Engineering Institute der Carnegie Mellon University als prominenter Forschungsvertreter auf diesem Gebiet

- **Die Definition von SW-A hat 4 Aspekte**
- **Zur Definition von SW-A müssen folgende vier Fragen gestellt werden:**
 - › In **welcher Phase des Entwicklungsprozesses** von SW spielt SW-A eine Rolle?
 - › Welche **Teile der zu entwickelnden SW** werden von der SW-A beschrieben?
 - › Wie **detailliert** beschreibt SW-A die zugrunde liegende SW?
 - › Was ist eine **gute SW-A**?

Zeitliche Einordnung

SW-A bildet die Brücke zwischen Anforderungsanalyse und Implementierung



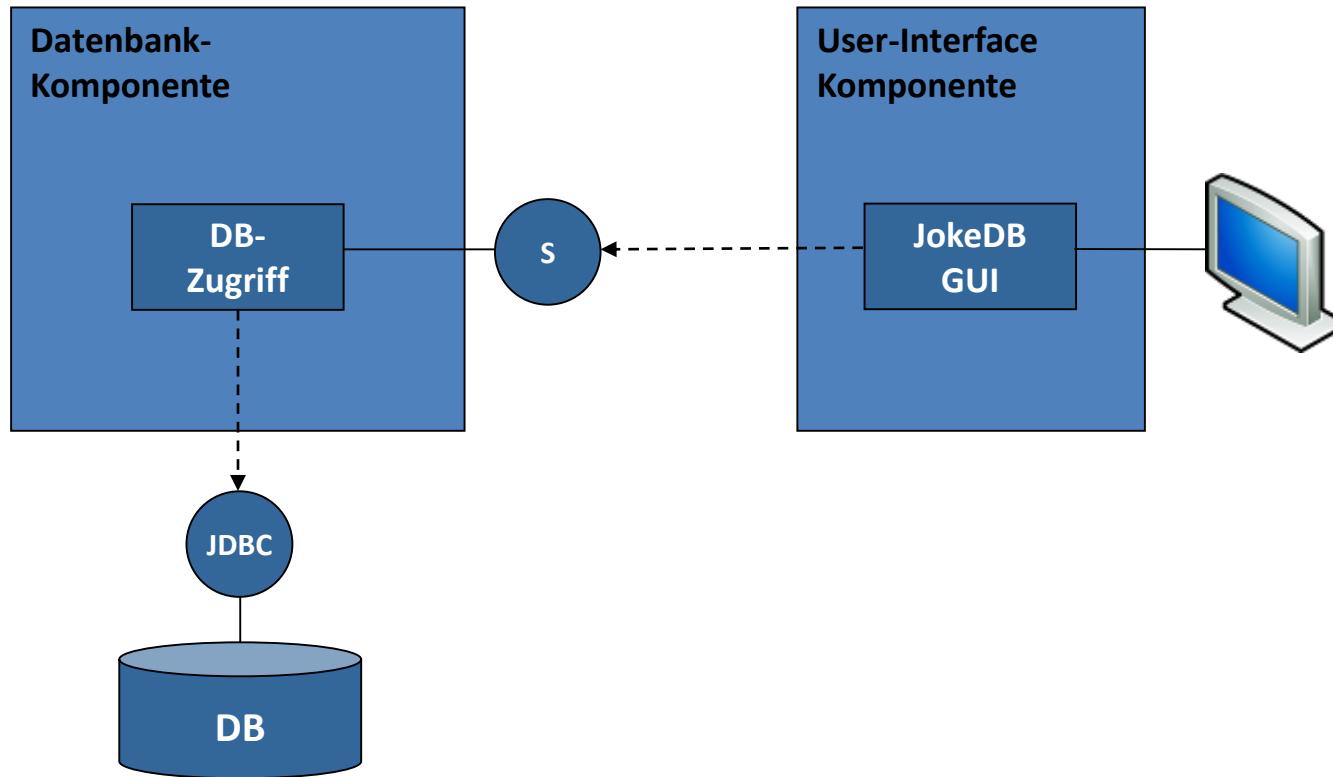
Inhaltliche Abgrenzung

Definition von SW-A nach Len Bass (2003)

- The Software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those components (interfaces) and the relationships among them.

Inhaltliche Abgrenzung

Beispiel: „The Joke Database“



Detaillierungsgrad der Beschreibung

- Architektur beschreibt SW auf den oberen Abstraktionsebenen
- Es existieren u. Ust. mehrere Architekturebenen
 - › Abstraktion von Details der Implementierung
 - › Innenleben der Elemente (Algorithmen) wird nicht betrachtet, außer bei der Spezifikation von externen Schnittstellen
- Detaillierungsgrad richtet sich nach
 - › Qualitätsanforderungen
 - › Risiko
 - › Projektmanagement

Was ist eine gute SW-Architektur?

- Es gibt keine generelle Aussage darüber, ob eine Architektur gut oder schlecht ist
- SW-Architekturen können nur im Zusammenhang mit den spezifischen Anforderungen in den Bereichen
 - › funktionaler Anforderungen (Verhalten)
 - › nicht-funktionaler Anforderungen (Qualität, Lebenszyklus)der SW bewertet werden.
- Architektur ist gut, wenn sie die diesen Anforderungen gerecht wird, z.B. Vergleich der Anforderungen an SW von Kohle- und Kernkraftwerk
- Jede gute Architektur ist eine LösungsMÖGLICHKEIT! Es kann durchaus mehrere denkbare gute Architekturen für eine bestimmte Software geben!!!

Was ist eine gute SW-Architektur? - Daumenregeln

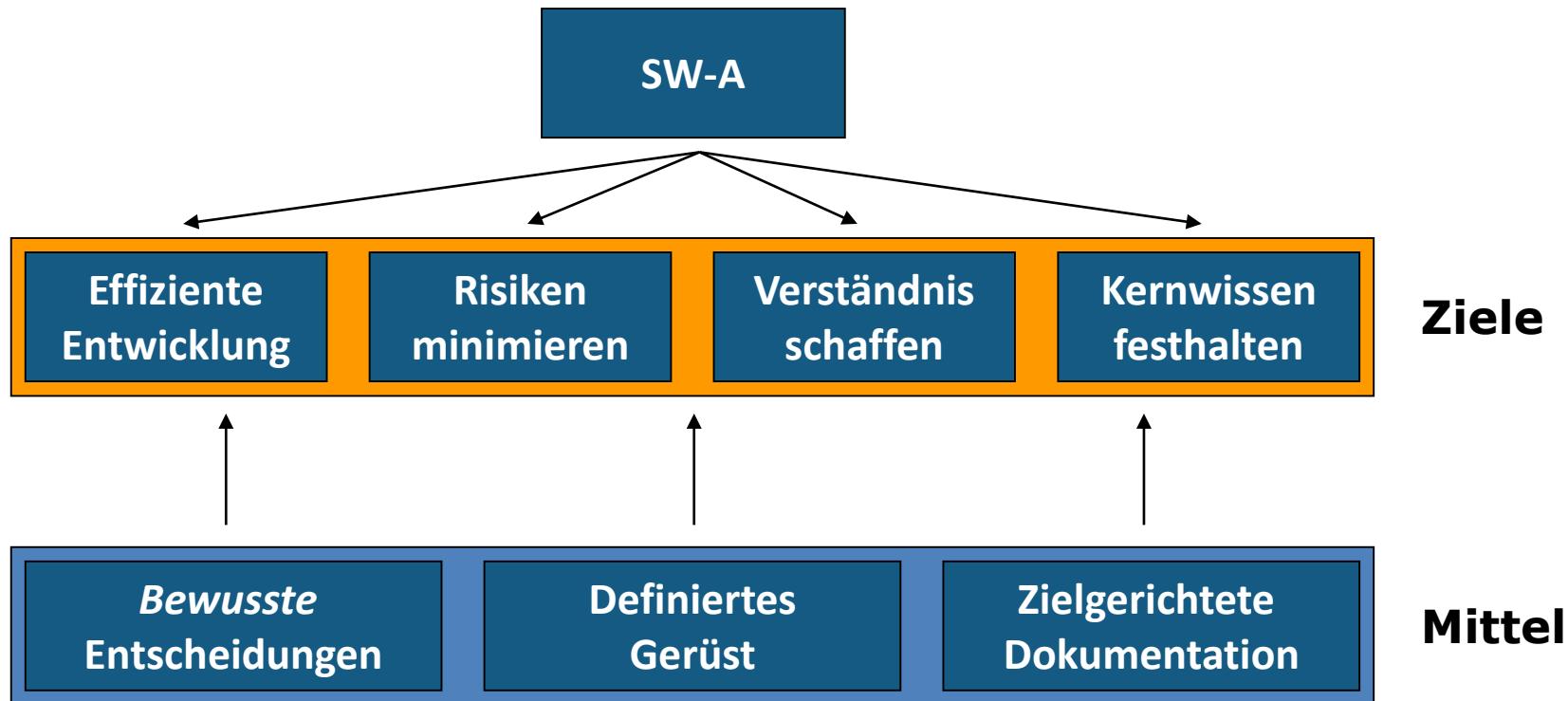
Prozess-orientiert

- nur **einer oder wenige Architekten** im Kernteam (mit **einem Chef!**)
- Architekt sollte über **strukturierte Liste** der funktionalen und nicht-funktionalen Anforderungen verfügen bevor die SW erstellt wird
- Architektur sollte **gut dokumentiert** werden (statisch und dynamisch)
- Architektur sollte von **Stakeholdern „reviewt“** werden
- **Leistungsfähigkeit** der Architektur sollte in frühem Stadium **getestet** werden
- Architektur sollte die **inkrementelle Erstellung** der SW unterstützen
- Architektur sollte **Regeln** für geplante **Ressourcen-Beanspruchung** enthalten

Struktur-orientiert

- Architektur sollte **sauber definierte Module** enthalten, deren Funktionalitäten dem **Prinzip des „Information-Hiding“** entsprechen
- Jedes Modul sollte **sauber definierte Schnittstelle** besitzen, die die **Implementierung** des Moduls nach außen **versteckt**
- Architektur sollte **niemals** auf einem bestimmten **kommerziellen Produkt oder Tool basieren**
- Architektur sollte auf einem Set von **„Interaction Patterns“ basieren**, die gewährleisten, dass die **selben Dinge** auch immer auf **gleiche Art und Weise gemacht** werden
- **Qualität** sollte durch Anwendung bestimmter „Taktiken“ **sichergestellt** werden

- Ziele und Aufgaben von SW-Architekturen



■ Ziele und Aufgaben von SW-A

- Frühe **Berücksichtigung der Einflussfaktoren**
- **Bewertung** der SW-A

- SW-A soll **wieder verwendbar** („architecture reuse“) sein und ist wertvolles geistiges Eigentum des Unternehmens
- Ohne SW-A **kein Lernen aus Erfahrungen** möglich

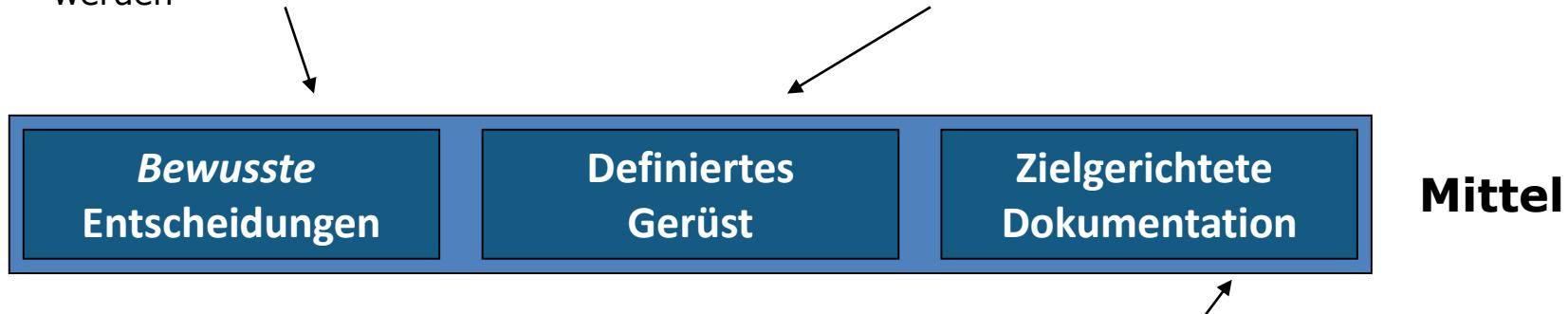


- SW-A ist **notwendiger Integrationsrahmen**, innerhalb dessen die Entwicklung inkrementell stattfindet („das große Ganze“)
- SW-A ist **Grundlage für Projektmanagement**
- SW-A ist **Fundament für verteiltes Arbeiten** in Projektteams

- SW-A als **Kommunikationsmedium** zwischen den **Stakeholdern** und **allen Mitarbeitern** des Entwicklungsprojekts
- SW-A als **Leitbild und Referenz** für alle Beteiligten (Code z.B. dafür ungeeignet, da ihn viele der Stakeholder nicht verstehen)

■ Ziele und Aufgaben von SW-A

- SW-A ist ein **konkretes physisches Ergebnis (Artefakt)**, z.B. in Form von UML Diagrammen, Anforderungsspezifikationen und anderen Dokumenten
- Um dieses Ergebnis zu erreichen müssen Entscheidungen **bewusst** getroffen, kommuniziert und diskutiert werden
- Beim Design der SW-A werden die **frühen Design-entscheidungen getroffen**
- Diese legen das **Gerüst der zu entwickelnden Software** fest und haben **weit reichende Auswirkungen** (auf das System selbst, aber auch z.B. auf die Projektorganisation)



- Dokumentation orientiert sich an den **verschiedenen Stakeholdern**, mittels Darstellung in **unterschiedlichen Sichten** (views), z.B. statische, dynamische oder Verteilungssicht

- **Einflussfaktoren von SW-Architekturen**

- › Was/Wer **lenkt die Entscheidungen** beim Entwurf der Architektur?
- › Was sind **treibende Faktoren**?

*Einflussfaktoren sind **Anforderungen**, **Umstände** oder **Tatsachen**,
die den **Entwurf der Architektur maßgeblich beeinflussen**.*

- **Bestimmung der Einflussfaktoren** gehört zu den **ersten Schritten** beim Design einer Software-Architektur
- **Diese sind Grundlage für Design-Entscheidungen**

■ Arten von Einflussfaktoren

Produktfaktoren	Technologische Faktoren	Organisatorische Faktoren
<ul style="list-style-type: none">• Funktionale Anforderungen (jedoch nicht Haupttreiber!)• Nichtfunktionale Anforderungen (Qualitätsanforderungen)<ul style="list-style-type: none">▪ Leistung▪ Sicherheit▪ Zuverlässigkeit▪ Testbarkeit▪ Änderbarkeit▪ Portierbarkeit▪ Wartbarkeit	<ul style="list-style-type: none">• Hardwaretechnologien• Softwaretechnologien• Architekturtechnologien• Standards	<ul style="list-style-type: none">• Management• Mitarbeiter• Weitere Stakeholder• Prozess- und Entwicklungsumgebung• Zeitplan• Budget• Unbekannte Größen

→ „**Jedes Design erfordert Kompromisse.**“

Wieso ist SW-Architektur so wichtig?

- SW-A sind das zentrale Artefakt bei der SW-Entwicklung
 - › SW-A stellt grundlegende Weichen und bestimmt somit, ob die SW später die Anforderungen erfüllen wird
 - › Investitionen in SW-A können zu hohem ROI führen, wenn die Architekturen später wieder verwendet werden; außerdem wird wichtiges und wertvolles Know-How konserviert, das als Wettbewerbsvorteil dienen kann
 - › Effektives Projektmanagement erst möglich durch SW-A
- SW-A ist das Rückgrat eines jeden SW-Projekts

- Was passiert, wenn eine SW-Architektur fehlt?
 - › Wichtige Anforderungen werden zu spät erkannt
 - › Projektleitern fehlt notwendiger Einblick und sind damit machtlos
 - › Keine Weiterverwendung von Architekturwissen

- Grundlagen der Software-Architektur
 - › Warum?
 - › Was?
 - › Bedeutung von SW-Architekturen
- Software-Architekturen in der Organisationsstruktur
 - › Wechselwirkung zwischen Architektur und Unternehmen
 - › Rolle des SW-Architekten
 - › Zusammenspiel SW-Architektur und Projektmanagement
- Fallbeispiel
 - › Software-Architektur des Airbus A380

Architecture Business Cycle (ABC)

- Die **SW-Architektur** und das **Unternehmen** beeinflussen sich gegenseitig im so genannten „Architecture Business Cycle“



Allgemeine Eigenschaften und Aufgaben des Architekten

- **Person**, die im **Mittelpunkt der Architektur** steht
- Sie **erschafft die Architektur** und ist die treibende Kraft für deren **Umsetzung und Einhaltung**
- **Vermittelt** zwischen Projektleiter, Entwicklern, Management und anderen Stakeholdern
- Ohne einen **wachsamen Architekten** besteht die Gefahr eines „**Architecture Drifts**“
- Sollte über eine **formale, im Unternehmen festgeschriebene Position verfügen**, um auch in schwierigen Situationen die **Architektur „verteidigen“** zu können

- Welche Fähigkeiten benötigt der Architekt hierzu?

Technische Fähigkeiten

- Technologien
- Methoden
- Werkzeuge



Soft Skills

- Kommunikation
- Führung
- Sicherer Verhandeln
- Organisation

- Die Architektenrolle steht in Wechselwirkung mit Projektbeteiligten und externen Rollen

Interne Beziehungen

- Entwickler
- Projektleiter
- Unterauftragnehmer
- Produktmanager
- Systemarchitekt
- Buildmeister

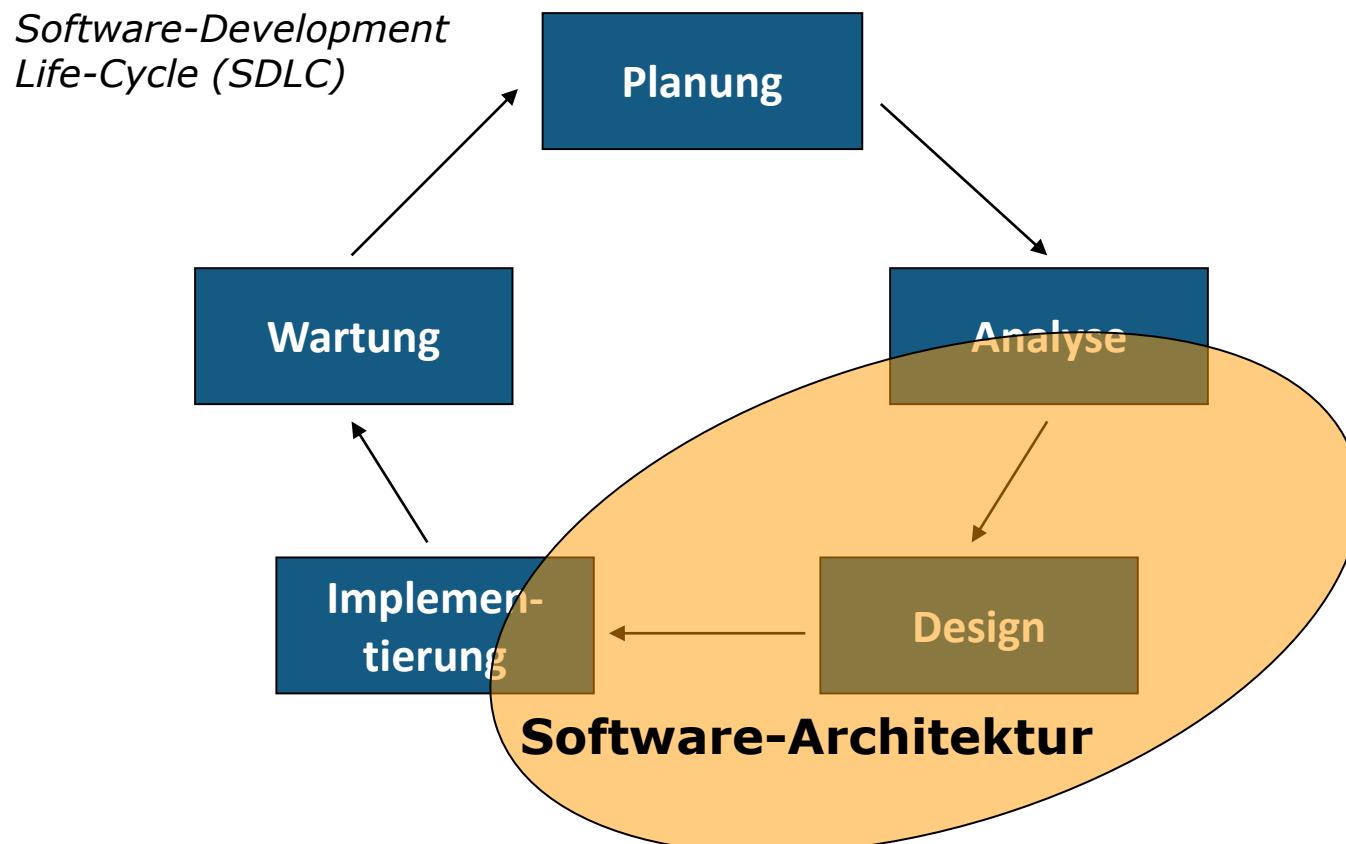


Externe Beziehungen

- Management
- Prozessverantwortliche
- Kollegen
- Forschung



- Relevante Phasen für den Software-Architekten



Phasenübergreifend

- Technische **Entscheidungen (frühzeitig) treffen**
- Als **technischer Berater fungieren**
- Mit **Projektmanager zusammenarbeiten**
- **Entwicklungsteams koordinieren**



- Anforderungen **überprüfen**
- Anforderungen **präzisieren**
- Anforderungen **verhandeln**
- **Risiken analysieren**

- **Entwurf erstellen**
- **Einflussfaktoren überwachen**
- **Abwägungen treffen**
- Entwurf **bewerten**
- **Codevorlagen und Architekturbausteine implementieren**
(Patterns etc.)
- Architektur **dokumentieren**

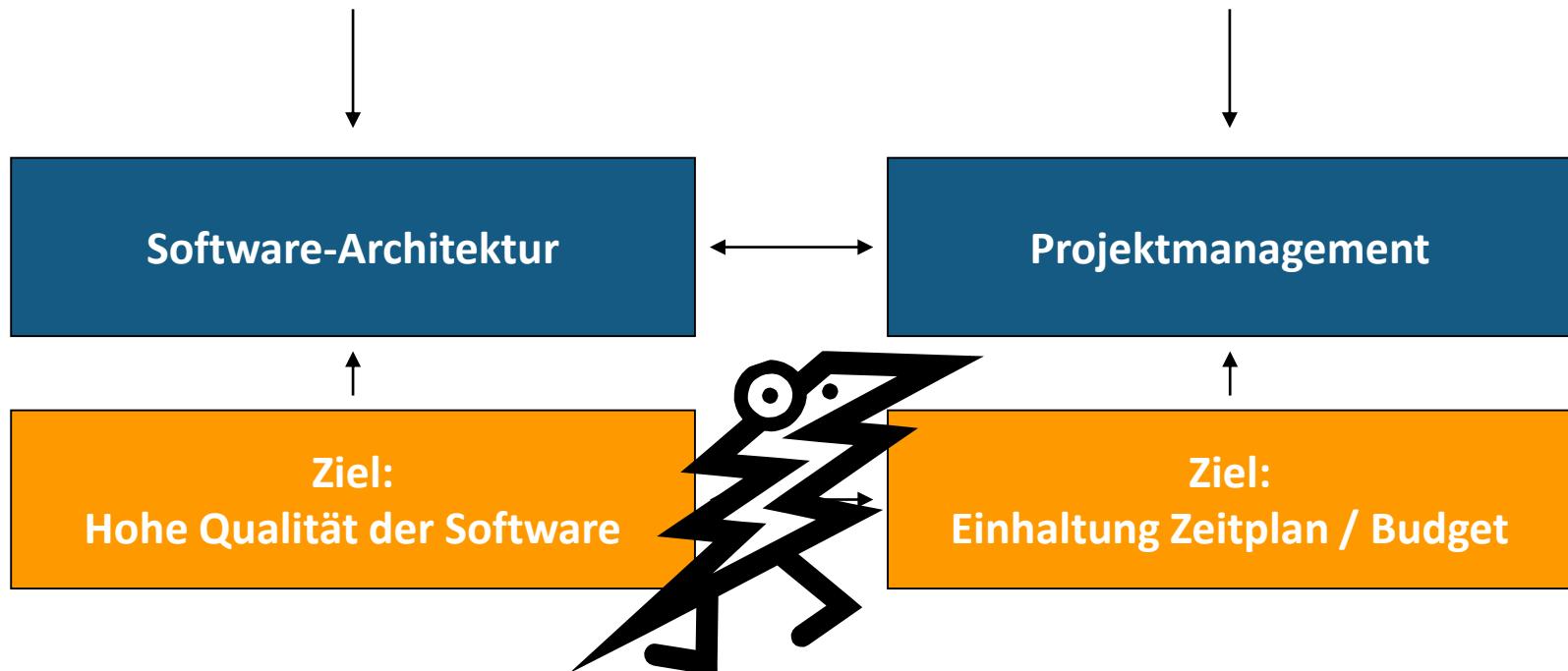
- Architektur **kommunizieren**
- **Verständnis schaffen**
- **Beraten**
- Entwickler **trainieren**
- Einhaltung der Architektur **kontrollieren**
- Architektur **pflegen**

- Architekturteam für komplexe und große Projekte
 - Werden Projekte **zu groß** und/oder erfordern Teilgebiete **sehr großes Fachwissen**, ist ein **Architekturteam notwendig**

- 1 Chefarchitekt
- 5 - 6 weitere Architekten, jedoch nicht mehr!
- Zusammensetzung und Verteilung der Aufgaben gemäß der Kompetenz und Erfahrung der einzelnen Architekten

- Einflussfaktoren
- organisatorisches Umfeld
- technologische Rahmenbedingungen
- ...

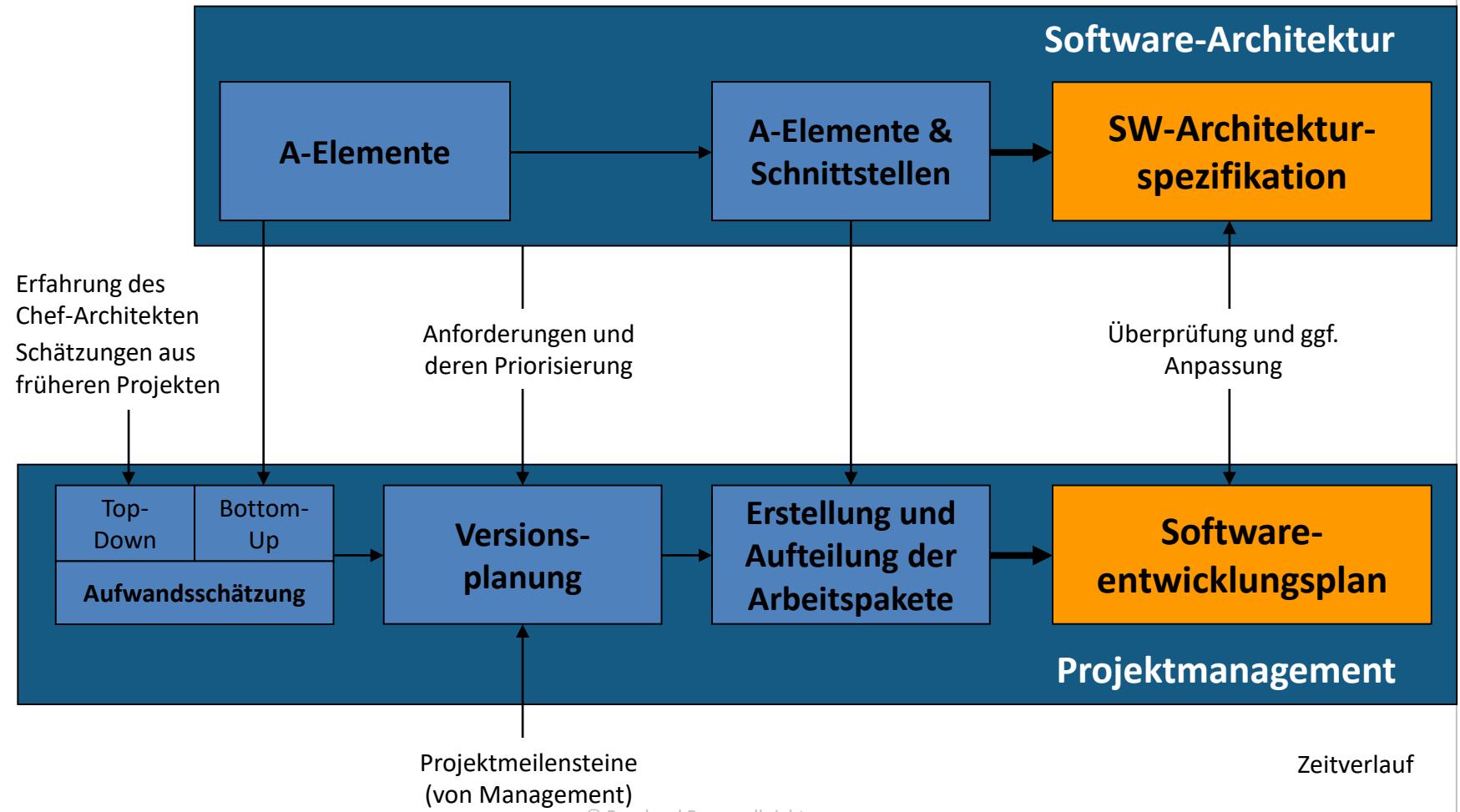
- Lieferzeitpunkt
- Qualität der Software
- Vorgaben des Managements
- ...



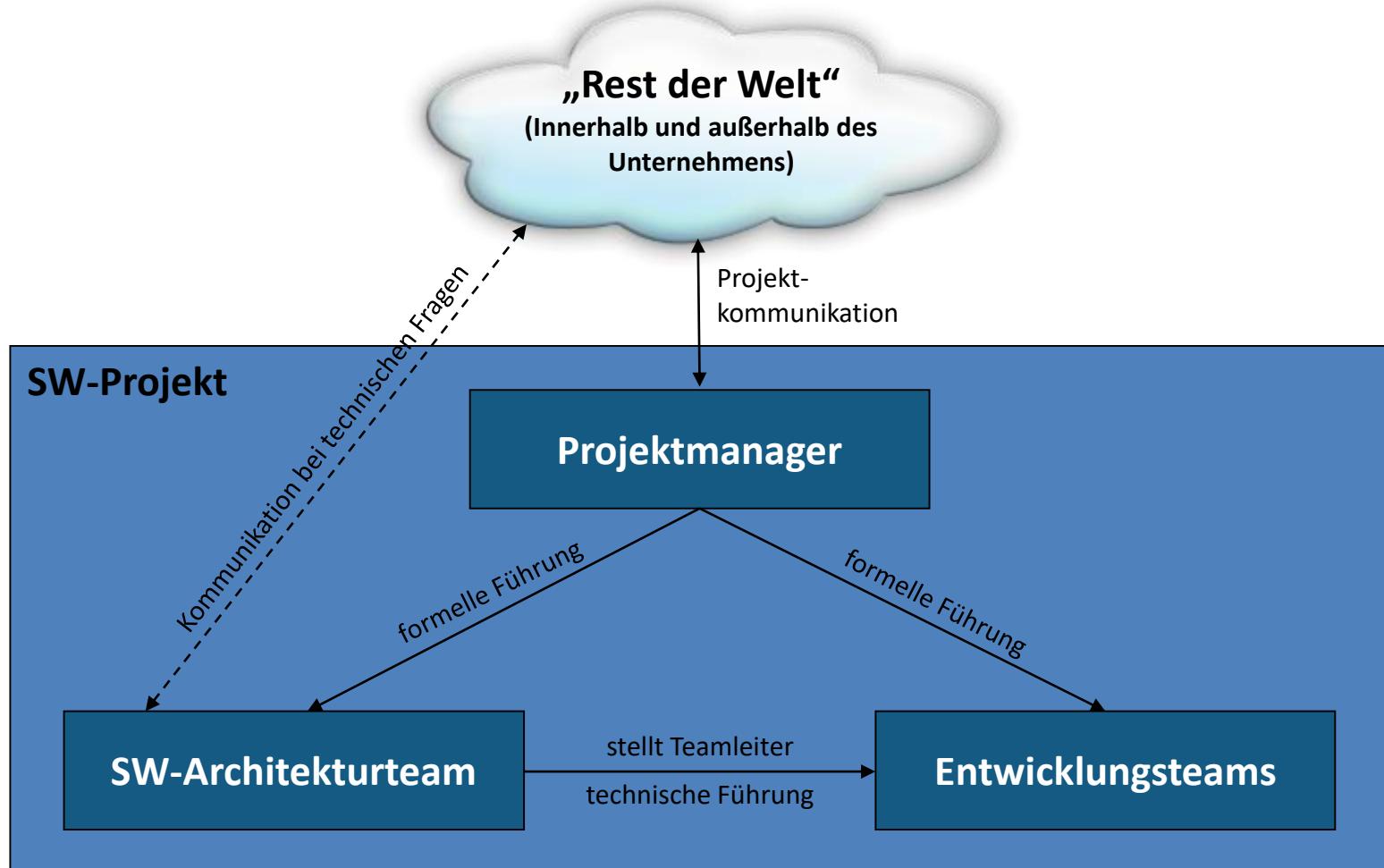
- **Hauptziel der SW-Architektur** ist die **hohe Qualität** der Software
- **Hauptziel des Projektmanagements** sind **geringe Kosten** und **kurze Projektdauer**
- **SW-Architektur** und **Projektmanagement** stehen deshalb grundsätzlich in einem **Interessenskonflikt**

- Beide Interessen sind **wichtig** und müssen ständig **ausbalanciert** werden
- Hierfür ist **entscheidend**, dass SW-A und PM von **unterschiedlichen Personen** durchgeführt werden, da nur so sichergestellt werden kann, dass **keiner der Aspekte vernachlässigt** wird

- Zusammenhang bei der Projektplanung

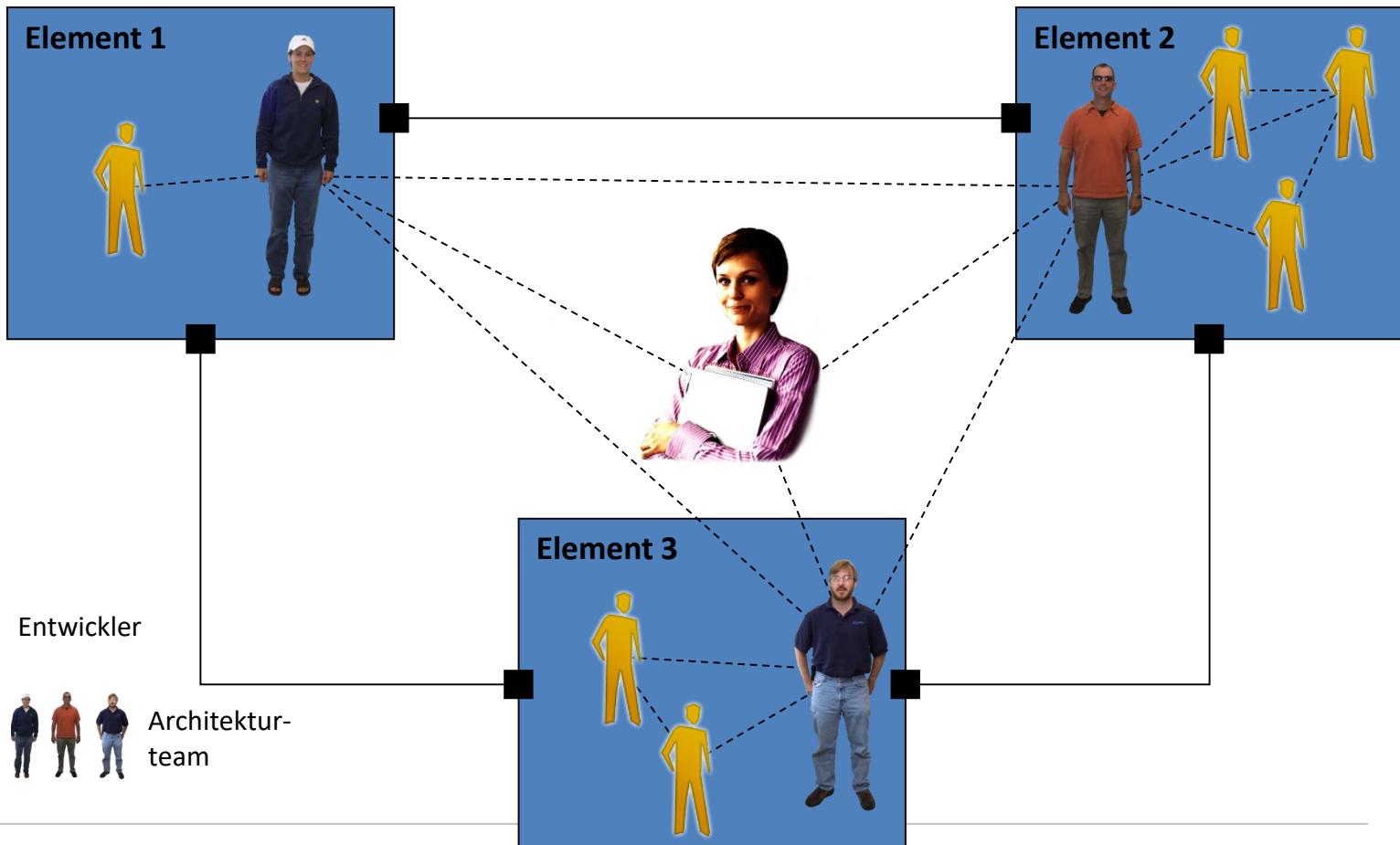


- Zusammenhang bei der Projektorganisation



- **Zusammenhang bei der Projektorganisation**

→ Conway's Law besagt, dass **Projektorganisation** mit der **statischen Struktur der SW-Architektur konsistent** sein sollte



■ Zusammenhang bei der Implementierung

- › Auch in der Implementierungsphase **versorgt der Architekt** das Projektmanagement **mit aktuellen Informationen**
- › Bei notwendigen Änderungen gibt die Architektur dem Projektmanager den **Handlungsspielraum für seine Entscheidungen vor**
- › SW-A ist auch **Grundlage** für die Durchführung von **Messungen** bzgl. Größe und Komplexität, die dem Projektmanagement **Einblick in den aktuellen Projekt-Status ermöglichen**

- Grundlagen der Software-Architektur
 - › Warum?
 - › Was?
 - › Bedeutung von SW-Architekturen
- Software-Architekturen in der Organisationsstruktur
 - › Wechselwirkung zwischen Architektur und Unternehmen
 - › Rolle des SW-Architekten
 - › Zusammenspiel SW-Architektur und Projektmanagement
- Fallbeispiel
 - › **Software-Architektur des Airbus A380**



■ Facts and Figures

- › Der A380 ist das **größte zivile Flugzeug** der Welt
(80 Meter Spannweite, 24 Meter hoch, 73 Meter lang)
- › Bis zu **853 Passagiere** können auf zwei Etagen befördert werden
(Boeing 747: max. 550 Passagiere)
- › Das maximale **Startgewicht** beträgt **560 Tonnen**
(Boeing 747: max. 395 Tonnen)
- › **Preis** liegt zwischen **273 und 293 Mio. USD**
(Boeing 747: 198-227 Mio. USD)
- › Der **A380 wurde komplett neu entwickelt** und ist in **vielen Belangen revolutionärer als die Concorde**
- › Auch die **IT-Systeme sind revolutionär** hinsichtlich **Komplexität und Funktionsweise**

■ Aufgaben der IT-Systeme

- › **Avionik**
(Fly-by-Wire, Schubkontrolle, Flight-Envelope-Protection etc.)
- › **Fuel & Energy Management**
- › **Assistenzsysteme** für die Piloten
(Onboard Information System, Terrain Awareness Warning System, Electronic Flight Bag, Onboard Maintenance System etc.)
- › **Kabinenmanagement**
(Sauerstoffversorgung, Temperatur, Beleuchtung, Doors and Slides Management System, Wassermanagement etc.)
- › **Convenience**
(Inflight Entertainment System, WLAN Internet Zugang, GSM Mobilfunk Netz etc.)

Rechtfertigung von SW und SW-A beim A380

- **Wieso erfordern diese Aufgaben den Einsatz von Software?**

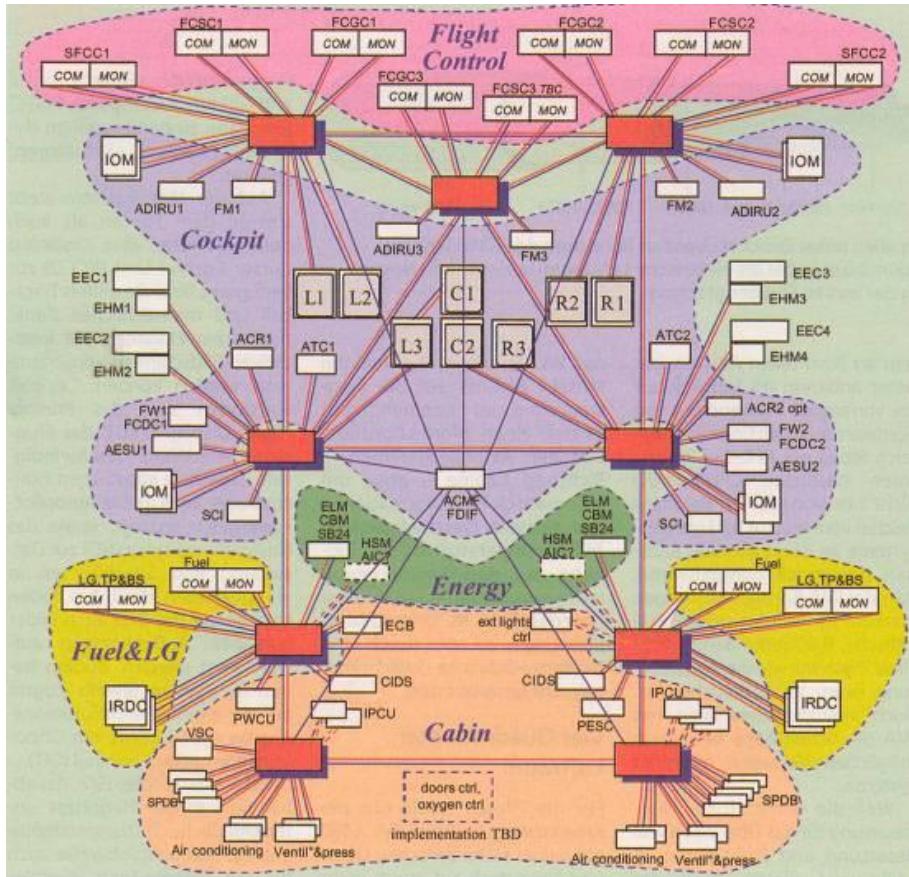
Verwendung von Software führt zu

- höherer Leistungsfähigkeit
- größerer Sicherheit
- geringeren Kosten

und ermöglicht so **Wettbewerbsfähigkeit**.

- **Wieso benötigt man beim A380 Software-Architektur?**

■ System-Architektur A380



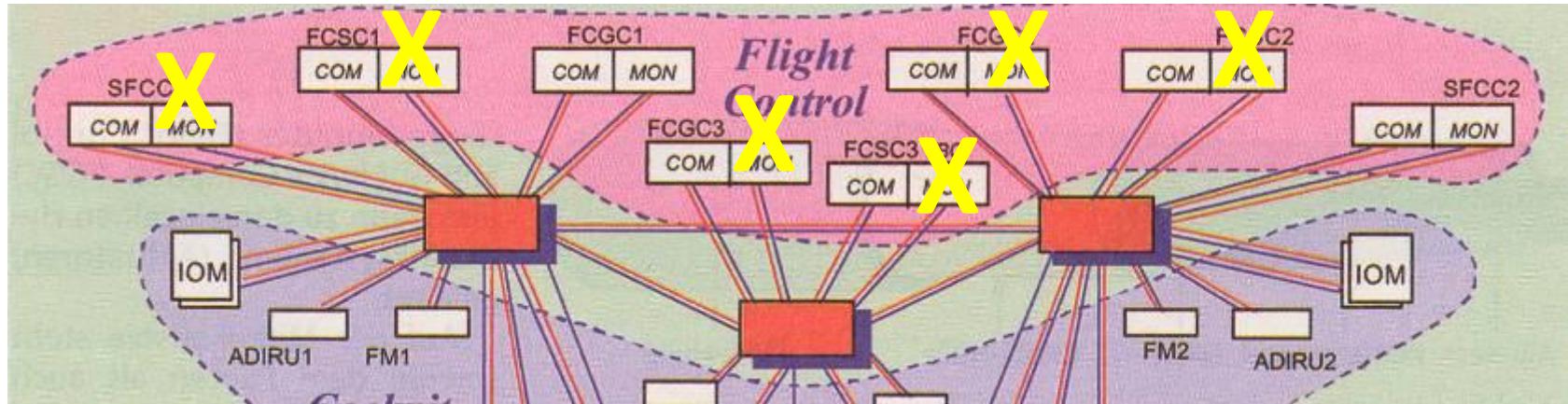
- **IT-System-Architektur des A380 ist ein hoch-komplexes Gebilde**
- **Ohne eine saubere Software-Architektur** wäre es praktisch **unmöglich** so ein System zu entwickeln
- **System-Architektur** gliedert sich in **fünf Domänen**
- **Jeder dieser Domänen hat eine** (den unterschiedlichen Anforderungen entsprechende) **eigene Software-Architektur**
- **Vernetzung** erfolgt durch **Avionics Full Duplex Switched Ethernet (AFDX)**

■ Einflussfaktoren beim A380 am Beispiel Avionik/KM

	Avionik (Flight Control Domain)	Kabinenmanagement (Cabin Domain)
Produktfaktoren (FA)	<ul style="list-style-type: none">• Avionik	<ul style="list-style-type: none">• Kabinenmanagement
Produktfaktoren (NFA)	<ul style="list-style-type: none">• Leistung• Sicherheit• Zuverlässigkeit	<ul style="list-style-type: none">• Wartbarkeit• Bedienbarkeit• Änderbarkeit• Portierbarkeit
Technologische Faktoren	<ul style="list-style-type: none">• Einsatz hochverfügbarer proprietärer Soft- und Hardware verschiedener Hersteller• Trennung von redundanten Hardwaresystemen	<ul style="list-style-type: none">• Einsatz von Hard- und Softwarestandards• Verteiltes System
Organisatorische F.	<ul style="list-style-type: none">• Keine Budgetrestriktion	<ul style="list-style-type: none">• Budgetrestriktionen

System-Architektur der Avionik

- Die System-Architektur der Avionik ist mehrfach redundant und diversitär ausgelegt
- 8 Systeme, die völlig unabhängig voneinander arbeiten



→ Auch nach dem Ausfall von 6 Systemen bleibt die Maschine voll flugfähig

System-Architektur des Kabinenmanagements

- Die System-Architektur des Kabinenmanagements ist modular und flexibel
- Verwendung von Hard- und Softwarestandards mit einheitlichen Schnittstellen
 - › Integrated Modular Avionics (IMA)
 - › Avionics Full Duplex Switched Ethernet
 - › PPC-7xx-basierte Consumer Hardware
- Kabinenmanagement ist eine verteiltes System, dadurch
 - › bessere Auslastung von Rechenkapazität
 - › Gewichtseinsparungen

■ Organisation und Projektmanagement

- › Die **meisten IT-Komponenten** des A380 kommen von **Fremdherstellern**
- › Dies liegt an der **hohen Komplexität** der **einzelnen Anwendungsdomänen**
- › **Ohne** eine vorher definierte **Software-Architektur** wäre so eine Projektdurchführung und Gestaltung der Projektorganisation **nicht möglich**



- Bass/Clements/Kazman (2005): Software Architecture in Practice; Second Edition; Addison-Wesley
- Hoffer/George/Valacich (2005): Modern Systems Analysis and Design; Fourth Edition; Pearson Education
- Posch/Birken/Gerdom (2004): Basiswissen Softwarearchitektur; dpunkt
- Siedersleben (2004): Moderne Software-Architektur; dpunkt
- Ziegler/Benz (2005): Fliegendes Rechnernetz in c't 17/2005; Heise

- Einführung in Software-Architekturen und Organisation
- **Entwurf von SW-Architekturen**
- Dokumentation von Software-Architekturen
- Evaluation / Bewertung von SW-Architekturen
- Toolbox des Softwarearchitekten
- Produktlinien für Software
- Enterprise Architecture Management

- Ziele/Motivation
- Grundlagen
 - › Kriterien für korrekten Entwurf
 - › Fundamentale Entwurfsprinzipien
 - › Sichten
- Konkretes Vorgehen
 - › Aktivitäten beim Architekturentwurf
 - › Konkrete Entwurfsschritte
 - › Heuristiken
- Fallbeispiel
- Zusammenfassung

- **Ziele/Motivation**
- **Grundlagen**
 - › Kriterien für korrekten Entwurf
 - › Fundamentale Entwurfsprinzipien
 - › Sichten
- **Konkretes Vorgehen**
 - › Aktivitäten beim Architekturentwurf
 - › Konkrete Entwurfsschritte
 - › Heuristiken
- **Fallbeispiel**
- **Zusammenfassung**

- Warum entwirft man eine Softwarearchitektur?
 - › Damit sie möglichst „gut“ ist
 - › Basis aller Projekttätigkeiten
 - › Bauplan für Erstellung, Wartung, Pflege und Weiterentwicklung
 - › Architekturbeschreibung als Kommunikations-/Diskussionsplattform und als Design-/Implementierungsplan
- Definition von „gut“
 - › Wohl definierte Abstraktionsschichten
 - › Jede Schicht ist in sich geschlossene Abstraktion mit definierten Schnittstellen
 - › Abstraktionsebenen bauen aufeinander auf
 - › Jede Schicht/jeder Baustein hat klare Trennung zwischen Implementierung und Schnittstelle
 - › So einfach und einheitlich wie möglich

- Ziele/Motivation
- **Grundlagen**
 - › Kriterien für korrekten Entwurf
 - › Fundamentale Entwurfsprinzipien
 - › Sichten
- Konkretes Vorgehen
 - › Aktivitäten beim Architekturentwurf
 - › Konkrete Entwurfsschritte
 - › Heuristiken
- Fallbeispiel
- Zusammenfassung

- Für einen korrekten Architekturentwurf müssen einige Kriterien eingehalten werden
 - › Kopplung
 - » Zu welchem Grad sind die Komponenten einer Software verbunden
 - › Kohäsion
 - » Beschreibt den Grad der Bindung der Elemente innerhalb eines Bausteins
 - › Zulänglichkeit
 - » Ist dann erfüllt, wenn die Funktionalität eines Bausteins den Anforderungen gerecht wird
 - › Vollständigkeit
 - » Ist gegeben, wenn die Schnittstelle eines Bausteines alle Anforderungen, die an den Baustein gestellt werden, erfüllt
 - › Einfachheit
 - » Formal: Anhaltspunkt für die Gratwanderung zwischen Zulänglichkeit und Vollständigkeit, d.h. ein Baustein ist einfach, wenn er ausreichend Funktionalität mit passenden Schnittstellen bietet

- Grundlegende Entwurfsprinzipien, die dem Architekten bei seiner Arbeit helfen
 - › Abstraktion
 - » Leichterer Umgang mit Komplexität, da nur noch die Informationen eines Objektes betrachtet werden, die für einen bestimmten Zweck relevant sind
 - › Kapselung
 - » Verbergen von Implementierungsdetails (Geheimnisprinzip)
 - » Trennung von Schnittstelle und Implementierung, der Zugriff auf einen Baustein erfolgt nur noch durch seine Schnittstellen
 - › Modularität
 - » Die Eigenschaft eines Systems, das in eine Menge von in sich geschlossenen und lose gekoppelten Modulen zerlegt wurde
 - › Hierarchie
 - » Abstraktionen können durch Hierarchien abgebildet werden
 - » Enthalten-in-/Besteht-aus-Hierarchien (Struktur)
 - » Ist-ein-Hierarchien (Generalisierung)
 - › Konzeptuelle Integrität
 - » Durchgängige Anwendung von Entwurfsentscheidungen im gesamten System
 - » Dadurch Vermeidung von Speziallösungen und Verwässerung des Konzepts

- Dienen der Beherrschbarkeit von Komplexität
- Je nach Blickwinkel erscheinen bestimmte Eigenschaften im Vordergrund, andere werden vernachlässigt
 - › Technische Sicht
 - » Beinhaltet technische Komponenten, deren Schnittstellen, Beziehungen zwischen den einzelnen Komponenten und die Abbildung der fachlichen Komponenten auf die technische Sicht
 - » Ziel: Softwaretechnische Lösung für das System inkl. der Umsetzung der fachlichen Sicht
 - › Entwicklungs-Sicht
 - » Darstellung der erstellten Beschreibung des Systems und der Vorgänge bei Entwicklung, Test und Integration des Systems
 - » Ziel: Unterstützung paralleler Arbeit einzelner Entwickler und Entwicklungsteams durch Entkopplung der Entwickler/Teams und Freigabe von Teilsystemen zur Integration in das bestehende System
 - › Deployment-Sicht
 - » Beschreibung der auszuliefernden Bestandteile, der Installation und der Konfiguration des Systems
 - » Ziel: Minimierung der Kosten für Installation, Konfiguration und Updates der Software

- Ziele/Motivation
- Grundlagen
 - › Kriterien für korrekten Entwurf
 - › Fundamentale Entwurfsprinzipien
 - › Sichten
- Konkretes Vorgehen
 - › Aktivitäten beim Architekturentwurf
 - › Konkrete Entwurfsschritte
 - › Heuristiken
- Fallbeispiel
- Zusammenfassung

■ Iteratives und inkrementelles Vorgehen

- › Der Entwurf einer Softwarearchitektur ist, ähnlich der Entwicklung von Software selbst, ein Prozess mit undefiniertem Anfang und Ende
- › Existierende Bausteine können so weiterentwickelt werden, dass sie einem neuen Projekt als Grundlage dienen
- › Iterativ: Die Ergebnisse der vorherigen Phase des Entwurfs werden für die jeweils nächste Iteration verwendet
- › Inkrementell: Schrittweise Verfeinerung der Architektur, indem sie von Iteration zu Iteration umfangreicher und näher an das gewünschte Ziel gebracht wird

■ Auswahl von Stilen, Vorlagen und Mustern

- › Der Architekt kann sich das Wissen von Vorgängern und Kollegen zu Nutze machen und auf eine Sammlung von Architekturstilen und -mustern zurückgreifen
- › Abgleich mit den Anforderungen, um Geeignete auszuwählen
- › Sog. Toolbox-Prinzip

■ Architekturtransformation

- › Eine Architektur muss nicht nur die funktionalen Anforderungen erfüllen, sondern auch Qualitätsattribute wie Skalierbarkeit, Performance, ...
- › Schwierig, das alles in der ersten Iteration unter einen Hut zu bekommen, deshalb:
 - » Versuchen in der ersten Iteration die funktionalen Anforderungen zu erfüllen (entspricht ungefähr dem fachlichen Modell aus den Vorleistungen)
 - » Im Anschluss die Architektur auf Basis von Architekturbewertungen so zu transformieren, dass sie auch die geforderten Qualitätsattribute aufweist
- › Zweigeteilter Entwurf in der Praxis schwierig

■ Erstellung von Modellen, Dokumentation

- › Dokumentation der Entscheidungen während des Entwurfs mittels Modellen
- › Modelle sind dabei vereinfachte Abbildungen der Realität, die es erlauben, Aussagen im Modell so dazustellen, dass sie von allen Beteiligten gleich interpretiert werden
- › Modelle zeigen Inhalt/Ergebnis der Entwurfsaktivitäten
- › Meist graphische Darstellung (z.B. UML-Diagramme)
- › Iterative und inkrementelle Verfeinerung der Modelle

- Bewertung des Entwurfs
 - › Durch Abgleich des Entwurfs mit den Einflussfaktoren und den elementaren Entwurfsprinzipien
 - › In unterschiedlichen Abständen und mit unterschiedlichem Aufwand
 - › Bewertung erfolgt
 - » In Reviews innerhalb des Teams
 - » Durch modellgetriebene Validierung und Verifikation
 - » Durch Prototyping
 - » Durch Tests
 - › Nach Abschluss des Architekturentwurfs: Abschließende Bewertung und Start der Implementierung

- Ziel: Zeitliche Abfolge von Schritten als grober Anhaltspunkt
- Zusätzlich: Dokumentation der Entwurfsaktivität
- Sammeln von Informationen
 - › Erstellen einer Faktorentabelle aus den Einflussfaktoren als Vorleistung
 - › Durchführung einer Risikoanalyse und Erarbeitung von Lösungsstrategien auf Basis der Einflussfaktoren
 - › Wichtige Fragestellungen
 - » Gibt es ähnliche, bereits existierende Systeme?
 - » Lassen sich Bausteine aus früheren Projekten wiederverwenden?
 - » Lassen sich Bausteine von Drittanbietern zukaufen?
 - › Systemidee nötig
 - » Kernaufgaben und Verantwortlichkeiten des Systems
 - » Nutzungsarten und Nutzung des Systems
 - » Schnittstellen zu Nutzern und externen Systemen, ...

- Erstellung des ersten Entwurfs
 - › Hat der Architekt nach Betrachtung vergleichbarer Systeme und dem Erstellen der Systemidee noch keinen Entwurf im Kopf, so kann er auf Architekturstile zurückgreifen
 - › Kernabstraktionen bilden und an Bausteine zuteilen
 - › Jeder Baustein sollte dabei klar definierte Verantwortlichkeiten haben
 - › Architekturmuster für alle auftretenden Systemaspekte finden
- Sicherstellen der konzeptuellen Integrität
 - › Sind Entwurfsentscheidungen nicht durchgehend einheitlich, ist das System für Entwickler und Benutzer evtl. nicht verständlich bzw. einfach bedienbar
 - › Für jeden vorhandenen Systemaspekt muss zur Umsetzung genau ein Konzept verwendet werden
 - › Dazu Auswahl eines passenden Architekturmusters für jeden Systemaspekt
 - › Auch bei Verbesserungen oder Ideen von anderen muss darauf geachtet werden, dass sie zum bisherigen Konzept passen

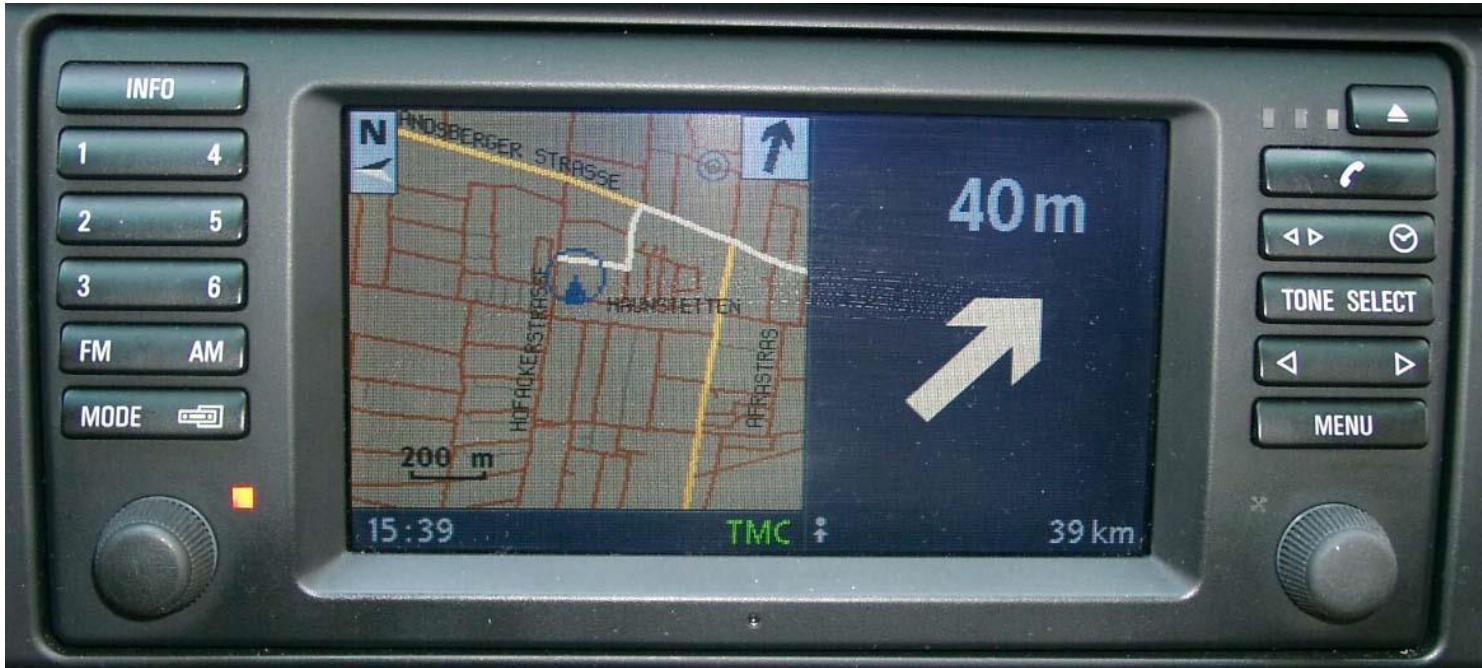
Abschließende Schritte

- › Iterativer und inkrementeller Verfeinerungsprozess der Architektur
- › Abwechselnde Durchführung von Bewertungen und Transformationen, um zu verhindern, an den Anforderungen vorbeizuentwickeln
- › Feedback von Kollegen zu den jeweiligen Zwischenergebnissen einholen

- Erfahrungsschatz von Heuristiken und Best Practices zur Erleichterung von Entscheidungen und Aktivitäten beim Architekturentwurf
- Ein Architekt kann auf diese Sammlung sowohl zugreifen als sie auch um eigene Erfahrungen aus bisherigen Projekten erweitern
- Keine fertigen Algorithmen, sondern Prinzipien, Regeln und Ratschläge, an denen sich der Architekt orientieren kann
 - › Heuristiken aus dem Projektumfeld
 - » Hinterfragen Sie Anforderungen!
 - » Betrachten Sie Use Cases!
 - › Heuristiken für den Entwurf
 - » So einfach wie möglich!
 - » Konzentrieren Sie sich auf Schnittstellen!
 - › Heuristiken zur Beherrschung von Komplexität
 - » Teile und herrsche!
 - » Verstecken Sie Details!
 - › Heuristiken zur Arbeitsmethodik
 - » Beginnen Sie mit den schwierigsten Teilen!
 - » Verwenden Sie Prototypen!

- Ziele/Motivation
- Grundlagen
 - › Kriterien für korrekten Entwurf
 - › Fundamentale Entwurfsprinzipien
 - › Sichten
- Konkretes Vorgehen
 - › Aktivitäten beim Architekturentwurf
 - › Konkrete Entwurfsschritte
 - › Heuristiken
- Fallbeispiel
- Zusammenfassung

- Erstellung der Software eines Navigationssystems



- Aufgabenstellung: Die Software muss mit Hilfe des vorhandenen Kartenmaterials die Route zwischen zwei Orten berechnen

■ Vorleistungen

- › Identifikation von Einflussfaktoren, Risiken und Lösungsstrategien
- › Erstellen von Faktorentabellen für die gefundenen Einflussfaktoren

Produktspezifische Faktoren	Flexibilität und Veränderbarkeit	Einfluss
P1: Funktionale Anforderungen		
...		
P2: Portierbarkeit		
P2.1: Externe Systeme		
Die SW soll zukünftig auf andere Plattformen portiert werden.	Es ist wahrscheinlich, dass in der nächsten Produktgeneration andere Plattformen zum Einsatz kommen.	Designentscheidungen
P3: Benutzerschnittstelle		
P3.1: Änderbarkeit		
Die Benutzerschnittstelle soll einfach auf eine reduzierte Darstellung, wie z.B. einfache Pfeildarstellungen im Radiogerät, geändert werden können.	Die Anpassung muss unterstützt werden können für die High-End- und eine zukünftige Low-End-Variante der Navigationssoftware.	Designentscheidungen
P4: ...		

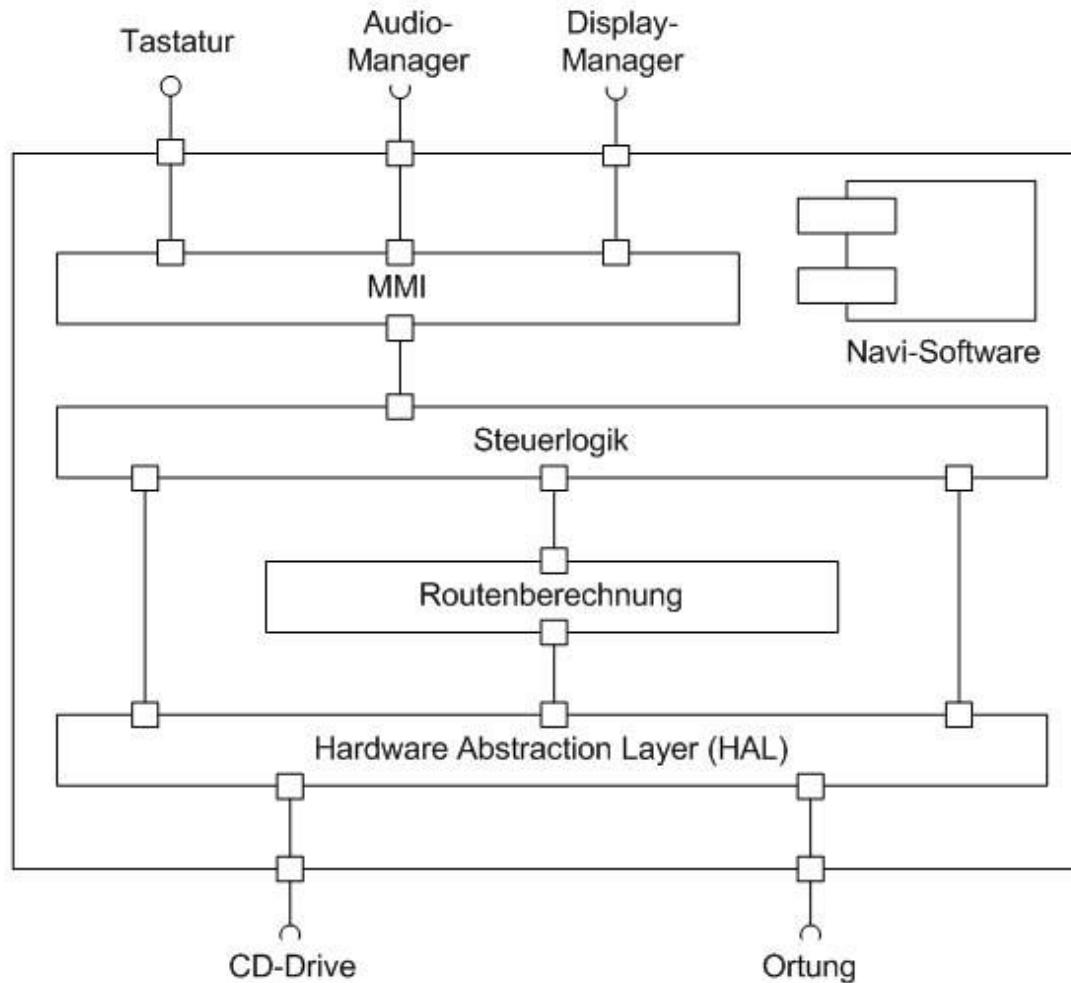
- › Präzisierung der Einflussfaktoren, so dass entschieden werden kann, ob das System die Anforderungen erfüllt

- Ableitung von Risiken aus den Einflussfaktoren (im Fallbeispiel die Einhaltung des Zeitplans und die Tragfähigkeit der Architektur für zukünftige Systeme, ...)
- Aufstellen von Lösungsstrategien für diese Risiken (→ Themenkarten)

Einhaltung des Zeitplans
Der Zeitplan ist sehr knapp bemessen und aufgrund des fixen Produktionsstarts nicht flexibel. Weitere Einflussfaktoren, wie das mangelnde Wissen zu spezifischen Themen, sowie Portierbarkeit verschärfen diese Problematik.
Beeinflussende Faktoren:
O2.1: fester Zeitplan
O4.1: Wissen über Routenberechnung
P2: Portierbarkeit
Lösung:
Über verschiedene Ansätze soll versucht werden, die benötigte Entwicklungszeit für die Software zu verkürzen.
Strategie: COTS
Es soll nach Komponenten gesucht werden, die fertig zugekauft werden können. Fertige Komponenten reduzieren die Komplexität in der eigenen Entwicklung und sparen Entwicklungszeit.
Strategie: Outsourcing
...
Strategie: Hoher Grad an paralleler Entwicklung
...
Verwandte Themen und Strategien:
O.3 Budget: Die Kosten im Projekt sind nicht kritisch.

- › Sammeln von Informationen über bereits existierende, ähnliche Systeme
 - › Der Architekt im Fallbeispiel sucht nach verfügbaren Zukaufkomponenten und findet einen passenden Navigationskernel
 - › Entwickeln der Systemidee → Benutzerschnittstelle, zentrale Steuerung, Hardware-Abstraktionsschicht und Navigationskernel
-
- Erster Entwurf
 - › Finden von Kernabstraktionen → Man Machine Interface (MMI), Steuerlogik, Routenberechnung und Hardware Abstraction Layer (HAL)
 - › Anwendung der Entwurfsprinzipien → Modularität, Kapselung
 - › Verwendung eines Kommunikationsframeworks zur Wahrung der konzeptuellen Integrität
 - › Durch die Untersuchung von Use-Case-Szenarien kann der Architekt die nötigen Schnittstellen der einzelnen Bausteine definieren
 - › Interne Struktur der Software:

Fallbeispiel



■ Weiteres Vorgehen

- › Iterativer und inkrementeller Verfeinerungsprozess der Architektur bis die gesamte geforderte Funktionalität vorhanden ist
- › Sinnvoll, nach jeder Iteration eine Bewertung der Architektur durchzuführen
- › Abschließende Bewertung
 - » Falls Bewertung positiv: Beginn der Implementierung
 - » Falls Bewertung negativ: Überarbeitung der Architektur

- **Ziele/Motivation**
- **Grundlagen**
 - › Kriterien für korrekten Entwurf
 - › Fundamentale Entwurfsprinzipien
 - › Sichten
- **Konkretes Vorgehen**
 - › Aktivitäten beim Architekturentwurf
 - › Konkrete Entwurfsschritte
 - › Heuristiken
- **Fallbeispiel**
- **Zusammenfassung**

- Ziel: Softwarearchitektur mit wohl definierten und aufeinander aufbauenden Abstraktionsebenen, die über wohl definierte Schnittstellen verfügen
- Erstellung der Architektur aber kein Vorgang nach Schema F
- Vielmehr ein iteratives und inkrementelles Vorantasten von den eigentlichen Anforderungen zu einer „guten“ Architektur
 - › Erfüllung bestimmter Kriterien: Kopplung, Kohäsion, Zulänglichkeit, Vollständigkeit, Einfachheit
- Hauptschwierigkeit beim Entwurf: Beherrschung von Komplexität
 - › Verwendung von Modellen und Sichten, um je nach Blickwinkel verschiedene Eigenschaften des Systems hervorzuheben
- Entwurf nach grundlegenden Prinzipien
 - › Abstraktion, Kapselung, Modularität, Hierarchie, konzeptuelle Integrität
- Benutzung von Heuristiken, um den Entwurf zu erleichtern

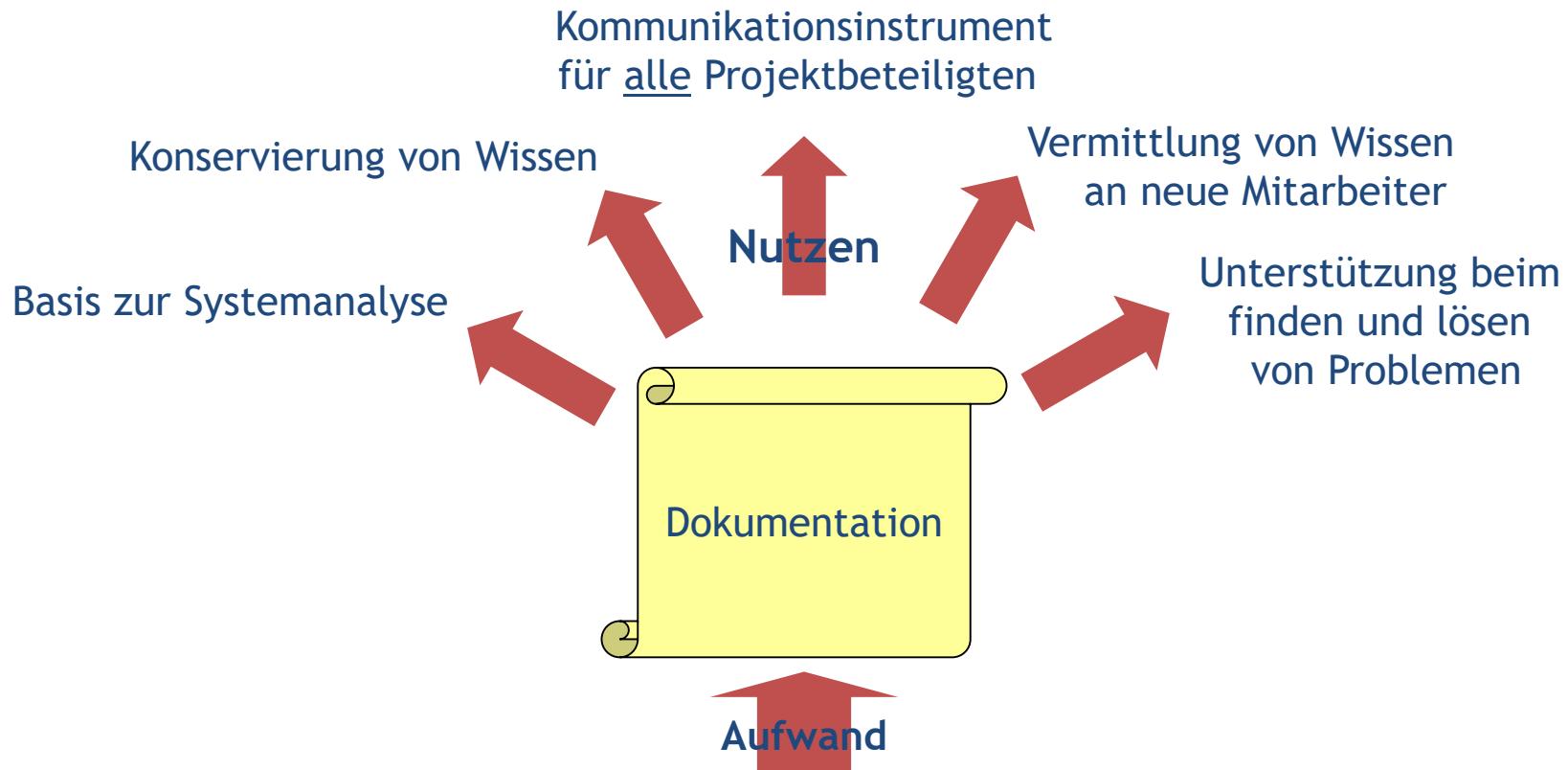
- Einführung in Software-Architekturen und Organisation
- Entwurf von SW-Architekturen
- **Dokumentation von Software-Architekturen**
- Evaluation / Bewertung von SW-Architekturen
- Toolbox des Softwarearchitekten
- Produktlinien für Software
- Enterprise Architecture Management

- Bedeutung
- Anforderungen
- Bestandteile
- Sichten
 - › Styles
 - › Typische Sichten und Styles
- Notationen
 - › Allgemeines
 - › Informale Notationen
 - › ADL
 - › UML 2.0
 - › Notationen für Sichten
 - › Beschreibung von Schnittstellen

- **Bedeutung**
- Anforderungen
- Bestandteile
- Sichten
 - › Styles
 - › Typische Sichten und Styles
- Notationen
 - › Allgemeines
 - › Informale Notationen
 - › ADL
 - › UML 2.0
 - › Notationen für Sichten
 - › Beschreibung von Schnittstellen

Bedeutung

- Die Architektur-Dokumentation ist ein wichtiges Kommunikationsinstrument



Kommunikation ist der Schlüssel zum Projekterfolg

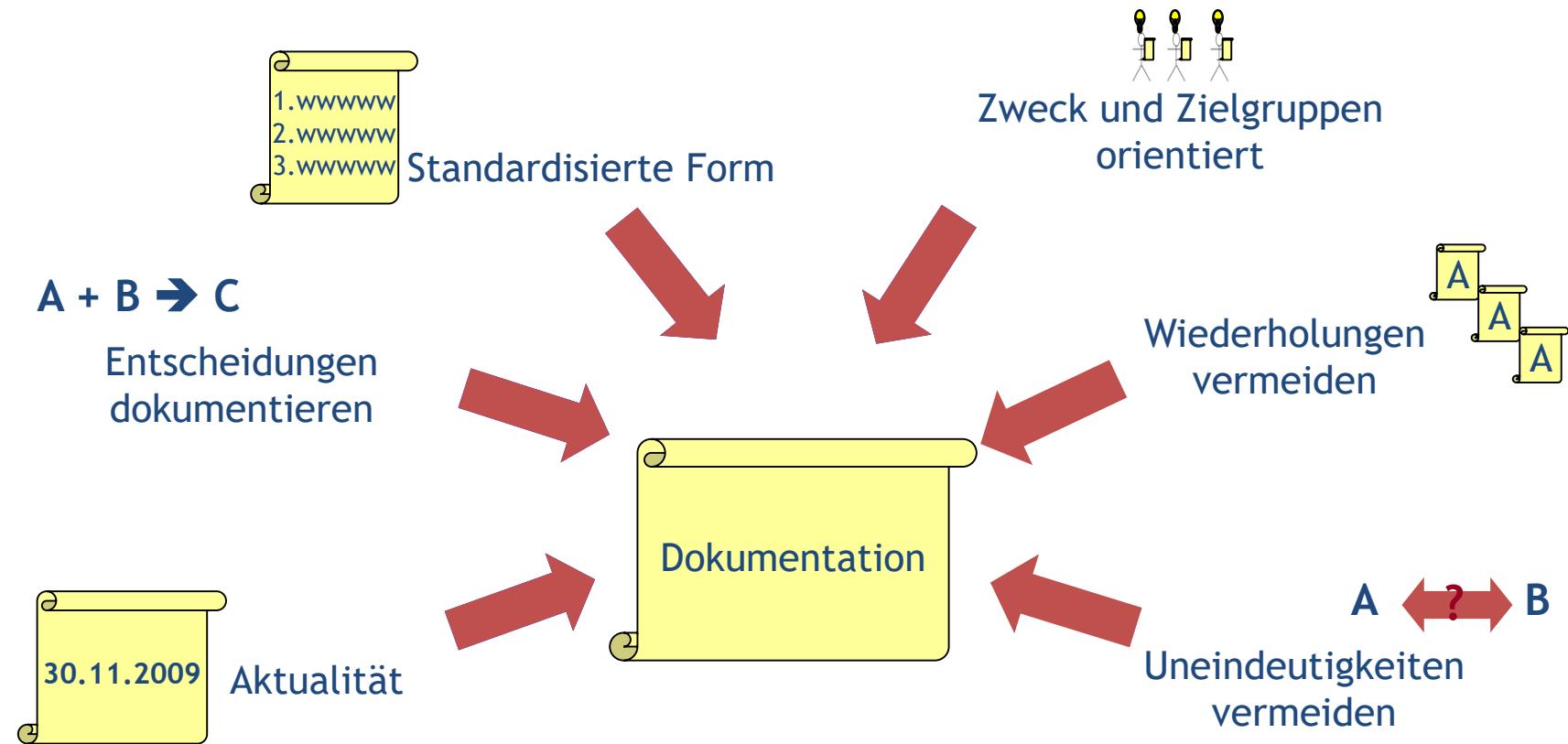
- Ausmaß heutiger Softwaresysteme erreicht einige Hunderttausend Zeilen an Quellcode
 - › Menschliches Gehirn kann daraus nicht die Architektur abstrahieren
- Softwaresysteme bleiben i.d.R. 5-10 Jahre in Betrieb und werden in dieser Zeit oft mehrmals erweitert und verändert
 - › Unmöglich, den Überblick zu behalten
- Dokumentation von Software Architekturen notwendig!

- Dokumentation hilft Architektur, ihre Ziele zu erfüllen
- Dokumentation als Mittel zur Wissensvermittlung / -bewahrung
 - › Hilft dabei, neue Teammitglieder, externe Analytiker oder neue Architekten in das System einzuführen
- Dokumentation als primäres Kommunikationsinstrument für Projektbeteiligte
 - › Präzise und eindeutig formulierte Dokumentation dient als gemeinsame Gesprächsgrundlage
- Dokumentation als Grundlage für Systemanalysen
 - › Notwendige Werte und Informationen für Systemanalyse (z.B. Ressourcenverbrauch, Abhängigkeiten und Fehlermaßnahmen) lassen sich der Dokumentation entnehmen
- Dokumentation als Mittel zur Risikominimierung
 - › Dokumentation stellt Risiken dar und bietet Strategien zur Minimierung an

- Bedeutung
- **Anforderungen**
- Bestandteile
- Sichten
 - › Styles
 - › Typische Sichten und Styles
- Notationen
 - › Allgemeines
 - › Informale Notationen
 - › ADL
 - › UML 2.0
 - › Notationen für Sichten
 - › Beschreibung von Schnittstellen

Anforderungen: Allgemein

- Um Nutzen zu erzielen muss jede Dokumentation einigen Anforderungen genügen



Anforderungen: Speziell

- Aus den Zielen einer Architektur leiten sich weitere Anforderungen für die Dokumentation ab



- Eine Architekturdokumentation adressiert viele Stakeholder

Stakeholder	Kommunikation	Stakeholder	Kommunikation
Architekt und Requirements-engineer	Verhandlung von Trade-offs zwischen Requirements	Designer von benachbarten Systemen	Schnittstellen und Protokolle
Architekt und Designer	Ressourcenverbrauch	Manager	Formen von Teams Arbeitsteilung Projektfortschritt
Programmierer	Freiheiten und Einschränkungen	Product line Manager	Feststellen der Linienzugehörigkeit
Tester und Integration	Korrekte Black-Box-Verhalten der Teile	Qualitätsüberwachung	Konformitätsüberprüfung
Wartung	Start von Wartungsarbeiten Impact-Analyse		

- Untergliederung in
 - › Funktionale Anforderungen
 - › Nicht-Funktionale Anforderungen
- Funktionale Anforderungen nach Starke
 - › Dokumentation muss Fragen von Projektbeteiligten beantworten können
 - » Allgemeine Auskünfte für alle Beteiligten
 - » Spezielle Fragen einzelner Beteiligter
- Allgemeine Auskünfte
 - › Wie fügt sich das System in seine Umgebung ein, insbesondere in seine technische Infrastruktur sowie die Projektorganisation?
 - › Wie ist das System als Menge von Implementierungseinheiten strukturiert und welche Beziehungen gibt es zwischen diesen?
 - › Wie verhalten sich die Bausteine zur Laufzeit und wie arbeiten sie zusammen?
- Spezielle Fragen
 - › Z.B. von Fachexperten, Entwicklern und Qualitätssicherern
 - » Welchen Teil des Systems entwerfe/realisiere/prüfe ich gerade und wie ordnet sich dieser in das Gesamtsystem ein?
 - » Aus welchen Gründen wurden Entwurfsentscheidungen getroffen?
 - » An welchen Stellen muss das System flexibel und variabel sein?

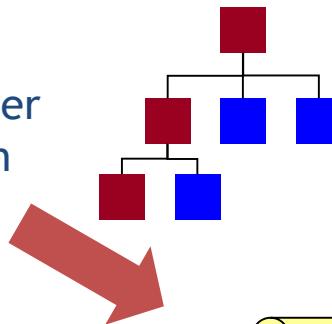
- „Seven Rules for Sound Documentation“ (Clements et al.)
 - › Regel 1: Schreiben Sie die Dokumentation aus der Sicht des Lesers!
 - » Je klarer und verständlicher, desto effizienter
 - › Regel 2: Vermeiden Sie unnötige Wiederholungen!
 - » Redundante Informationen sind schwer zu pflegen und verwirren u.U. den Leser
 - › Regel 3: Vermeiden Sie Mehrdeutigkeiten!
 - » Führt zu falschen Annahmen und falscher Arbeitsgrundlage
 - › Regel 4: Verwenden Sie eine standardisierte Form!
 - » Leser findet sich besser zurecht, dokumentieren wird leichter
 - › Regel 5: Dokumentieren Sie die Gründe Ihrer Entscheidungen!
 - » Entscheidungen können besser hinterfragt und nachvollzogen werden
 - › Regel 6: Halten Sie die Dokumentation aktuell, aber nicht zu aktuell!
 - » Dokumentation zu bestimmten Meilensteinen aktualisieren
 - › Regel 7: Lassen Sie überprüfen, ob die Dokumentation ihren Zweck erfüllt!
 - » Dokumentation vor Freigabe von potentiellen Lesern auf Zweckdienlichkeiten untersuchen lassen

- Bedeutung
- Anforderungen
- **Bestandteile**
- Sichten
 - › Styles
 - › Typische Sichten und Styles
- Notationen
 - › Allgemeines
 - › Informale Notationen
 - › ADL
 - › UML 2.0
 - › Notationen für Sichten
 - › Beschreibung von Schnittstellen

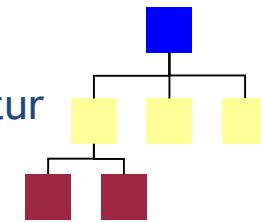
- Dokumentation besteht aus
 - › Mehreren Sichten auf die Architektur
 - » beantworten Fragen der Projektbeteiligten
 - › Informationen, die für mehr als eine Sicht relevant sind („Documentation Across / Beyond Views“)
 - » Helfen, sich in der Dokumentation zurechtzufinden
- Sichten
 - › Definition:
„Eine Sicht ist eine Repräsentation eines Satzes von Systemelementen und ihrer Beziehung zueinander.“ (Clements et al.)
 - › Analogie aus dem Immobilienwesen: Es gibt mehrere verschiedene Pläne für den Bau eines Gebäudes (Grundrisse, Elektro-, Sanitär- und Raumpläne). Alle Pläne sind nötig um das Gebäude zu bauen, allerdings stellt jeder Plan eine andere Sicht auf das Gebäude dar. Jeder Projektbeteiligte (Käufer, Maurer, Architekt, Installateur, Zimmermann etc.) benötigt i.d.R. nicht alle dieser Informationen, sondern nur die für ihn relevanten.
- Architektsichten
 - › verschiedene Blickwinkel auf die Software Architektur

- Die Qualität der Dokumentation wird von ihrer Navigierbarkeit bestimmt

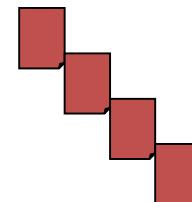
Zielgruppenorientierter
Navigationsleitfaden



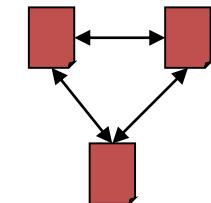
Zielgruppenorientierte Struktur



Sichten



Zusammenspiel der Sichten

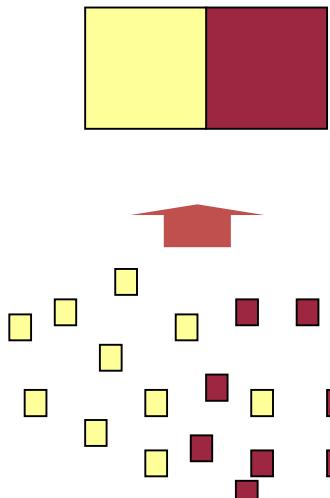


- Bedeutung
- Anforderungen
- Bestandteile
- **Sichten**
 - › Styles
 - › **Typische Sichten und Styles**
- Notationen
 - › Allgemeines
 - › Informale Notationen
 - › ADL
 - › UML 2.0
 - › Notationen für Sichten
 - › Beschreibung von Schnittstellen

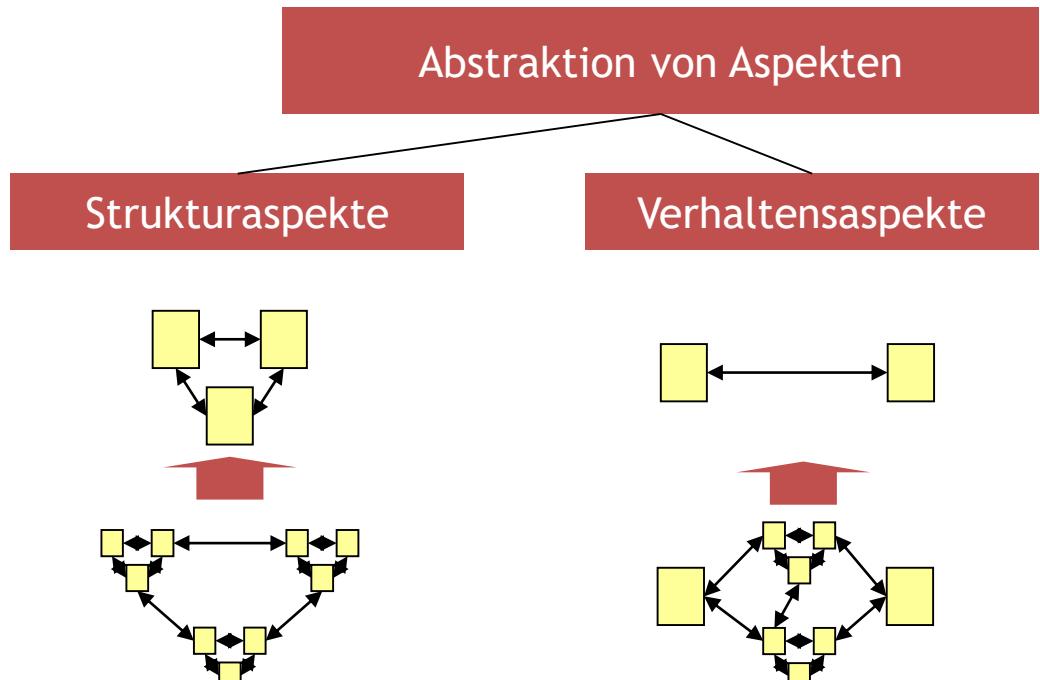
- Durch Abstraktion konzentrieren Sichten die Aufmerksamkeit auf bestimmte Problemstellungen
- Sichten: Definition

„Eine Sicht ist eine Repräsentation eines Satzes von Systemelementen und ihrer Beziehung zueinander.“¹

Abstraktion von Details

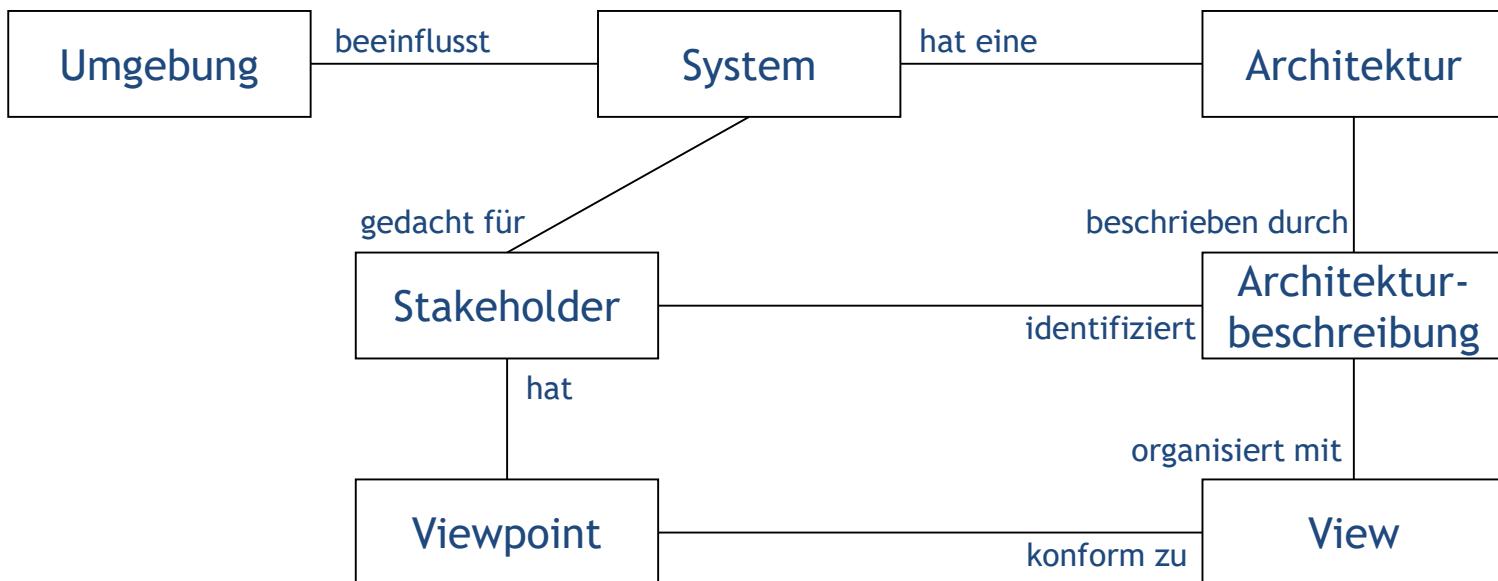


Abstraktion von Aspekten



Sichten: Definition - IEEE Framework

- Liefert wichtige Anhaltspunkt, was für eine Architekturdokumentation wichtig ist
- Beschreibt keine Notation



Styles und Pattern - Styles

- „Ein Architekturstil definiert eine Familie von Software-Systemen anhand ihrer strukturellen Organisation. Ein Architekturstil beschreibt Komponenten, Beziehungen zwischen ihnen, Einschränkungen, denen sie unterliegen, und Kombinations- und Entwurfsregeln für ihre Konstruktion.“ [BMR+98]
- Stile sind immer wiederkehrende Lösungsmuster für Architekturprobleme
- z.B. Pipe-and-Filter



- Unterschied zum Pattern, liegt in der reduzierten Dokumentation
 - › Keine Dokumentation des Kontexts
 - › Keine genaue Definition des Problems

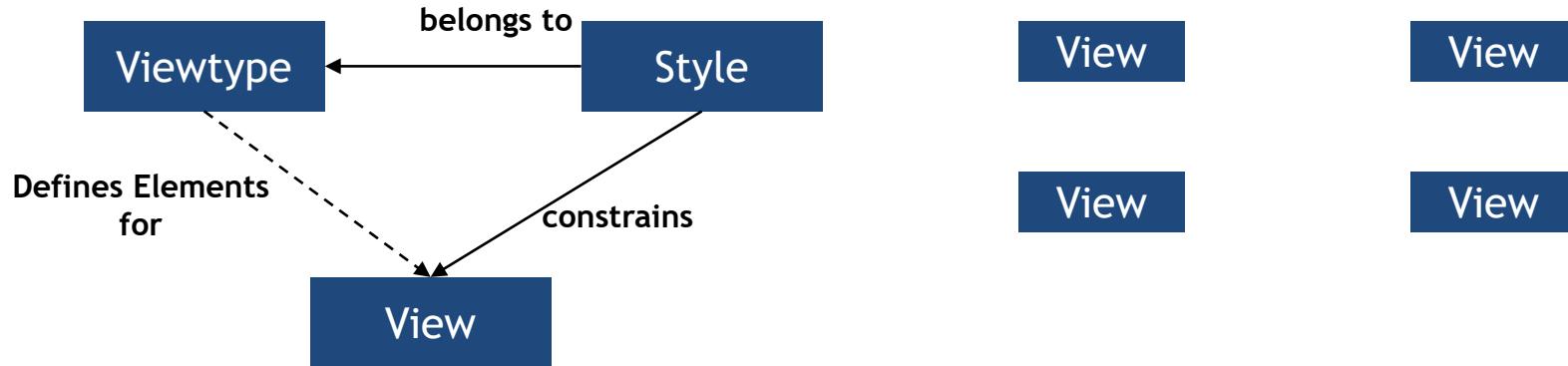
Eine Standarddokumentation von Stilen ist für das Verständnis hilfreich

- Overview
 - › Warum ist dieser Stil nützlich?
 - › Worum geht es?
 - › Wie unterstützt es die Analyse von Systemen?
- Elemente, Relationen, Eigenschaften
 - › Bausteine des Stils
 - › Die Beziehungen zwischen den Bausteinen
 - › Eigenschaften von Relationen und Elementen
- Wofür ist der Stil gut und wofür nicht?
- Notationen
- Beziehungen zu anderen Sichten
- Beispiel

Definition – Viewtypes, View

[CBB+05]

[PBG04]



- Viewtypes
 - Module
 - Component & Connector
 - Allocation

- Sichten
 - Kontext
 - Struktur
 - Verhalten
 - Abbildung

Es gibt kein einheitliches Auswahlkriterium für Sichten

Definition - Verschiedene Sichten

[CBB+05]

Module

Component
&
Connector

Allocation

[PBG04]

Kontext

Struktur

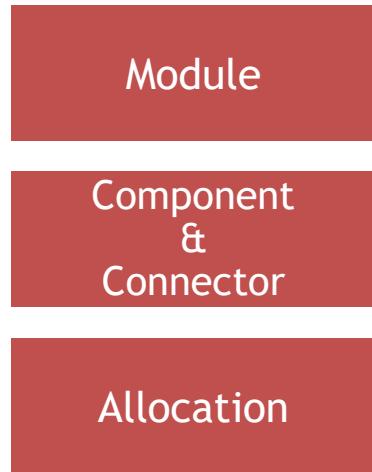
Verhalten

Abbildung

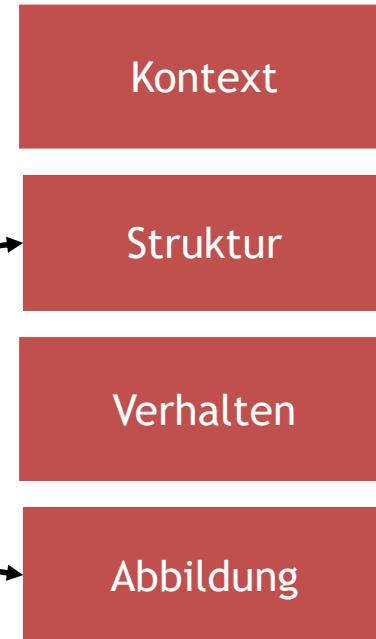
Definition - Verschiedene Sichten

[CBB+05]

[PBG04]



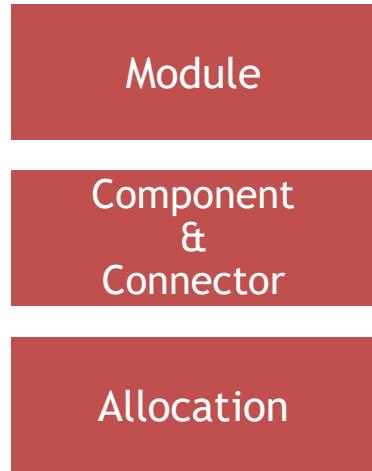
Ähnliche Zielsetzung



Definition - Verschiedene Sichten

[CBB+05]

[PBG04]



Spezialfall



Definition - Verschiedene Sichten

[CBB+05]

[PBG04]

Requirements Spezifikation:
Jedoch Praktische Relevanz
im Architekturdesign

Module

Component
&
Connector

Allocation

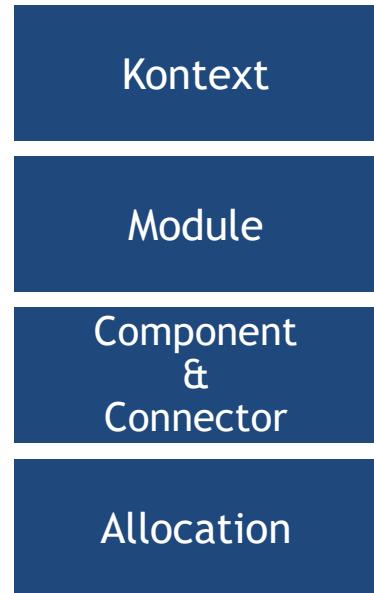
Kontext

Struktur

Verhalten

Abbildung

Definition - Verschiedene Sichten



- Die Kontextsicht beschreibt die Einbettung des Systems in seine Umgebung

Ziele

- Darstellung des Systems als Black-Box
- Identifizieren der Systemgrenzen
- Schnittstellen des Systems festlegen

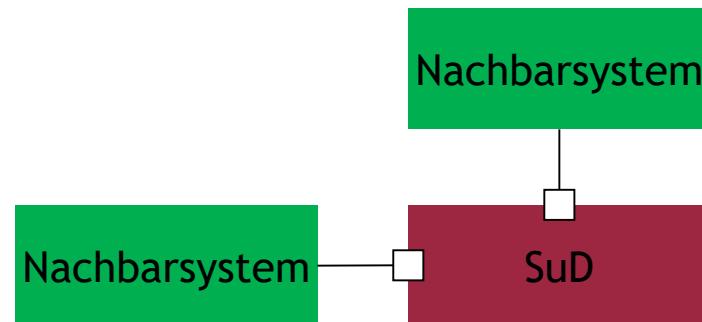
Elemente

- Externe Elemente
 - Nachbarsysteme
 - Benutzer
- System (SuD)
- Schnittstellen

Styles

n/a

Beispiel



- In der Modulsicht wird das System in einfachere Teile zerlegt

Ziele

- Systemkonstruktion
- Aufteilen von Verantwortlichkeiten
- Impact-Analyse
- Kommunikation der Systemfunktionalität

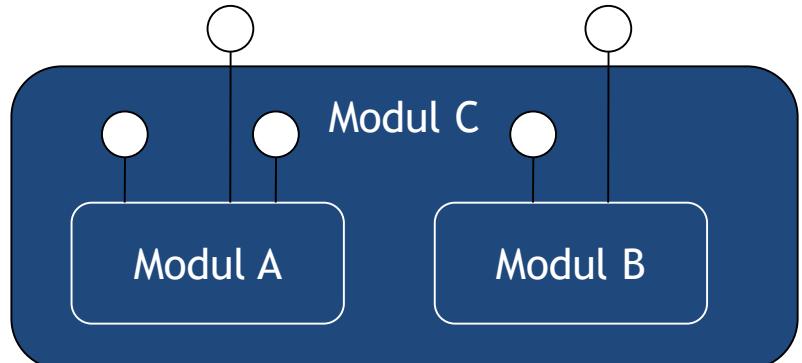
Elemente

- Module (d.h. Klassen, Layer, eine Gruppe von Klassen, jede Aufteilung von Code-Einheiten)
- Relationen
 - is-part-of
 - depends-on
 - is-a

Styles

- Decomposition
- Uses
- Generalization
- Layered

Beispiel



- Komponenten beschreiben eher Laufzeiteinheiten
- Module beschreiben eher Designeinheiten
- In der Literatur unterschiedliche Unterscheidungen
- Hier Benutzung im Sinne von [CBB+05]
 - › Module
 - » Entität während des Designs
 - » Zur Beschreibung der Systemstruktur
 - › Komponenten
 - » Laufzeitentitäten
 - » Zur Beschreibung von Daten- und Kontrollflüssen

- *Decomposition* teilt das System hierarchisch auf
- *Uses* beschreibt Abhängigkeiten zwischen Modulen

Sichten: Modulsicht: Decomposition, Uses

Decomposition

Elements: Module / Subsystems
Relations: is-part-of
Topology: keine Schleifen
ein Modul kann nicht zu mehreren gehören

Beispiel

```
System A
  Subsystem A1
  Subsystem A2
    SubSubsystem A2a
System B
```

Uses

Elements: Module / Subsystems
Relations: uses (specialized depends-on)
Topology: n/a

Beispiel

uses	A	B	C
A			✓
B	✓		
C		✓	

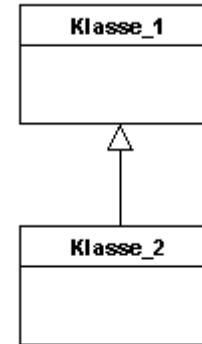
- *Generalization* beschreibt Vererbungsstrukturen
Eigenschaften zur Portierbarkeit werden mit *Layered* dargestellt

Sichten: Modulsicht: Generalization, Layered

Generalization

Elements: Module / Subsystems
Relations: is-a
Topology: keine Schleifen
mehrfache Eltern sind zu vermeiden

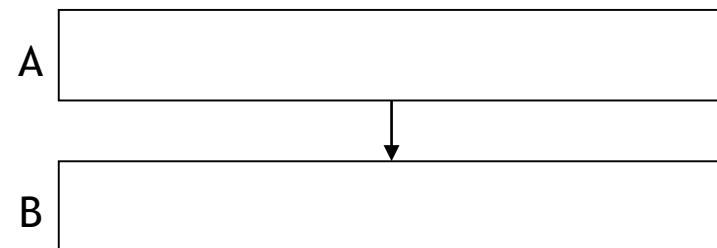
Beispiel



Layered

Elements: Layers
Relations: allowed-to-use
Topology: Layer A über Layer B
→ Layer B nicht über Layer A

Beispiel



- Die Component-and-Connector Sicht gibt Aufschluss über die Struktur des Systems zur Laufzeit

Sichten: Component-and-Connector Sicht

Ziele

- Modellierung von (Laufzeit-) Komponenten und ihr Zusammenspiel
- Abschätzung von Qualitätsattributen (z.B. Performance, Zuverlässigkeit,...)
- Datenflussmodellierung

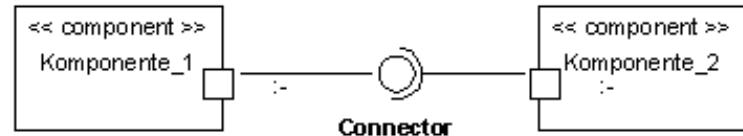
Elemente

- Komponenten
- Ports
- Connector
- Attachments (zuordnung von Rollen zu Ports und Connector)

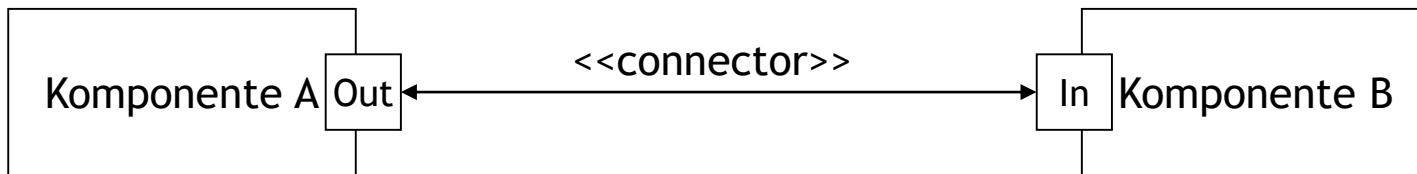
Styles

- Pipe and Filter
- Shared Data
- Publish Subscribe
- Client Server
- Peer To Peer
- Communicating Processes

Beispiel



- Connectoren und Ports sind ein Mittel zur Abstraktion
- Connector
 - › Interaktionspfad zur Laufzeit
- Port
 - › Namentlichen Interaktionspunkt einer Komponente
- Schnittstelle
 - › Mögliche Typisierung von Ports



- Pipe-and-Filter und Shared Data sind zwei Stile, die den Datenfluss beschreiben

Sichten: Component and Connector Sicht: Pipe-and-Filter, Shared Data

Pipe-and-Filter

Elements: Filter-Komponenten
(Input-/Output-Ports)

Relations: Pipe-Connector

Topology: Pipes verbinden Output-Ports
mit Input-Ports

Beispiel



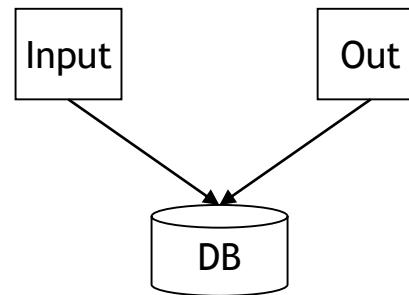
Shared Data

Elements: Repositories und
Data Accessor

Relations: Read-/Write-Connectoren

Topology: Connectoren verbinden Data
Accessors und
Repositories

Beispiel



- Publish-Subscribe ist ein Stil für den Kontrollfluss
Im Client-Server Stil wird Funktionalität in Services gekapselt

Sichten: Component and Connector Sicht: Publish-Subscribe, Client-Server

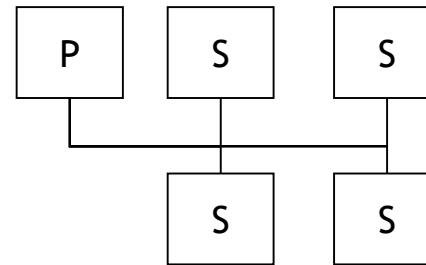
Publish-Subscribe

Elements: Publisher-/Subscriber-
Komponenten

Relations: Publish-Subscribe-Bus

Topology: Alle Komponenten sind mit
einem Event-Distributor
gekoppelt

Beispiel



Client-Server

Elements: Client- und Server-
Komponenten

Relations: request/reply

Topology: (Optionale Restriktionen)
Anzahl der Verbindungen
Server-Servern-Verbindungen
Tiers

Beispiel



Sichten: C&C View Type- Styles

- Peer-to-Peer-Stil eignet sich für Distributed Computing
Communicating Processes beschreibt komplexe Interprozesskommunikation

Sichten: Component and Connector Sicht: Peer-to-Peer, Communicating Processes

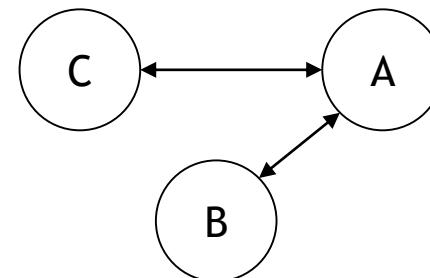
Peer-to-Peer

Elements: Peers

Relations: Invokes Procedure

Topology: (optionale Restriktionen)
Verbindungen
Sichtbarkeit

Beispiel



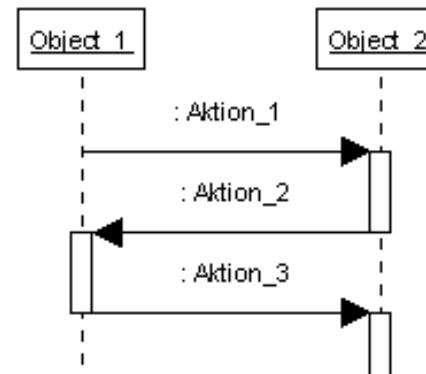
Communicating Processes

Elements: Concurrent Units

Relations: data exchange, sync,
control, etc.

Topology: -

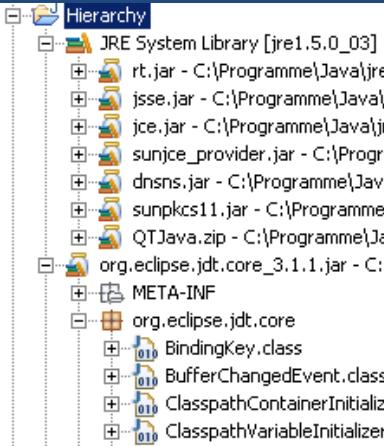
Beispiel



Sichten: Allocation View Type

- Im Allocation Viewtype werden Softwareelemente Objekten der realen Welt zugeordnet

Sichten: Allocation

Ziele	Elemente
<ul style="list-style-type: none"> Organisatorische Aspekte <ul style="list-style-type: none"> Configuration Management Arbeitsaufteilung Verteilung auf Hardware 	<ul style="list-style-type: none"> Softwareelemente Umgebungselemente Allocated-to Relationen
Styles	Beispiel
<ul style="list-style-type: none"> Deployment Implementation Work Assignment 	 <p>The screenshot shows a file tree titled 'Hierarchy'. It displays the contents of the 'JRE System Library [jre1.5.0_03]' and 'org.eclipse.jdt.core_3.1.1.jar' jars. The 'JRE System Library' contains several JAR files like rt.jar, jsse.jar, jce.jar, sunjce_provider.jar, dnsns.jar, and sunpkcs11.jar. The 'org.eclipse.jdt.core' jar contains a 'META-INF' folder and several class files including BindingKey.class, BufferChangedEvent.class, ClasspathContainerInitializer, and ClasspathVariableInitializer.</p>

Sichten: Allocation View Type - Styles

- Deployment beschreibt die Laufzeitaufteilung
- Implementation beschreibt das Configurationmanagement

Sichten: Allocation: Deployment, Implementation

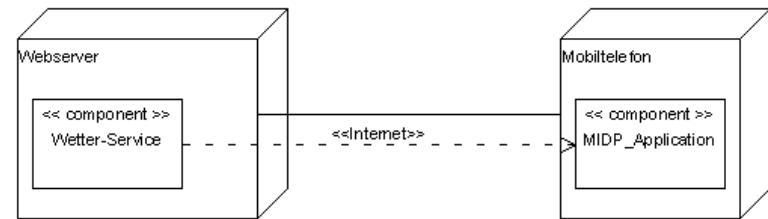
Deployment

Elements: Software Element (Prozess),
 Computerhardware

Relations: Allocated-to, migrates-to,
 copy-migrates-to, etc.

Topology: -

Beispiel



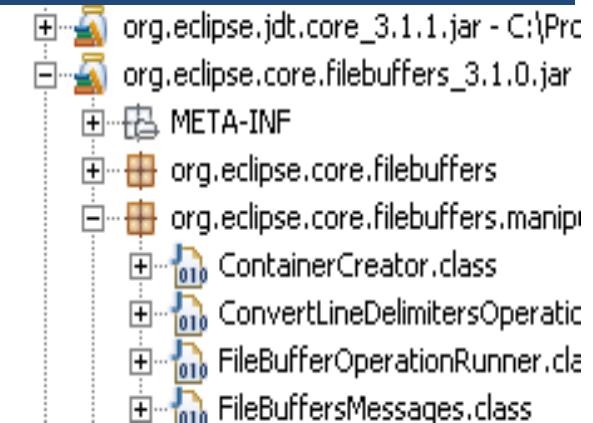
Implementation

Elements: Modul,
 Configuration Item

Relations: Containment
 allocated-to

Topology: hierarchisch

Beispiel



- Work Assignment ordnet Module als Arbeitspakete Organisationseinheiten zu

Sichten: Allocation: Work Assignment

Work Assignment

Elements: Module,
Organisationseinheiten

Relations: allocated-to

Topology: -

Beispiel

Team	Task
Team A	Modul 3
Team B	Modul 2
Team C	Modul 1

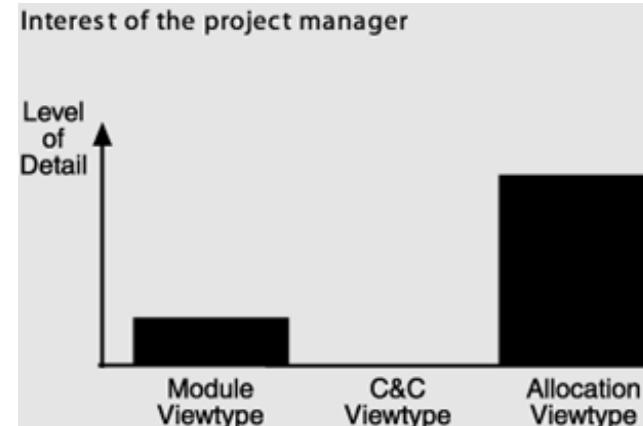
Auswahl der Views für eine Dokumentation: Adressaten

- Für Auswahl der Views ist es wichtig zu wissen:
 - › Wer wird die Dokumentation nutzen?
 - › Welche Informationen braucht jeder dieser Nutzer?
- Häufige Nutzer (=Stakeholder) einer Dokumentation sind:
 - › Projekt-Manager
 - › Entwickler
 - › Tester und Integratoren
 - › Entwickler andere Systeme
 - › Software-Wartungspersonal
 - › Anwendungs-Erststeller
 - › Kunden
 - › Endbenutzer
 - › Architekten (aktuelle und zukünftige)

1. Beispiel: Projekt-Manager

- Interessen:
 - › Überblick über System
 - › Zeitplanung, Fortschritt der Arbeiten
 - › Verteilung der Ressourcen (Arbeitskraft, etc.)
 - › Vorbereitung der Verbindung zu anderen Systemen und der Hardwareumgebung

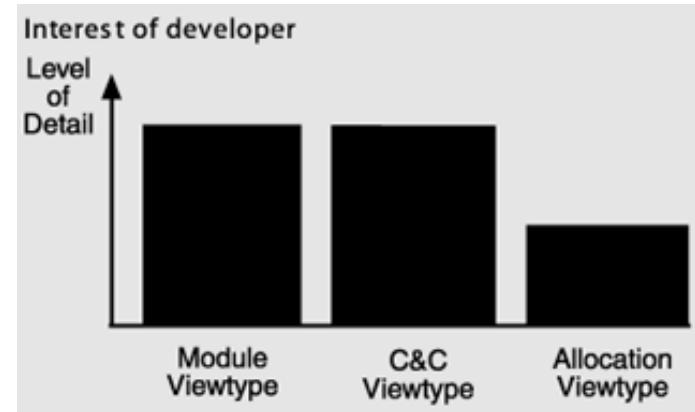
- Bedarf:
 - › Top-Level-Kontextdiagramm
(Module Viewtype)
 - › Decomposition View
(Module Viewtype)
 - › Work-Assignment View
(Allocation Viewtype)
 - › Deployment View
(Allocation Viewtype)
 - › Die von Views unabhängigen
Teile der Dokumentation



2. Beispiel: Entwickler

- Interessen:
 - › Vorgaben für Implementierung ihm zugewiesener Elemente
 - › Bei Kauf von Komponenten (vgl. Fallstudie): Entwickler macht Anpassungen

- Bedarf:
 - › Kontextdiagramm seiner Elemente
(Module Viewtype)
 - › Decomposition View
(Module Viewtype)
 - › Views zu Interaktion seiner Elemente
(C&C Viewtype)
 - › Schnittstellenbeschreibungen
(Module oder C&C Viewtype)
 - › Etc.



Auswahl der Views für eine Dokumentation: Vorgehensweise zur Auswahl der Views

1. Liste der View-Kandidaten erstellen

- Nachfrage bei den Stakeholdern ermitteln (z.B. in Gesprächen, Workshops, etc.)
- Ergebnisse in Stakeholder/View-Tabelle erfassen

Auswahl der Views für eine Dokumentation: Stakeholder/View–Tabelle

Stakeholder	Module Viewtype					C & C Viewtype					Allocation Viewtype		
	Decompo-sition	Genera-lization	Uses	Layered	Pipe-And-Filter	Shared-Data	Client-Server	Peer-To-Peer	Com.-Processes	Deploy-ment	Imple-mentation	Work-Assignment	
Architekt	D	D	D	D	D	D	D	D	D	D	T	T	
Projekt-Manager (Staat)	D	Ü	Ü	T	Ü	T	Ü	Ü	Ü	T		D	
Projekt-Manager (Unternehmen)	T	Ü	T	T	Ü	T	T	T	Ü	D	T	D	
Entwickler	D	D	D	D	Ü	D	D	D	D	T	T	D	
Tester u. Integr.	T	T	D	T	Ü	D	D	D	T	T	D		
Wartungspers.	D	D	D	D	Ü	D	D	D	D	T	T	T	
Komponenten-Ingenieure	D	T		D		D	D	D	T	D		D	
Analyst (Performance)	D	T	D	T	Ü	D	D	D	D	D			
Analyst (Datenintegrität)	T	T	T	D	Ü	D	D	D	D	D			
Analyst (Sicherheit)	D	T	D	D	Ü	T	D	D	D	D	Ü	Ü	
Analyst (Verfügbarkeit)	D	S	D	D				T	T	D		Ü	
Nutzer (Notfall)	Ü				Ü								
Nutzer	Ü				Ü	Ü				Ü			

Auswahl der Views für eine Dokumentation: Vorgehensweise zur Auswahl der Views

2. Reduzieren der View-Anzahl:

- Können Informationen auch aus anderen Views gewonnen werden?
- Kann man ähnlich nachgefragte Views kombinieren?

Auswahl der Views für eine Dokumentation: Stakeholder/View–Tabelle

Work-Assignment und Implementation:

- Guter Standard-Kandidat für Kombination
- Nachfrage fast identisch
- Inhaltlich zur Kombination geeignet

Aber:

- Großes Projekt
 - Mehrere Unternehmen
→ Arbeitsaufteilung in den Unternehmen
- Views getrennt lassen

Stakeholder	Allocation Viewtype		
	Deploy- ment	Imple- mentation	Work- Assignment
Architekt	D	T	T
Projekt-Manager (Staat)	T		D
Projekt-Manager (Unternehmen)	D	T	D
Entwickler	T	T	D
Tester u. Integr.	T	D	
Wartungspers.	T	T	T
Komponenten- Ingenieure	D		D
Analyst (Performance)	D		
Analyst (Datenintegrität)	D		
Analyst (Sicherheit)	D	Ü	Ü
Analyst (Verfügbarkeit)	D		Ü
Nutzer (Notfall)			
Nutzer	Ü		

D: Detaillierte Informationen / T: Detaillierte Informationen zu Teilbereichen / Ü: Übersichtsverschaffende Informationen

Auswahl der Views für eine Dokumentation: Stakeholder/View–Tabelle zur Fallstudie

*Client-Server und
Peer-To-Peer:*

- Nachfrage fast identisch
- Inhaltlich zur Kombination geeignet

→ Kombination zu neuem View

Stakeholder	C & C Viewtype				
	Pipe-And-Filter	Shared-Data	Client-Server	Peer-To-Peer	Com.-Processes
Architekt	D	D	D	D	D
Projekt-Manager (Staat)	Ü	T	Ü	Ü	Ü
Projekt-Manager (Unternehmen)	Ü	T	T	T	Ü
Entwickler	Ü	D	D	D	D
Tester u. Integr.	Ü	D	D	D	T
Wartungspers.	Ü	D	D	D	D
Komponenten-Ingenieure		D	D	D	T
Analyst (Performance)	Ü	D	D	D	D
Analyst (Datenintegrität)	Ü	D	D	D	D
Analyst (Sicherheit)	Ü	T	D	D	D
Analyst (Verfügbarkeit)				T	T
Nutzer (Notfall)	Ü				
Nutzer	Ü	Ü			

D: Detaillierte Informationen / T: Detaillierte Informationen zu Teilbereichen / Ü: Übersichtsverschaffende Informationen

Auswahl der Views für eine Dokumentation: Stakeholder/View–Tabelle zur Fallstudie

Stakeholder	Module Viewtype					C & C Viewtype				Allocation Viewtype		
	Decompo-sition	Genera-lization	Uses	Layered	Pipe-And-Filter	Shared-Data	C-S/ P2P	Com.-Proces-ses	Deploy-ment	Imple-mentation	Work-Assig-nment	
Architekt	D	D	D	D	D	D	D	D	D	T	T	
Projekt-Manager (Staat)	D	Ü	Ü	T	Ü	T	Ü	Ü	T		D	
Projekt-Manager (Unternehmen)	T	Ü	T	T	Ü	T	T	Ü	D	T	D	
Entwickler	D	D	D	D	Ü	D	D	D	T	T	D	
Tester u. Integr.	T	T	D	T	Ü	D	D	T	T	D		
Wartungspers.	D	D	D	D	Ü	D	D	D	T	T	T	
Komponenten-Ingenieure	D	T		D		D	D	T	D			D
Analyst (Performance)	D	T	D	T	Ü	D	D	D	D			
Analyst (Datenintegrität)	T	T	T	D	Ü	D	D	D	D			
Analyst (Sicherheit)	D	T	D	D	Ü	T	D	D	D	Ü	Ü	
Analyst (Verfügbarkeit)	D	S	D	D			T	T	D		Ü	
Nutzer (Notfall)	Ü				Ü							
Nutzer	Ü				Ü	Ü			Ü			

Auswahl der Views für eine Dokumentation: Stakeholder/View–Tabelle zur Fallstudie

Pipe-And-Filter und Shared-Data:

- Inhaltlich gut zur Kombination geeignet
- Nutzerkreis fast identisch
- Kombination zu neuem View (Wurde bereits gezeigt)

Stakeholder	C & C Viewtype			
	Pipe-And-Filter	Shared-Data	C-S/ P2P	Com.- Processes
Architekt	D	D	D	D
Projekt-Manager (Staat)	Ü	T	Ü	Ü
Projekt-Manager (Unternehmen)	Ü	T	T	Ü
Entwickler	Ü	D	D	D
Tester u. Integr.	Ü	D	D	T
Wartungspers.	Ü	D	D	D
Komponenten-Ingenieure		D	D	T
Analyst (Performance)	Ü	D	D	D
Analyst (Datenintegrität)	Ü	D	D	D
Analyst (Sicherheit)	Ü	T	D	D
Analyst (Verfügbarkeit)			T	T
Nutzer (Notfall)	Ü			
Nutzer	Ü	Ü		

D: Detaillierte Informationen / T: Detaillierte Informationen zu Teilbereichen / Ü: Übersichtsverschaffende Informationen

Auswahl der Views für eine Dokumentation: Stakeholder/View–Tabelle

Stakeholder	Module Viewtype				C & C Viewtype			Allocation Viewtype		
	Decompo-sition	Genera-lization	Uses	Layered	P&F/ S-D	C-S/ P2P	Com.- Proces-ses	Deploy- ment	Imple- men-tation	Work- Assignment
Architekt	D	D	D	D	D	D	D	D	T	T
Projekt-Manager (Staat)	D	Ü	Ü	T	T	Ü	Ü	T		D
Projekt-Manager (Unternehmen)	T	Ü	T	T	T	T	Ü	D	T	D
Entwickler	D	D	D	D	D	D	D	T	T	D
Tester u. Integr.	T	T	D	T	D	D	T	T	D	
Wartungspers.	D	D	D	D	D	D	D	T	T	T
Komponenten- Ingenieure	D	T		D	D	D	T	D		D
Analyst (Performance)	D	T	D	T	D	D	D	D		
Analyst (Datenintegrität)	T	T	T	D	D	D	D	D		
Analyst (Sicherheit)	D	T	D	D	T	D	D	D	Ü	Ü
Analyst (Verfügbarkeit)	D	S	D	D		T	T	D		Ü
Nutzer (Notfall)	Ü				Ü					
Nutzer	Ü				Ü			Ü		

Auswahl der Views für eine Dokumentation: Stakeholder/View–Tabelle

Uses und
Decomposition:

- Inhaltlich zur Kombination geeignet
 - Nutzerkreis fast identisch
 - Benötigte Details ähnlich
- Kombination zu neuem View

Stakeholder	Module Viewtype			
	Decompo-sition	Genera-lization	Uses	Layered
Architekt	D	D	D	D
Projekt-Manager (Staat)	D	Ü	Ü	T
Projekt-Manager (Unternehmen)	T	Ü	T	T
Entwickler	D	D	D	D
Tester u. Integr.	T	T	D	T
Wartungspers.	D	D	D	D
Komponenten-Ingenieure	D	T		D
Analyst (Performance)	D	T	D	T
Analyst (Datenintegrität)	T	T	T	D
Analyst (Sicherheit)	D	T	D	D
Analyst (Verfügbarkeit)	D	S	D	D
Nutzer (Notfall)	Ü			
Nutzer	Ü			

D: Detaillierte Informationen / T: Detaillierte Informationen zu Teilbereichen / Ü: Übersichtsverschaffende Informationen

Auswahl der Views für eine Dokumentation: Stakeholder/View–Tabelle

Stakeholder	Module Viewtype			C & C Viewtype			Allocation Viewtype		
	Decomp./ Uses	Genera- lization	Layered	P&F/ S-D	C-S/ P2P	Com.- Processe s	Deploy- ment	Imple- mentation	Work- Assignment
Architekt	D	D	D	D	D	D	D	T	T
Projekt-Manager (Staat)	D	Ü	T	T	Ü	Ü	T		D
Projekt-Manager (Unternehmen)	T	Ü	T	T	T	Ü	D	T	D
Entwickler	D	D	D	D	D	D	T	T	D
Tester u. Integr.	T	T	T	D	D	T	T	D	
Wartungspers.	D	D	D	D	D	D	T	T	T
Komponenten- Ingenieure	D	T	D	D	D	T	D		D
Analyst (Performance)	D	T	T	D	D	D	D		
Analyst (Datenintegrität)	T	T	D	D	D	D	D		
Analyst (Sicherheit)	D	T	D	T	D	D	D	Ü	Ü
Analyst (Verfügbarkeit)	D	S	D		T	T	D		Ü
Nutzer (Notfall)	Ü			Ü					
Nutzer	Ü			Ü			Ü		

3. Prioritäten setzen:

- Für Projektplanung benötigte Views (Überblick, Arbeitsaufteilung) vorziehen!
- Falls Analysen/Verifikationen Views benötigen, diese vorziehen!
- Fallstudie:
 - Vorziehen, da wichtig für Koordination der Unternehmen:
 - Decomposition View (Aus welchen Teilen besteht das System)
 - Work-Assignment View (Wer macht was)
 - Implementation View:
 - Vollständige Erstellung erst möglich, wenn Implementierungsdetails durch die Architekten der einzelnen Unternehmen festgelegt sind
(Jeder Architekt in einem Unternehmen entscheidet selbst, was er wie implementiert)
 - Evaluation
 - bei diesem Projekt sehr wichtig
 - Benötigte Views vorziehen

- Grundprinzipien
 - › Hierarchischer Aufbau
 - › Zielgruppenorientierung
 - › Klare Beziehungen zwischen den Views
- Grundentscheidung: Einzeldokumente oder Gesamtdokument?
 - Abhängig von:
 - › Systemgröße
 - › Geplante View Packets
 - › vorhandene Konventionen
 - › Fallstudie: Großes System, mehrere ausführende Unternehmen
 - Einzeldokumente

- Für jedes Dokument sollte angegeben sein:
 - › Erstelldatum und Status
 - › Herausgeber
 - › Changelog
 - › Zusammenfassung
- Vollständigkeit der Beschreibung (Descriptive Completeness)
 - › Vollständigkeit: Wenn Element in View zu Systemteil fehlt, existiert es auch nicht!
 - › Keine Vollständigkeit:
Aussage nicht möglich, da Element evtl. auch erst in Refinement enthalten ist!
 - › Fallstudie:
Für die Übersichtlichkeit ist es sinnvoller, Details in Refinements aufzudecken
→ Keine Vollständigkeit

Zusammenstellen der Dokumentation: Umfang einer View-Dokumentation

- Primäre Präsentation (meist graphisch):
 - › Hauptelemente und –relationen
 - › alle wichtigen Informationen
 - › Weglassen von z.B. Fehlerbehandlung
- Elementkatalog (Details zu allen Elementen)
 - › Eigenschaften
 - › Schnittstellen
 - › Verhalten
 - › Relationen
- Kontextdiagramm:
 - › Einordnung des Ausschnitts in das System
 - › Beziehungen zwischen Systemauschnitt und dessen Umgebung
- Variability Guide:
 - › Dokumentation vorhandener Veränderlichkeit oder Dynamik
- Hintergrund der Architektur:
 - › Begründung für Wahl der Architektur

Die Angaben neben den Views beantworten die Fragen:

- Wie ist die Dokumentation organisiert?
- Was stellt die Architektur dar?
- Warum wurde die Architektur so gewählt?

- Wie ist die Dokumentation organisiert?
 - › Documentation Roadmap
 - » Umfang der vorliegenden Dokumentation verdeutlichen
 - » Beschreibung zu jedem der Teile:
 - Abstract
 - Autor
 - Position in der Dokumentation
 - Versionsnummer
 - » Wie Leser die Dokumentation benutzen können:
 - Beispielszenarien für verschiedene Nutzergruppen
 - › View Template
 - » Ähnlich wie zuvor gesehener Aufbau einer View-Dokumentation (jedoch detaillierter)
 - » Hilft Leser beim Auffinden der gesuchten Information
 - » Schreiber hat Vorgabe zur einheitlichen Organisation von Views (wichtig v.a. bei vielen Schreibern!)
 - » Fallstudie: Dokumentation wird von vielen Autoren verfasst (Detail-Views durch Architekt im ausführenden Unternehmen)
→ Sehr exaktes View-Template notwendig, damit Dokumentation einheitlich ist

- Was stellt die Architektur dar?
 - › Zweck des Systems
 - › Benutzer des Systems
 - › Hintergrundwissen
 - › Existierende Beschränkungen
 - › Zusammenhang zwischen den Views erklären
 - » z.B. durch Mapping-Tabellen
 - › Elementverzeichnis
 - » Verzeichnis aller Elemente aus allen Views
 - » Verweis auf die Definition eines Elements
 - › Glossar + Abkürzungsverzeichnis für Begriffe mit besonderer Bedeutung

- Warum wurde die vorliegende Architektur gewählt?
 - › Darlegung der Entscheidungen eines Architekten
 - › Betrachtete Alternativen erläutern
 - › Begründungen gegen diese Alternativen
 - › Zeigen, warum die getroffene Entscheidung die Beste war
 - › Aus Entscheidung resultierende Auswirkungen, z.B. für
 - » Entwickler
 - » Benutzer

Zusammenstellen der Dokumentation:

Validierung der Dokumentation

- Nicht nur Software, sondern auch Dokumentation validieren!
- Validierung z.B. durch Fragenkatalog (Auszug):
 - › Passt Dokumentation zu den Nutzern und deren Informationsbedarf?
 - › Ist die Dokumentation konsistent logisch?
 - » Frei von Widersprüchen?
 - » Frei von Mehrdeutigkeiten?
 - › Passt die Dokumentation zur Architektur?
 - › Ist die Dokumentation noch aktuell?
 - › Fallstudie: Sicherheitskritische Anwendung!
 - Sehr genaue Validierung der Dokumentation durchführen!
 - Fehlerhafte Dokumentation als Basis für Erweiterungen gefährlich!

- Bedeutung
- Anforderungen
- Bestandteile
- Sichten
 - › Styles
 - › Typische Sichten und Styles
- Notationen
 - › **Informale Notationen**
 - › **ADL**
 - › **UML 2.0**
 - › **Notationen für Sichten**
 - › **Beschreibung von Schnittstellen**

- Viele Möglichkeiten der Dokumentation
 - › Dokumentation durch Text
 - › Tabellen
 - › Kästchen und Pfeile
- Vorteile
 - › Anpassung auf bestimmtes Problem
 - › Schnelle Einarbeitung und Anwendung
- Nachteile
 - › Kein Allgemeines Verständnis
 - › Können zu Missverständnissen führen
 - › Zusätzlicher Dokumentationsaufwand um Syntax und Semantik festzuhalten

- Formale Notationen
- Häufig gibt es Tools zur Umsetzung in graphische Notationen
- Vorteile
 - › Strikte Notation
 - › Wenige Freiheiten lassen wenig Spielraum für Missverständnisse
 - › Viewtype, Style, View-Konzept 1:1 umgesetzt
- Nachteile
 - › Analyse schwierig
 - › Unhandliches Kommunikationsmittel
 - › Schlechter Tool-Support

```
Family PipeFilter = {  
    Port Type OutputPort;  
    Port Type InputPort;  
    Role Type Source;  
    Role Type Sink;  
  
    Component Type Filter;  
    Connector Type Pipe = {  
        Role Src : Source;  
        Role snk : Sink;  
        Properties {  
            latency : int;  
            pipeProtocol:String = ...;  
        }  
    };
```

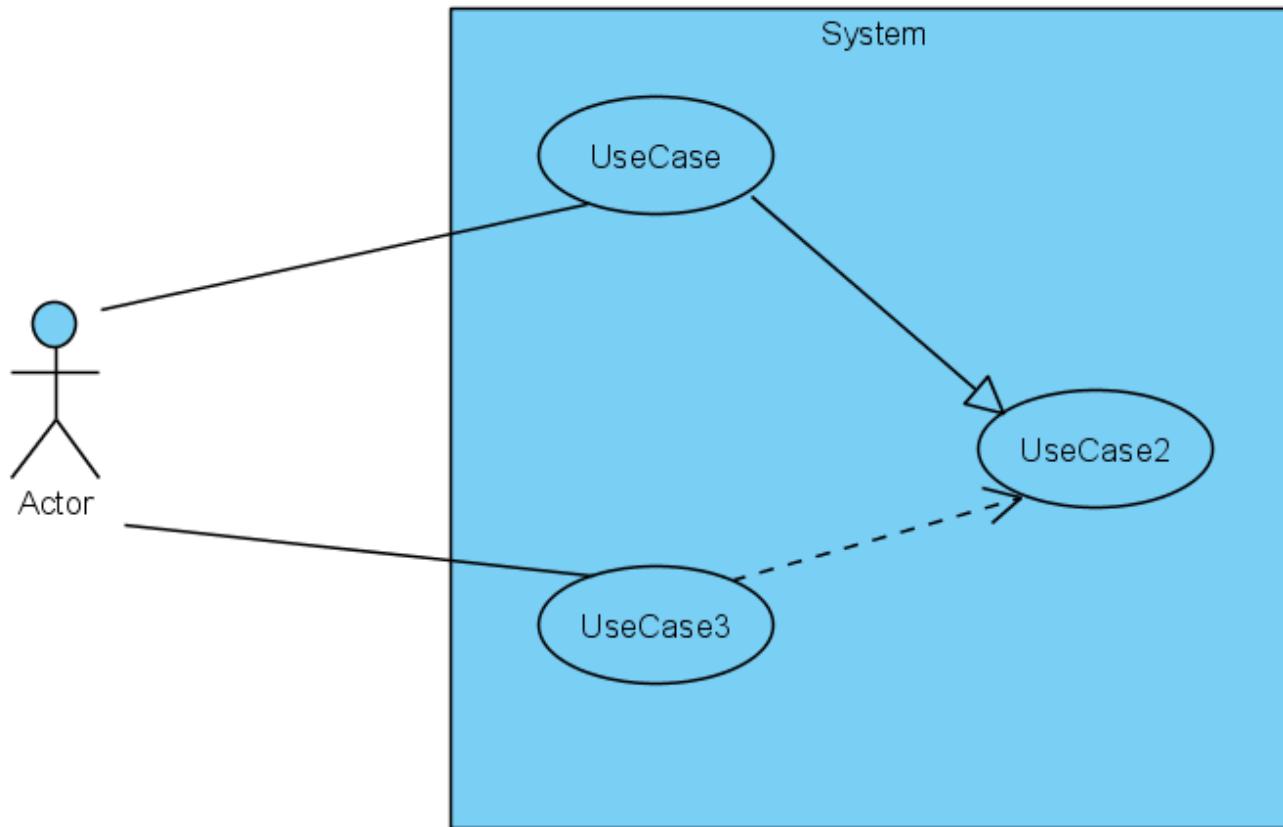
```
System simple : PipeFilter = {  
    Component Splitter : Filter = {  
        Port pIn : InputPort = new Inputport;  
        Port pOut1: OutputPort = new OutputPort;  
        Port pOut2: OutputPort = new OutputPort;  
    };
```

- UML 2.0 ist weit verbreitet und wird von vielen Tools unterstützt
- Semi-Formale Beschreibungssprache
- Entwicklung der Spezifikation durch die Object Management Group (OMG)
- Unterschiedliche Diagrammarten
- Nachteile
 - › Diagramme verführen zur Überladung:
z.B. Klassendiagramm geeignet für viele Stile
→Vermischung von Problemen
- Vorteile
 - › Verbreiteter Standard
 - » Leichtes gemeinsames Verständnis schaffen
 - » Gute Unterstützung in Tools
 - › An Anwendungsgebiete anpassbar
 - » Stereotype
 - » Constraints
 - » Tagged values

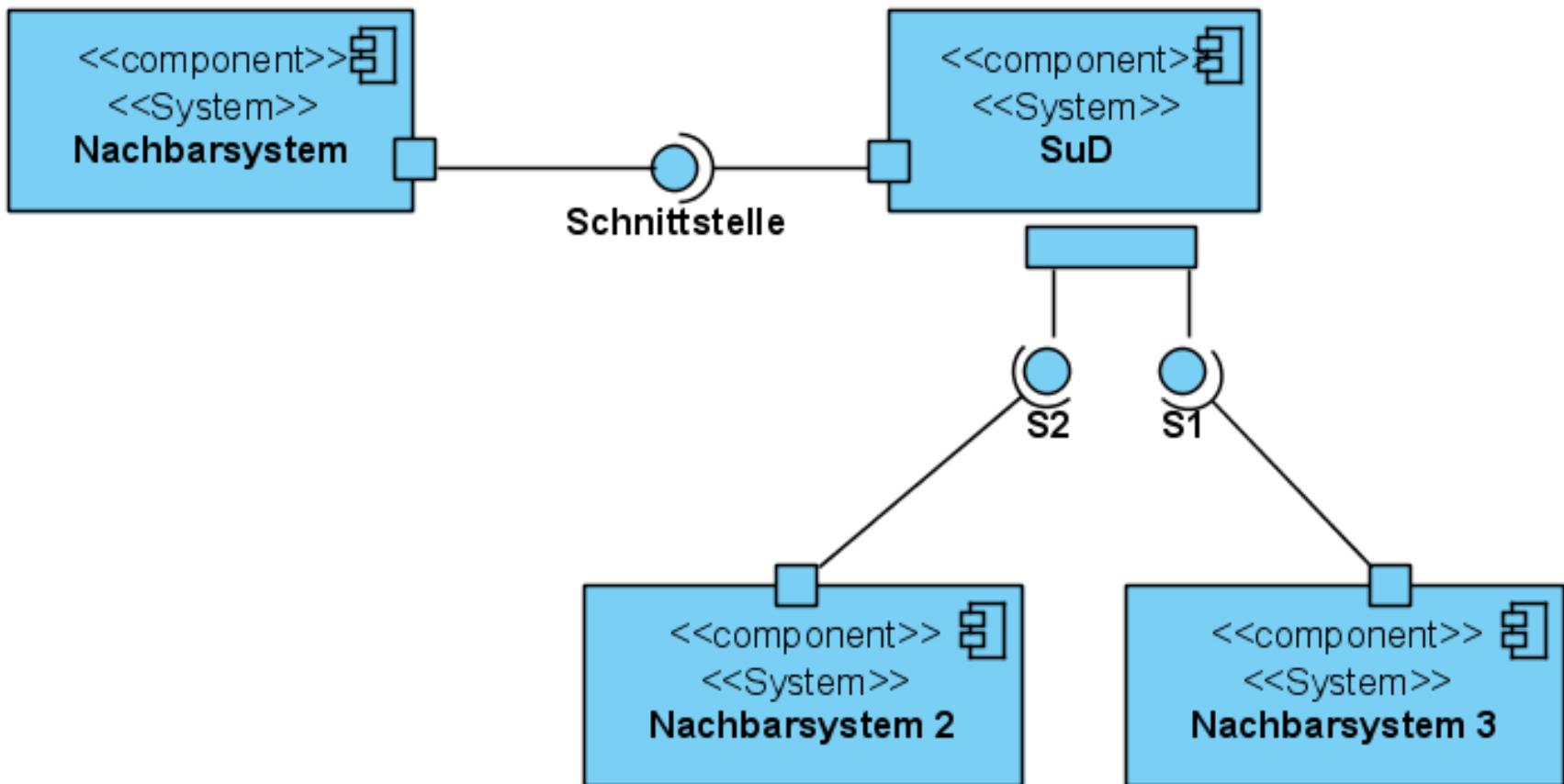
- UML 2.0 definiert Diagramme für die unterschiedlichsten Zwecke
- Strukturdiagramme
 - › Klassendiagramme
 - › Komponentendiagramme
 - › Interne Strukturdiagramme
 - › Verteilungsdiagramme
 - › Paketdiagramme
 - › Objektdiagramme
- Verhaltensdiagramme
 - › Interaktionsdiagramme
 - » Sequenzdiagramme
 - » Kommunikationsdiagramme
 - » Interaktionsübersichtsdiagramme
 - » Timingdiagramme
 - › Zustandsdiagramme
 - » Protokollzustandsdiagramme
 - » Verhaltensdiagramme
 - › Use-Case-Diagramme
 - › Aktivitätsdiagramme

Kontextsicht: Use-Case-Diagramme

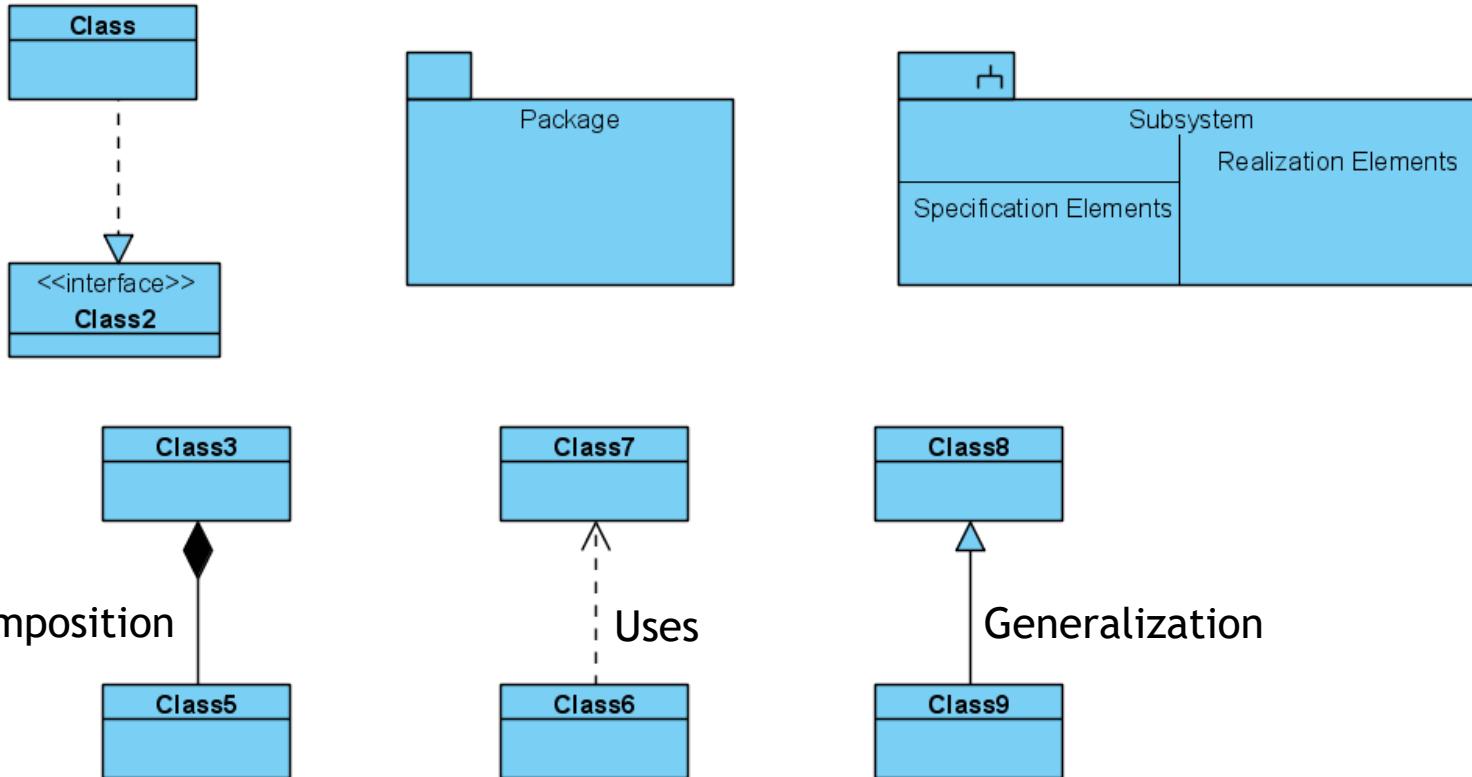
- Durch ein Komponentendiagramm lassen sich die Schnittstellen zu Nachbarsystemen verdeutlichen



- Durch ein Komponentendiagramm lassen sich die Schnittstellen zu Nachbarsystemen verdeutlichen

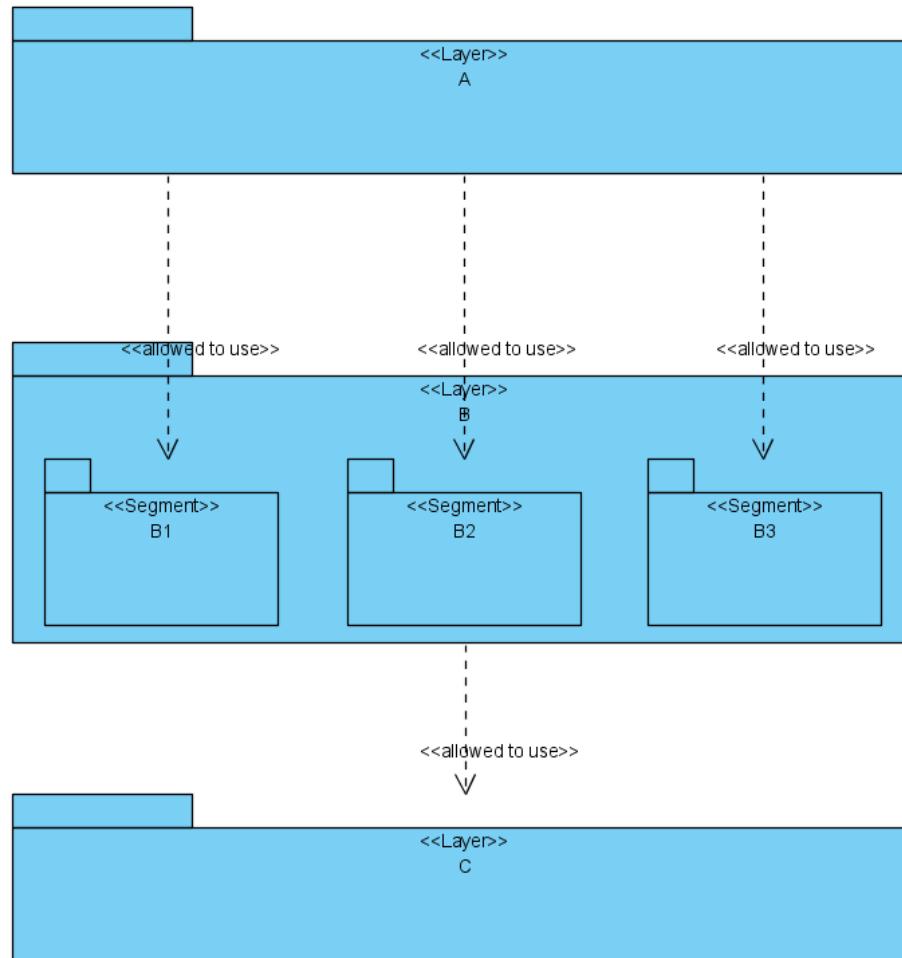


- Klassen- und Packagediagramme eignen sich für die Darstellung von Modulen



Modulsicht: Layers

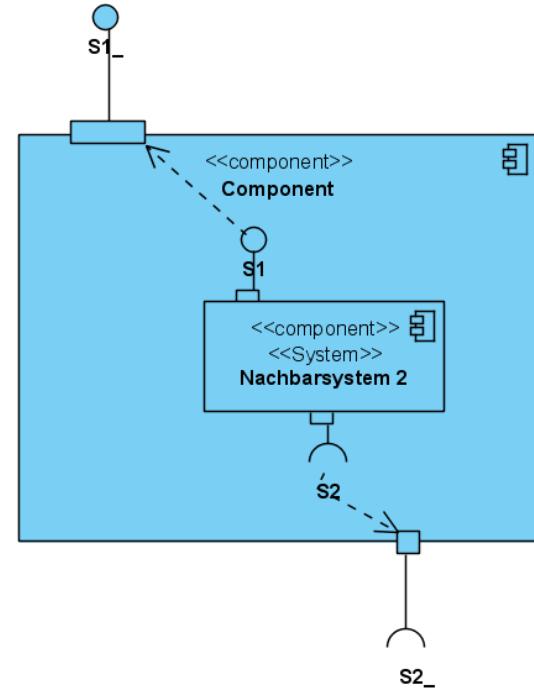
- Mittels Packages lassen sich Layer darstellen



Segmentierte Layer

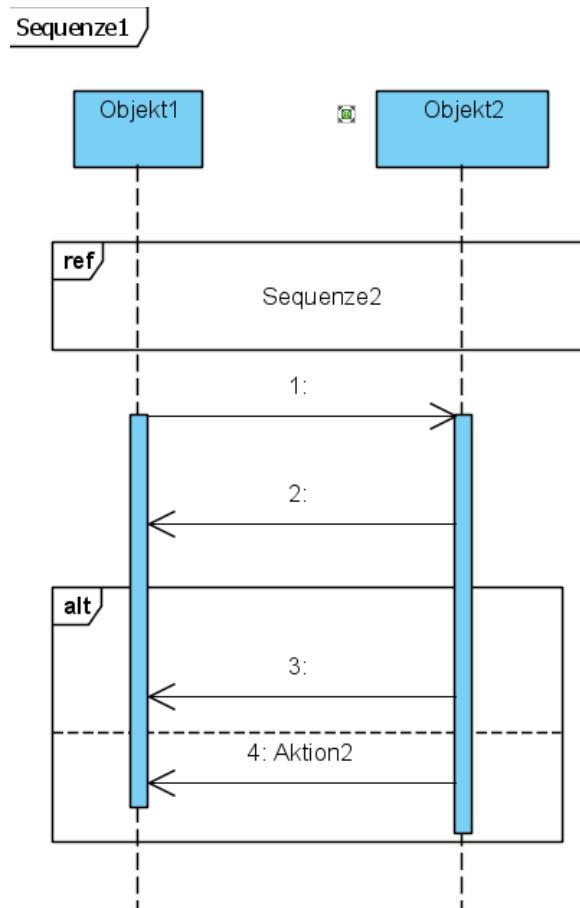
Component-and-Connector Viewtype

- Viele Diagramme sind mächtiger als für Views benötigt wird
- Komponentendiagramme erlauben es is-part-of-Beziehungen darzustellen
- Mittels Stereotypen lassen sich Elemente der Views darstellen
 - › <<Peer>>, <<invokes>>
 - › <<Pipe>>, <<Filter>>
- Für den Communicating Processes Stil eignen sich Interaktionsdiagramme
- Auch Klassendiagramme sind möglich

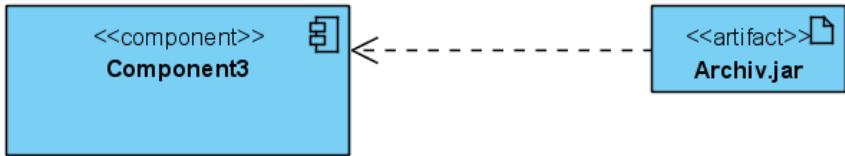


Component-and-Connector Viewtype

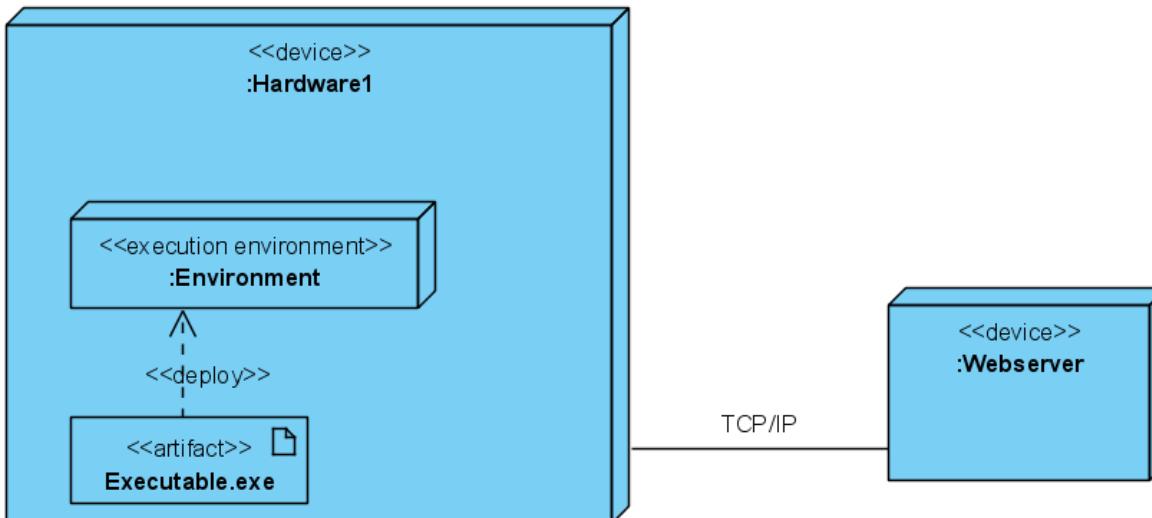
- Für den Communicating Processes Stil eignen sich Interaktionsdiagramme



- Implementation- und Deployment-Stil lassen sich mit UML-Diagrammen darstellen
- Implementation Stil



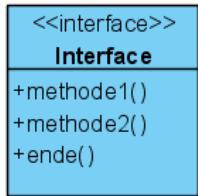
- Deployment Stil



- Man kann zwischen Benutzungsschnittstellen und Programmschnittstellen unterscheiden
- Unterscheidung:
 - › Benutzungsschnittstelle
 - » Benutzerhandbücher
 - » GUI
 - › Programmschnittstellen
- Folgende Informationen müssen berücksichtigt werden
 - › Syntax
 - › Semantik
 - › Protokoll
 - › Nichtfunktionale Eigenschaften

Beschreibung von Schnittstellen - Syntax

- Syntax kann mit verschiedenen Mitteln beschrieben werden
- UML



- Programmiersprache
 - › Meist mittels IDL (Interface Definition Language) der OMG
 - » Datentypen
 - » Operationen
 - » Attribute
 - » Exceptions
 - › Einsatz z.B. bei CORBA (Common Object Request Broker Architecture)
 - › Beispiel:

```

Interface Article{
    const string prefix= "A_";
    typedef unsigned long QuantityType;
    exception InvalidPrice{ double price; }
    readonly attribute QuantityType minStock;
    void setPrice(in double price)
        raises(InvalidPrice);
    double getPrice();
}
  
```

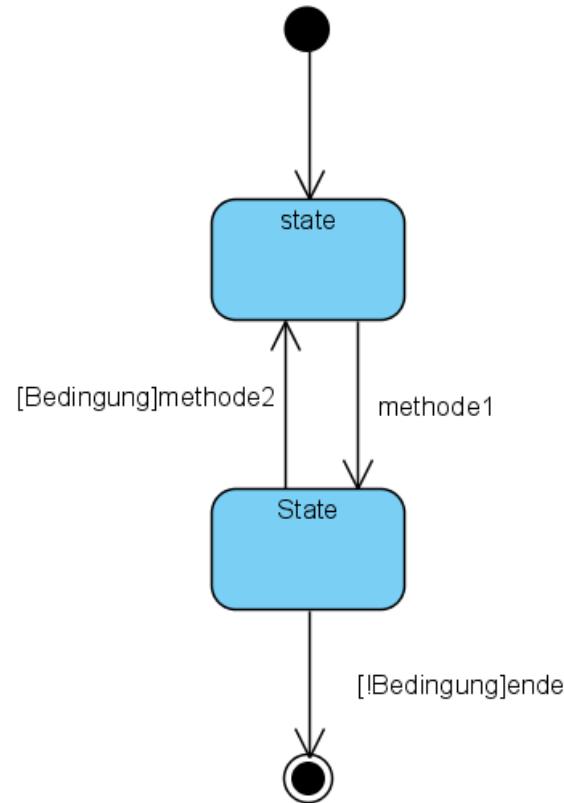
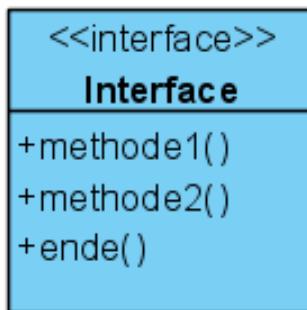
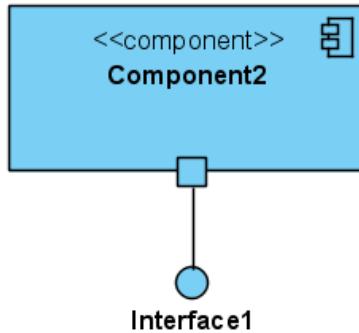
- Pseudocode

- Semantik kann mit verschiedenen Mitteln beschrieben werden
- Formal
 - › Pre- und Postconditions
 - › Input / Outputs
 - › Effects
- z.B. mit OCL
- Textuell

Method Summary

char	charAt (int index) Returns the char value at the specified index.
int	codePointAt (int index) Returns the character (Unicode code point) at the specified index.
int	codePointBefore (int index) Returns the character (Unicode code point) before the specified index.
int	codePointCount (int beginIndex, int endIndex) Returns the number of Unicode code points in the specified text range of this String.
int	compareTo (String anotherString)

- Auch mittels Diagrammen lässt sich die Semantik einer Schnittstelle beschreiben

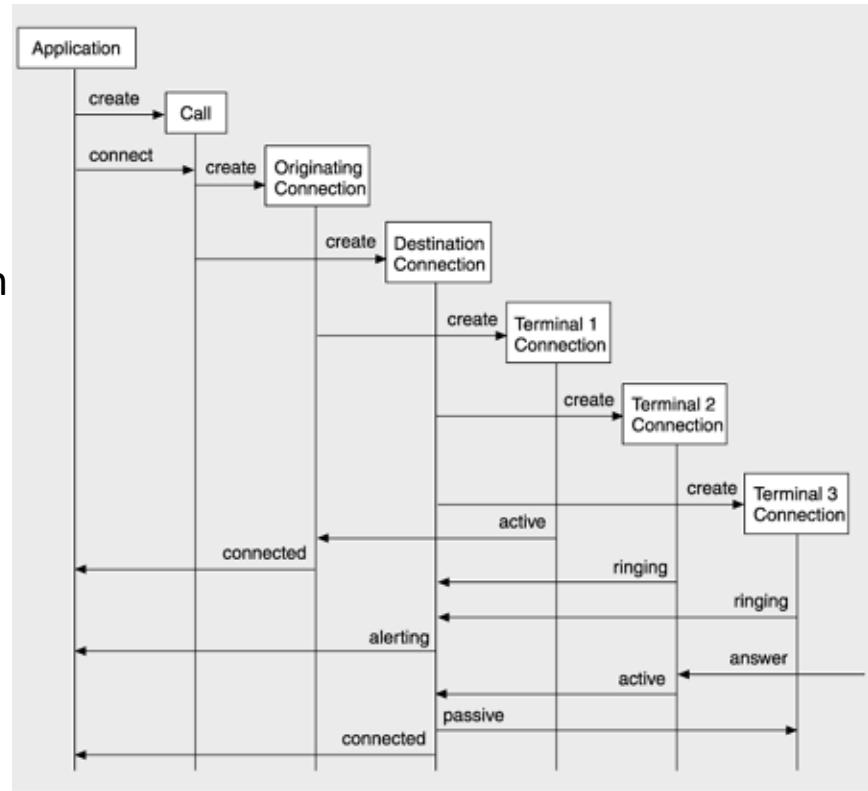


- Ergänzt strukturelle Beschreibung um Semantik
- Liefert Informationen zu
 - › Auslösern von Aktionen
 - › Informationsaustausch zwischen Elementen
 - › Zeitbezug
 - › Reihenfolge von Ereignissen (sofern wichtig für Ausführung)
 - › Evtl. Einschränkungen, wann Ausgaben überhaupt sinnvoll
- Erläutert Vorgänge in einem Element
 - › Gute Basis für spätere Tests / Simulationen:
Entspricht das implementierte Verhalten dem erwarteten (dokumentierten)?
 - › Ermöglicht analytische Betrachtung des Systems:
z.B. wie wahrscheinlich ist ein Deadlock?

Dokumentation von Verhalten: Dokumentationsarten

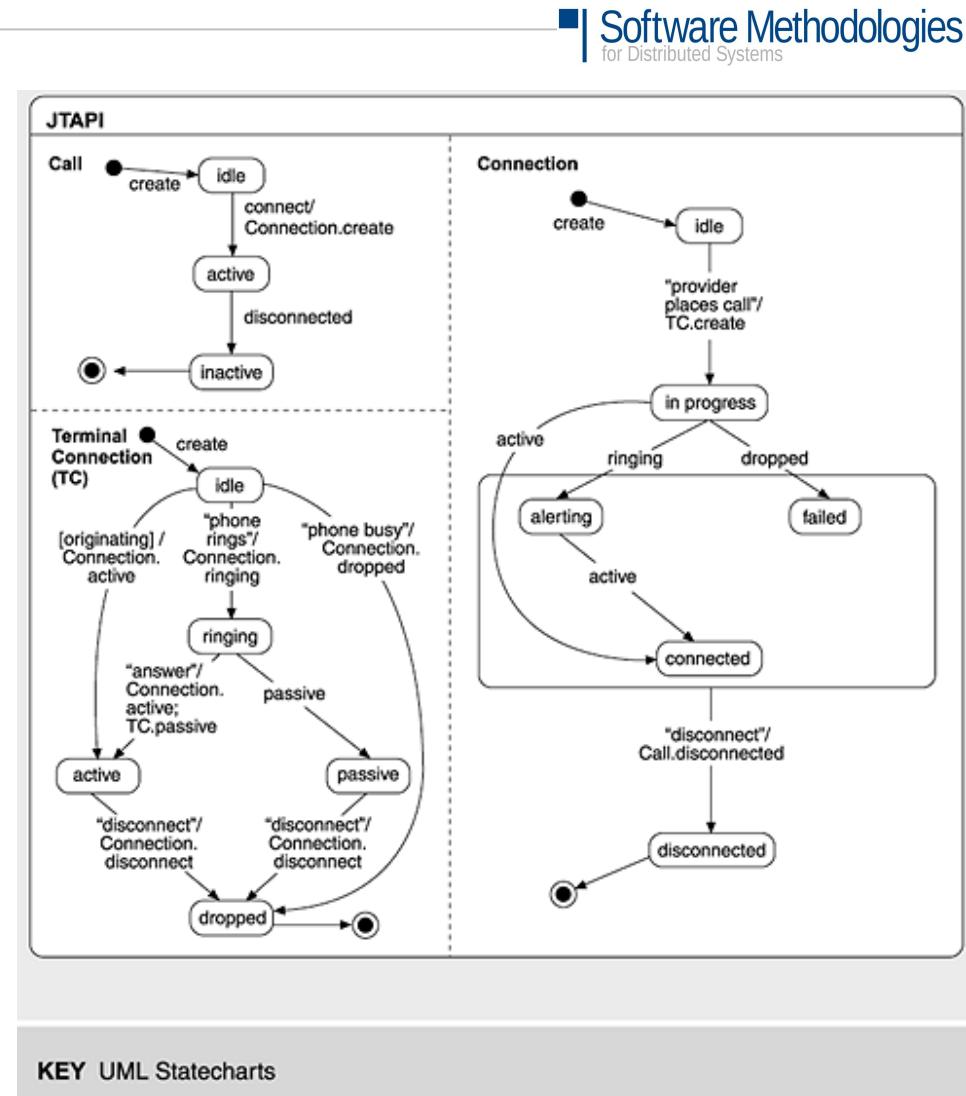
Traces	Statische Modelle
<ul style="list-style-type: none">▪ „Protokollierung“ des Durchlaufs eines Szenarios▪ Verhaltensausschnitt element-übergreifend dokumentiert▪ Nur am Szenario beteiligte Elemente und nur die verwendeten Ressourcen aus deren Schnittstellen▪ Nur vollständig bezüglich spezifischen Auslösers und Systemzustands▪ Kombination aller denkbaren Traces praktisch unmöglich → Kein vollständiges Systemmodell	<ul style="list-style-type: none">▪ Vollständiges Modell, das jedes denkbare Verhalten von einem Element erfasst▪ Liefert nur Verhalten für betrachtetes Element▪ Alle Ressourcen aller Schnittstellen des dokumentierten Elementes▪ Für alle Zustände vollständig, da meist zustandsbasierend: System = Endlicher Automat▪ Kombination aller statischen Modelle ergibt vollständiges Systemmodell

- Use Cases
 - › Interaktionen zwischen einzelnen Elementen und Akteuren der Umwelt in Textform
- Use Case Maps
 - › Grafische Darstellung verschiedener Use-Cases
- Message Sequence Charts
 - › Aus Bereich der TK-Schaltsysteme
 - › Für Interaktionen zwischen Systemen
- Sequenzdiagramme
 - › Geordnete Interaktionen
 - › Zeitlinie
 - › Bsp: Software-Telefon
- Kollaborationsdiagramme
 - › Geordnete Interaktionen
 - › Nummerierung der Abläufe



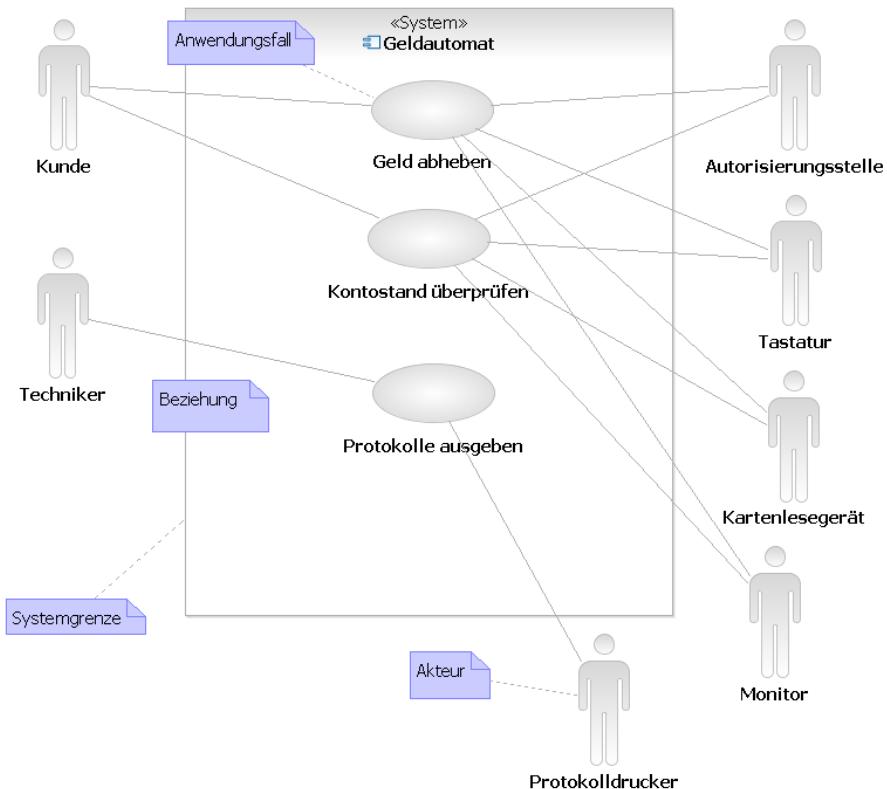
Dokumentation von Verhalten: Notationen für dynamische Modelle

- Zustandsdiagramme
 - › Bestandteil von UML
 - › Zeigen Systemverhalten für bestimmte Eingabe
 - › Verschiedene Pfade abhängig von Laufzeitwerten (Bsp.)
- SDL
(Specification and Description Language)
 - › Formale Sprache der ITU (International Telecommunications Union)
 - › Objektorientiert
 - › Hauptsächlich internes Verhalten
- Z
 - › Mathematische Sprache
 - › Basierend auf Prädikatenlogik und Mengenlehre
 - › Führt zu exakten Verhaltensmodellen
 - › Gute Basis z.B. für Model Checking



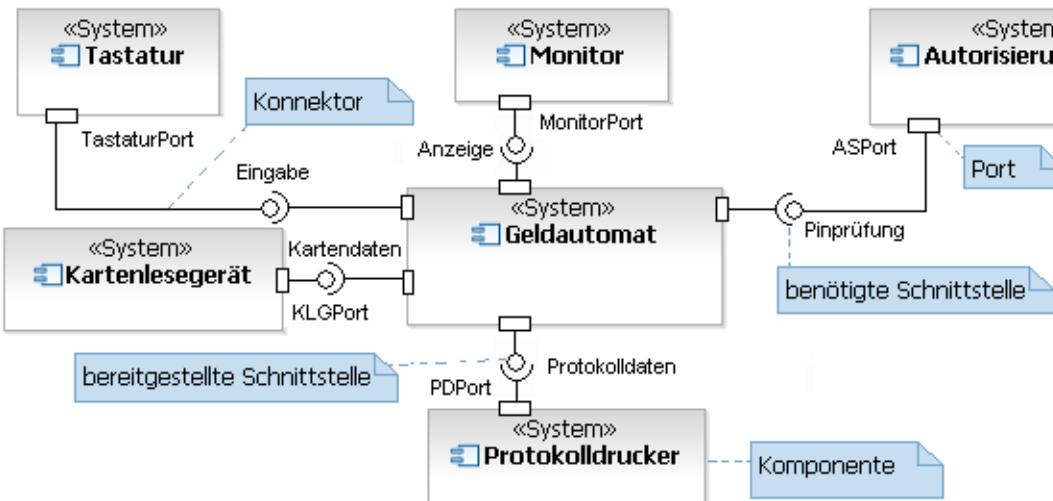
- Ziel
 - › Vorstellung der Sichten von Posch, Birken und Gerdom
 - › Vorstellung der Unified Modeling Language 2 als Notationsmöglichkeit für diese Sichten
- Fallstudie: Geldautomat
 - › Für die Softwaresteuerung eines Geldautomaten wurde eine Architektur entworfen
 - › Sichten auf diese Architektur sollen nun mit UML 2 dokumentiert werden
 - › Geldautomat
 - » Verfügt über Tastatur, Monitor, Kartenlesegerät
 - » Kunde kann Geld abheben und Kontostand überprüfen
 - » Techniker kann Protokolle ausdrucken
- Sichten von Posch, Birken und Gerdom
 - › Kontextsicht
 - › Struktursicht
 - › Verhaltenssicht
 - › Abbildungssicht

- Kontextsicht zeigt System als Blackbox im Kontext seiner Nachbarsysteme
- Zwei Arten von Diagrammtypen für Kontextsicht
 - › Für die Anforderungsanalyse
 - » Use-Case-Diagramme
 - » Aktivitätsdiagramme
 - › Für das Architekturdesign
 - » Komponentendiagramme
- Use-Case-Diagramm
 - › Anwendungsfälle
 - › Systemgrenzen
 - › Akteure (Person, Organisation, anderes System)

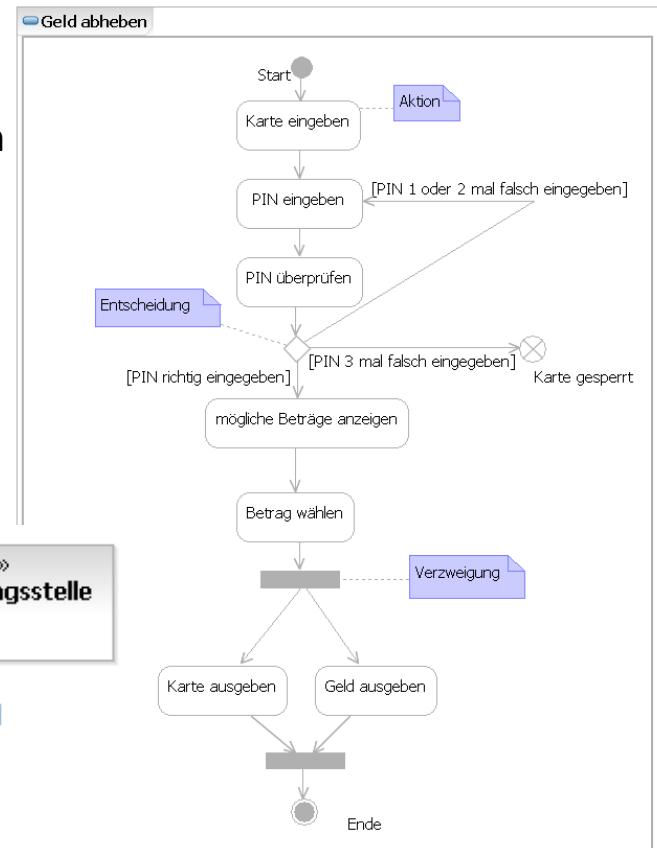


Fallstudie – Kontextsicht

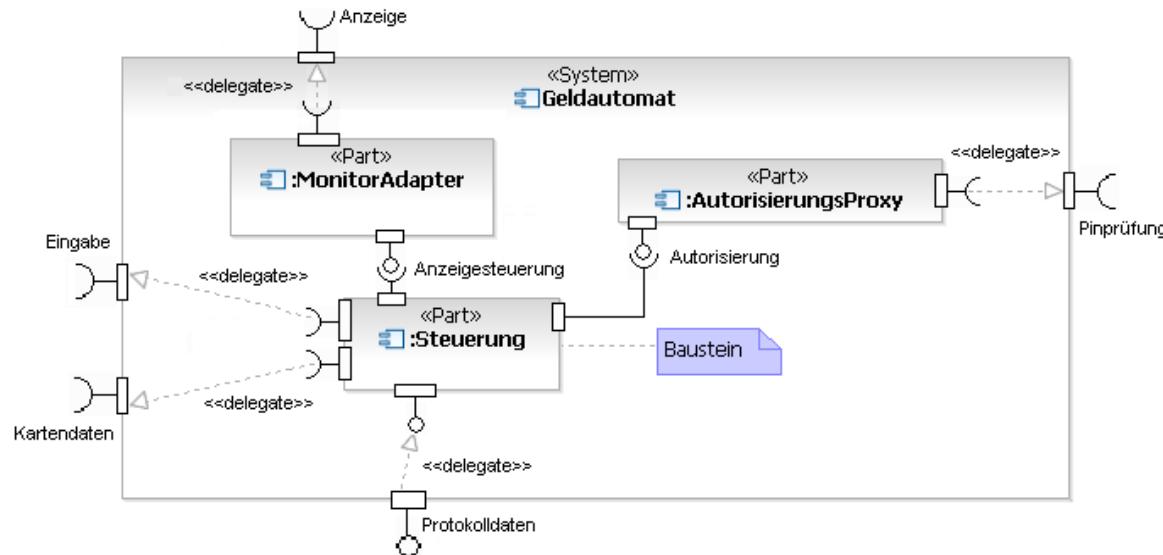
- Aktivitätsdiagramm
 - › Beschreibt Anwendungsfall aus Use-Case-Diagramm
- Komponentendiagramm
 - › System mit Schnittstellen und Kommunikationsmöglichkeiten zu Nachbarsystemen



Software Methodologies

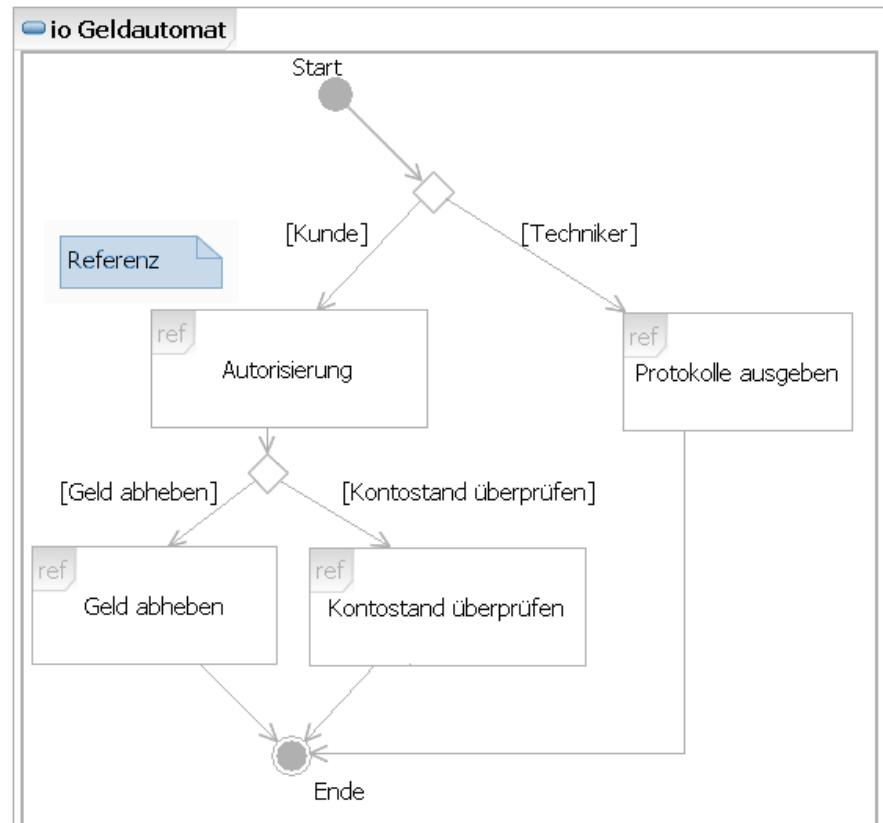


- Struktursicht
 - › Zeigt statische Sicht auf Innenleben des Systems
 - › Zeigt Bausteine und deren Kommunikationsbeziehungen
 - › Zeigt nur Kommunikationsmöglichkeiten, nicht tatsächliche Wechselwirkungen
 - › Kann bei Bedarf weiter hierarchisch zerlegt werden
- Internes Strukturdiagramm
 - › <<delegate>> ordnet externen Schnittstellen / Ports interne zu



Fallstudie – Verhaltenssicht

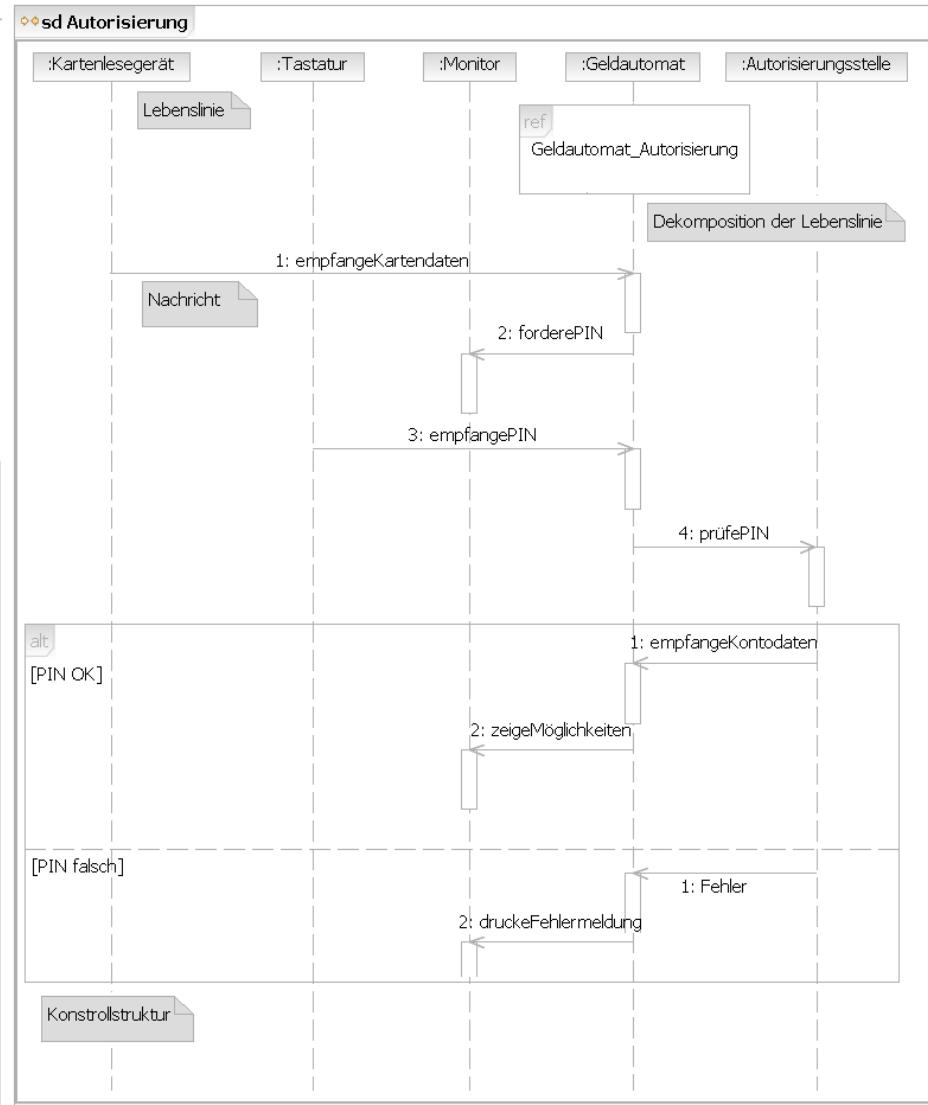
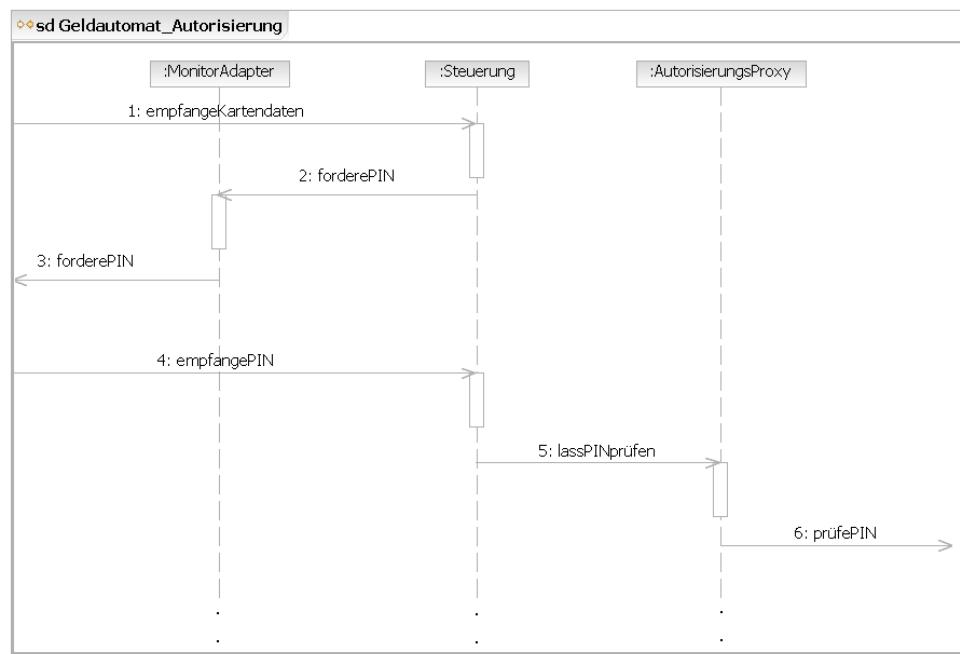
- Verhaltenssicht beschreibt dynamische Aspekte des Systems
 - › Kommunikationsbeziehungen zwischen Bausteinen
 - › Zustände und Zustandsveränderungen von Bausteinen
- Zwei Arten von Diagrammtypen für Verhaltenssicht
 - › Interaktionsdiagramme
 - » Interaktionsübersichtsdiagramme
 - » Sequenzdiagramme
 - › Zustandsdiagramme
 - » Protokollzustandsdiagramme
 - » Verhaltenszustandsdiagramme
- Interaktionsübersichtsdiagramm
 - › Zeigt mögliche Interaktionen anderer Akteure mit dem System
 - › Setzt Interaktionen zueinander in Beziehung
 - › Enthält Verweise auf Sequenzdiagramme



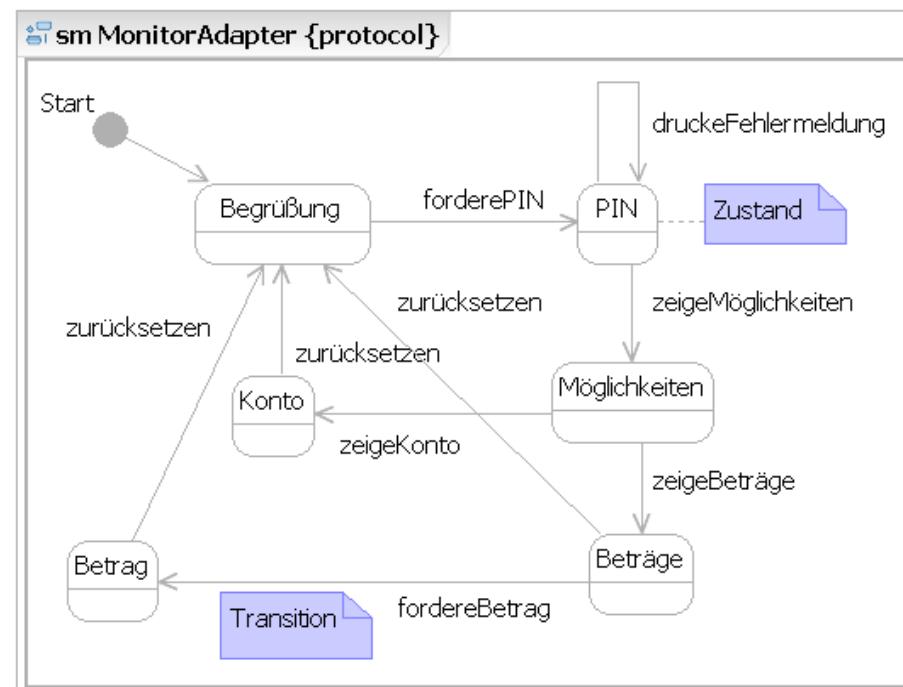
Fallstudie – Verhaltenssicht

Sequenzdiagramm

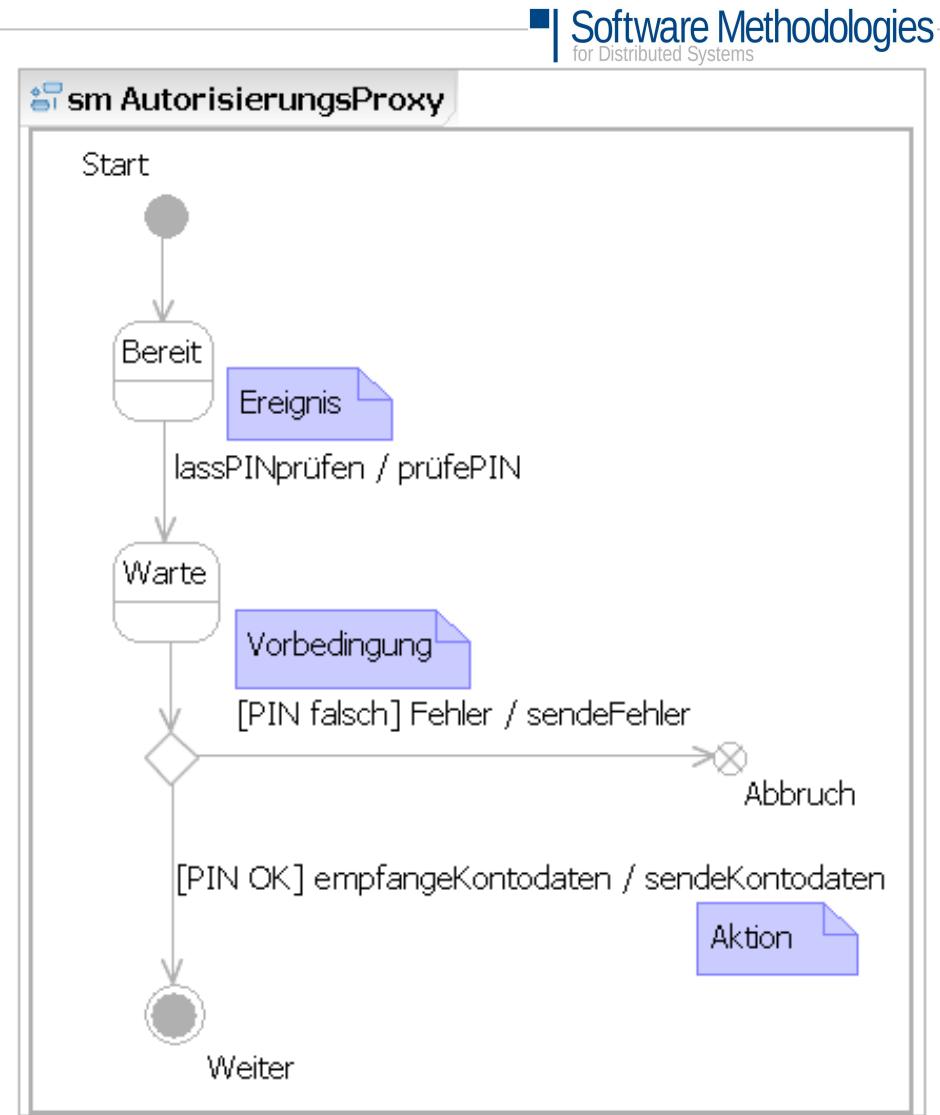
- Zeigt Interaktionen zwischen Komponenten oder Bausteinen entlang eines zeitlichen Verlaufs
- Kann weiter untergliedert werden (Dekomposition)



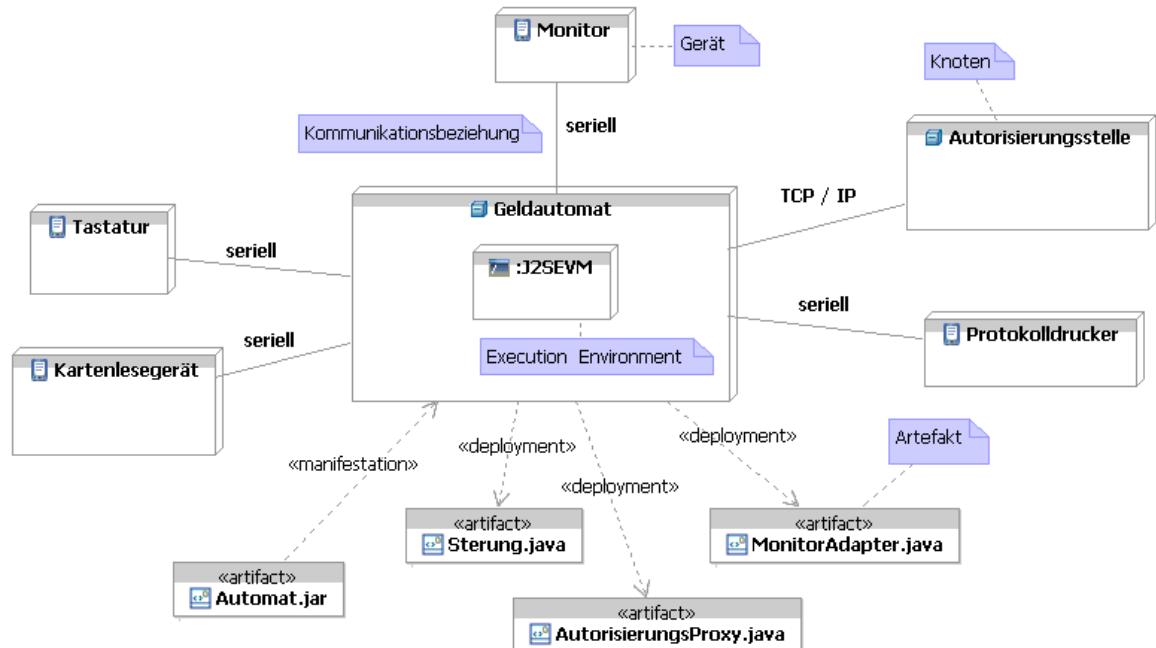
- Protokollzustandsdiagramm
 - › Ergänzt statische Schnittstelleninformation eines Bausteins um dynamische Elemente
 - » Baustein kann diskrete Zustände annehmen
 - » Operationen lassen Baustein in anderen Zustand wechseln



- Verhaltenszustandsdiagramm
 - › Ähnelt Protokollzustandsdiagramm
 - › Zeigt isoliertes Verhalten eines Bausteins zur Laufzeit
 - › Transitionen können hier auch mit Vorbedingungen und Aktionen beschriftet sein



- Abbildungssicht beschreibt technische und organisatorische Aspekte des Systems
- Verteilungsdiagramm
 - › Ordnet Komponenten und Bausteinen physikalische Objekte (Artefakte) zu
 - › Beschreibt Art der Systeme
 - › Beschreibt Kommunikationsbeziehungen



- [PBG04] T. Posch, K. Birken, M. Gerdom. Basiswissen Sofwarearchitektur: Verstehen, entwerfen und dokumentieren. Dpunkt.verlag, Heidelberg 2004.
- [BMR+98] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerland, M. Stal. Pattern-orientierte Software-Architektur. Addison-Wesley, Bonn, Paris [u.a.] 1998.
- [CBB+05] P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, R. Nord, J. Stafford. Documenting Software Architectures: Views and Beyond. Addison-Wesley, Boston, San Francisco [u.a.] 2005.

- Einführung in Software-Architekturen und Organisation
- Entwurf von SW-Architekturen
- Dokumentation von Software-Architekturen
- **Evaluation / Bewertung von SW-Architekturen**
- Toolbox des Softwarearchitekten
- Produktlinien für Software
- Enterprise Architecture Management

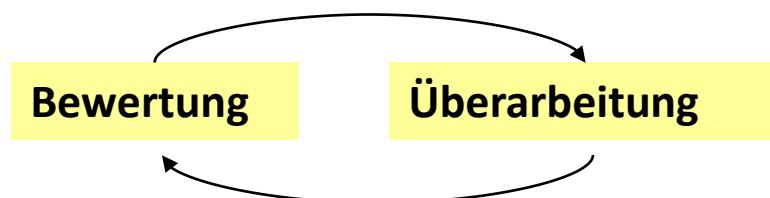
- Problemstellung und Motivation
- Grundlagen
 - › Allgemeines Vorgehen
 - › Arten von Bewertungen
- Bewertungsmethoden
 - › Qualitative Methoden
 - › Quantitative Methoden
- Szenariobasierte Ansätze
 - › ATAM
 - › SAAM
 - › Vergleich
- Kosten und Nutzen
- Bewertungsbeispiel

- **Problemstellung und Motivation**
- **Grundlagen**
 - › Allgemeines Vorgehen
 - › Arten von Bewertungen
- **Bewertungsmethoden**
 - › Qualitative Methoden
 - › Quantitative Methoden
- **Szenariobasierte Ansätze**
 - › ATAM
 - › SAAM
 - › Vergleich
- **Kosten und Nutzen**
- **Bewertungsbeispiel**

- Software-Architektur als „Bauplan“ eines Systems beeinflusst spätere Funktionalität und Kosten
 - › Je später Fehler erkannt werden, desto teurer
- Deshalb Bewertung der entworfenen Architektur um:
 - › Sicher zu stellen, dass die richtige Architektur verwendet wird
 - › Qualitätsmerkmale von Beginn an zu berücksichtigen
 - › Anpassungserfordernisse bereits vor Umsetzung zu erkennen
- Problem:
 - › System liegt noch nicht vor → Messung von Eigenschaften?
- Deshalb:
 - › Abschätzung des Potenzials der Architektur zur Erfüllung der geforderten Eigenschaften

- Problemstellung und Motivation
- **Grundlagen**
 - › Allgemeines Vorgehen
 - › Arten von Bewertungen
- Bewertungsmethoden
 - › Qualitative Methoden
 - › Quantitative Methoden
- Szenariobasierte Ansätze
 - › ATAM
 - › SAAM
 - › Vergleich
- Kosten und Nutzen
- Bewertungsbeispiel

- Allgemeines Vorgehen:
 - › Bewertungskriterien festlegen
wesentliche, bei der Spezifikation der Einflussfaktoren gefundene Risiken
 - › Messbarkeit der Kriterien sicherstellen
Einflussfaktoren sauber und messbar definiert?
 - › Bewertungsarten und Methoden festlegen
 - › Durchführen der Bewertung
 - › Ergebnisse zusammenfassen
 - » Hinweise, wo sich in der Architektur Problembereiche befinden
 - » Nachweis, ob Architektur allen Anforderungen entspricht
 - › Maßnahmen ergreifen



- Am besten iterativ, soweit Kosten es zulassen.

■ Umfangreiches, szenariobasiertes Assessment

› **Zeitpunkt:**

- » Früh genug, so dass getroffene Entscheidungen ohne allzu großen Aufwand rückgängig gemacht werden können.
- » Spät genug, so dass wesentliche Entscheidungen bereits getroffen wurden, um die Berücksichtigung der Einflussfaktoren in der SW-A bewerten zu können.
- » Aber auf alle Fälle bevor Entwicklungsteams beginnen Entscheidungen zu treffen, die auf die SA aufbauen

› **Teilnehmer:**

- » Projektmanager,
- » Architekturteam,
- » alle wichtigen Stakeholder,
- » projektexternes Bewertungsteam

› **Aufwand:** hoch

› **Zeitpunkt:**

- » Beginn des letzten Drittels der Architekturentwicklungsphase

› **Ziel:**

- » Sicherstellung der Berücksichtigung aller Einflussfaktoren

› **Beispiele:** ATAM; SAAM

■ Discovery Review

- › Ergänzend zum umfangreichen Assessment
- › Entwurf der SW-A noch nicht weit fortgeschritten:
- › Wesentliche Entwurfsentscheidungen wie die Auswahl von Architekturstilen bereits festgelegt aber noch nicht in Stein gemeißelt.
- › Vermeidung das falsche Richtung eingeschlagen wird
- › **Teilnehmer:**
 - » Projektmanager,
 - » Vertreter des Architekturteams,
 - » evtl. ausgewählte Stakeholder
- › **Aufwand:** mittel
- › **Zeitpunkt:** Ende des ersten Drittels der Architekturentwicklungsphase
- › **Ziel:** Bewertung der Tragfähigkeit der Grundlagenentscheidungen
- › **Beispiel:** szenariobasierte „Minibewertung“

■ Gezielte Überprüfungen

- › **Umfang:**
 - » aufwändig, wenn Prototypen oder Simulationsumgebungen eingesetzt werden
- › **Zeitpunkt:**
 - » genau dann, wenn man bei der Spezifikation der Einflussfaktoren die Risiken identifiziert hat und entscheidet, welche Maßnahmen man durchführt
 - » gezielt im Laufe der Architekturentwicklungsphase
 - » Beispiel: System muss hohe Anzahl an gleichzeitigen Zugriffen verkraften → Test mit Prototypen
- › **Teilnehmer:**
 - » Projektmanager,
 - » Vertreter des Architekturteams,
 - » ausgewählte Stakeholder
- › **Aufwand:** z.T. sehr hoch
- › **Ziel:**
 - » Absicherung einzelner, definierter Risiken
- › **Beispiele:**
 - » Prototypen; Simulationen;

■ Ad-hoc-Bewertungen

- › **Teilnehmer:**
 - » Softwarearchitekt
 - » evtl. Teil oder ganzes Architekturteam
- › **Aufwand:** niedrig
- › **Zeitpunkt:** spontan, im Laufe der Entwicklung
- › **Ziel:**
 - » Überprüfung einzelner Entscheidungen
 - » Ergebnisse sollten auf jeden Fall in einem kurzen Protokoll festgehalten werden.
- › **Beispiele:**
 - » schlanke, szenariobasierte Bewertung;
 - » kleinere Prototypen;
 - » mathematische Berechnungen;
 - » logisches Schlussfolgern

- Problemstellung und Motivation
- Grundlagen
 - › Allgemeines Vorgehen
 - › Arten von Bewertungen
- **Bewertungsmethoden**
 - › Qualitative Methoden
 - › Quantitative Methoden
- Szenariobasierte Ansätze
 - › ATAM
 - › SAAM
 - › Vergleich
- Kosten und Nutzen
- Bewertungsbeispiel

▪ Qualitative Methoden

- › Szenariobasierte Bewertung
 - › Fragebögen
 - › Checklisten
 - › Auf Erfahrung basierende Argumentation
-
- › Gedankenexperiment zur Vorhersage des Verhaltens eines Systems
 - › Zur Bewertung jeglicher Art von Faktoren geeignet
- 
- Fragetechniken

- **Szenariobasierte Bewertung**
- Definition (Szenario):
 - › Ein Szenario ist eine kurze Beschreibung eines Anwendungsfalls. D.h. kann ein gewünschter Anwendungsfall oder aber auch der Auftritt eines Fehlers sein

■ Szenariobasierte Bewertung

- › **Ausgangspunkt:** Bewertung komplexer Qualitätsmerkmale
- › Abbildung der **Qualitätsanforderungen** als Szenarien.
- › Untersuchung der Auswirkungen die sich bei Eintritt eines oder mehrerer Szenarien auf die Architektur ergeben.
- › Szenarien werden im Rahmen der Entwicklung erarbeitet
- › **Qualität** der Ergebnisse hängt von Qualität der gewählten Szenarien ab
- › Wie gut eine Architektur ein bestimmtes Qualitätsmerkmal erfüllt erkennt man meist **nur im Kontext**
- › **Weiteres Bsp.:** Änderung der GUI, falls sich herausstellt dass dadurch sehr viele Komponenten betroffen sind, solle man dieses Designmerkmal besser kapseln

→ erlaubt Bewertung komplexer Qualitätsmerkmale

- › z.B. Wartbarkeit, Modifizierbarkeit, Robustheit, Flexibilität, etc.
- › **Beispiel:**
 - » **Faktor:** Portierbarkeit
 - » **Szenario:** Portierung des Systems von Windows XP auf Linux
 - » **Ergebnis:** Identifizierung der betroffenen Architekturkomponenten bei Eintritt des Szenarios

■ Fragebögen

- › Existieren bereits vor Beginn des Projekts
- › Entstehen aus längerer Erfahrung
- › Allgemeine, relativ offen formulierte Fragen
- › Überprüfung formaler Kriterien einer Architektur

→ Frühzeitige Adressierung immer wieder auftauchender Probleme

- › Beispiel:
 - » „Wurde die Architektur anhand mehrerer Sichten dokumentiert?“
 - » „Ist eine Verteilungssicht vorhanden?“

■ Checklisten

- › Existieren bereits vor Beginn des Projekts
- › Entstehen aus Erfahrung bei der Entwicklung von gleichartigen Systemen
- › Fragen detaillierter als bei Fragebögen

→ Überprüfung spezifischer, typischer Probleme aus dem Anwendungsbereich des Systems

- › Beispiel:
- › „Erlaubt die Architektur eine einfache Umstellung auf eine andere Sprache?“

■ Auf Erfahrung basierende Argumentation

- › Bewertung auf Basis subjektiver Einschätzungen
- › Grundlage ist das Wissen und die Erfahrung des Architekten, externer Berater oder anderer Projektmitglieder
- › Impliziter Prozess, kaum erkennbar und z.T. schwer nachvollziehbar
- › **Problem:**
 - » Ergebnisse schwer nachvollziehbar
 - » Damit andere Beteiligte teilhaben können ist eine logische Argumentation, Kommunikation und Dokumentation notwendig
- › **Herausforderung:**
 - » Entscheidung anhand objektiver und logischer Argumentation
 - » Kommunikation
 - » Dokumentation

■ Quantitative Methoden

- › Metriken
 - › Rate Monotonic Analysis (RMA)
 - › Prototypen/Simulationen
-
- Ausgereifter als Fragetechniken
 - Konzentration auf einzelne Qualitätsmerkmale
-
- › Aber:
 - » Weniger flexibel
 - » Benötigen ausführliche Design- bzw. Implementierungsartefakte

■ Metriken

- › Zahlenmäßige Interpretation bestimmter Eigenschaften
- › Liefern „handfeste“ Ergebnisse
- › Allgemein anwendbar oder auch domänen-spezifisch
- › Anwendung, wenn die SW-A bereits zu einem gewissen Grad ausgearbeitet

→ Überprüfung einzelner Qualitätsmerkmale

- › Beispiel:
 - » LOC oder Anzahl Generalisierungen für die Komplexität

- **Rate Monotonic Analysis (RMA)**

- › Bekannte Bewertungsmethode
- › Ist eigentlich auch eine Metrik
- › Konkretes mathematisches Modell
- › Benötigt keine Implementierung
- › Unkomplizierte Anwendung

→ Planung der Ausführungszeiten von Prozessen auf einer CPU

- Prototypen/Simulationen

- › Überprüfung, ob Entwurfsentscheidungen mit der eingesetzten Technologie umgesetzt werden können
- › Unter Umständen sehr zeit- und kostenaufwändig
- › Idealerweise Teil des Entwicklungsprozesses
- › Wegen des erheblichen Aufwands sollten Prototypen und Simulationen nicht nur zur Bewertung der SW-A verwendet werden, sondern Teil des Entwicklungsprozesses sein

→ Klärung einzelner Aspekte der Architektur

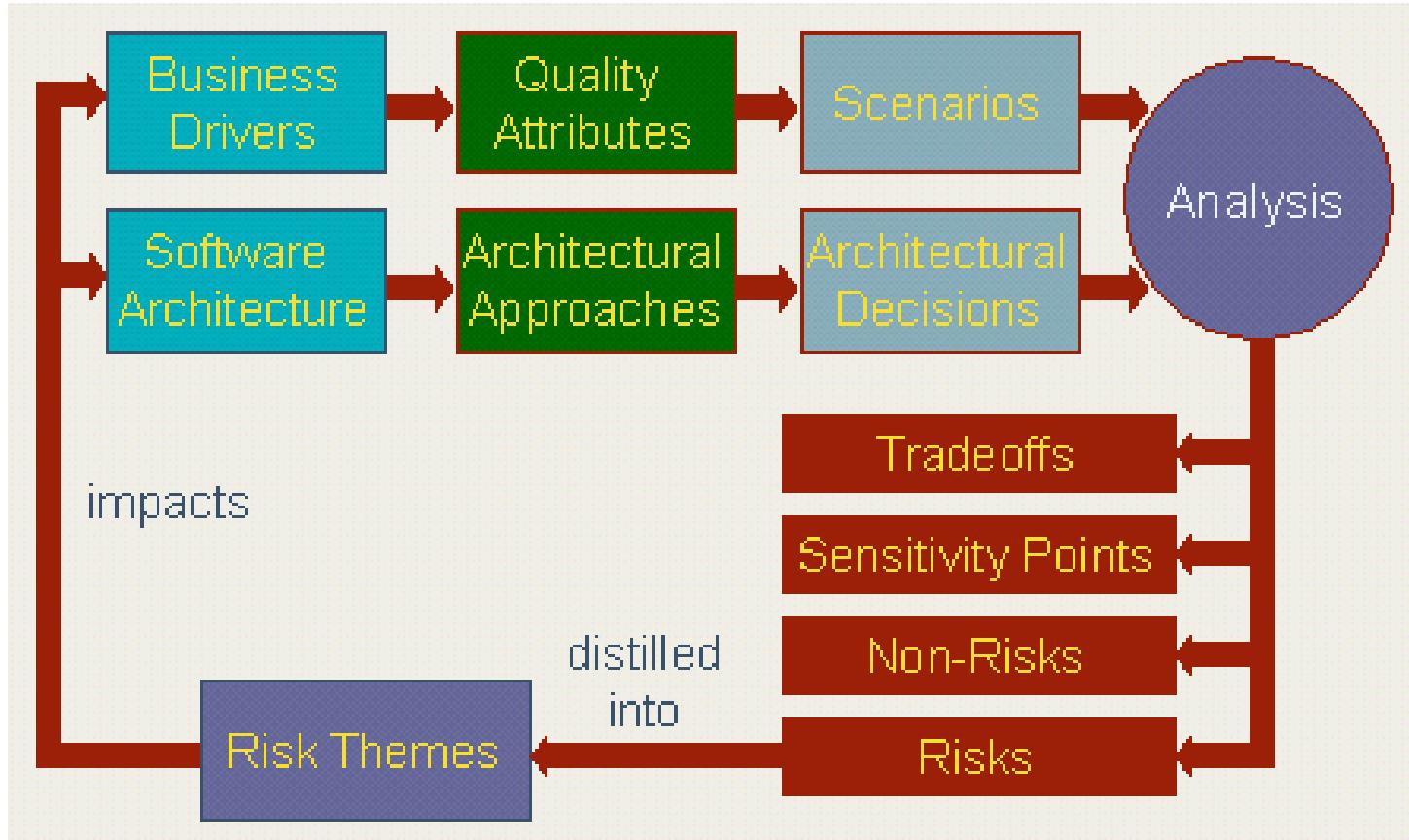
- › Beispiele:
 - » Messung von Rechenzeiten
 - » Test der Belastbarkeit bei hohen Zugriffsralten

- Problemstellung und Motivation
- Grundlagen
 - › Allgemeines Vorgehen
 - › Arten von Bewertungen
- Bewertungsmethoden
 - › Qualitative Methoden
 - › Quantitative Methoden
- **Szenariobasierte Ansätze**
 - › ATAM
 - › SAAM
 - › ARID
 - › CBAM
 - › Vergleich
- Kosten und Nutzen
- Bewertungsbeispiel

- Ziel wie bei allen Bewertungsmethoden: Erfüllt die Architektur bestimmte Faktoren, z.B. Qualitätsmerkmale
- Qualitätsmerkmale sind schwer zu messen, bzw. in Zahlen auszudrücken, deshalb Operationalisierung durch Szenarios, Qualität einer Software erkennt man oft nur im Kontext
- Was sind die Vorteile?
 - › Für nahezu jegliche Art von Projekt geeignet
 - › Gutes Kosten Nutzenverhältnis
 - › Kann in abgespeckter Version auch in andere Bewertungsarten einfließen (z.B. Discovery Review)
 - › Gründliche Strukturierung der Ansätze ermöglicht Wiederholbarkeit der Bewertung

- **ATAM - Architecture Tradeoff Analysis Method**
 - › Nachfolger der SAAM (Software Architecture Analysis Method)
 - › Sinnvoll für die Evaluation von sich in der Entwicklung oder bereits im Einsatz befindlichen Systemen
 - › Analysiert die Zusammenhänge zwischen den aus den Business Driver hervorgehenden Quality Attributes und architekturellen Entscheidungen.
Zeigt, ob ein System diese Quality Attributes erfüllt
 - » Quality Attribute: Eigenschaft eines Systems, dass dessen Qualität angibt (z.B. Leistungsfähigkeit, Sicherheit etc.)
 - » Architekturelle Entscheidung: Entscheidung, wie etwas in einem System umgesetzt wird
 - » Business Driver: Definiert das Ziel / den Sinn eines Systems
 - › Zeigt die Kompromisse (Tradeoff) auf, die eine architekturelle Entscheidung mit sich bringt
 - › Analysiert nicht die funktionale Korrektheit eines Systems!

- Konzeptueller Fluss in der ATAM



- Überblick
 - › Vorbereitung endet mit Kickoffmeeting
 - › Eigentliche Bewertung in Phase 1 & 2 in Form von zwei Workshops

Phase 0: Vorbereitung	<ul style="list-style-type: none">✓ Bewertungsteam aufsetzen✓ Zusammenarbeit zwischen Projekt- und Bewertungsteam abstimmen✓ Zeitplanung Organisation✓ Kick-Off-Meeting
Phase 1: Bewertung – Architektur zentriert	<ul style="list-style-type: none">✓ Schritt 1-3: ATAM, Geschäftsziele und Architektur präsentieren✓ Schritt 4: Architekturansätze identifizieren✓ Schritt 5: Utility Tree erstellen✓ Schritt 6: Architekturansätze analysieren (1)
Phase 2: Bewertung – Stakeholder zentriert	<ul style="list-style-type: none">✓ Schritt 7: Szenario-Brainstorming✓ Schritt 8: Architekturansätze analysieren (2)✓ Schritt 9: Ergebnisse präsentieren
Phase 3: Nachbearbeitung	<ul style="list-style-type: none">✓ Abschlussbericht✓ Verbesserung des Bewertungsprozesses✓ Artefakte archivieren

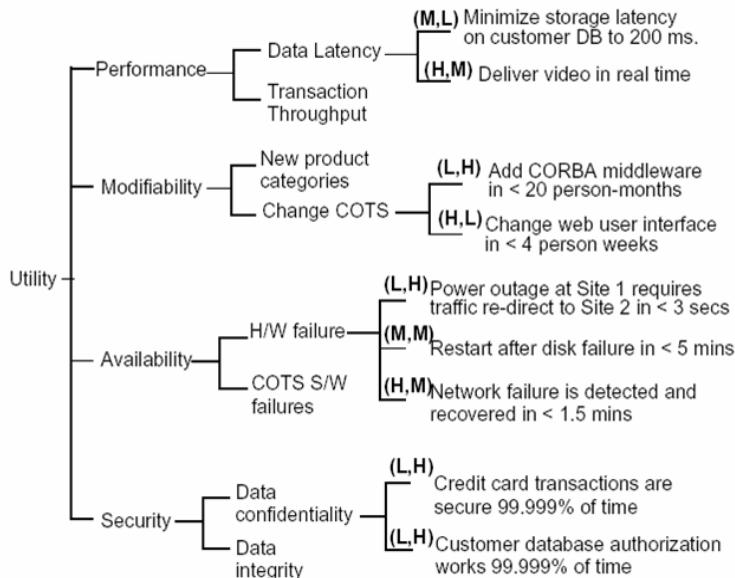
- Inhalt von Phase 0
 - › Aufbau der Partnerschaft
 - » Erste Treffen
 - » Auftraggeber sollte selbst eine gewisse Kenntnis der ATAM besitzen
 - › Analyse des bisherigen Systems auf Basis von Dokumentation → Treffen einer „go / no-go“ Entscheidung
 - › Bilden des Teams
 - › Ausarbeiten von
 - » Terminplänen
 - » Logistischen Details
 - » etc.

- Schritt 1: Präsentation der ATAM
 - › Wie ist sie aufgebaut?
 - › Wie läuft sie ab?
 - › Was für Techniken werden für die Analyse hergenommen?
 - › Wie werden Informationen gesammelt?
 - › Was für Resultate sind zu erwarten?
- Schritt 2: Präsentation der Business Driver
 - › In welchen geschäftlichen Kontext ist das System eingebettet?
 - › Was sind die wichtigsten Funktionen des Systems?
 - › Relevante technische und/oder geschäftliche Einschränkungen
 - › Welche Business Driver gibt es?
 - › Wer sind die Hauptinteressensgruppen?
 - › Die architekturellen Treiber (Quality Attributes)

- Schritt 3 – Präsentation der Architektur
 - › Welche technischen Einschränkungen gibt es?
 - › Existiert Interoperabilität mit anderen Systemen?
 - › Etc.
- Schritt 4 – Identifikation der architekturellen Herangehensweisen
 - › Evaluationsteam und Architekt analysieren die Architektur und versuchen, architekturelle Herangehensweisen zu identifizieren, um zu sehen wie der Architekt die Quality Attributes erfüllen wollte
 - » Konkret: Wie sind wichtige Strukturen des Systems aufgebaut?
 - » Wie und warum hat der Architekt seine Entscheidungen so getroffen?
 - » ABAS (Attribute based architectural style) als spezielle Notation, um Zusammenhang zwischen Architektureller Entscheidung \leftrightarrow Quality Attribute besser darzustellen
 - › Relevante Informationen: Beschreibung der Komponente, Topologie, Datenstrukturbeschreibung etc. sowie die für diese architekturelle Entscheidung typischen Vor- und Nachteile

Phase 1 – Evaluation der Architektur

- Schritt 5 – Generierung des Quality Attribute Utility Tree
 - › Kern für Rest der Evaluation
 - › Verfeinert die Quality Attributes hin zu Szenarien
 - › Nur die wichtigsten Szenarien werden weiterhin begutachtet
 - › Sorgt dafür, dass insbesondere die für das Unternehmen wichtigen Punkte begutachtet werden



- Schritt 6 – Analyse der architekturellen Herangehensweise
 - › Vergleicht die Ergebnisse von Schritt 4 (architekturellen Herangehensweisen) mit denen von Schritt 5 (Szenarien).
 - › Risiken, Vorteile, potenzielle Schwachstellen und Kompromisse werden dokumentiert
 - › Definition Kompromiss: Wenn eine architekturelle Entscheidung eine mögliche Schwachstelle für mehr als ein Quality Attribute darstellt

- Lücke zwischen Phase 1 und 2
 - › Neue Teilnehmer aus den verschiedenen Interessengruppen aussuchen
- Schritt 7 – Brainstorming und Priorisierung von Szenarien
 - › Neuen Teilnehmer sollen Szenarien finden
 - › Auch bisher nicht beachtete Szenarien aus dem Utility Tree dürfen neu überdacht werden
- Schritt 8 – Analyse der architekturellen Herangehensweise
 - › Deckungsgleich mit Schritt 6
 - › Hat die Charakteristik eines Testschritts: Desto weniger neue Informationen gefunden werden, desto besser hat die Evaluation in Phase 1 geklappt.

- Schritt 9 – Präsentation der Resultate
 - › Die Resultate beinhalten
 - » Die architekturellen Herangehensweisen
 - » Szenarien mitsamt Priorisierung
 - » Utility Tree
 - » Risiken, Vorteile, potenzielle Schwachstellen und Kompromisse
 - › Zusätzlich Gruppierung der Risiken in Themengebiete und mit Business Drivers in Verbindung bringen
 - › Ringschluss zum Anfang der Evaluation und macht es Nicht-Technikern verständlicher

■ Inhalt

- › Anfertigen eines schriftlichen Reports für das Unternehmen
- › Update der eigenen Datenrepositories für evt. zukünftige Evaluationen

■ Einführung

- › ECS – EOSDIS (Earth Observing System Data Information System) Core System
- › Empfängt 24/7 Daten von die Erde beobachtenden Satelliten
- › Daten müssen archiviert und aufgearbeitet werden → Mehrere Terabyte an neuen Daten pro Tag
- › Neue Daten sollen weltweit verfügbar sein

- Phase 0 – Partnerschaft und Vorbereitung
 - › ECS bereits im Einsatz
 - › NASA Mitarbeiter hatten gutes Vorwissen über ATAM
 - › Evaluation sollte insbesondere auf die Performance in zukünftigen Ausbaustufen achten
- Erfahrungen
 - › Phase dient dem Kennenlernen der Architektur
 - › Wenn das System nicht gut dokumentiert ist, kann je nach Kenntnisstand des Architekten trotzdem weiter verfahren werden
 - › Ev. zwei Teamleiter, um den Evaluierungsprozess „frisch“ zu halten
 - › Ev. auch Mitarbeiter des Auftraggebers in das Evaluierungsteam aufnehmen

- Schritt 1 – Präsentation der ATAM
 - › Mitarbeiter der NASA waren gut vorbereitet
 - › Hatten sich schon mit Verfeinerung der Quality Attributes für den Utility Tree beschäftigt
- Erfahrung
 - › Die Präsenz wichtiger Personen sollte explizit hervorgehoben werden
 - › Keine „wir gegen sie“ Stimmung aufbauen

- Schritt 2 – Präsentation der Business Driver
 - › ECS sammelt 15 Informationen von 7 Satelliten
 - › Täglich 1,5 Terabyte an neuen Daten
 - › Wird an 4 Standorten in USA gespeichert
 - › Business Driver:
 - » Unterstützung von fachübergreifenden Geowissenschaften
 - » Ortsunabhängige Benutzung
 - » Unterstützung von extern entwickelten wissenschaftlichen Algorithmen
 - » Gleichzeitiger Dateninput
 - » Produktion und Verteilung von insgesamt großen Datenmengen
 - » Etc.
- Erfahrung
 - › Auftraggeber besitzt häufig keine ausgereiften Präsentationsfolien → Keine Garantie, dass die richtigen Informationen genannt werden

- Schritt 3 – Präsentation der Architektur
 - › Vorstellung der einzelnen Komponenten des ECS durch den Projektmanager und Architekten
 - » Data Management (nach Daten suchen) und Data Server Subsystems (Datenhaltung)
 - » Data Ingestion Subsystem (Verantwortlich für neu eingehende Daten)
 - » Etc.
- Erfahrung
 - › Nicht zu tiefer Detaillevel in dieser Präsentation
 - › Architekt sollte auf die „Architectural Views“ zurückgreifen, die während der Erstellung des Systems am wichtigsten waren

- Schritt 4 – Identifikation der architekturellen Herangehensweise
 - › Client – Server Ansatz bei datenzentrischen Systemen → Ev. Durchsatzproblem
 - › Verteilte Datenhaltung für die Verteilung der Daten hin zur Benutzer Community → Schnell und Betriebssicher, aber ev. Inkonsistenzen und schlechte Veränderbarkeit
 - › Three-Tier Ansatz → Kann ev. Auswirkungen auf die Performance haben
 - › Metadaten zur Erhöhung der Benutzbarkeit → Ev. eingeschränkte Modifizierbarkeit
- Erfahrung
 - › Punkte können für gewöhnlich schnell aufgelistet, da schon Kenntnis der Architektur existiert

- Schritt 5 – Generierung des Quality Attribute Utility Tree
 - › Ausgangspunkt war die erste von den NASA Mitarbeitern erarbeitete Version
 - › Beispiel für die Verfeinerung:
 - » Quality Attribute (Level 2): Wartbarkeit
 - » Quality Attribute Verfeinerung (Level 3): Änderungen an einem Subsystem verlangen keine Veränderungen an anderen Subsystemen
 - » Quality Attribute Szenario: Installiere die nächste Version (5B) des Science Data Server mit einem Update der geowissenschaftlichen Daten Typen und Breiten-/Längengrad Unterstützung in das bestehende System (5A). Dieser Vorgang darf nicht länger als 8 Stunden dauern. Ebenso dürfen andere Subsysteme oder die Suchen, Browse und Anforderung Funktionalitäten nicht in ihrer Verfügbarkeit beeinträchtigt werden.
 - » Priorisierung: 30/30
 - › Gewählte Skala war Numerisch → Einfache Aufaddierung der Werte zu einem „Score“
- Erfahrung
 - › Auslöser – Antwort – Umwelt Form der Szenarien
 - › Ein Wort kann unterschiedliche Bedeutungen bei verschiedenen Benutzern haben → Automatische Verfeinerung auf Level 3

- Schritt 6 – Analyse der architekturellen Herangehensweisen

Scenario #: M1.1	Scenario: (M1.1) Deploy 5B version of the science data server with an update to the earth science data types and latitude/longitude box support into 5A baseline in less than 8 hours with no impact on other subsystems or search, browse, and order availability.			
Attribute(s)	Maintainability			
Environment	During routine maintenance			
Stimulus	Deploy 5B version of the science data server with an update to the earth science data types and latitude/longitude box support into 5A baseline.			
Response	Less than 8 hours with no impact on other subsystems or search, browse, and order availability.			
Architectural Decisions	Risk	Sensitivity	Tradeoff	Nonrisk
AD 1 Backward compatibility of interface	R1			
AD 2 Static linkage of client stubs in servers (static binding of libraries)	R2			
AD 3 Single copy of key operational databases	R3	S1	T1, T2	

■ Schritt 6 – Erfahrungen

- › Es existiert kein Leitweg, um Kompromisse, Risiken, Vorteile zu finden
- › Vorteil ATAM: Zeit hilfreiche / interessante Stellen zum finden:
- › Utility Tree: Sagt, worauf man sich fokussieren soll
- › Identifizierung der architekturellen Herangehensweisen: Zeigt, wo man suchen muss

- Schritt 7 – Brainstorming und Priorisierung von Szenarien
 - › 13 neue Szenarien wurden gefunden
 - › Davon wurden 3 für die weitere Evaluation ausgewählt
 - › Endgültiger Utility Tree besaß 50 Szenarien
- Erfahrung
 - › Beiträge dürfen von beiden Seiten eingereicht werden
 - › Auch alte, bisher nicht weiter beachtete Szenarien sollen nochmals überdacht werden
 - › Priorisierung nach dem 30% Regel – Anzahl der gefundenen neuen Szenarien * 0.3 = Anzahl der Stimmen pro Teilnehmer

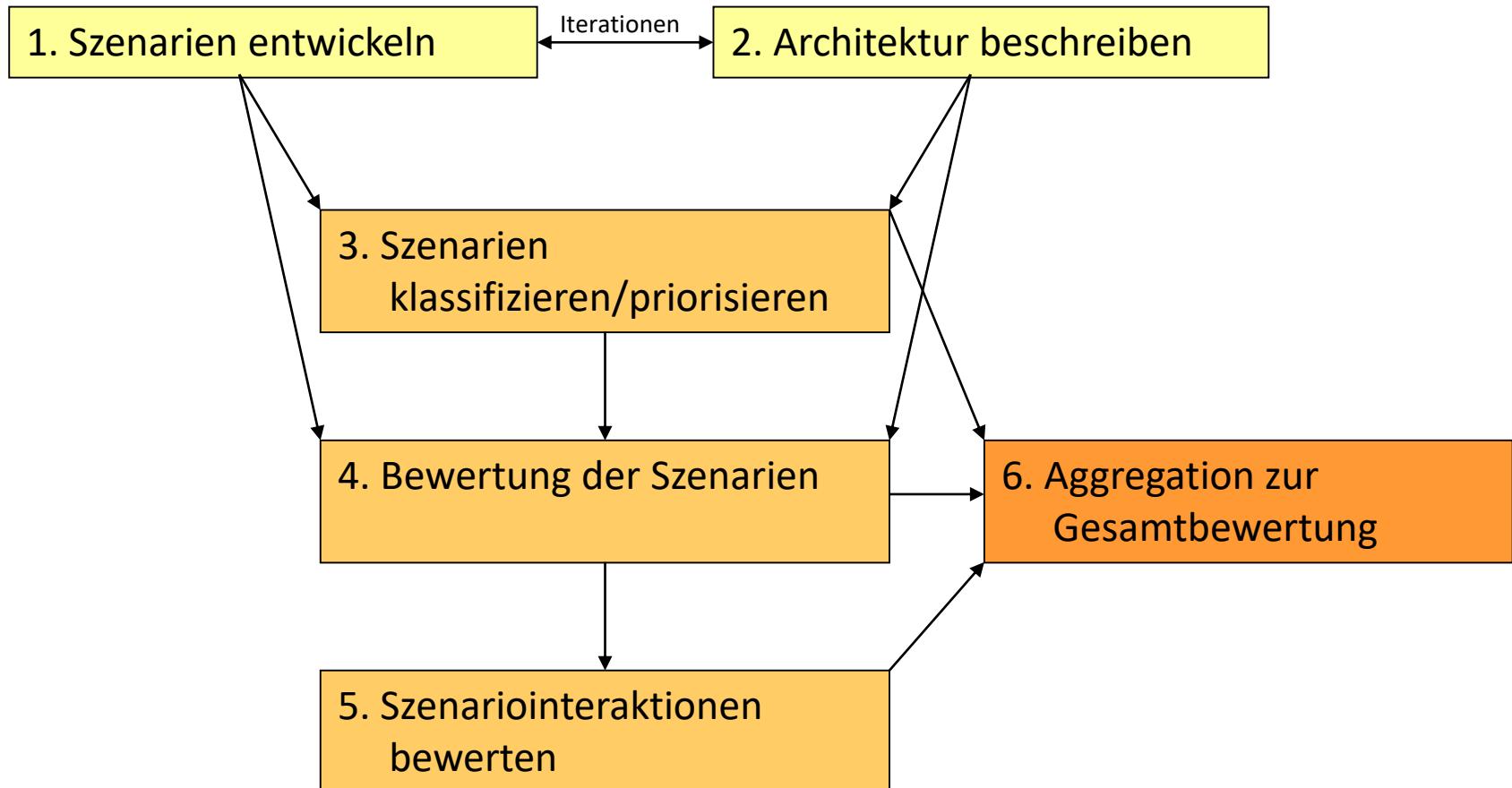
- Schritt 8 – Analyse der architekturellen Herangehensweisen
 - › Analog zu Schritt 6
- Erfahrung
 - › Kann vorkommen, dass alle in Schritt 7 gefundenen Szenarien einer Kategorie angehören, die Teilnehmer sich aber für die Analyse anderer Szenarien aussprechen
 - › Entscheidung hierfür liegt in der Hand des Evaluationsleiters

- Schritt 9 – Präsentation der Resultate
 - › Risikogruppen:
 - » 1. Keinerlei konsistente Strategie zur Kontrolle und Referenzierung von Datentypen und Interfaces
 - » 2. Lediglich eine Datenbank war online → Leistungs- und Verfügbarkeitskompromisse
 - » 3. Extern eingekaufte Datenbank war langsam
 - » 4. Objekt Beschreibungssprachen Bibliothek nicht sonderlich leistungsfähig
 - » 5. Langwieriger Ausschaltprozess
 - › 2, 3 und 4 betreffen insbesondere den Business Driver des gleichzeitigen Dateninputs sowie Produktion und Verteilung von großen Datenmengen
- Erfahrung
 - › Kurze Präsentation möglich, da von Anfang an Konzentration auf die wichtigsten Dinge
 - › Durch Einbindung der Mitarbeiter sind die Ergebnisse kein Schock für den Auftraggeber
 - › Gruppierung der Risiken und mit den Business Drivers in Verbindung bringen macht die Ergebnisse für alle verständlich

- Inhalt
 - › Abschließender Report war 48 Seiten lang
 - › 128 Mann Stunden von Seiten des Evaluationsteams
 - › 154 Mann Stunden von Seiten der Projektorganisation
- Erfahrung
 - › Post-Mortem Treffen abhalten

- ATAM
 - › Modernes und noch recht junges Verfahren
 - › Gut zeitlich planbar, da Fokus nur auf den wichtigsten Szenarien
 - › Gute Kosten / Nutzen Relation
 - › Leicht verständlich (auch für Nicht – „ITler“)
 - › Fördert „Teamspirit“
 - › Momentan führendes Verfahren
- Aber auch die ATAM ist nicht der Weißheit letzter Schluss
 - › Ev. nehmen zu viele Personen teil, so dass Entscheidungsprozesse zu lange dauern → Andere Verfahren wie die CBAM (Cost Benefit Analyse Method) sinnvoller...

■ Überblick



- Beteiligte Akteure (analog zu ATAM):
 - › Bewertungsteam:
 - » 3-4 Projektfremde, unparteiische Personen
 - » Festes Team oder für aktuelle Bewertung ausgewählt
 - › Vertreter des Projektteams:
 - » Projektleiter
 - » Architekt
 - » Auftraggeber der Bewertung
 - › Stakeholder:
 - » Entwickler
 - » Tester
 - » Systemintegrierer
 - » Wartungspersonal
 - » Users
 - » Etc.

- Inputs:
 - › Vorhandene Architekturdokumentation (Unzulänglichkeiten werden aufgedeckt)
 - › Szenarios
- Outputs
 - › Potenzial zur Erfüllung der Anforderungen
 - › Besseres Verständnis für die Architektur und dessen Funktionalität
 - › Bessere Dokumentation
 - › Verbesserte Kommunikation
 - › Abbildung der Szenarien auf die Architektur
 - › Vergleichbar machen von unterschiedlichen Architekturen hinsichtlich Modifizierbarkeit und Komplexität
 - › Priorisierung der Szenarios

- 1. Szenarien entwickeln
 - › Bewertung von Architekturkomponenten durch entsprechende Szenarien
 - › Entwickeln von Szenarien durch Brainstorming
 - › Abbildung aller wichtigen
 - Aktivitäten,
 - User,
 - Änderungen am System und
 - Qualitätsmerkmale,
- denen das System gerecht werden soll

- 2. Architektur beschreiben
 - › Untersuchung der Szenarien erfordert Dokumentation der betroffenen Architektureile
 - › Beschreibung der Architektur hinsichtlich Funktion, Kommunikation und Daten
 - › Wahl einer für alle Beteiligten verständlichen Notation (formale Beschreibungsprachen oder natürlichsprachlich)

■ 3. Szenarien klassifizieren/priorisieren

› Direkte Szenarien:

- » Direkte Szenarien sind Szenarien, die direkt zu den Anforderungsspezifikationen der Entwurfsphase passen
- » Primäres Ziel ist nicht das Aufdecken von Fehlern, sondern Förderung des Verständnisses der Stakeholder für die Architektur
- » Direkte Szenarien sind analog zu Use Cases in UML-Szenarien, die von der Architektur explizit, d.h. ohne notwendige Änderungen, unterstützt werden
- » Demonstration wie das jeweilige Szenario die Architektur „durchläuft“

› Indirekte Szenarien:

- » Indirekte Szenarien nennt man manchmal auch Change Cases
- » Szenarien, die eine Anpassung von Teilen der Architektur erfordern
- » Änderungen können sein:

- Hinzufügen oder Entfernen von Komponenten, Verknüpfungen, Schnittstellen, bzw. einer Kombinationen davon
- Essenziell für die Beurteilung von „Flexibilität“

■ 3. Szenarien klassifizieren/priorisieren

- › Klassifizierung:
 - » Mehrere Szenarien, anhand denen man das gleiche Qualitätsmerkmal testen kann, werden zu Klassen zusammengefasst.
 - » z.B. Robustheit: Szenario für Fehleingaben, Szenario für Stromausfall
 - » Vor Priorisierung
- › Priorisierung:
 - » Wichtigkeit: Stakeholder bringen ihre Interessen oder auch Befürchtungen ein. Festlegung der Prioritäten durch Abstimmen.
 - » Stellt sicher, dass alle wichtigen Szenarien innerhalb der begrenzten Zeit bewertet werden können
 - » Die „Wichtigkeit“ wird allein durch die Stakeholder definiert
- › Abstimmungsprozess: Jeder Stakeholder bekommt eine fixe Anzahl an Stimmen, typischerweise 30% der Gesamtsumme der Szenarien

■ 4. Bewertung der Szenarien

- › Stakeholder meist weniger an Szenarien interessiert, die „glatt“ durchlaufen, sondern eher an indirekten Szenarien
- › Mapping der Szenarien mit Architekturbeschreibung (Architekt)
 - » Mapping deckt Schwächen in der Architektur oder ihrer Dokumentation auf
 - » Direktes Szenario:
 - Präsentation, wie das Szenario in der Architektur verarbeitet wird
 - » Indirektes Szenario:
 - Dokumentation, der notwendigen Änderungen und dessen zeitlichen und finanziellen Aufwand
- › Stakeholder und Bewertungsteam bekommen mehr Einblick in die Struktur der Architektur und die dynamische Interaktionen der einzelnen Komponenten
- › Diskussion:
 - » Versteht jeder das gleiche unter den Szenarien? Werden die gemappten Architekturlösungen den Szenarien gerecht?
 - » Diskussion ist wichtig, um Missverständnissen vorzubeugen

■ 5. Szenariointeraktionen bewerten

- › Interaktion:
- › Zwei oder mehrere indirekte Szenarien interagieren, wenn sie Änderungen an ein und derselben Architekturkomponente erfordern
- › Interaktion von unabhängigen Szenarien, deckt Komponenten auf, die bezugsfremde Funktionen ausführen
- › Bereiche mit hoher Interaktion deuten auf schlechte Separation von Funktionalitäten hin (Störanfälligkeit)
- › Schlechte Dokumentation als weiterer Grund für (scheinbare) Interaktion (→ Schritt 2)

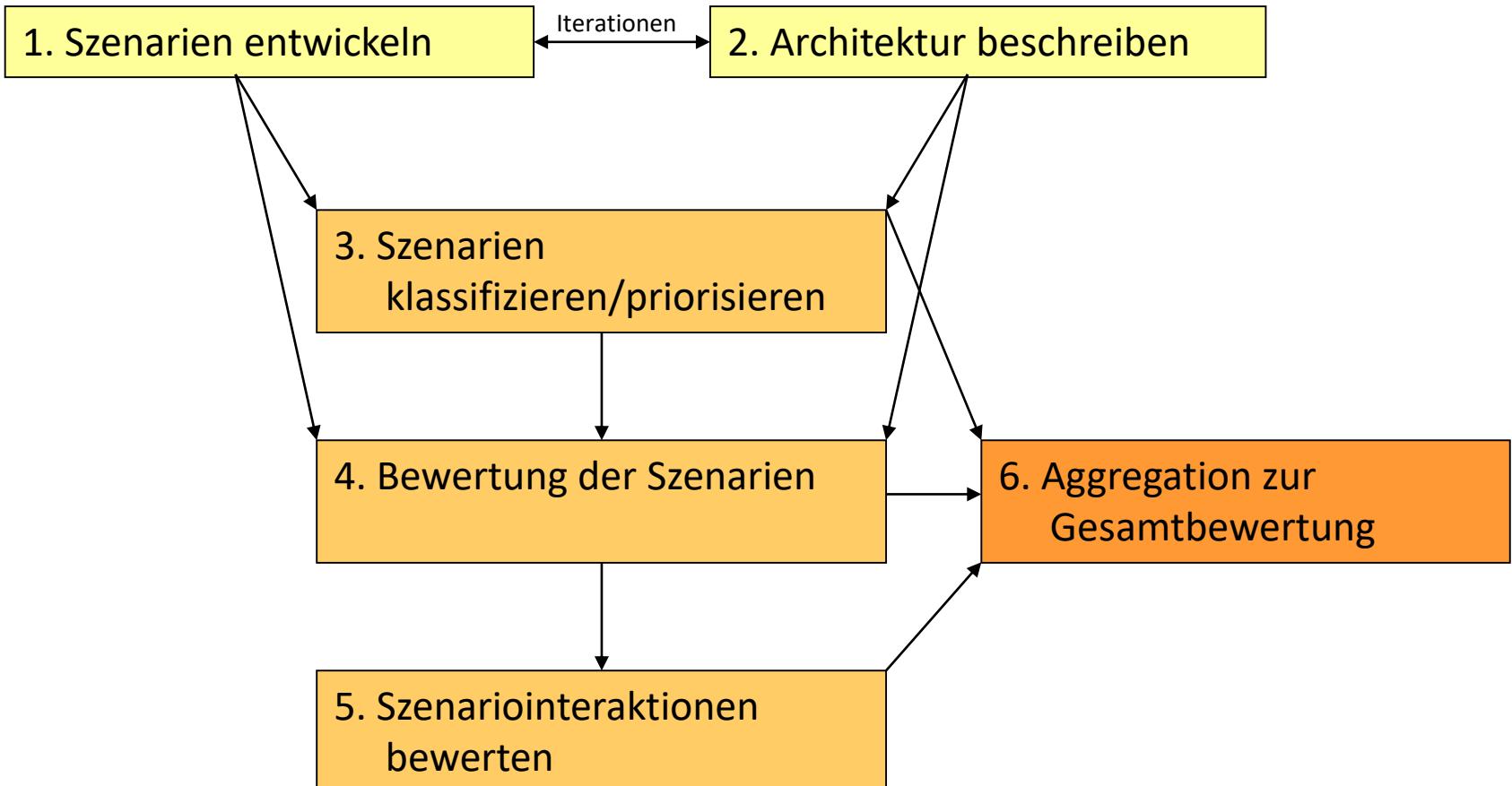
- **6. Aggregation zur Gesamtbewertung**
 - › Gewichtung der Szenarien nach relativer Bedeutung des zugrunde liegenden Erfolgsfaktors
 - › Ermöglicht Ranking von mehreren zur Auswahl stehenden Architekturlösungen
 - › Wichtig: Wahl der Gewichte sollte unter Einbezug aller Stakeholder offen diskutiert werden

- 6. Aggregation zur Gesamtbewertung
 - › Noten von 1 bis 10
 - › Hier jedes Szenario gleich wichtig. Man könnte die einzelnen Szenarien auch noch gewichten → gewichtete Gesamtnote

Beispiel für eine Gesamtbewertungstabelle:

	Szenario 1	Szenario 2	Szenario 3	Szenario 4	Interaktion	Gesamt	
○ Architektur- lösung 1	5	5	1	4	6	21	○
Architektur- lösung 2	5	5	6	10	7	33	

- Überblick
 - › Direktes Szenario: „Normales“ Szenario, welches ohne Anpassung der Architektur verarbeitet werden kann
 - › Indirektes Szenario: Szenario, welches eine Änderung an der Architektur erfordert



Beispielhafte Agenda für eine Bewertung nach SAAM

Day 1

Time	Agenda item
08:15-08:45	Introductions, statement of evaluation goals, overview of evaluation method, ground rules
08:45-09:00	Overview of system being evaluated, including goals for the architecture
09:00-10:00	Develop scenarios (Step 1)
10:00-10:30	Break
10:30-12:00	Develop scenarios (Step 1)
12:00-01:00	Lunch
01:00-02:30	Describe the architecture(s) (Step 2)
02:30-03:00	Break
03:00-03:30	Develop scenarios (Step 1)
03:30-05:00	Classify and prioritize scenarios (Step 3)

Beispielhafte Agenda für eine Bewertung nach SAAM

Day 2

Time	Agenda item
08:15-10:00	Individually evaluate indirect scenarios (Step 4)
10:00-10:30	Break
10:30-12:00	Individually evaluate indirect scenarios (Step 4)
12:00-01:00	Lunch
01:00-02:30	Assess scenario interactions (Step 5)
02:30-03:30	Create the overall evaluation (Step 6)
03:30-04:00	Break
04:00-05:00	Wrapup, summarization, reports



- ATAM und SAAM sind sehr effektive, aber auch aufwändige Verfahren zur Bewertung komplexer Gesamtarchitekturen
- Gesamtarchitekturen bestehen (meist) aus kleineren Subarchitekturen, deren Qualität maßgeblich zur Leistungsfähigkeit des Gesamtsystems beitragen
- Diese sollten deshalb so früh wie möglich evaluiert werden, wofür ein schlankes Verfahren benötigt wird, das dem Architekten die Sicherheit gibt „auf dem richtigen Weg zu sein“
- **Active Reviews for Intermediate Designs (ARID)**

Definition & Charakteristika

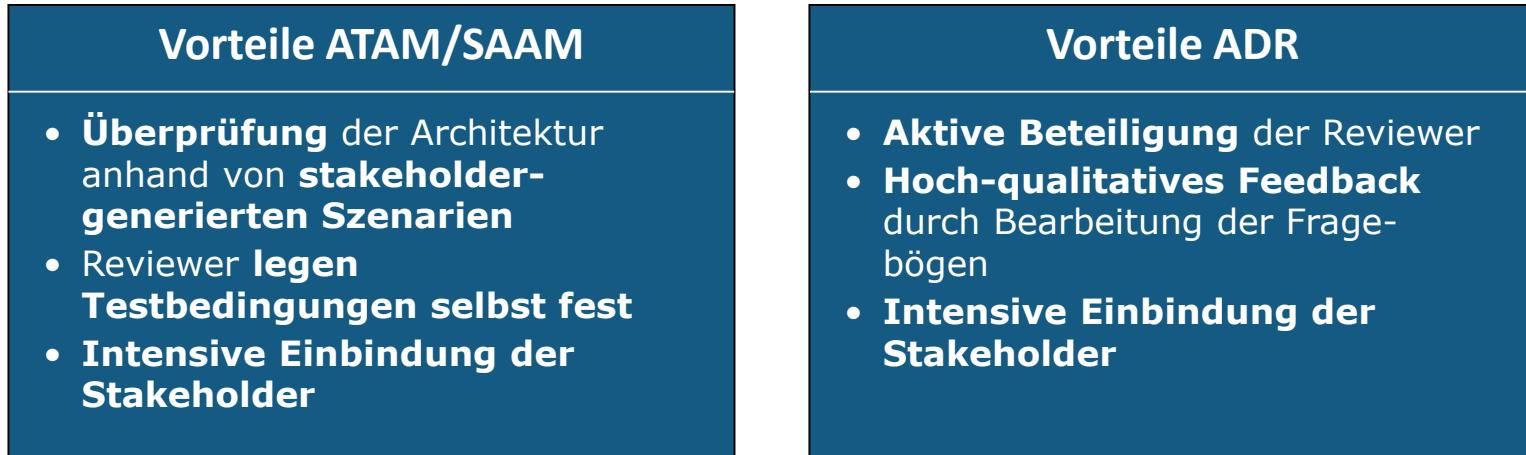
- ARID ist eine schlanke Bewertungsmethode, um Teilstrukturen einer Software-Architektur frühzeitig zu evaluieren.

- Einbindung von Stakeholdern (z.B. Entwicklern, Kunden) in die Bewertung
- Eine detaillierte Dokumentation wird nicht benötigt
- Kombiniert die Vorteile von szenario-basierten Ansätzen (ATAM, SAAM) und Active Design Reviews
- Kann relativ zügig durchgeführt werden
- Bewertet nur, ob der gewählte Ansatz grundsätzlich passend ist oder nicht (keine Details)

Active Design Reviews (ADR)

- Effektive Art der Befragung von Review-Teilnehmern
- Aktive Teilnahme der Reviewer:
Jeder bekommt vorab einen Teil der Dokumentation und muss einen Fragebogen mit Übungen dazu bearbeiten; er muss die Architektur anwenden und ist so gezwungen sich Gedanken darüber zu machen
- Führt zu tiefem Verständnis anstelle von oberflächlichem Suchen nach offensichtlichen Fehlern
(à la „Was meint ihr dazu?“)
- Bringt Struktur und Nachvollziehbarkeit in den Befragungsprozess

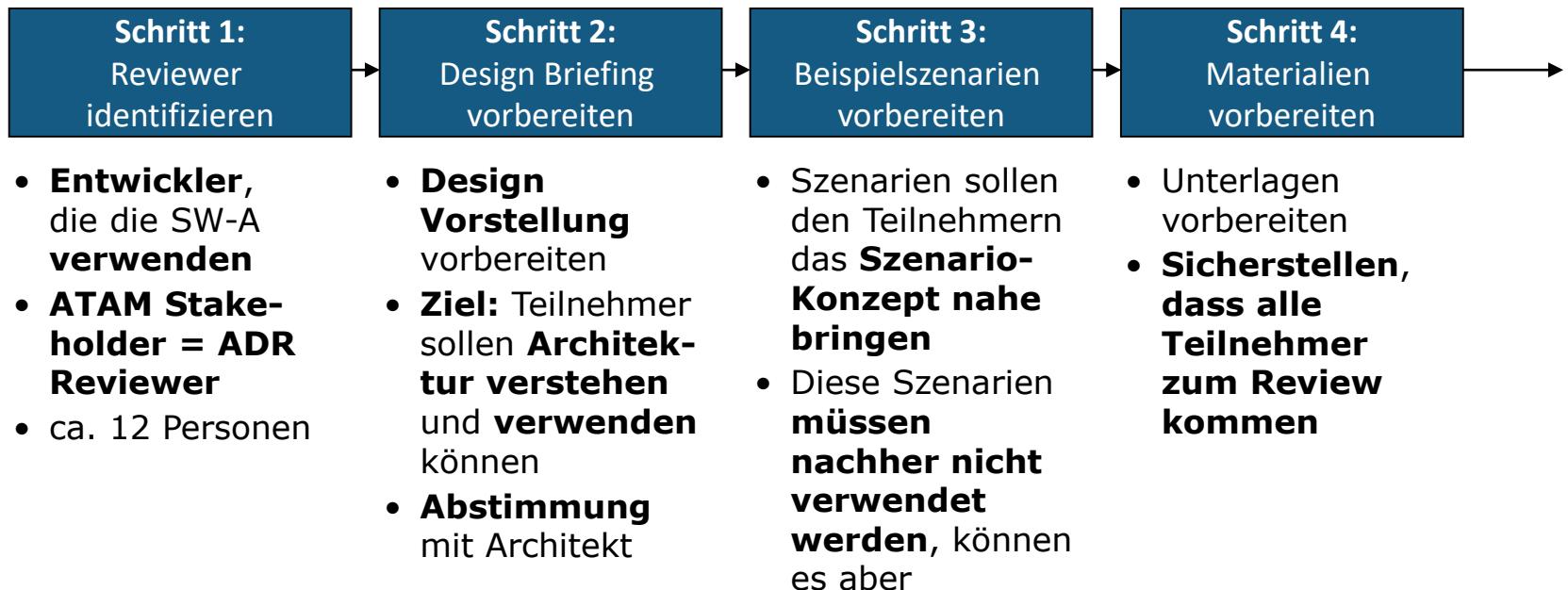
- ARID kombiniert die Vorteile von ATAM/SAAM und ADR



- ARID läuft in 2 Phasen ab

Phase 1: Vorbereitung

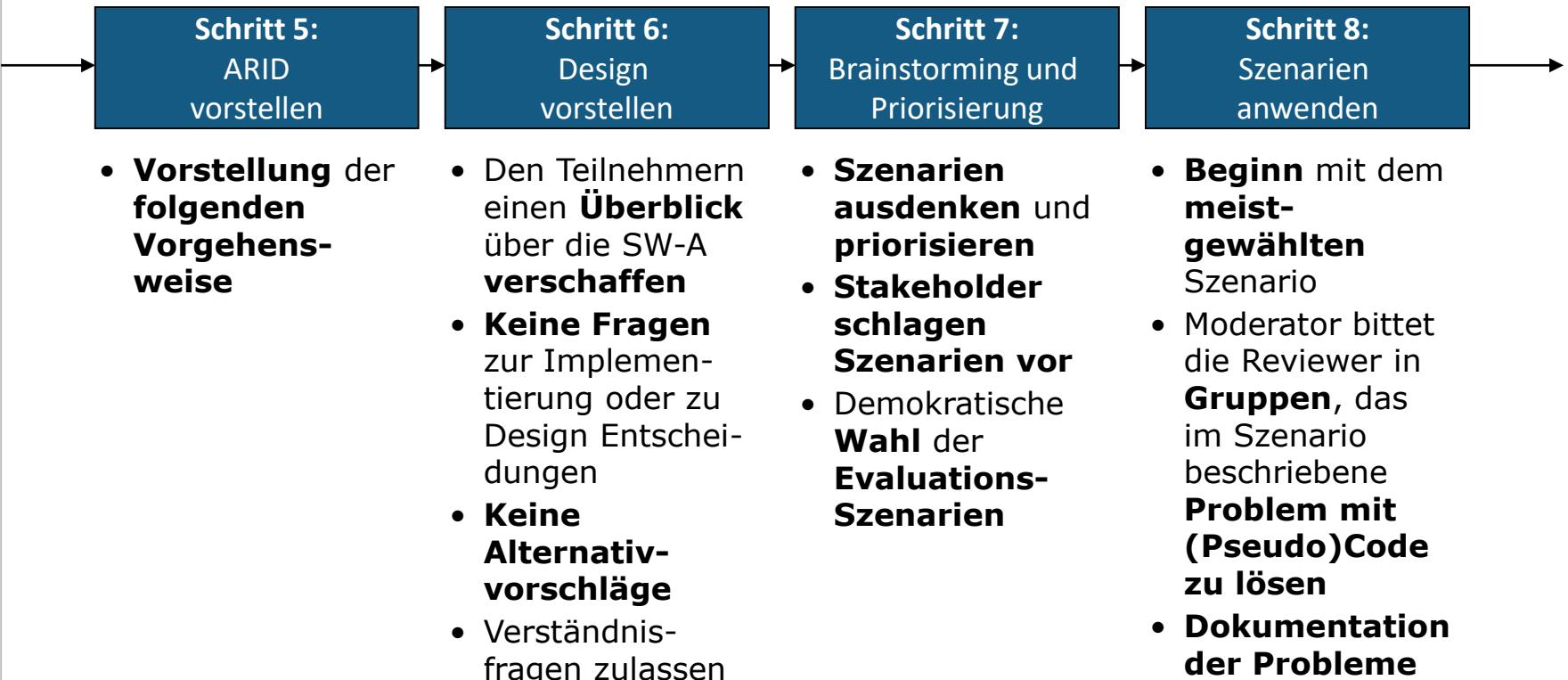
- **Teilnehmer:** Architekt und Review-Moderator
- **Dauer:** etwa 1 Tag



- ARID läuft in 2 Phasen ab

Phase 2: Review

- **Teilnehmer:** Architekt, Review-Moderator, Reviewer
- **Dauer:** etwa 1,5 Tage



- ARID läuft in 2 Phasen ab

Phase 2: Review

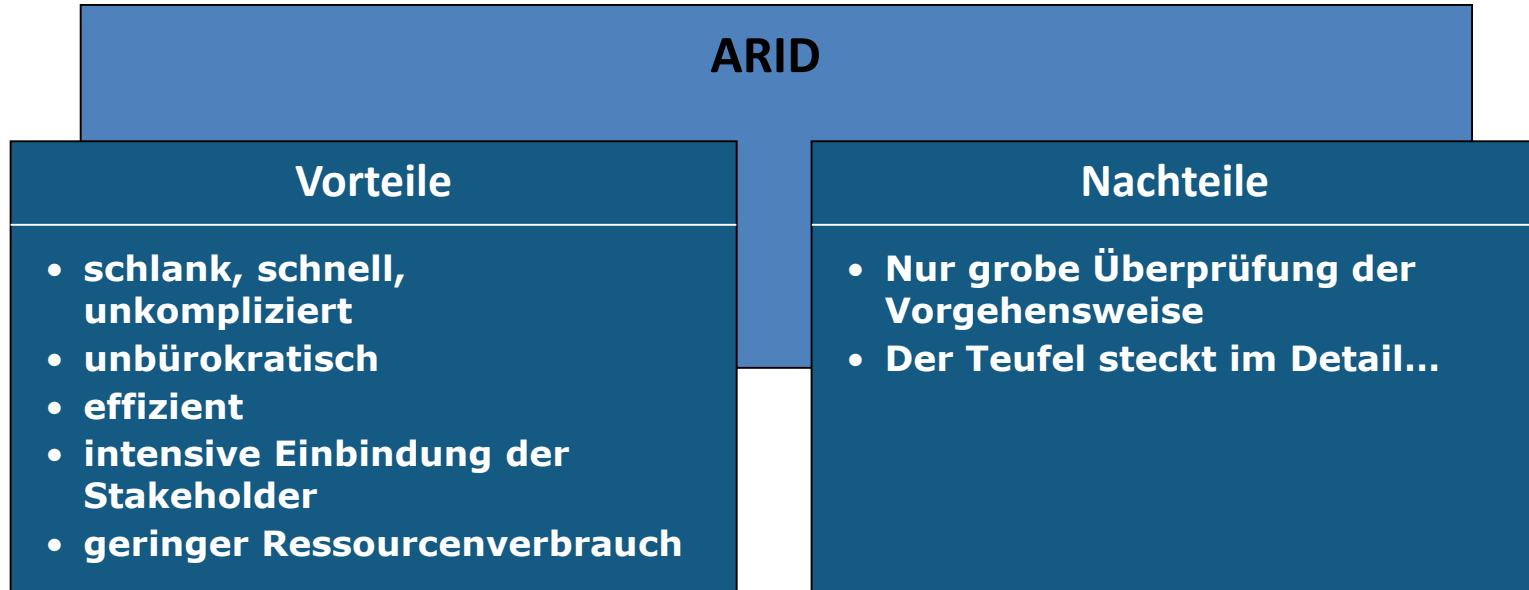
- **Teilnehmer:** Architekt, Review-Moderator, Reviewer
- **Dauer:** etwa 1,5 Tage

Schritt 9:
Zusammen-
fassung



- **Festhalten der Ergebnisse, Probleme, Hürden und Meinungen**

- ARID kombiniert die Vorteile von ATAM/SAAM und ADR



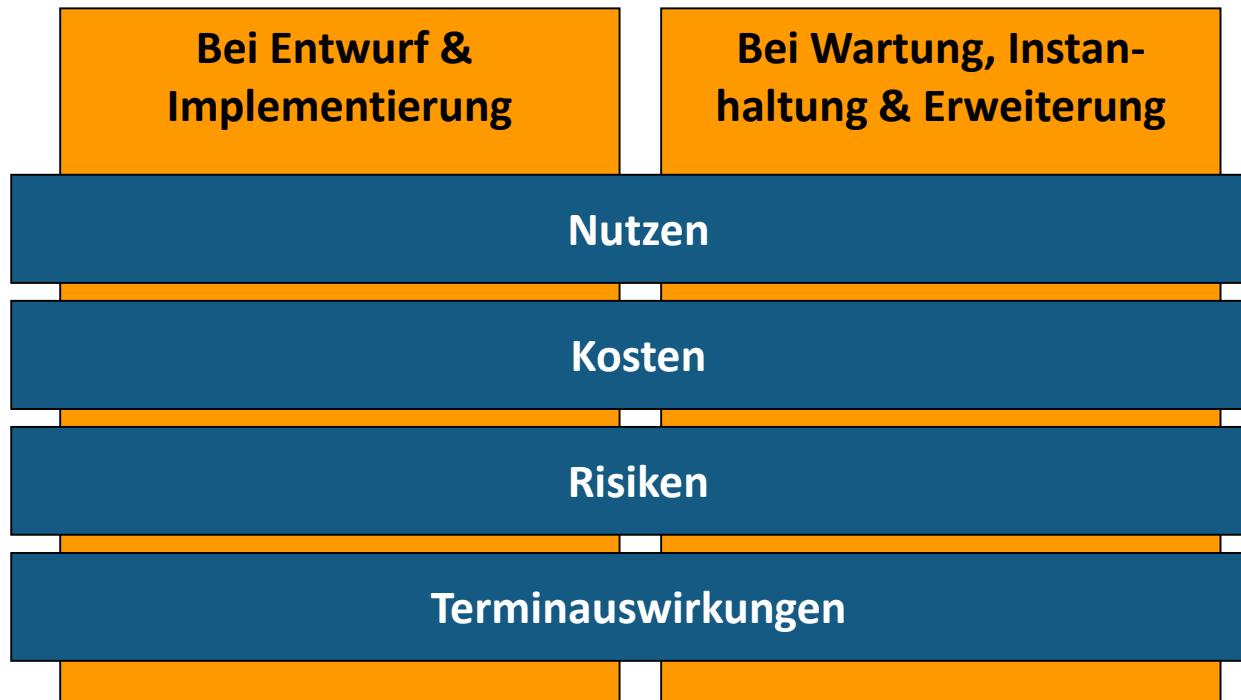
Motivation - Cost Benefit Analysis Method (CBAM)

- ATAM unterstützt den SW-Architekten mit der Analyse der technischen Trade-Offs bezüglich der Design und Wartungsphase der Software
- Eine entscheidende Betrachtung fehlt jedoch:

Die wichtigsten Trade-Offs bei großen und komplexen Systemen beruhen auf wirtschaftlichen Motiven.

- Wie kann eine Organisation ihre (knappen) Ressourcen einsetzen, um Gewinne zu maximieren und Risiken zu minimieren?
- In der Vergangenheit wurden bei der Bewertung von SW-A meist nur die Implementierungskosten betrachtet. Das ist aus zweierlei Hinsicht falsch:
 - › 1. Müssen vor allem auch die Kosten die für Wartung, Instandhaltung und Erweiterung/Anpassung anfallen betrachtet werden
 - › 2. Es müssen genauso die Nutzen betrachtet werden

- Um die Frage beantworten zu können, wird zusätzlich zur technischen Bewertung ein ökonomisches Modell benötigt, dass die folgenden Faktoren berücksichtigt:

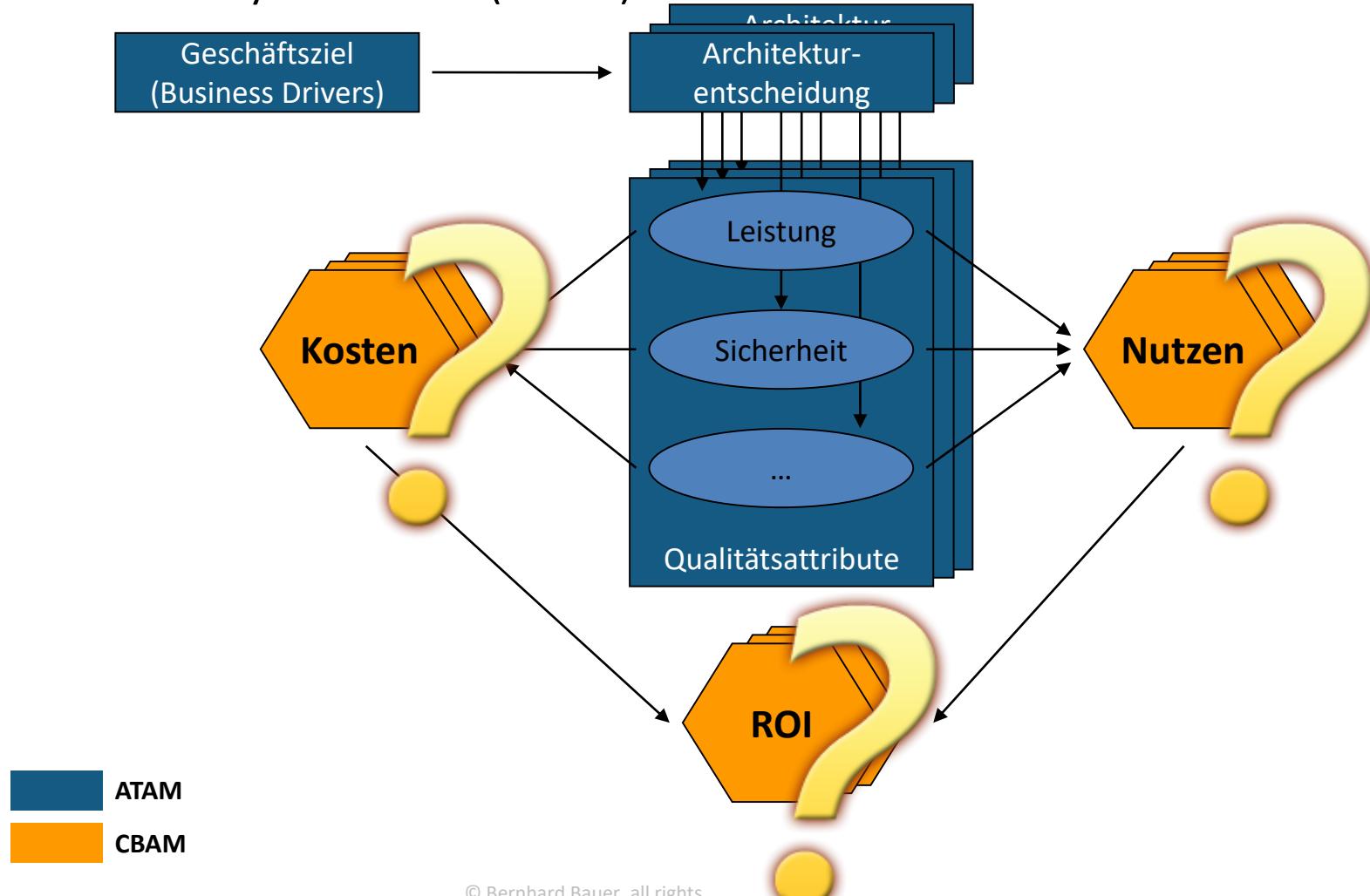


Definition & Charakteristika

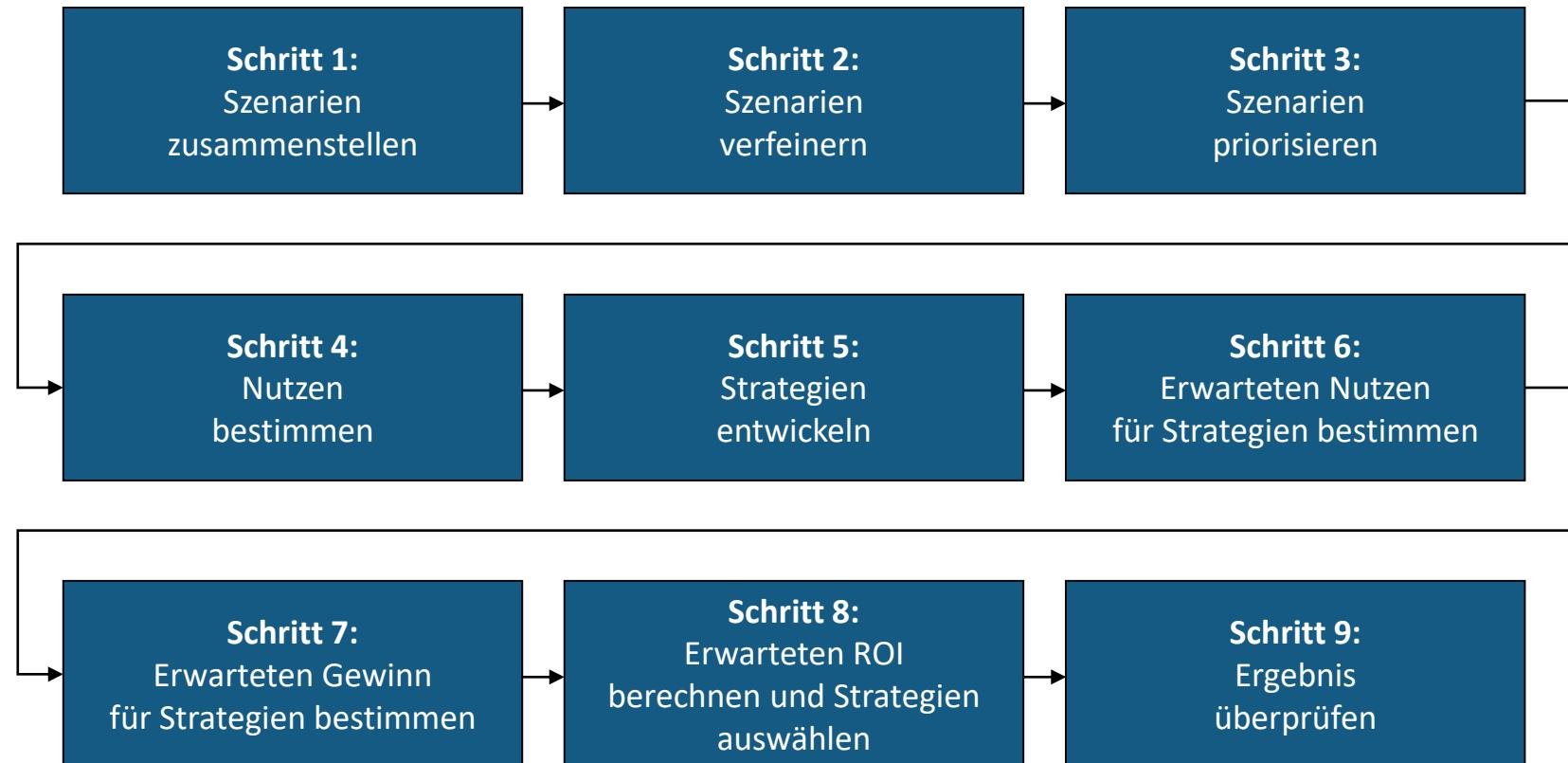
- **Cost Benefit Analysis Method (CBAM)** unterstützt den SW-Architekten bei der **Maximierung des Gewinns** aufgrund von Architektur-Entscheidungen
- CBAM **baut auf die Ergebnisse** aus der **ATAM** Bewertung auf
- Damit ist es **keine Alternative** zu **ATAM**, sondern eine **Ergänzung** der Evaluation um die **finanzielle Sichtweise**

CBAM ist ein Rahmenwerk, das bei der Zuordnung von **Kosten, Nutzen, Risiken und Zeitimplikationen** zu **Architektur-Entscheidungen** hilft. Basierend darauf können die Stakeholder einen **rationalen Entscheidungsprozess** aufsetzen, der ihre **Bedürfnisse** und **Risikoeinstellung** berücksichtigt.

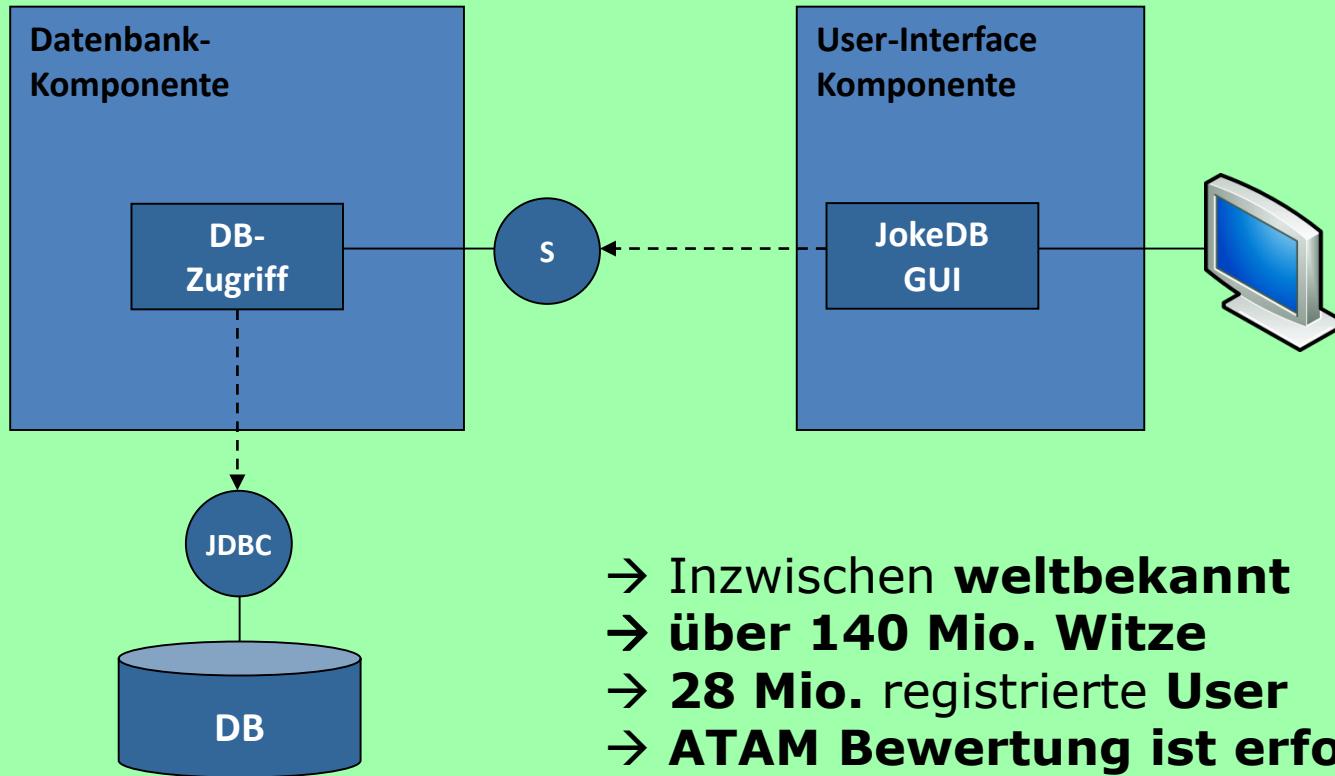
- Cost Benefit Analysis Method (CBAM)



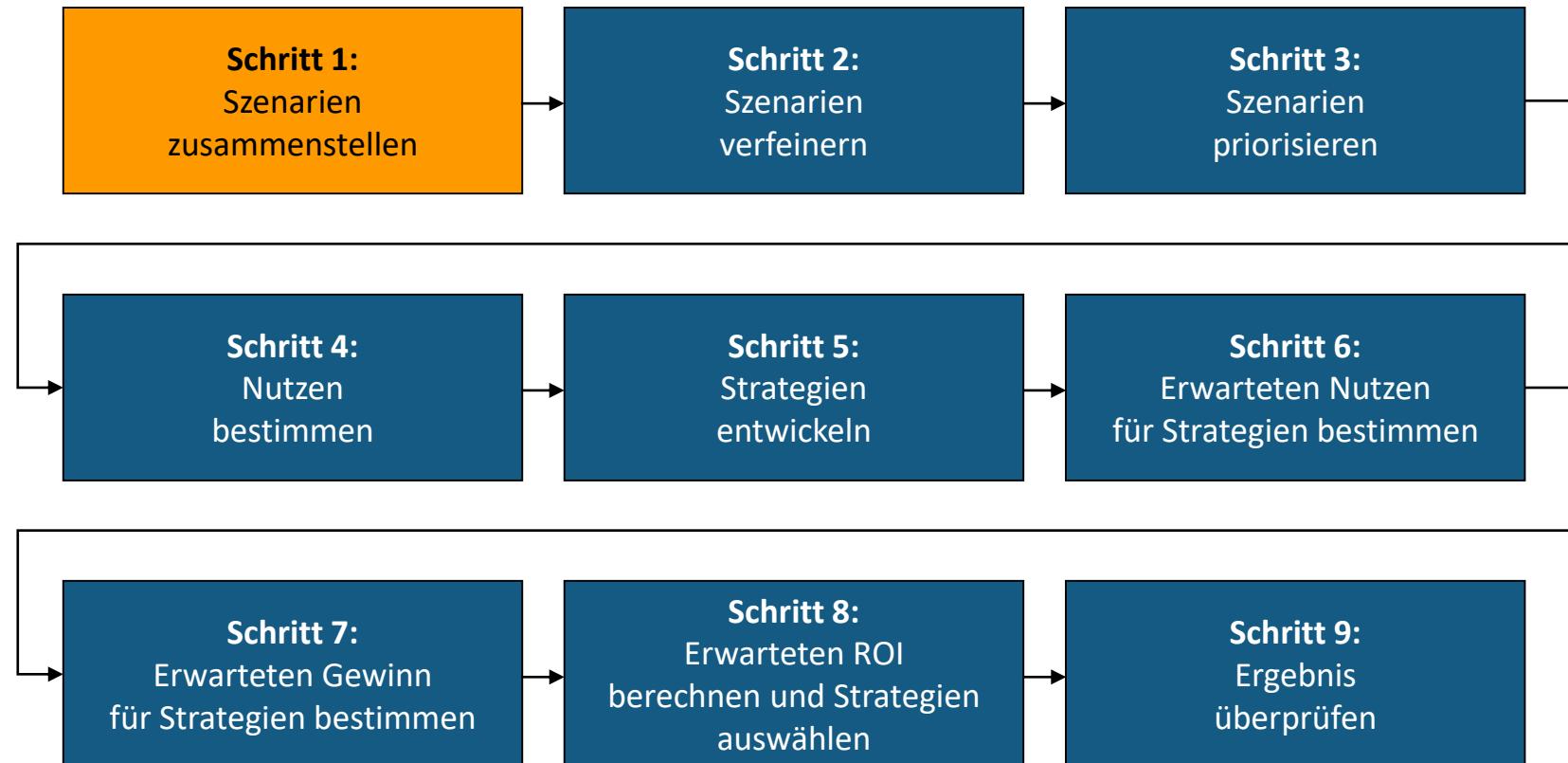
- CBAM läuft in neun Schritten ab



Hands-On: „The Joke Database“

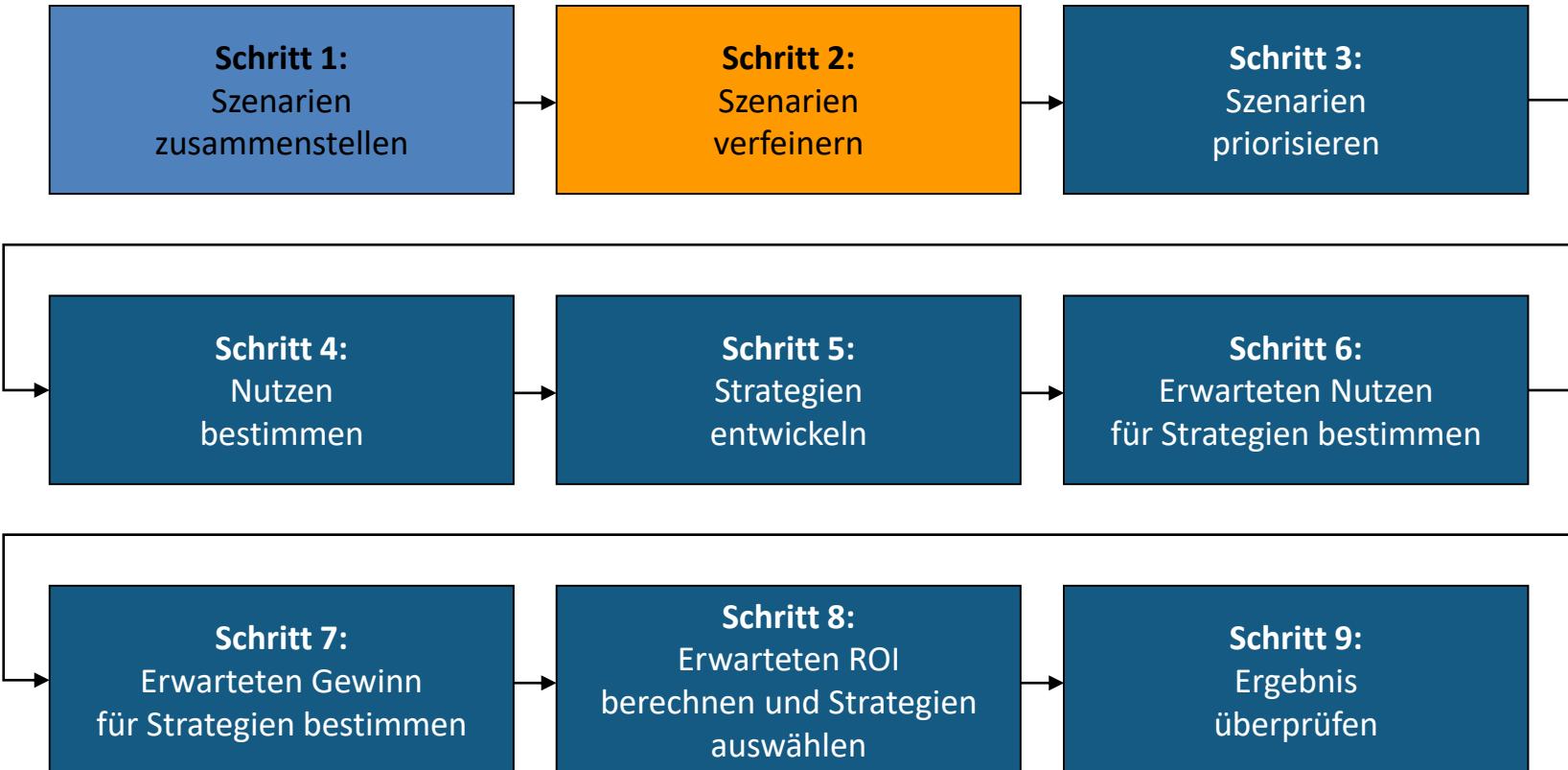


- CBAM läuft in neun Schritten ab



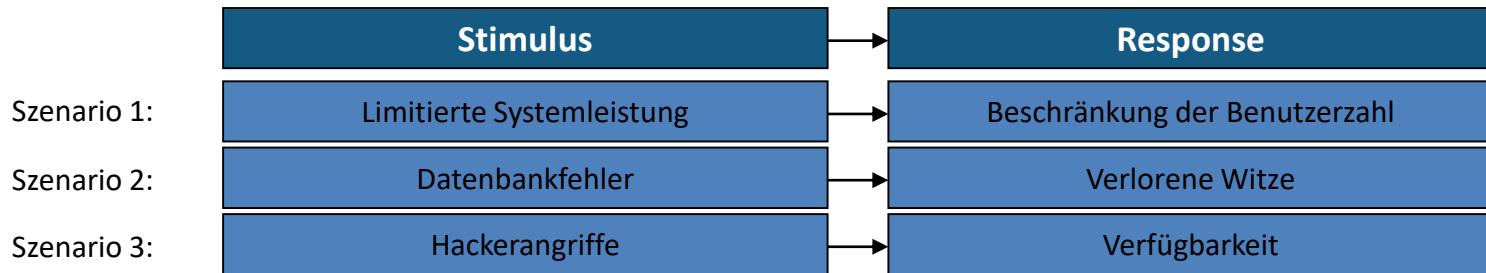
- Schritt 1: Szenarien zusammenstellen
 - › Szenarien aus der ATAM Bewertung übernehmen
 - › Zusätzlich können die Stakeholder auch neue Szenarien einbringen
 - » Insgesamt: N Szenarien
 - › Diese Szenarien bzgl. Erfüllung der Geschäftsziele priorisieren und ein Drittel davon auswählen
 - › am Ende: $N/3$ Szenarien
- Beispiel:
 - › Szenario 1: Wegen einer limitierten Systemleistung muss die Anzahl der Benutzer, die gleichzeitig eingeloggt sind, beschränkt werden.
 - › Szenario 2: Reduktion der verlorenen Witze auf Grund von Datenbankfehlern.
 - › Szenario 3: Reduktion der Down-Time wegen Hackerangriffen.

- CBAM läuft in neun Schritten ab



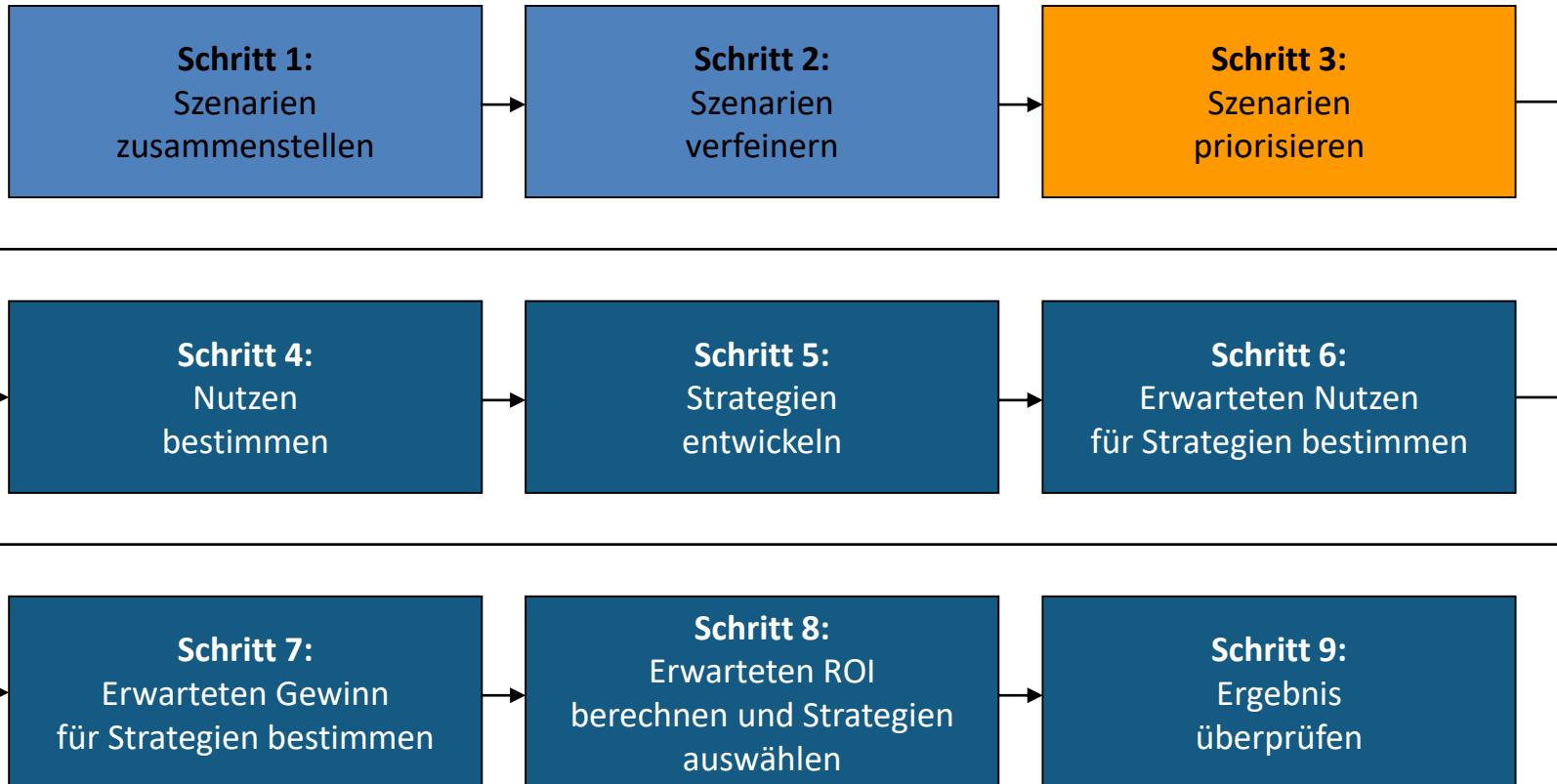
- Schritt 2: Szenarien verfeinern
- Szenarien in Hinblick auf Stimulus-Response formulieren
- Stimulus ist (im weitesten Sinne) eine Interaktion mit dem System
- Response ist die resultierende Ausprägung des Qualitätsattributes
- Ausprägungen des Qualitätsattributs dieses Szenarios für Best-Case, Worst-Case, Ist und Soll bestimmen
 - › Best-Case: Eine weitere Verbesserung bringt keinen zusätzlichen Nutzen
 - › Worst-Case: Unterhalb dieser Ausprägung kein Nutzen
 - › Ist: Derzeitige Ausprägung eines Qualitätsattributes
 - › Soll: Gewünschte Ausprägung eines Qualitätsattributes

■ Schritt 2: Beispiel



	Szenario 1 Gleichzeitige Benutzer	Szenario 2 Verlorene Witze	Szenario 3 Verfügbar- keit
best-case	8 Mio.	0 %	100 %
soll	4 Mio.	0 %	95 %
ist	2 Mio.	30 %	70 %
worst-case	0	100 %	0 %

- CBAM läuft in neun Schritten ab



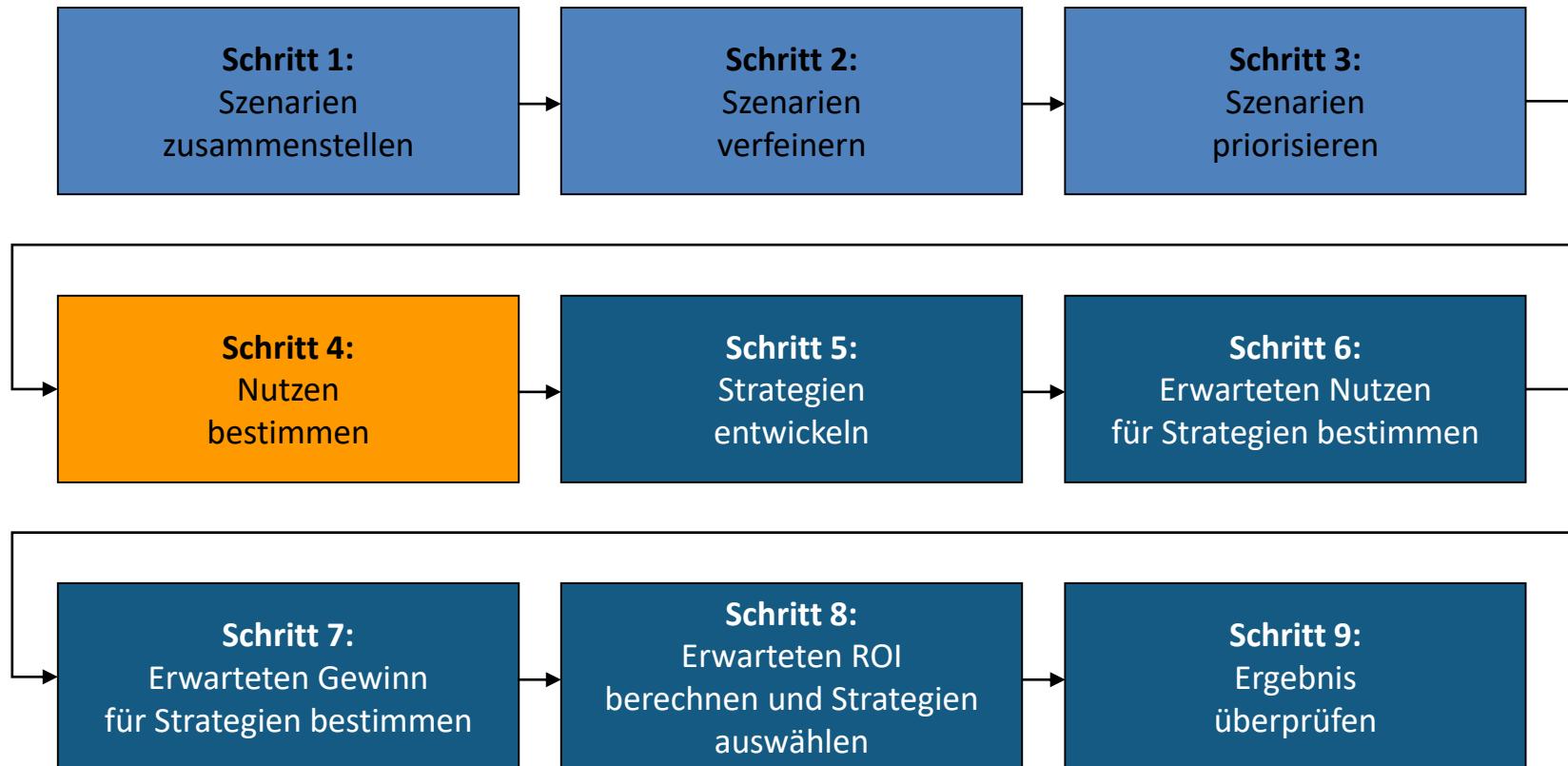
■ Schritt 3: Szenarien priorisieren

- › Jeder Stakeholder bekommt (z.B. 100) Stimmen, die er frei auf die Szenarien verteilen kann
- › Wahl bezieht sich auf Soll-Niveau der Qualitätsattribute
- › Nur die obere Hälfte der Szenarien wird weiter betrachtet
- › Es bleiben $N/6$ Szenarien
- › Die Stimmen, die die Szenarien erhalten haben werden später als Gewichtungen verwendet

- Schritt 3: Beispiel

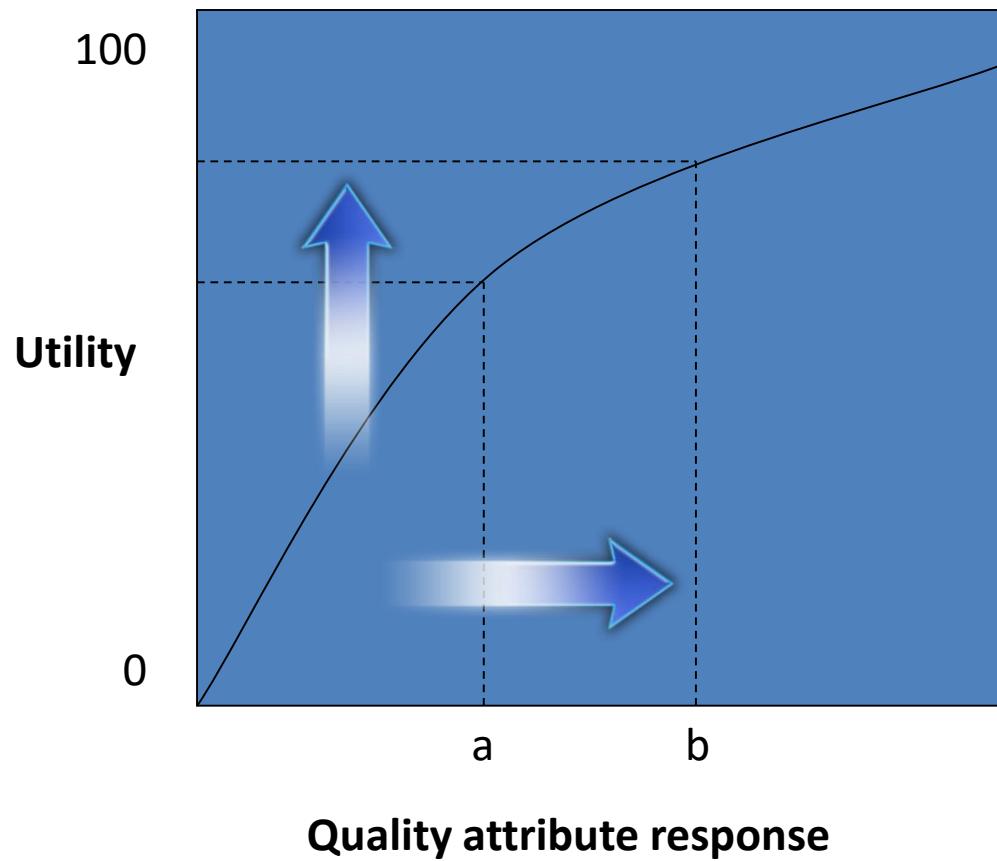
Stakeholder A	Szenario 1	Szenario 2	Szenario 3
40	30	30	
15	60	25	
40	40	20	
SUMME	95	130	75
Gewichtung w_j	0,32	0,43	0,25

- CBAM läuft in neun Schritten ab

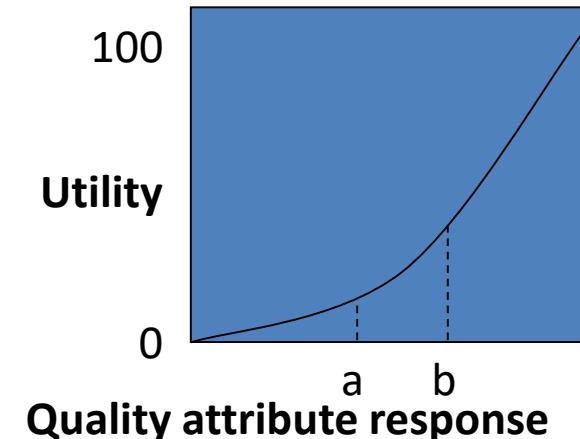
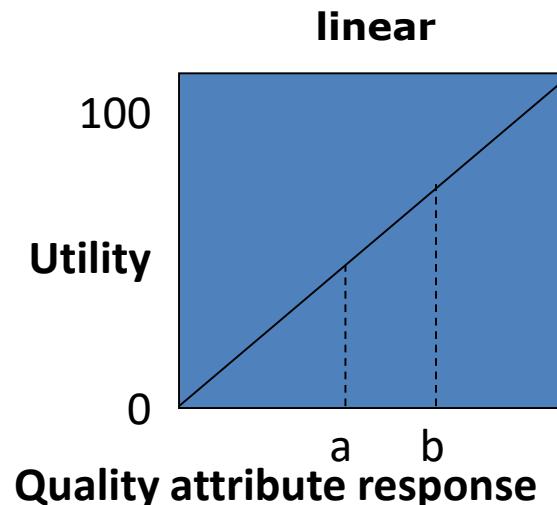
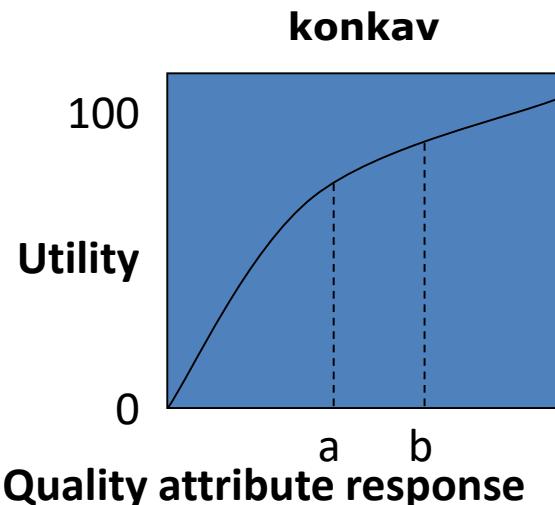
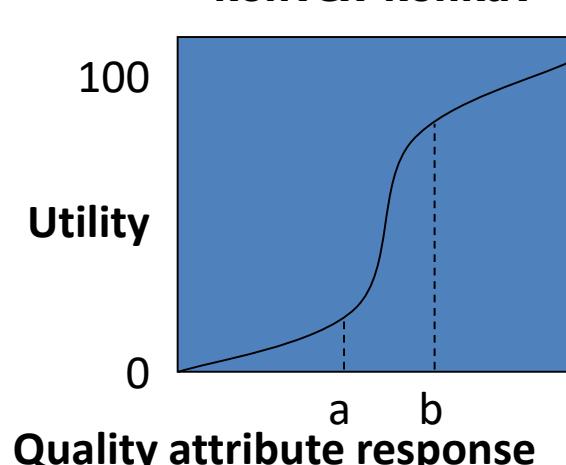


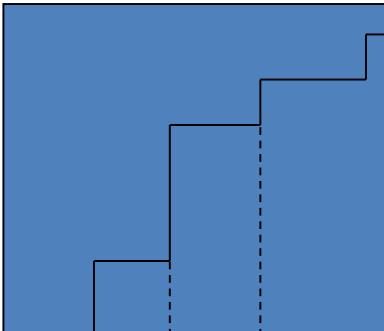
- **Schritt 4: Nutzen bestimmen**
 - › Den unterschiedlichen Ausprägungen eines Qualitätsattributs werden nach Wichtigkeit und Risikoeinstellung unterschiedliche Nutzenniveaus (z.B. zwischen 0 und 100) zugewiesen
 - › Das Ergebnis ist eine Utility-Response-Curve für jedes Szenario

- Vergleich des Nutzenniveaus verschiedener Szenarien liefert die Utility-Response-Curve



Vorgehensweise – Schritt 4 – Utility-Response-Curve

konvex**konvex-konkav****diskret**



This graph shows a discrete utility response curve. The vertical axis is labeled "Utility" with values 0 and 100. The horizontal axis is labeled "Quality attribute response" with points "a" and "b". The utility is zero until point "a", then jumps to a constant value. At point "b", it jumps again to a higher constant value. Dashed lines connect points "a" and "b" on the curve to their corresponding values on the axes.

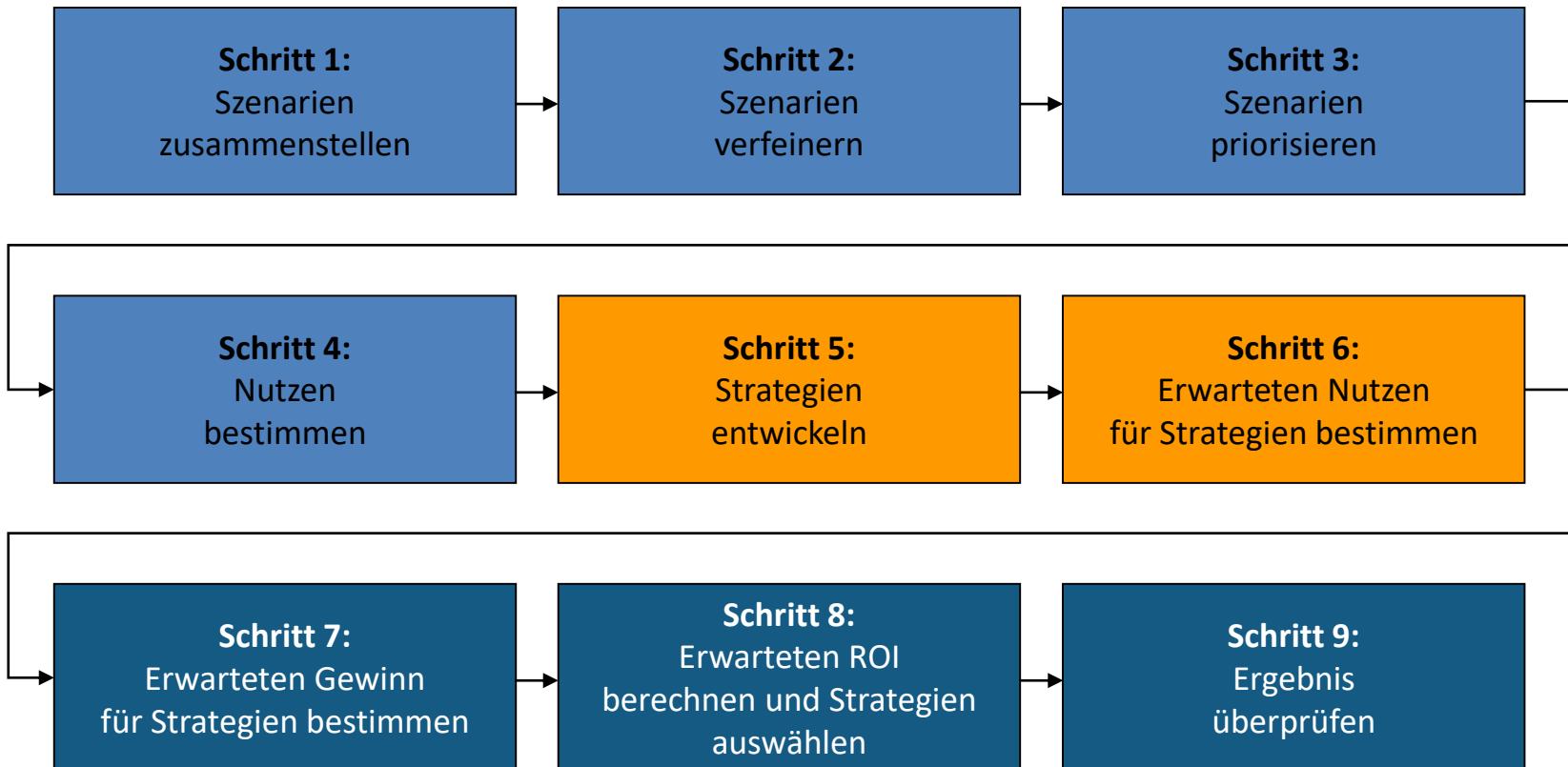
© Bernhard Bauer, all rights
 reserved 2018

262

Schritt 4: Beispiel

	Szenario 1 Gleichzeitige Benutzer	Nutzen	Szenario 2 Verlorene Witze	Nutzen	Szenario 3 Verfügbar- keit	Nutzen
best-case	8 Mio.	100	0 %	100	100 %	100
soll	4 Mio.	90	0 %	100	95 %	95
ist	2 Mio.	45	30 %	30	70 %	70
worst-case	0	0	100 %	0	0 %	0

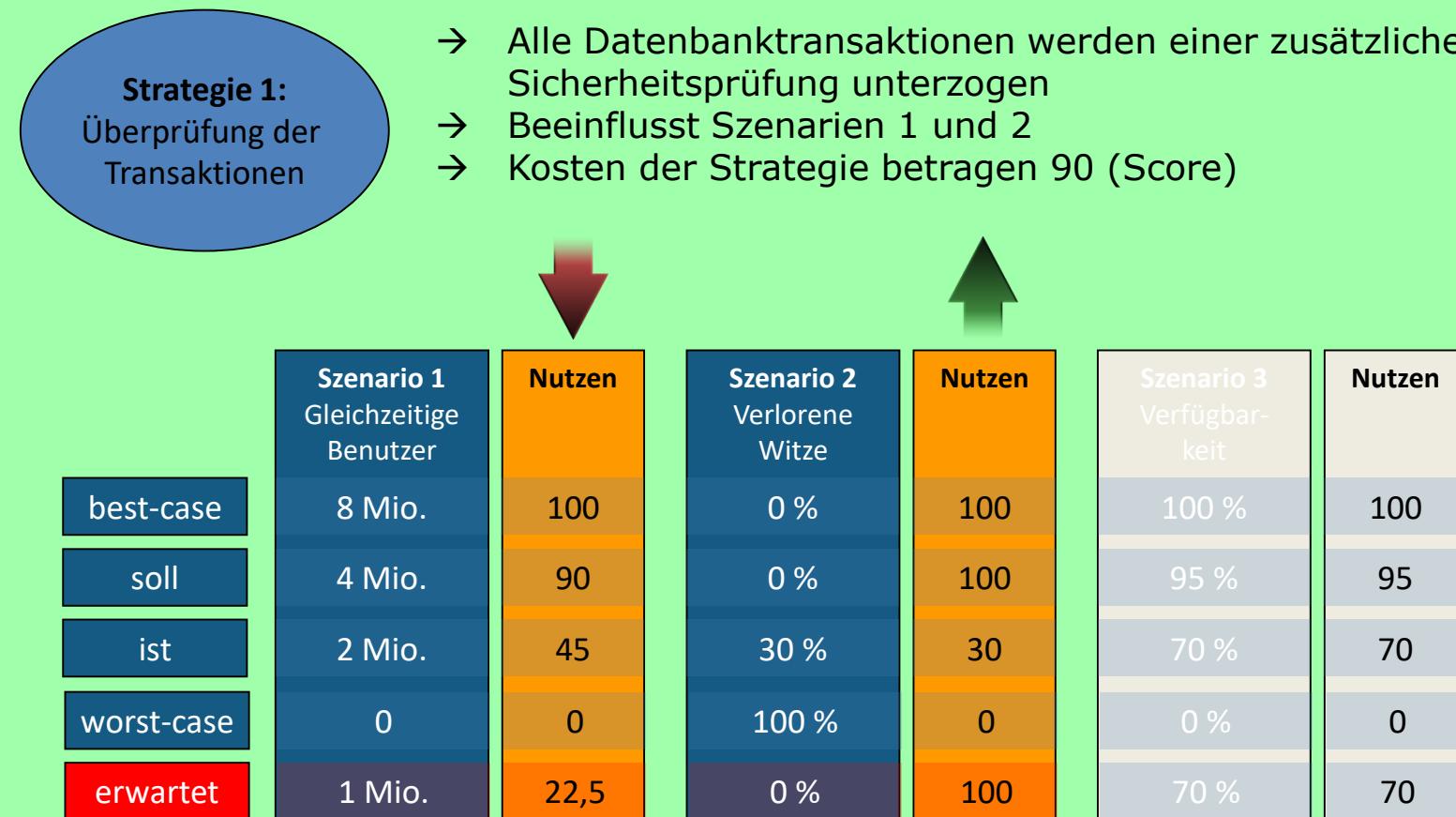
- CBAM läuft in neun Schritten ab



Schritt 5 & 6: Strategien entwickeln und Nutzen bestimmen

- Strategien entwickeln, die die Szenarien adressieren um die Qualitätsattribute auf Soll-Niveau zu bringen
- Diese Aufgabe wird vom Architekten(team) wahrgenommen
- Erwartete Ausprägungen der Qualitätsattribute von jedem Szenario sowie dazugehörigen erwarteten Nutzen durch Interpolation bestimmen
- Es kann durchaus sein, dass eine Strategie Auswirkungen auf mehrere Qualitätsattribute hat, dies sind Nebeneffekte bzw. Trade-Offs
- Schließlich müssen die erwarteten Kosten für jede Strategie mit einem zur Architektur und zur Umgebung passenden Kosten-Modell bestimmt werden

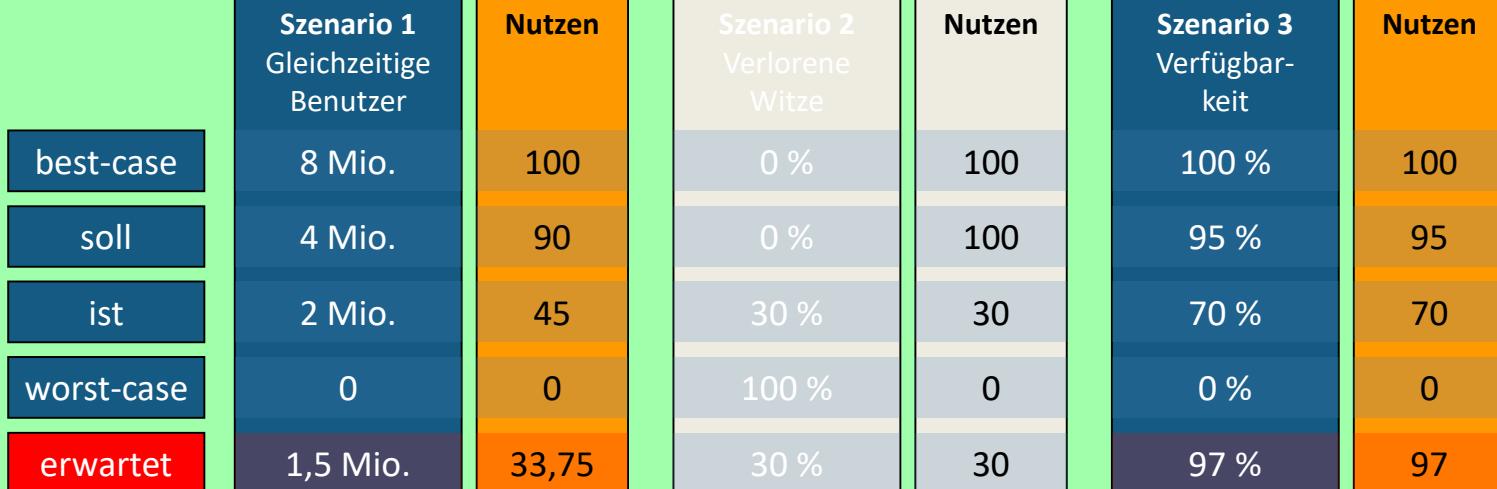
Schritt 5: Beispiel



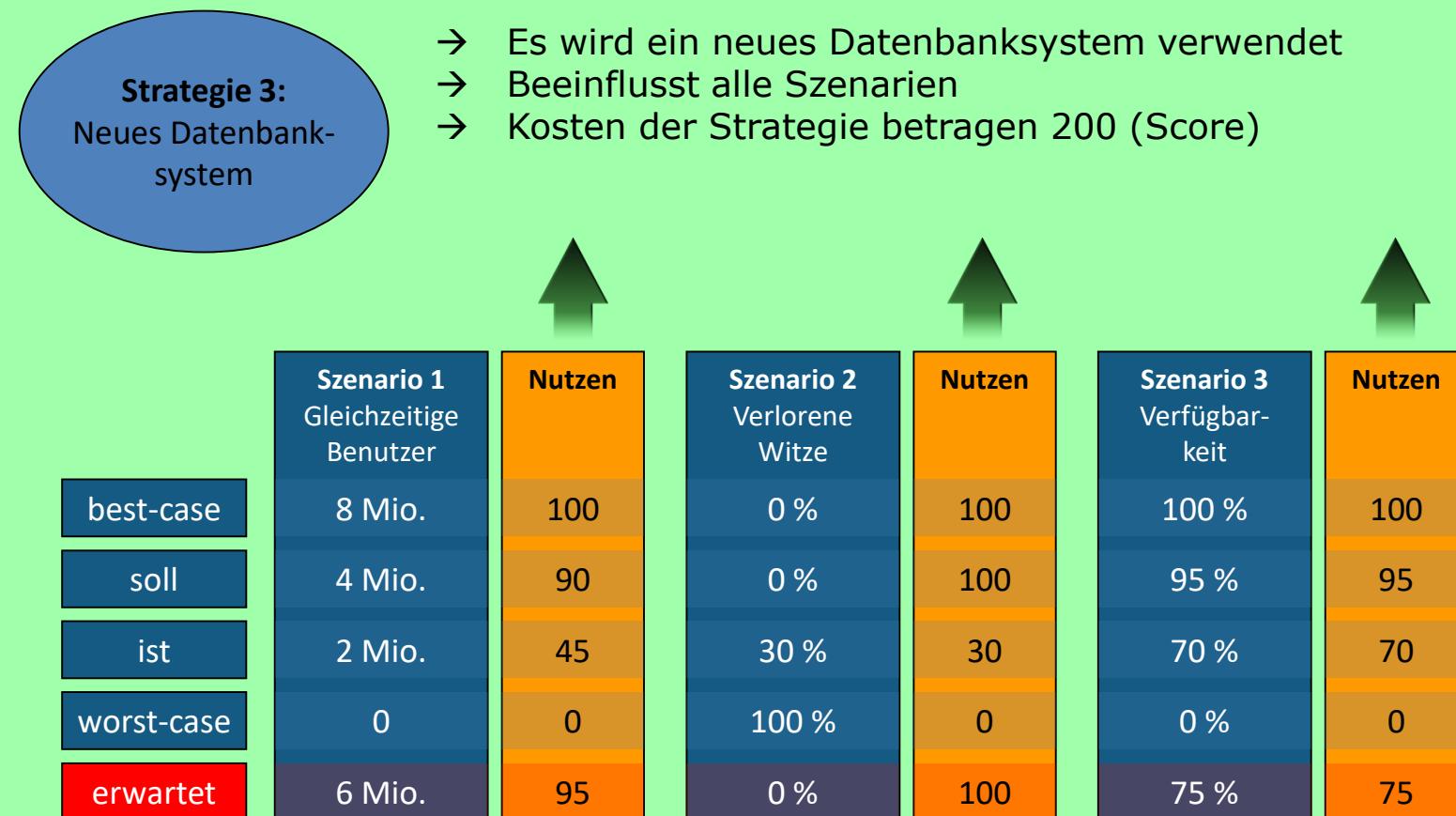
Schritt 5: Beispiel

Strategie 2:
Firewall
installieren

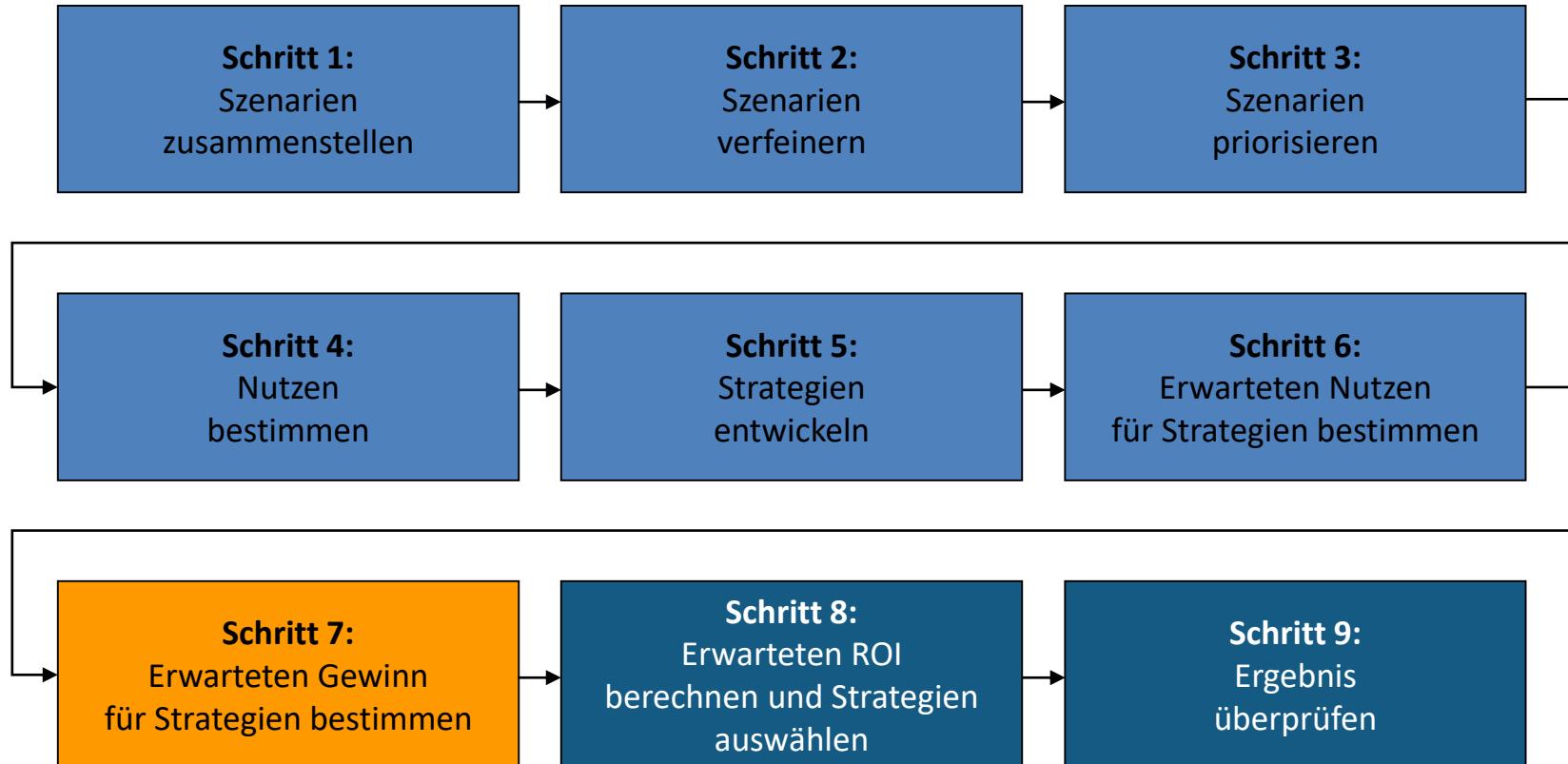
- Es wird eine Firewall installiert um externe Angriffe abzublocken
- Beeinflusst Szenarien 1 und 3
- Kosten der Strategie betragen 45 (Score)



Schritt 5: Beispiel



- CBAM läuft in neun Schritten ab



Schritt 7: Erwarteten Gewinn für die Strategien bestimmen

- Für jede Strategie i wird nun der erwartete Gewinn B_i wie folgt bestimmt
- Zunächst wird die absolute Nutzenänderung bei jedem von der Strategie i betroffenen Szenario j berechnet:

$$b_{i,j} = U_{expected} - U_{current}$$

- Anschließend wird die Nutzenänderung gewichtet und zum erwarteten Gewinn der Strategie i aufsummiert

$$B_i = \sum_j (b_{i,j} * w_j)$$

Schritt 7: Beispiel



	Szenario 1 Gleichzeitige Benutzer	Nutzen	Szenario 2 Verlorene Witze	Nutzen	Szenario 3 Verfügbar- keit	Nutzen
best-case	8 Mio.	100	0 %	100	100 %	100
soll	4 Mio.	90	0 %	100	95 %	95
ist	2 Mio.	45	30 %	30	70 %	70
worst-case	0	0	100 %	0	0 %	0
erwartet	1 Mio.	22,5	0 %	100	70 %	70

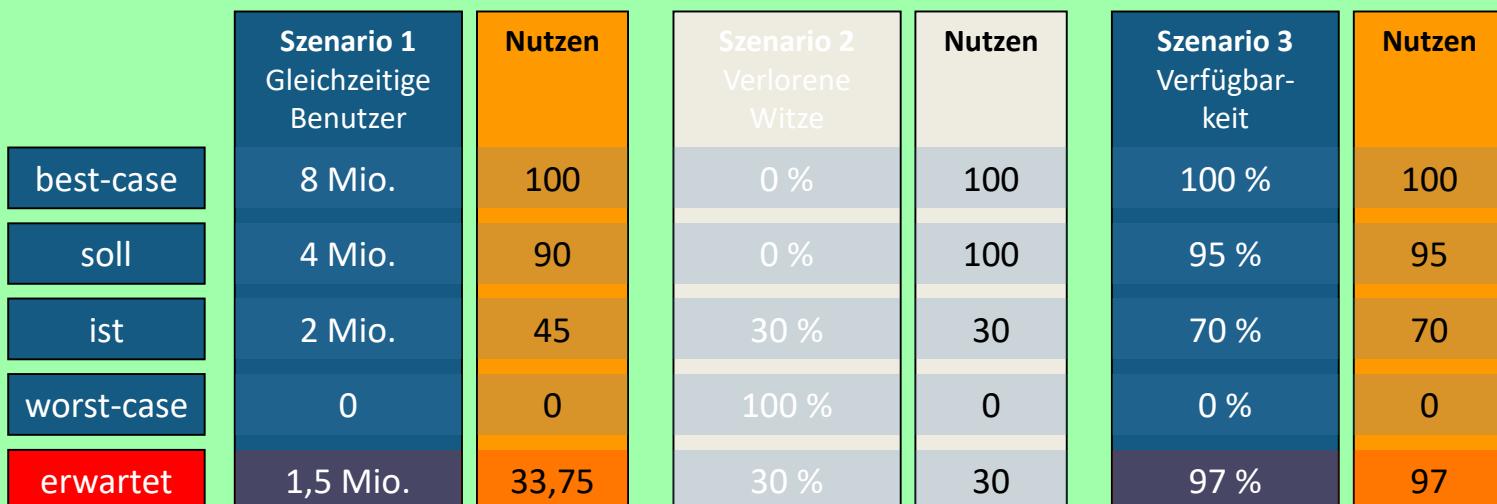
Schritt 7: Beispiel

Strategie 2:
Firewall
installieren

$$\rightarrow b_{2,1} = 33,75 - 45 = -11,25$$

$$\rightarrow b_{2,3} = 97 - 70 = 27$$

$$\rightarrow B_2 = (-11,25 * 0,32) + (27 * 0,25) = 3,15$$



	Szenario 1 Gleichzeitige Benutzer	Nutzen
best-case	8 Mio.	100
soll	4 Mio.	90
ist	2 Mio.	45
worst-case	0	0
erwartet	1,5 Mio.	33,75

	Szenario 2 Verlorene Witte	Nutzen
best-case	0 %	100
soll	0 %	100
ist	30 %	30
worst-case	100 %	0
erwartet	30 %	30

	Szenario 3 Verfügbar- keit	Nutzen
best-case	100 %	100
soll	95 %	95
ist	70 %	70
worst-case	0 %	0
erwartet	97 %	97

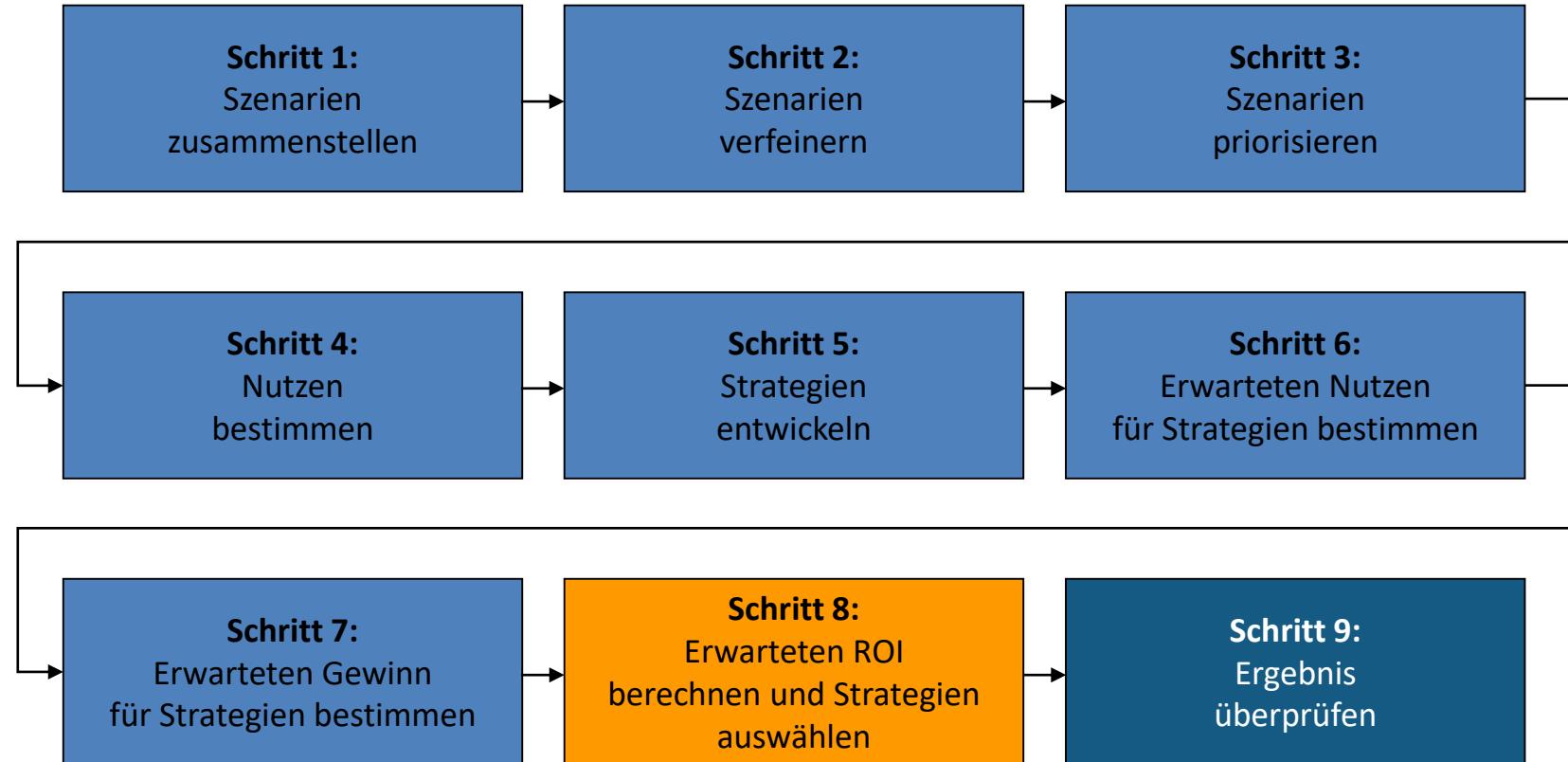
Schritt 7: Beispiel



	Szenario 1 Gleichzeitige Benutzer	Nutzen	Szenario 2 Verlorene Witze	Nutzen	Szenario 3 Verfügbar- keit	Nutzen
best-case	8 Mio.	100	0 %	100	100 %	100
soll	4 Mio.	90	0 %	100	95 %	95
ist	2 Mio.	45	30 %	30	70 %	70
worst-case	0	0	100 %	0	0 %	0
erwartet	6 Mio.	95	0 %	100	75 %	75

$$\begin{aligned}\rightarrow b_{3,1} &= 95 - 45 = 50 \\ \rightarrow b_{3,2} &= 100 - 30 = 70 \\ \rightarrow b_{3,3} &= 75 - 70 = 5 \\ \rightarrow B_3 &= (50 * 0,32) + (70 * 0,43) + \\ &\quad (5 * 0,25) = 47,35\end{aligned}$$

- CBAM läuft in neun Schritten ab



Schritt 8: Erwarteten ROI der Strategien bestimmen

- Für jede Strategie i wird nun der erwartete ROI Score R_i (Return on Investment) bestimmt

$$R_i = \frac{B_i}{C_i}$$

- Schließlich werden die Strategien nach dem ROI Score geranked
- Zusätzlich muss noch die Implementierungsduer berücksichtigt werden
- Die Strategien mit dem höchsten Ranking werden nacheinander ausgewählt, bis das Budget erschöpft ist

Schritt 8: Beispiel

Strategie 1:
Überprüfung der
Transaktionen

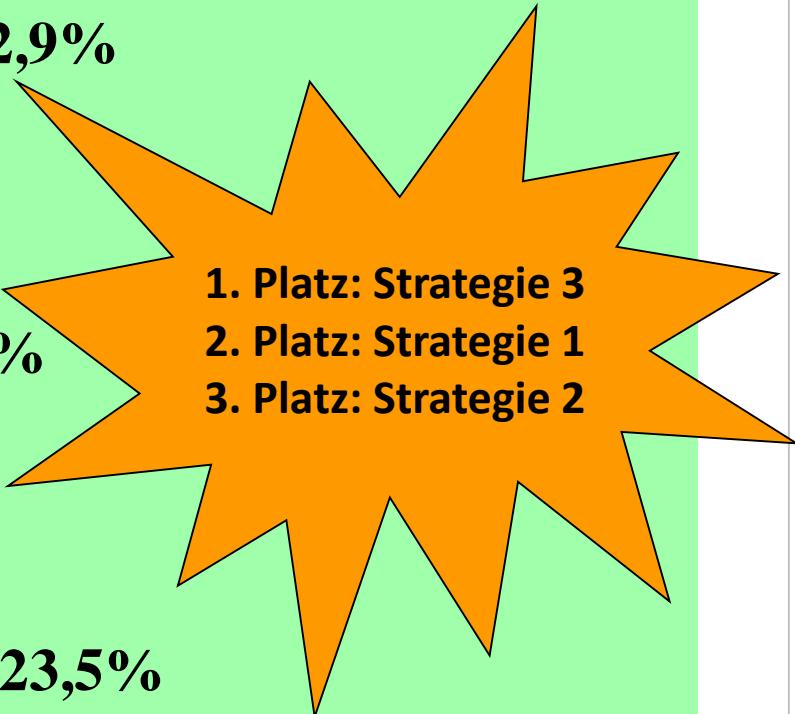
Strategie 2:
Firewall
installieren

Strategie 3:
Neues Datenbank-
system

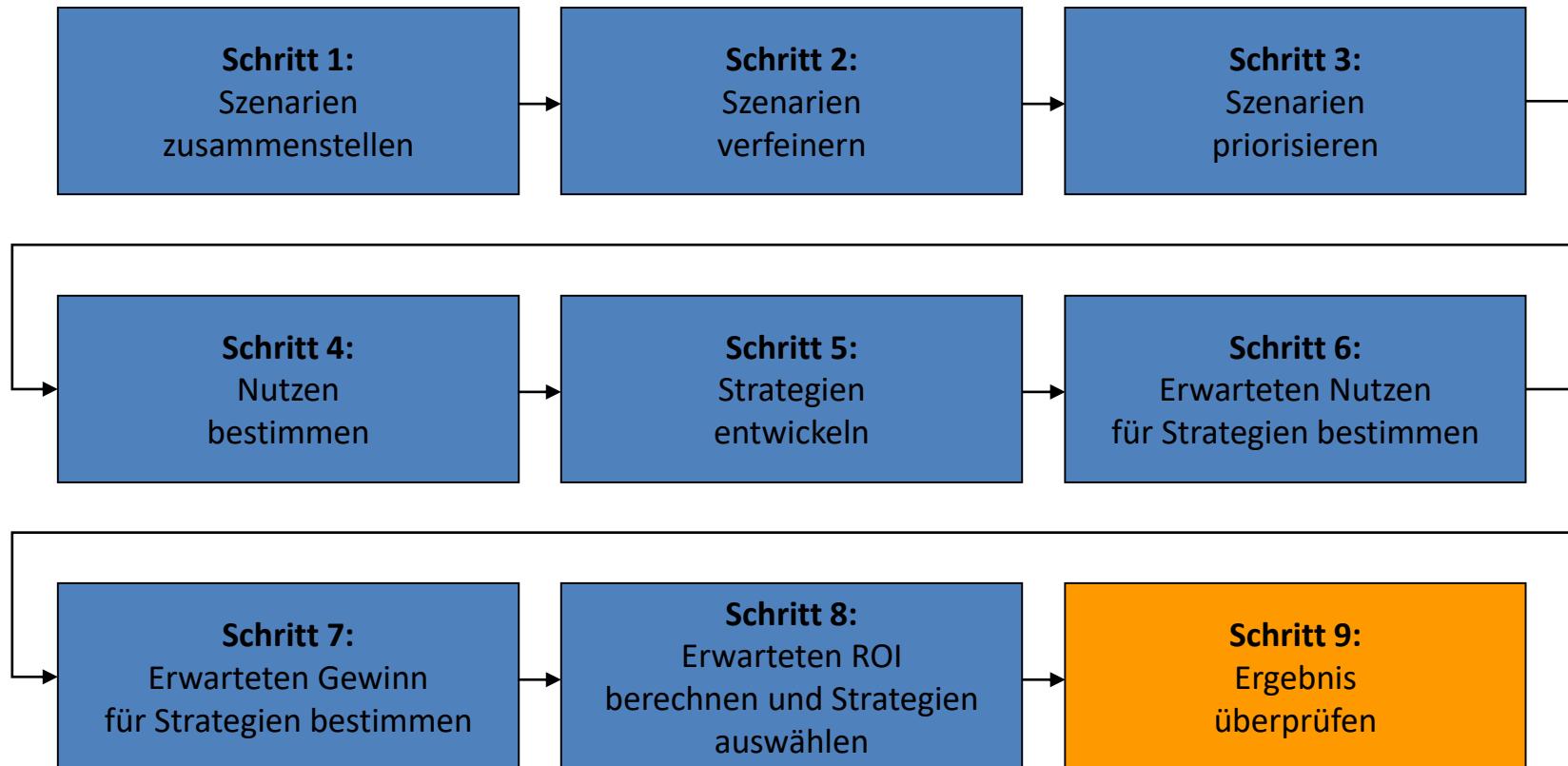
$$R_1 = \frac{B_1}{C_1} = \frac{22,9}{100} = 22,9\%$$

$$R_2 = \frac{B_2}{C_2} = \frac{3,15}{45} = 7\%$$

$$R_3 = \frac{B_3}{C_3} = \frac{47,35}{200} = 23,5\%$$



- CBAM läuft in neun Schritten ab



- Schritt 9: Gewählte Strategien überprüfen
- Niemals blind auf quantitative Verfahren verlassen
 - › Darüber nachdenken, ob das Ergebnis der CBAM Analyse Sinn macht
 - › Wurden andere wichtige Qualitätsattribute übersehen?

■ Bewertung von CBAM

CBAM

Vorteile

- Bezieht nicht nur Kosten, sondern auch Nutzen, Risiken und Terminimplikationen mit ein
- Betrachtet obige Größen während Implementierungs- und Wartungsphase
- Versuch mit einem rationalen und stabilen Prozess Architekturen zu bewerten

Nachteile

- Viele Schätzungen und Annahmen im Modell
- Unter Umständen falsche Entscheidung, da wichtige qualitative Faktoren außer acht gelassen werden
- Kann nur ergänzend eingesetzt werden
- Stakeholder können das Verfahren subjektiv beeinflussen
- Keine Sensitivitätsanalyse

Vergleich der Bewertungsmethoden

	ATAM	SAAM	ARID	CBAM
Betrachtete Qualitätsattribute	Keine spezifischen QA jedoch historisch: Veränderbarkeit, Sicherheit, Zuverlässigkeit, Leistung	Primär Veränderbarkeit und Funktionalität	Eignung des gewählten Design Ansatzes	Keine spezifischen QA
Analysierte Objekte	Architektur-ansätze und Architekturstile	Architekturdokumentation	Schnittstellenbeschreibungen der Elemente	Architekturstrategien
Projektstadium bei Anwendung	Nach der Auswahl der Architektur-ansätze	Nach der Zuweisung der Funktionalitäten zu den Modulen	Während dem Architektur-design	Nach der ATAM Bewertung
:	:	:	:	:

Vergleich der Bewertungsmethoden

	ATAM	SAAM	ARID	CBAM
Verwendete Methoden	Utility Trees & Szenarien um Anforderungen an QA zu analysieren. Analyse der Architekturansätze zur Bestimmung von Sensitivity-Points, Trade-Off-Points, Risiken.	Szenarien um Anforderungen an QA zu analysieren. Szenario walk-throughs um Funktionalität zu testen und Änderungskosten zu bestimmen.	Szenarien. Active Design Reviews.	Szenarien um Anforderungen an QA zu analysieren. Nutzwert-analyse um Nutzen zu bestimmen. Strategie-analyse um Kosten und Auswirkungen zu bestimmen.
Benötigte Ressourcen	3 Tage plus Vor- & Nachbereitung. Teilnehmer sind Kunde, Architekt, Stakeholder und 4-köpfiges Evaluations-Team.	2 Tage plus Nachbereitung. Teilnehmer sind Kunde, Architekt, Stakeholder und 3-köpfiges Evaluations-Team.	2 Tage plus Vor- & Nachbereitung. Teilnehmer sind Architekt, Stakeholder und 2-köpfiges Evaluations-Team.	2 Tage à 4h plus Vorbereitung. Teilnehmer sind Kunde, Architekt, Stakeholder und 3-köpfiges Evaluations-Team.

- Bass/Clements/Kazman (2005): Software Architecture in Practice; Second Edition; Addison-Wesley
- Clements/Kazman/Klein (2002): Evaluating Software Architectures; Addison-Wesley
- Hoffer/George/Valacich (2005): Modern Systems Analysis and Design; Fourth Edition; Pearson Education
- Kazman/Asundi/Klein (2002): Making Architecture Design Decisions: An Economic Approach; CMU/SEI-2002-TR-035
- Moore/Kazman/Klein/Asundi (2003): Quantifying the Value of Architecture Design Decisions: Lessons from the Field

- Einführung in Software-Architekturen und Organisation
- Entwurf von SW-Architekturen
- Dokumentation von Software-Architekturen
- Evaluation / Bewertung von SW-Architekturen
- Toolbox des Softwarearchitekten
 - › Motivation
 - › Lösungsvorlagen & Methoden
 - » Architekturstile
 - » Architekturmuster
 - » Entwurfsmuster
 - › Technologien & Werkzeuge
 - » Betriebssysteme & Programmiersprachen
 - » Bibliotheken & Frameworks
 - » Modellierung & Generierung
 - » Analyse & Rekonstruktion
 - › Zusammenfassung
- Produktlinien für Software
- Enterprise Architecture Management

Problem

Der Nagel muss in das Brett



Lösung

„Werkzeug“ Hammer



Motivation „Toolbox“

Problem

Soll der Vorstand der ABC AG die XYZ GmbH übernehmen?

Lösung

„Werkzeug“ Geschäftsbericht,
„Werkzeug“ Rating, etc.



The screenshot shows the Deutsche Bank Annual Report 2004 website. The main header reads "Geschäftsbericht 2004" and "Deutsche Bank". Below the header, there's a navigation bar with links like "Download", "Datensammlung", "Info Services", "Bestellservice", "Feedback", and "Schlagworte". A sidebar on the left lists menu items such as "Brief des Vorstandsvorsprechers", "Konzern Deutsch Bank", "Zielgruppen", "Lagebericht", and "Konzernabschluss". The central content area displays the "Konzernbilanz" (Consolidated Balance Sheet) in a tabular format. The table has columns for "Aktiva" (Assets), "Passiva" (Equity and Liabilities), and "Handelsaktivia" (Trading Assets). The table includes detailed financial figures and notes. To the right of the main content, there's a sidebar titled "Service Funktionen" with links to "Mehr Informationen", "Vorjahresvergleich", and "Jahres-Pressekonferenz 2005 Analytikerkonferenz".

Problem

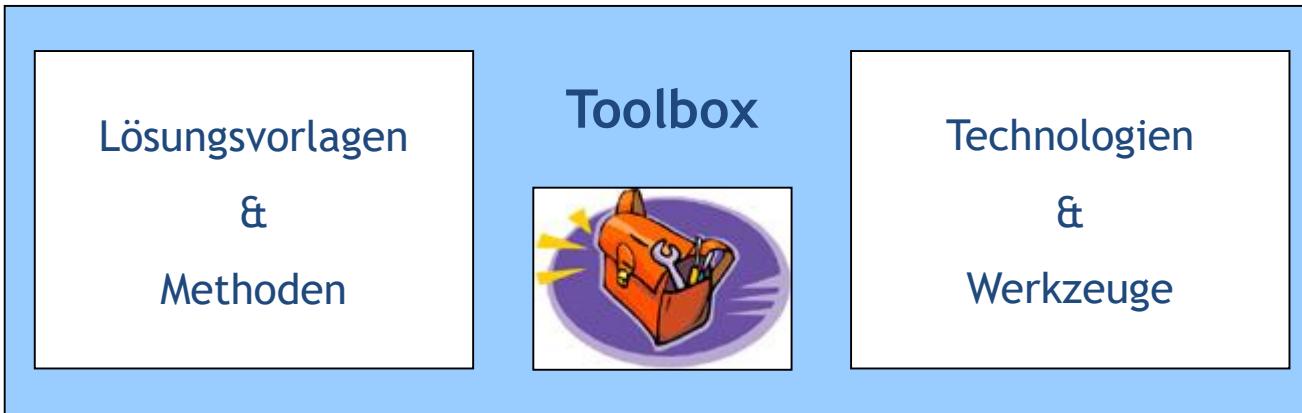
Entwurf einer Softwarearchitektur



Lösung

„Toolbox des Softwarearchitekten“





- katalogisierte *Lösungsvorlagen & Methoden*
 - › Wissenskatalog
 - › Methoden, Muster, ...
- *Technologien & Werkzeuge*
 - › Betriebssysteme, Programmiersprachen
 - › Frameworks
 - › Software (CASE-Tools, ...)

Vorteile durch den Einsatz einer Toolbox

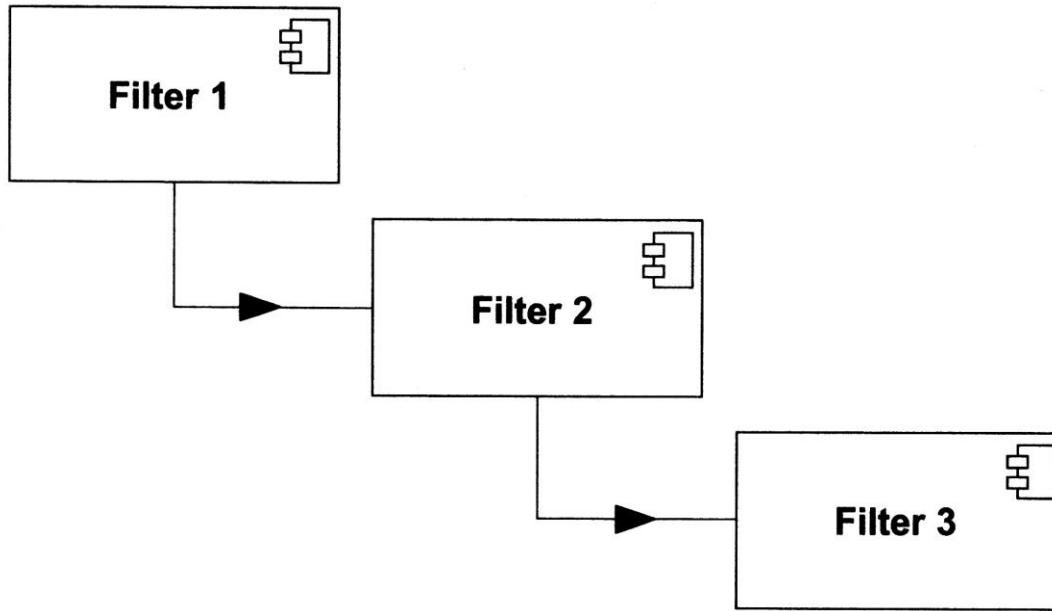
- Jede Problemlösung muss nicht immer neu erfunden werden
- Bewährte Lösungen
 - *Minimierung von Risiken*
- Nutzung von Standardelementen aus der Toolbox
 - *Minimierung der Komplexität*
- Schnellerer Entwurf der Architektur
 - › Viele Bausteine und ihr Zusammenspiel ist bekannt
 - › *Reduktion des Aufwands*
- Besseres Teamwork
 - › namentliche Methoden und Werkzeuge
 - › Diskussionsgrundlage
- Problem: Toolbox muss *up to date* gehalten werden!

Bezeichnung	Zweck	Beispiele
<i>Architekturstil</i>	globale Prinzipien zur Strukturierung	Pipes and Filters, Blackboard, Layers
<i>Architekturnuster</i>	querschnittliche Aspekte	Nebenläufigkeit, Persistenz
<i>Entwurfsmuster</i>	lokale Anwendung von bewährtem Entwurfswissen	Observer, Abstract Factory, Singleton

- globale Strukturierungs- bzw. Organisationsprinzipien für Softwarearchitekturen
- Architekturen sind instanzierte Architekturstile
- in der Literatur oft auch als architectural pattern bekannt
- Der Architekturstil beeinflusst die gesamte Softwarearchitektur

- Vermischung von Architekturstilen
 - › problematisch, da die Auswirkungen der Stile evtl. gegenläufig sind
 - › auf unterschiedlichen Abstraktionsebenen in der Regel unproblematisch
 - › sollte wegen der konzeptuellen Integrität vermieden werden

- Architekturstile (nach [Bosch00]):
 - › Pipes and Filters
 - › Blackboard
 - › Layers
- Der Architekturstil hat u.a. Einfluss auf die folgenden Qualitätsattribute:
 - › Leistung
 - › Flexibilität
 - › Sicherheit, ...
- Um Qualitätsanforderungen zu erfüllen, kann ein bestimmter Stil genutzt werden



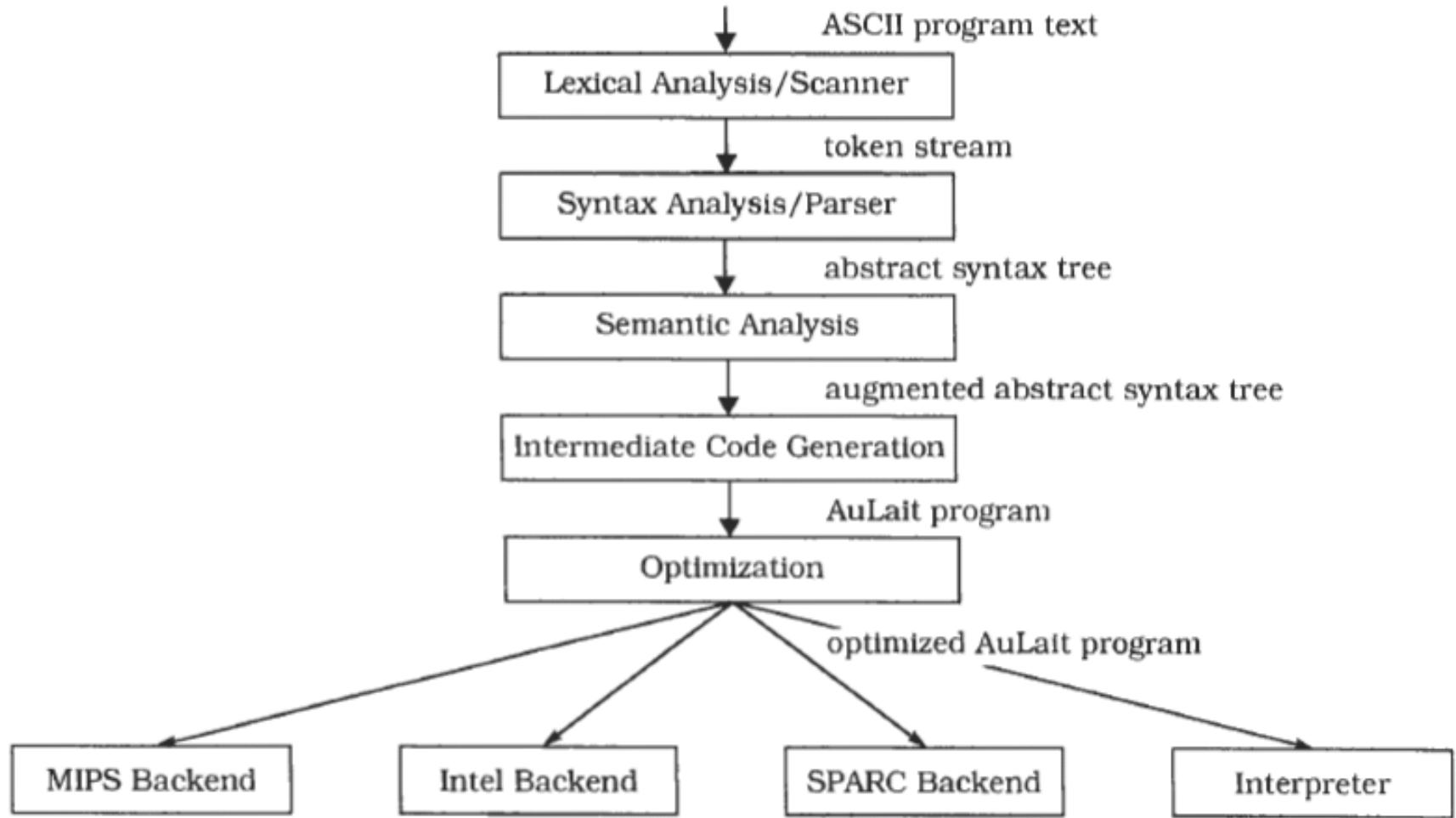
aus [Posch04]

- Jeder Baustein hat Eingänge und Ausgänge
- Die Verbindungen zwischen den Bausteinen heißen *Pipes*
- Durch diese Pipes fließt ein Datenstrom

Pipes and Filters

Einsatz	Sehr gut geeignet für die Verarbeitung von kontinuierlichen Datenströmen.
Vorteile	<ul style="list-style-type: none">▪ Filter arbeiten asynchron und unabhängig voneinander▪ Einfache Aufteilung auf mehrere Entwicklungsteams▪ Einfache Wiederverwendung in einem anderen Kontext
Nachteile	<ul style="list-style-type: none">▪ Geringe Performance wegen dem Datentransport▪ Schlechte Wartbarkeit wegen der Abhängigkeiten der Filter▪ Geringe Zuverlässigkeit und Sicherheit durch die Verkettung
Beispiele	<ul style="list-style-type: none">▪ Compiler▪ UNIX-Programme

Architekturstil: Pipes and Filters

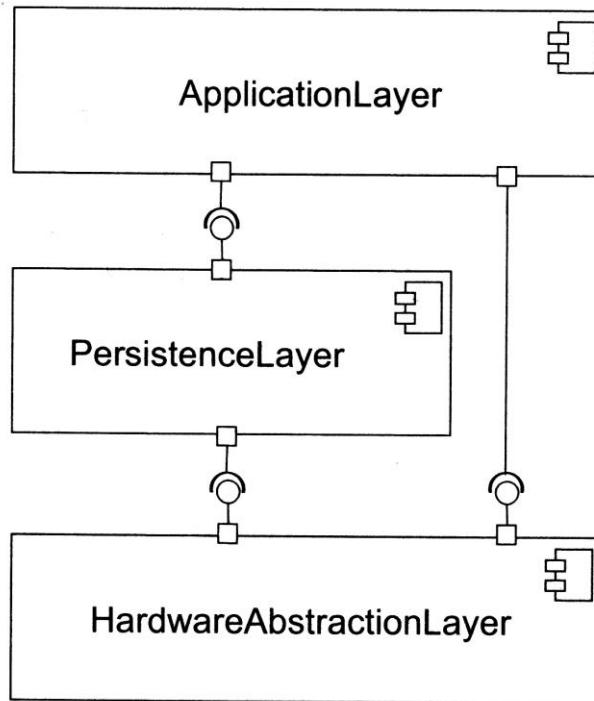


- Alle Bausteine des Stils sind um einen zentralen Baustein (*Blackboard*) angeordnet
- Blackboard ist für die zentrale Datenhaltung zuständig
- Die anderen Bausteine
 - › überwachen den Inhalt des Blackboards
 - › erkennen, wann sie Aufgaben von dem Blackboard aufgreifen und bearbeiten
 - › und schreiben die Lösung dann auf das Blackboard zurück

Blackboard

<i>Einsatz</i>	sehr gut geeignet, wenn es für Probleme keine deterministischen Lösungsstrategien gibt
<i>Vorteile</i>	<ul style="list-style-type: none">▪ Möglichkeit zum Experimentieren mit Lösungsansätzen▪ Gute Änderbarkeit und Wartbarkeit
<i>Nachteile</i>	<ul style="list-style-type: none">▪ Fehlender Determinismus → keine garantiierte Lösung▪ Zentrale Datenhaltung
<i>Beispiele</i>	<ul style="list-style-type: none">▪ Spracherkennungssysteme▪ Aktien-Trendanalyse

- Jede horizontale Schicht
 - › repräsentiert eine bestimmte Abstraktionsebene
 - › verwendet Operationen der darunter liegenden Schichten
 - › bietet der nächst höheren Schichte komplexere Operationen an

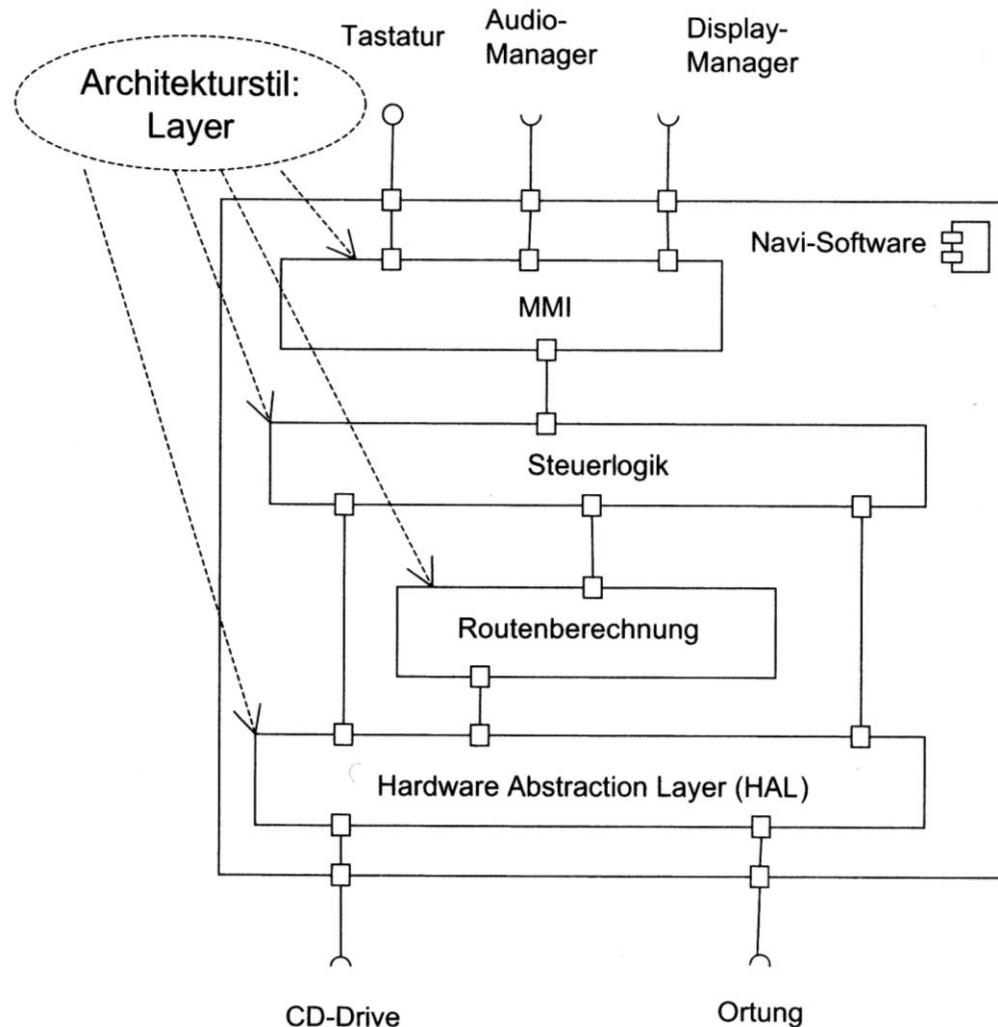


aus [Posch04]

Layers

<i>Einsatz</i>	sehr gut geeignet für die Aufteilung großer Systeme in abstrahierte Komponenten
<i>Vorteile</i>	<ul style="list-style-type: none">▪ Verfügbarkeit von verschiedenen Abstraktionsebenen▪ Wiederverwendung von Schichten▪ Erhöhte Sicherheit durch Einfügen von Überwachungsschichten
<i>Nachteile</i>	<ul style="list-style-type: none">▪ Geringere Performance▪ Richtige Granularität der Schichten?
<i>Beispiele</i>	<ul style="list-style-type: none">▪ ISO-OSI-7-Schichten-Modell für Kommunikationssysteme▪ Typische Betriebssysteme (Treiber-Kernel-UI)▪ Informationssysteme (DB – Anwendungslogik - GUI)

Architekturstil: Layers



aus [Posch04]

- Architekturmuster bieten Lösungen für so genannte *Querschnittsaspekte*
 - › Nebenläufigkeit
 - › Persistenz
 - › Verteilung
 - › Graphische Benutzerschnittstelle
- Unabhängig vom gewählten Architekturstil
- Architekturmuster adressieren bestimmten Teil der Systemfunktionalität
 - › nicht fachlich, sondern technisch
 - › Auswirkungen auf die Qualitätsattribute des Systems

Problem

- Anwendung führt eine größere Berechnung durch
- GUI aktualisieren ständig die angezeigten Daten
- Benutzer wechselt zwischen verschiedenen Ansichten für die Daten

Alles soll gleichzeitig in der Anwendung möglich sein

Lösung

- Architekturmuster für die Nebenläufigkeit der verschiedenen Aufgaben
- (Parallelität)

Nebenläufigkeit

<i>Prozesse des Betriebssystems</i>	<ul style="list-style-type: none">▪ Jeder Baustein hat als Prozess seinen eigenen Adressbereich▪ Durch den eigenen Adressraum geschützt▪ Kommunikation über Nachrichten oder gemeinsamer Speicher
<i>Threads des Betriebssystems</i>	<ul style="list-style-type: none">▪ Leichtgewichtige Prozesse▪ Einfachere Kommunikation durch gemeinsamen Adressraum▪ Taskwechsel ist weniger aufwendiger als bei Prozessen▪ Synchronisation beim Zugriff auf gemeinsame Daten

Nebenläufigkeit

<i>Kooperative Threads</i>	<ul style="list-style-type: none">▪ Jeder Thread bestimmt selbst, wann er den Prozessor freigibt▪ Benötigt Spezialwissen der Entwickler (→ Deadlock)▪ Nötig für Echtzeitanforderungen oder▪ Nötig für spezielle Leistungseigenschaften
<i>Scheduling auf Applikationsebenen</i>	<ul style="list-style-type: none">▪ Bei Entwicklung ohne Betriebssysteme▪ Applikation übernimmt Zuweisung des Prozessors (Scheduling)▪ Benötigt Spezialwissen der Entwickler

Problem

Nach dem Beenden und erneuten Starten der Anwendung sollen die Daten unverändert vorliegen

Lösung

Architekturmuster für Persistenz

- Lebensdauer der Daten wird über die Lebensdauer der Anwendung hinaus verlängert
- bei Objektorientierung: Beziehung zwischen Objekten und dauerhaften Daten
- Transaktionen: Änderung mehrerer Daten als einzelne Operation

Persistenz

<i>Datenbank-managementsystem (DBMS)</i>	<ul style="list-style-type: none">▪ Auf effiziente Speicherung von Daten optimierte Software▪ Relationale Datenbanken mit SQL▪ In der Regel bieten DBMS Unterstützung für Transaktionen
<i>Persistenz auf Applikationsebene</i>	<ul style="list-style-type: none">▪ Wenig Ressourcen oder wenig benötigte Funktionalität → kein DBMS▪ Applikation muss die Persistenz der Daten sicherstellen▪ Unterstützung durch Programmiersprache oder Bibliotheken<ul style="list-style-type: none">▶ Serialisierung in Java▶ XML-Bibliotheken (JDOM, etc.)

Problem

Die Bausteine eines Systems laufen nicht nur auf einem Rechner, sondern verteilt auf mehreren Rechner.

- Sind die Verbindungen zwischen den Systemen statisch oder dynamisch?
- Ist die Verteilung transparent für die einzelnen Bausteine?
- Wie kommunizieren die einzelnen Bausteine untereinander?

Lösung

Architekturmuster für Verteilung

Verteilung

<i>Vermittler</i>	<ul style="list-style-type: none">▪ Stellt Funktionalität bereit,<ul style="list-style-type: none">▶ Damit sich Bausteine gegenseitig finden können▶ Damit sich Bausteinen zueinander verbinden können▪ Keine statische Verbindung → erhöhte Flexibilität
<i>Entfernter Methodenaufruf (RPC/RMI)</i>	<ul style="list-style-type: none">▪ Aufruf von Methoden eines Bausteins/Objekts auf einem anderen Rechner▪ Verpacken und Auspacken der Parameter notwendig▪ Aufwendiger als lokaler Methodenaufruf

Problem

- Wie trennt man die graphische Benutzeroberfläche sinnvoll von der Anwendungslogik?

Lösung

Architekturmuster für graphische Benutzerschnittstellen

- Graphische Benutzerschnittstelle muss in der Regel öfters geändert und angepasst werden, als die Kernfunktionalität des Systems.

Graphische Benutzerschnittstellen

Model-View-Controller	<ul style="list-style-type: none">▪ Model: Datenbasis und Kernfunktionalität▪ View: Ansichten präsentieren die Daten▪ Controller: Steuerungskomponente verarbeiten Eingaben▪ Durch Notifikationsmechanismus wird Konsistenz zwischen Model und Views sichergestellt
Präsentation-Abstraktion-Steuerung	<ul style="list-style-type: none">▪ Funktionale Anforderungen sind auf verschiedene Bausteine verteilt → <i>Agents</i>▪ Jeder Agent hat<ul style="list-style-type: none">▶ Einen Präsentationsteil mit dem sichtbaren Verhalten▶ Einen Abstraktionsteil mit dem Datenmodell▶ Einen Steuerungsteil für die Inter-Agenten-Kommunikation▪ Agenten sind hierarchisch angeordnet

- Entwurfsmuster sind formalisiertes und dokumentiertes Wissen erfahrener Entwickler, welches somit wieder verwendet werden kann.
- Häufig wiederkehrende Problemstellungen beim Softwareentwurf können eben durch diese Entwurfsmuster (engl. *Design Pattern*) gelöst werden.

- Die Entwurfsmuster haben ihren Ursprung in der Architektur. Christopher Alexander hat in den 1970er mehrere Bücher über Muster in der Architektur geschrieben.
- Gamma, Helm, Johnson und Vlissides (*Gang Of Four*) führen die Entwurfsmuster 1994 in die Softwareentwicklung ein (vgl. [Gamma94]).

■ Vorteile

- › *mentale Bausteine*: Man kann während des Entwurfs mit Entwurfsmustern und deren Kombinationen jonglieren
- › *effektives Kommunikationsmittel*: Da die Namen der Muster geläufig sind, stellen sie ein gemeinsames Verständnis her
- › *verbesserte Qualitätsattribute*: Jedes Muster hat positive und negative Auswirkungen. Durch gezielten Einsatz von Mustern lassen sich die Qualitätsattribute beeinflussen
- › *Wiederverwendung von bewährten Lösungen*

■ Entwurfsmuster wirken lokal

- › Sie berühren nicht die Gesamtarchitektur
- › Entwurfsmuster können ohne die konzeptuelle Integrität zu verletzen kombiniert werden

- Voraussetzung für die effektive Verwendung eines Pattern:
- Das richtige Pattern schnell und einfach identifizieren!
- Eine reine Auflistung der Muster ist zu wenig, da
 - › sich Muster gegenseitig ergänzen
 - › Muster oft kombiniert eingesetzt werden
- Klassifikation von Entwurfsmustern
 - › Erzeugungsmuster (creational patterns)
 - › Strukturmuster (structural patterns)
 - › Verhaltensmuster (behavioral patterns)
- Nach [Posch04] sollte sich ein Softwarearchitekt ein Vokabular von 20-30 Muster erarbeiten

Name	Essenz des Musters
Klassifizierung	Einordnung in die geltende Klassifizierung
Grundprinzip	Was leistet das Pattern? Welches Problem löst es?
Motivation	Szenario zur Darstellung des Problems
Struktur	Graphisches Modell (z.B. in UML)
Rollen	beteiligte Objekte und ihre Rolle
Kooperation	Wie arbeiten die Klassen zusammen?
Konsequenzen	Ziele und Auswirkungen des Pattern
Implementierung	bspw. Techniken, ...
Beispiel	Anwendungsfall und beispielhafte Implementierung
Verwandte Muster	Beziehung zu anderen Mustern

Problem

- In einem System ändern sich Daten. Nun soll
 - › die jede Anzeige in der GUI aktualisiert werden.
 - › der Benutzer informiert werden.
 - › ...

Vorschlag 1

- Alle Element fragen nach, ob sich die Daten geändert haben.
- → Schlecht, da unter Umständen viele unnötige Anfragen.

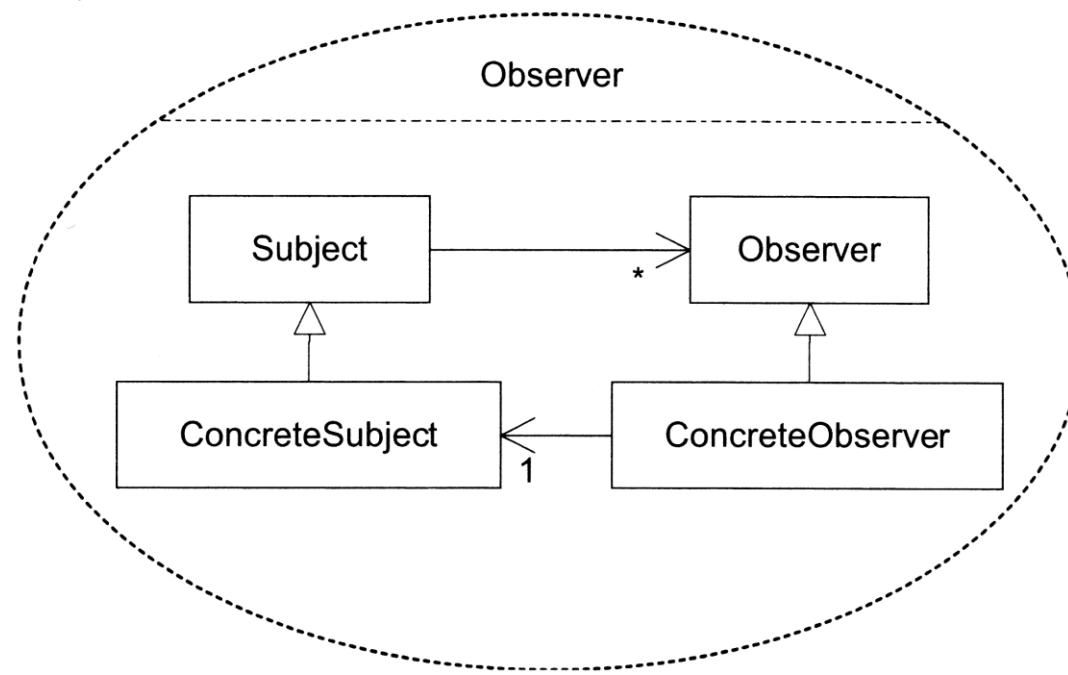
Problem

- In einem System ändern sich Daten. Nun soll
 - › die jede Anzeige in der GUI aktualisiert werden.
 - › der Benutzer informiert werden.
 - › ...

Vorschlag 2

- Die Elementen werden über die Änderungen informiert.
- → Aber wie können viele unterschiedliche Elemente informiert werden?

Beispiel für Design Patterns (I)



*Lösung: **Observer Pattern***

- Observer Pattern
 - › auch bekannt als *Publish-Subscribe-Pattern*
 - › Verhaltensmuster (*behavioral pattern*)
- Vorteile
 - › direkte Benachrichtigung, kein Aufwand durch Überwachung
 - › Beobachter und Subjekt sind entkoppelt
 - › neue Beobachter können einfach hinzugefügt werden
- Nachteile
 - › Vorsicht bei einer kaskadierenden Anwendung des Pattern
 - › Zyklen
- Beispiele
 - › GUIs in Java (AWT, Swing)

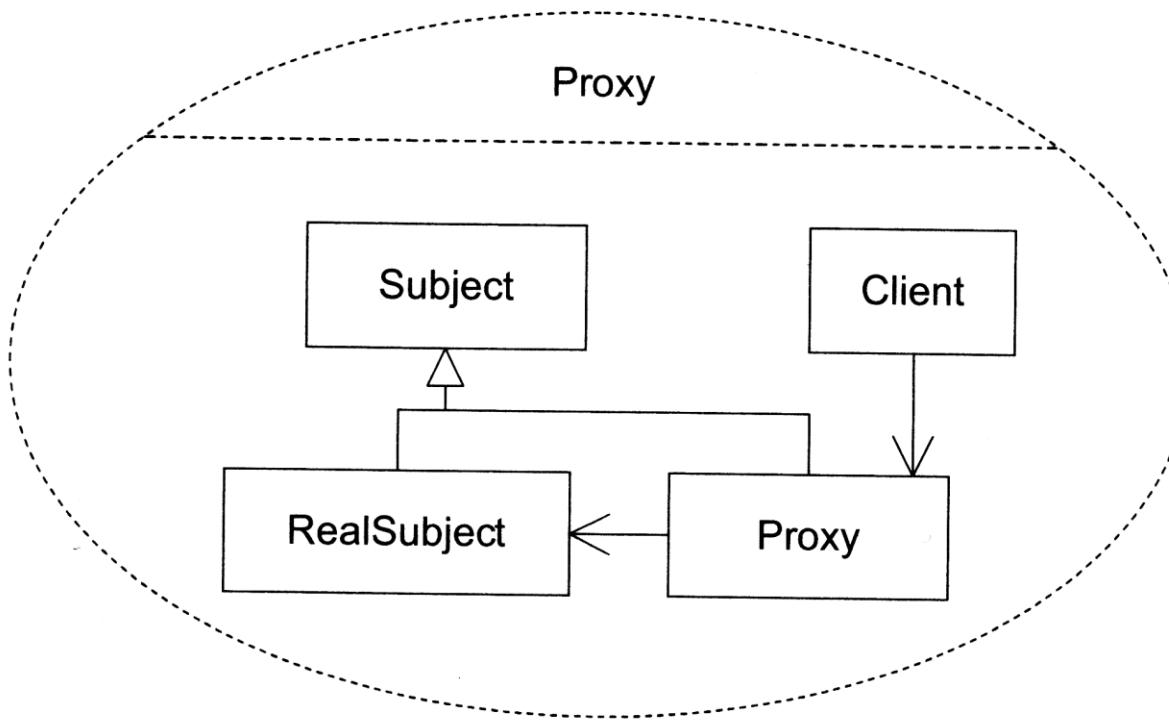
Problem

In einer Anwendung muss über das Netzwerk auf sehr große Objekte zugegriffen werden und ein Teil der im Objekt gespeicherten Daten gelesen werden. Die Übertragung der Objekte ist über das Netzwerk sehr langsam.

Vorschlag

- Da Objekt wird geladen und in einem Cache gespeichert.
- → Was passiert wenn sich die Daten öfters ändern?

Beispiel für Design Patterns (II)



Lösung: Proxy Pattern

Beispiel für Design Patterns (II)

- Proxy (Stellvertreter)
 - › Strukturmuster (*structural patterns*)
- Vorteile
 - › Man muss die eigentliche Komponente nicht kennen
→ Transparenz der Verteilung
 - › Proxy kann Kontrollfunktionalität erledigen
→ Effizienzsteigerung beim dynamischen Nachladen von Daten
→ Sicherheitsüberprüfung vor dem Zugriff auf das wirkliche Objekt
- Beispiele
 - › bei Java-RMI (Stub)
 - › Bei Axis: Es wird auf einen Proxy statt auf den Web Service zugegriffen

Beispiel für Design Patterns (III)

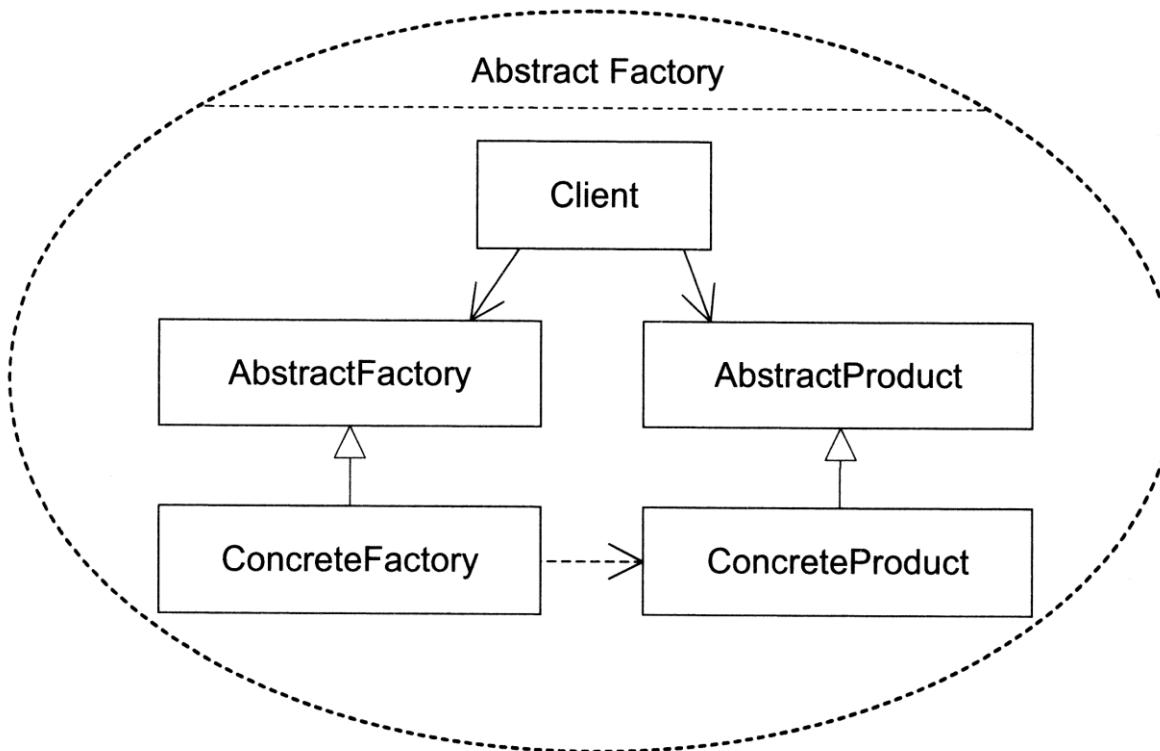
Problem

In einer Anwendung werden Nachrichten erzeugt und verschickt. Abhängig von der Einstellung, dem Netzwerk etc. werden unterschiedliche Nachrichtenformate verwendet. Dennoch haben alle Nachrichtenformate einen Empfänger, Sender, Betreff und den Text gemeinsam.

Vorschlag

- Das Erzeugen und Versenden der Nachrichten wird für jedes Format separat implementiert. Die Anwendung ruft anhand des Format dann die entsprechende Funktion auf.
- → Wie können neue Formate einfach eingefügt werden?

Beispiel für Design Patterns (III)



Lösung: *Abstract Factory*

- Abstract Factory
 - › Erzeugungsmuster (*creational patterns*)
- Vorteile
 - › Prozess der Erstellung ist unabhängig von der konkreten Ausprägung der Produkte
 - › zusätzliche Produkt (inkl. Fabrik) können leicht integriert werden
 - › Erzeugung zusammengehöriger Objekte kann kontrolliert ablaufen
- Beispiele
 - › GUI Toolkits

- Architekturstile und –muster

- › [Buschmann96]

Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad und Michael Stal: Pattern-Oriented Software Architecture: A System of Patterns.
John Wiley & Sons, 1996

- › [Bosch00]

Jan Bosch: Design and use of software architectures. Addison-Wesley, 2000

- Entwurfsmuster

- › [Gamma94]

Erich Gamma, Richard Helm, Ralph Johnson und John Vlissides:
Design Patterns: Elements of Reusable Object-Oriented Software.
Addison-Wesley, 1994

- › [Grand02]

Mark Grand: Patterns in Java, Volume 1. John Wiley & Sons, 2002

- Einsatzbereiche bei der Softwarearchitektur
 - › Entwurf der Architektur
 - Modellierungswerzeuge
 - › Bestandteile der Architektur
 - Betriebssysteme, Bibliotheken, Off-the-Shelf-Komponenten
 - › Umsetzung der Architektur
 - Programmiersprachen
 - › Test der Architektur
 - › Rekonstruktion von Architekturen bestehender Systeme

- engl. *Libraries*
- bei objektorientierten Programmiersprachen
 - › Zusammengehörige Klassen werden in einer Bibliothek abgelegt
 - › Basis für neue Klassen (z.B. durch Vererbung)
- Beispiele
 - › Standard Template Library für C++
 - › Java API
 - » Listen (Vector, ArrayList)
 - » Hash-Tabellen (HashMap)
 - » Sortieralgorithmen

- Vorteile von Bibliotheken
 - › *Strukturierungsmittel*
Kapselung ähnlicher und zusammengehöriger Funktionalität
 - › *Wiederverwendung von Softwarebausteinen*
 - › *Binärausleiferung*
Quellcode bleibt beim Anbieter
- *Komponenten*
 - › sind unabhängig von einer Programmiersprache
 - › genau definierte externe Schnittstelle

- Programmgerüst bzw. Applikationsrahmen
- Framework gibt die Architektur vor
 - › Wiederverwendung von Design und Softwarearchitekturen
 - › *Frameworks sind ein zentrales Mittel zur Formulierung von wiederverwendbarer Softwarearchitektur*
- Funktionsweise von Frameworks
 - › Framework besitzt die Ablaufhoheit
 - › Hollywood-Prinzip: *Don't call us, we call you!*
 - › Ableitung projektspezifischer Klassen aus Frameworkklassen
 - › und die Implementierung von Callback-Funktionen

- *Fachliche Frameworks*
 - › Spezifische Anwendungsgebiete
- *Technische Frameworks*
 - › Kommunikation, GUI, Persistenz
- Beispiele
 - › *Microsoft MFC* für die Erstellung von Benutzeroberflächen
 - › *JUnit* für die Entwicklung von Testsuiten für Java-Programme
 - › *Eclipse* für die Entwicklung von integrierten Entwicklungsumgebungen

- Modellierung der Architektur durch Bausteine und Abhängigkeiten
 - › *gute Unterstützung durch Softwarewerkzeuge möglich*
 - › CASE-Tools (*computer aided software engineering*)
- Anforderungen an CASE-Tools
 - › Unterstützung von Sichten
 - › Strukturaspekte
 - › Verhaltensaspekte
 - › Standardisierte Modellierungssprache (z.B. UML)

- Bewertung von CASE-Tools
 - › Ist die verwendete Modellierungssprache geeignet für das Problem?
 - › In welchen Branchen wird das Werkzeug bisher eingesetzt?
 - › Lässt sich das Werkzeug komfortabel benutzen?
 - › Lassen sich Diagramme und andere Informationen aufbereiten und als Kommunikationsmedium mit anderen Projektbeteiligten nutzen
 - › Gibt es Integrationsmöglichkeiten mit anderen Werkzeugen?
 - › Gibt es Unterstützung für Codegenerierung oder Reverse Engineering?

- Codegenerierung
 - › Struktur der Architektur → Struktur des Quellcodes
 - › *Generierung ermöglicht die automatische Wiederholung der Entwicklungsleistung.*
 - › Klar definierte Semantik der Modelle
 - › Je mehr Information im Modell steckt, je mehr Code kann generiert werden
 - › Entwicklung auf einer höheren Abstraktionsebene
 - › Weniger Routinearbeit, mehr Entwurfsarbeit

■ *Model Driven Architecture*

- › Fachliches Modell
 → *platform independent model* (PIM)
- › Modell der technischen Gegebenheiten
 → *platform specific model* (PSM)
- › Beide Modelle können unabhängig voneinander bearbeitet und weiterentwickelt werden
- › Der Codegenerator verknüpft beide Modelle und erzeugt plattformspezifischen Quellcode

- Einhaltung der Architektur
 - › Entwickler implementieren die vorgegebene Architektur (unter bestimmten ergänzenden Richtlinien)
 - › Entwickler haben bei der Implementierung Freiheiten
 - › Softwarearchitekt muss die Vorgaben der Architektur überprüfen und überwachen → *Werkzeuge zur Überprüfung der Architektur*
- Rekonstruktion
 - › Rekonstruktion der Architektur eines fertigen bzw. halbfertigen Systems
 - › Analyse des Quellcode bzw. anderer Artefakte → Ableiten der Architektur

- *statische Analyse*
 - › Codeanalyse
 - › Standardprodukte
 - › eigene, spezialisierte Werkzeuge mit Skriptsprachen erstellen
- *dynamische Analyse*
 - › Analyse des Ablaufs
 - › Standardprodukte, z.B.:
 - » Profiling-Werkzeuge für den Zeitbedarf
 - » Aufspüren von fehlerhafter Speicherzugriffe
 - › projektbezogene Trace-Ausgaben

Vier goldenen Regeln für die Toolbox des Softwarearchitekten

- Muster auf möglichst vielen Ebenen der Granularität einsetzen und erfolgreiche Muster in der Toolbox sammeln!
- Wissen zu Technologien und Werkzeugen verfeinern! Auf dem neusten Stand bleiben!
- Die Toolbox mit dem Wissen aus erfolgreichen Projekten kontinuierlich erweitern!
- Toolbox-Wissen mit anderen Architekten austauschen!

- Einführung in Software-Architekturen und Organisation
- Entwurf von SW-Architekturen
- Dokumentation von Software-Architekturen
- Evaluation / Bewertung von SW-Architekturen
- Toolbox des Softwarearchitekten
- **Produktlinien für Software**
 - › Was sind Produktlinien für Software?
 - › Wie werden Produktlinien eingeführt?
 - › Wie werden Produktlinien betrieben?
 - › Architektur und Software Engineering
- Enterprise Architecture Management

„Eine Menge von softwarelastigen Systemen, die einen gemeinsamen, kontrollierten Satz von Produkteigenschaften teilen, die auf die speziellen Bedürfnisse eines einzelnen Marktsegments ausgerichtet sind und die ausgehend von einem gemeinsamen Vorrat an Softwaregütern auf eine vorgeschriebene Art und Weise entwickelt werden.“

- Aus einem Vorrat an bereits fertigen Softwarekomponenten und anderen Geistesgütern nach streng definierten Regeln und Schnittstellen Softwareprodukte erstellen.
 - Die Softwarearchitektur ist ein Kernelement von erfolgreichen Softwareproduktlinien, da die Produkte in der Linie auf der gleichen Architektur basieren.
-
- 
- Die Herstellung von Softwareprodukten verlagert sich vom Programmieren zum Integrieren

- Eine Referenzarchitektur ist „eine generische Architektur für Anwendungen aus einer bestimmten Fachdomäne“
- Wie sieht der Zusammenhang zwischen Produktlinien und Referenzarchitekturen aus?
 - › Die Softwarearchitektur einer Produktlinie ist oft die Grundlage für die Entwicklung einer Referenzarchitektur
 - › Die Herausforderungen bei der Erstellung einer guten Produktlinienarchitektur sind denen für eine gute Referenzarchitektur sehr ähnlich
 - › Strategische Überlegungen sind sowohl bei Produktlinienarchitekturen als auch bei Referenzarchitekturen wichtig

- Ausschlachten bzw. Klonen von vorherigen Projekten: Projekte haben keine gemeinsame Basis
- Feingranulare Wiederverwendung: Nicht die Wiederverwendung vorhandener Software, sondern die Wiederverwendung der Architektur
- Reine komponentenbasierte Entwicklung: die Softwarekomponenten einer Produktlinie werden in dieser festgelegt und speziell für diese entwickelt
- Neue Versionen eines bestimmten Produktes: jedes Produkt (und jede Version davon) ist eine Instanz der Produktlinie
- Eine Sammlung von technischen Standards: Standards und Vorschriften sind nur die Randbedingung für die Softwareproduktlinie

- Softwareproduktlinien lassen sich am besten für die Produktion vieler ähnlicher Systeme innerhalb eines Unternehmens nutzen
- Es müssen drei Randbedingungen erfüllt sein:
 - › Die einzelnen Systeme müssen genügend Gemeinsamkeiten haben
 - › Die Anforderungen müssen verhandelbar sein
 - › Das Eigentum an den entwickelten Gütern darf nicht an den Kunden übergehen

- Wirtschaftliche Vorteile
 - › Einsparung bei der Entwicklung
 - › Kürzere Zeiten bis zur Fertigstellung und damit früherer Markteintritt
- Vorteile für die Organisation
 - › Bei erhöhter Nachfrage kann das organisatorische Wachstum besser bewältigt werden
 - › Geringere und dadurch leichter zu steuernde Anzahl an Entwicklern
 - › Entwickler können ihr Wissen leichter von Projekt zu Projekt übertragen
 - › Bei allgemein höherer Nachfrage nach Softwareprodukten werden sich überschneidende Aufgaben vermieden, Unabhängigkeit von eventuell raren Experten
 - › Bei stagnierenden Geschäften kann der Durchsatz mithilfe von Produktlinien erhöht werden

- Der **Geschäftsführer** kann Nischen schnell besetzen, oder die Präsenz in bereits erschlossenen Märkten verstärken
- Der **Projektleiter** und **technische Geschäftsführer**: bessere Planbarkeit von technischen Errungenschaften und organisatorischen Aspekten
- Im **Marketing** und **Vertrieb**: Die Produkte haben einen nicht verhandelbaren, gemeinsamen Teil und einen flexiblen Teil, der ein Eingehen auf die Bedürfnisse des Kunden erlaubt. Die bessere Planbarkeit des Entwicklungsprozesses hilft die Kundenzufriedenheit zu steigern
- Der **Kunde**: höhere Qualität, verlässlicherer Liefertermin und Preis sowie von den Synergieeffekten durch die gemeinsamen Produkteigenschaften, die zu niedrigerem Preis und weniger Fehlern führen

- Anfangsinvestition
 - › Im Allgemeinen der größte Teil der Kosten
 - › Zusätzliche Entwicklungsarbeit
 - › Organisatorische Änderungen
 - » Evtl. eine eigene Abteilung für die zentralen Güter der Produktlinie
 - » Zielkonflikte mit Projektdeadlines
- Laufende Kosten, um die grundlegenden Güter auf dem Stand zu halten

1. Welche Güter werden im Rahmen der Produktlinie produziert und wiederverwendet?
 - Softwarearchitektur, muss alle Produkte einer Linie abdecken
 - Komponenten
 - Zu entwickelnde Systeme
2. Welche unternehmerischen Aspekte bestimmen die Produktlinie?
 - Wirtschaftlichkeit
 - Organisatorische Sicht
 - Prozesssicht
 - Technologie
3. Welche Phasen gibt es im Lebenszyklus der Güter einer Produktlinie?
 - Entwicklung: Güter, insbes. Architektur und Basiskomponenten erstellen
 - Verwendung: Bausteine aus der Produktlinie werden zusammengesetzt
 - Weiterentwicklung

„Von allen Gütern, die im Rahmen eines Softwareprojektes entwickelt werden, ist die Softwarearchitektur eines derjenigen Güter, in die ein wesentlicher Anteil der Wertschöpfung fließt. Die Softwarearchitektur enthält wertvolles Wissen, Erfahrung und Know-How der besten Mitarbeiter eines Unternehmens. Deshalb ist es wichtig, die gleiche Softwarearchitektur in möglichst vielen Projekten einzusetzen.“

Wie soll die Wiederverwendung von Softwaregütern innerhalb einer Organisation betrieben werden?

Bottom-up-Prinzip versus Top-down-Prinzip

Bottom-Up-Prinzip: nicht aufeinander abgestimmte Bausteine werden eingesetzt, um beliebige Systeme zu entwickeln

- Nur bis zu einer gewissen Größe durchführbar

Top-Down-Prinzip: Ein vorgegebene Struktur, in die sich die Komponenten einfügen müssen

- Liegt Softwareproduktlinien zugrunde
- In der Praxis nicht immer ganz einzuhalten (Legacy-Software)

Welche Artefakte haben Potential zur Wiederverwendung?

- **Anforderungen:** Bestimmung gemeinsamer Anforderungen mindert Aufwand bei der Anforderungsanalyse
- **Architektur:** Unternehmenswissen wird wiederverwendet, Risiko einer fehlerhaften Architektur wird vermindert
- **Komponenten und Frameworks:** gute Designlösungen, Dokumentation,...
- **Ergebnisse aus Analysen:** Leistungsanalysen, Betrachtungen zur Fehlertoleranz
- **Tests:** Testpläne, Kommunikationspfade zur Fehlerbehebung
- **Planung:** Terminplanung, Teamorganisation,...
- **Prozesse, Methoden, Werkzeuge:** bereits etablierter Entwicklungsprozess
- **Personal:** Mitarbeiter beschäftigen sich mit projektübergreifenden Themen, sind unterschiedlichen Projekten leichter zuzuordnen
- **Beispielsysteme:** Alle fertigen Systeme können als Beispielsysteme dienen
- **Fehlerbehebung:** Jedes weitere Produkt profitiert von Fehlerbehebungen bei vorangegangenen Produkten

1. **Wirtschaftlichkeitsanalyse:** Lohnt sich die Einführung einer Produktlinie? Wie groß ist der Unterschied zwischen aktueller Situation und Situation nach Einführung?

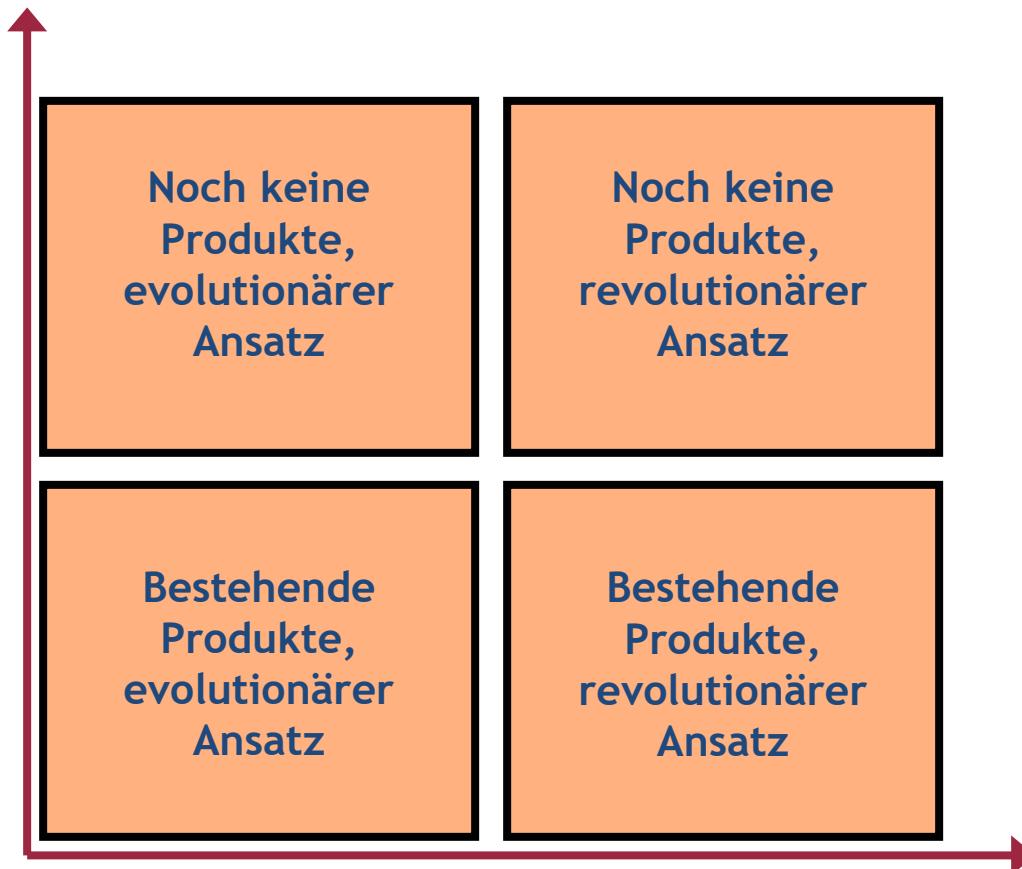
2. **Festlegung des Anwendungsbereichs (Scoping):** Welche Produkte sollen in der Linie enthalten sein? Welche Eigenschaften soll die Architektur berücksichtigen?
Zu eng gesteckt: nur wenige Produkte ableitbar
Zu weit gesteckt: Aufwand zur Entwicklung jedes einzelnen Produkts zu groß

3. **Planung der zukünftigen Entwicklung:** Welche Produkte und Produkteigenschaften werden in der Zukunft erwartet? Wie sollen diese berücksichtigt werden?

4. **Design der Produktlinienarchitektur:** Welche Softwarearchitektur soll der Produktlinie zugrunde liegen? Ist diese eine tragfähige Basis für die Linie? Zentraler Schritt!!
5. **Festlegung von Anforderungen an die einzelnen Komponenten:** Welche Anforderungen gibt die Linienarchitektur vor? Wie sieht die Konfiguration der einzelnen Komponenten für die jeweiligen Produkte aus?
6. **Validierung:** Wird der geforderte Anwendungsbereich abgedeckt? Können alle Produkte aus der Produktlinie entwickelt werden? Ist sie erweiterbar?

Wie werden Produktlinien eingeführt?

- Je nach Ausgangslage und Vorgehensweise vier Möglichkeiten:



- **Ausgangslage:**
 - › Die bereits vorhandenen Produkte wurden so gut wie unabhängig voneinander entwickelt
 - › Entscheidung: die einzelnen Produkte werden schrittweise umgestellt
- Komponenten suchen, die Anforderungen von mehreren Produkten umsetzen; diese verallgemeinern
- Produktlinienarchitektur muss ausgehend von den einzelnen Softwarearchitekturen definiert werden
- **Vorteil:** geringe anfängliche Investition, Produkte können während der Einführung der Produktlinie weiterentwickelt
- **Nachteil:** insgesamt höhere Kosten durch längere Übergangsphase, ggf. provisorische Übergangsversionen während der komponentenweisen Umstellung

- **Ausgangslage:**
 - › Die bereits vorhandenen Produkte wurden so gut wie unabhängig voneinander entwickelt
 - › Entscheidung: die einzelnen Produkte werden in einem großen Schritt umgestellt
- Anforderungen für die Produktlinienarchitektur werden aus der Summe der Anforderungen der Einzelprodukte abgeleitet, wobei mögliche zukünftige Erweiterungen berücksichtigt werden
- **Vorteile:** insgesamt geringere Einführungskosten
- **Nachteile:** Zeitverzögerung bis zur Auslieferung des ersten Produktes kann ein K.O.-Kriterium sein, hohe Anfangsinvestition erforderlich, die bei Anforderungsänderung wertlos werden kann

- **Ausgangslage:**
 - › Unternehmen plant die Entwicklung neuer Produkte
 - › die einzelnen Produkte werden schrittweise umgestellt
- **Vorteile:** geringere Anfangskosten, kürzeres Time-to-Market, benötigtes Know-How kann langsam aufgebaut werden, anfängliche Fehlentscheidung nicht unbedingt katastrophal/können wieder korrigiert werden
- **Nachteile:** gesamten Kosten hoch

- **Ausgangslage:**
 - › Unternehmen plant die Entwicklung neuer Produkte
 - › Zuerst wird die Produktlinie entwickelt, dann die Produkte
- Zunächst müssen die Anforderungen an die Produkte definiert werden, von dieser Schätzung hängt der Erfolg der Produktentwicklung ab
- **Vorteile:** insgesamt geringere Kosten, neue Produkte können in schnellerer Folge entwickelt werden
- **Nachteile:** hohes Risiko des Scheiterns, da noch wenig Erfahrung im Unternehmen vorhanden und deshalb nur unzuverlässige Schätzung der Anforderungen möglich

Wie werden Produktlinien eingeführt?

- Je nach Ausgangslage und Vorgehensweise vier Möglichkeiten:



Vorbedingung:

- Die Entscheidung für die Einführung einer Produktlinie ist getroffen
- Die Vorgehensweise (evolutionär oder revolutionär) ist festgelegt

Phase	Aktivität
Entwicklung	Entwurf einer Softwarearchitektur, die die Anforderungen an die Produkte der Familie unterstützt
	Entwicklung von Komponenten (traditionelle Komponenten, aber auch Frameworks)
Verwendung	Konfiguration der einzelnen Produkte aus den Komponenten im Rahmen der Produktlinienarchitektur (dies sollte ein viel geringerer Aufwand sein als bei einer reinen Produktentwicklung)
	a) Konfiguration z.B. durch Parameter oder durch Codegenerierung
	b) Falls nötig: Entwicklung von produktspezifischen Erweiterungen für Komponenten
	Zusätzliche Entwicklung von produktspezifischer Software (nach Bedarf)

Phase	Aktivität
Weiterentwicklung	<p>Alle Güter (Architektur, Komponenten, Produkte) werden kontinuierlich weiterentwickelt, um neue Anforderungen erfüllen zu können</p> <ul style="list-style-type: none">a) Die Architektur muss neue Komponenten und neue Beziehungen zwischen Komponenten einführen bzw. berücksichtigenb) Komponenten werden in Bezug auf ihre Funktionsweise und Eigenschaften, aber auch ihre Schnittstellen weiterentwickeltc) Für Produkte werden Nachfolgeversionen entwickelt und ausgeliefert. Alternativ werden Produkte beim Kunden durch den Austausch oder die Ergänzung von Komponenten neu konfiguriert (sog. Dynamische Architekturen)

- 1. Entwurf der Produktlinienarchitektur**
- 2. Bewertung der Produktlinienarchitektur**
- 3. Wiederverwendung von Gütern**
- 4. Weitere Aufgaben**

- **Einzelnes Softwareprojekt:**
- Trennung der langfristigen, konstanten Aspekte des Systems von den variablen

- **Produktlinienarchitektur:**
- Definition von Variationspunkten, die die Erzeugung verschiedener Produkte einer Linie erlauben
- Vordefinierte, gemeinsame Qualitätseigenschaften sicherstellen
- Bzgl. Tauglichkeit für die Produktlinie bewerten

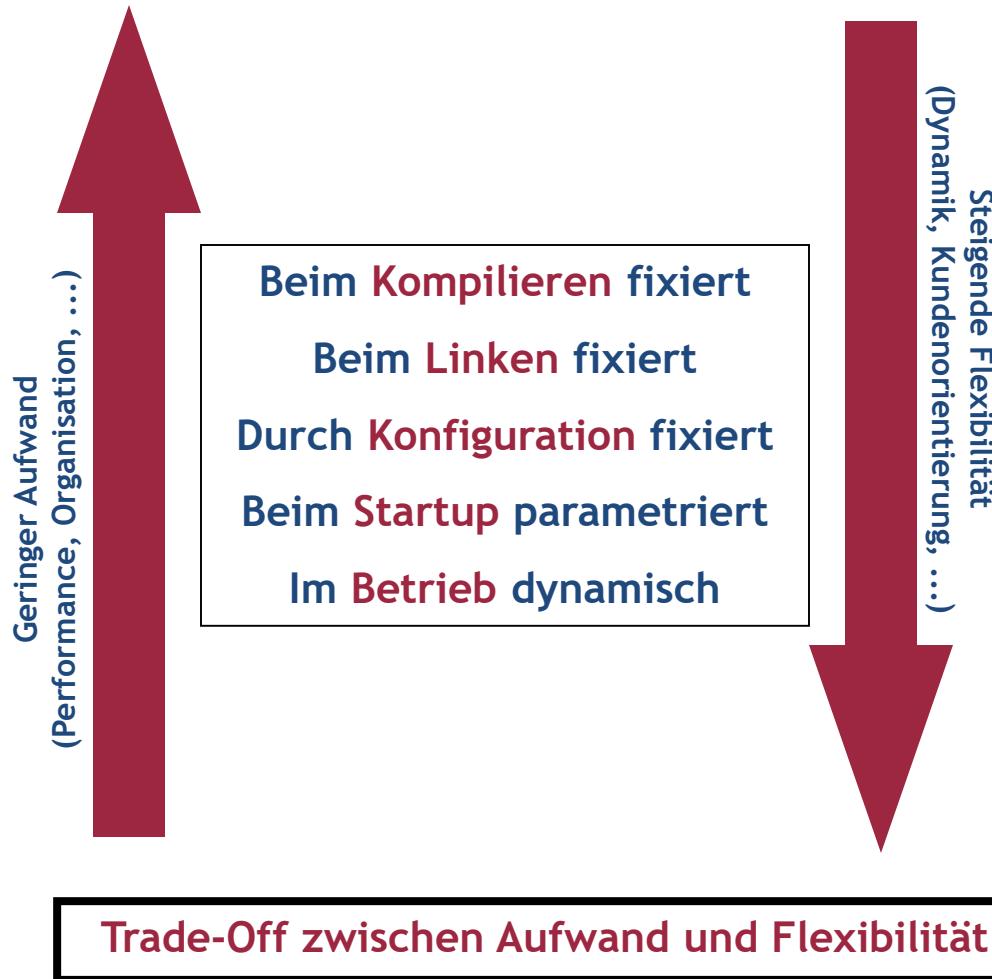
- Organisatorische, technische und produktspezifische Einflussfaktoren
- Aufgaben sind z.B. Verantwortlichkeiten identifizieren, Schnittstellen finden
- **Abstraktion und Variationspunkte:**
 - › Softwarearchitektur ist eine Abstraktion
 - › Erlaubt verschiedene Realisierungen des Systems, die integrierbar sein müssen
 - › Deshalb Variationspunkte
- **Product Builder's Guide**
- Dokumentation und Anleitung zur Konstruktion von Produkten erstellen!
- **Risiken beim Entwurf**
 - › Ursachen: z.B. unerfahrener Architekt, fehlende Kommunikation, ...
 - › Folgen: nicht zusammenpassende Komponenten, nicht erfüllte Anforderungen,...

Variationspunkte finden	<ul style="list-style-type: none">■ In den Anforderungen an die Produkte einer Produktlinie■ Während des Architekturentwurfs■ Aus der Implementierung der grundlegenden Komponenten und der ersten Produkte■ Typisch: Entscheidungen, die aufgrund fehlender Information erst später getroffen werden können
Variationspunkte umsetzen	<ul style="list-style-type: none">■ Auf Komponentenebene:<ul style="list-style-type: none">▶ optionale Komponenten vorsehen▶ verschiedene Komponenten mit gleicher Schnittstelle einsetzen■ Auf Quellcodeebene:<ul style="list-style-type: none">▶ Variabilität durch Codeänderung▶ Spezialisierung und Generalisierung▶ Parameter,▶ ...

Variationspunkte dokumentieren und bewerten

- Variationspunkte müssen dokumentiert werden, da sie die einzige Differenzierung von Produkten in einer Linie darstellen
- Wie lassen sich konkrete Produkte anhand der Variationspunkte realisieren?
- Welche Kombinationen sind erlaubt?
- Bewertung auch anhand der Variationspunkte:
 - ▶ Sind sie flexibel genug?
 - ▶ Können Produkte in angemessener Zeit erzeugt werden?

Aufgaben des Architekten - Wie viele Variationspunkte sollen gesetzt werden?



- Produktlinienarchitektur hat wesentliche Auswirkung auf Umsetzbarkeit, Qualität und damit gesamten Lebenszyklus der resultierenden Systeme
 - › Ungereimtheiten und Fehler wirken sich weithin aus und sind im Nachhinein schwer zu korrigieren

Bewertung bereits ab frühen Phasen

→ Bewertung auf zwei Ebenen:

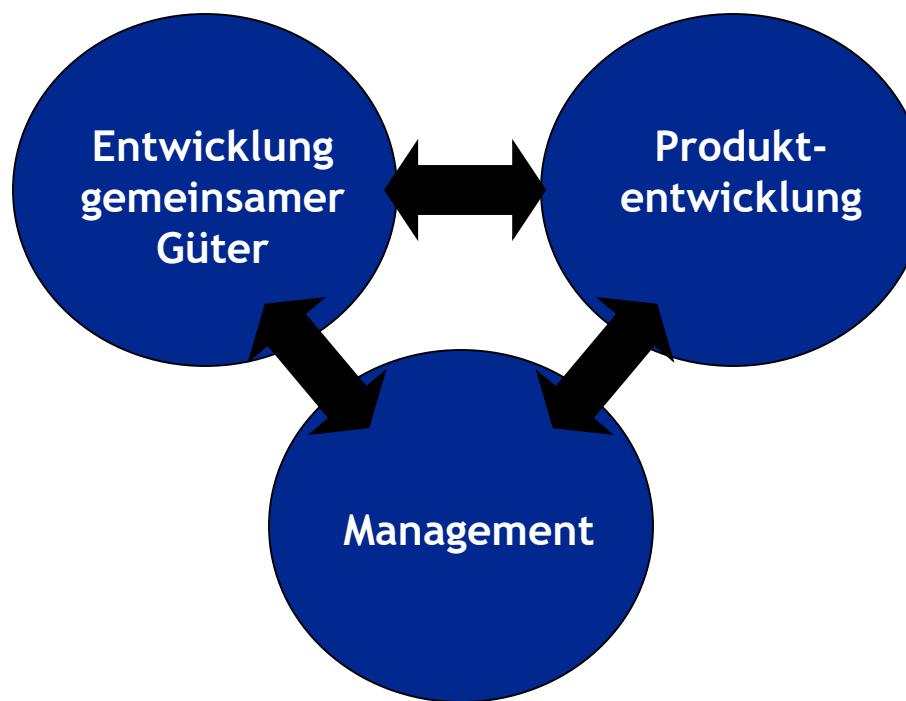
- › Gemeinsame Produktlinienarchitekturen: Allgemein und robust genug?
- › Architektur der abgeleiteten Produkte: genügt Architektur den abgeleiteten Anforderungen und Qualitätsattributen der Produkte?

- Meist existierende Systeme vorhanden
 - › Geistige Werte sollen gerettet werden
 - › Sollen in die neue Produktlinienstruktur überführt werden
 - › Wiederverwendung von Quellcode nur ein Teil davon
- Asset Mining
- Softwarearchitektur des bestehenden Systems als wertvollstes Gut
 - › Falls diese für Produktlinie wiederverwendet werden kann, können evtl. auch die existierenden Komponenten für die Produktlinie aufbereitet werden
 - › Vollständiges und gründliches Verständnis der existierenden Architektur ist dafür Voraussetzung!
 - › Für wiederverwendete Güter dieselben Qualitätsansprüche wie sonst!
 - › Kosten der Wiederverwendung dürfen die einer Neuerstellung nicht überschreiten!

- **Komponenten-Entwicklung:** Planung und Leitung der Komponentenentwicklung, erstellt eine Liste von Komponenten, die als Bausteine zusammengefügt werden und die einzelnen Produkte bilden
- **Verwendung von COTS-Komponenten¹:** COTS-Komponenten mit geeigneten Variationspunkten auswählen
- **Definition von Anforderungen:** Erstellen eines gemeinsamen Satzes von Anforderungen mit Variationspunkten
- **Software-Systemintegration:** Zusammenfügen der Bausteine zu einem kompletten Produkt. Die Integrationsfähigkeit steht und fällt mit den Schnittstellen, es ist hilfreich, bei den grundlegenden Gütern bereits vorab integrierte Subsysteme vorzuhalten

1) Commercial-off-the-shelf: extern entwickelte und zugekaufte Komponenten

Wesentliche Aktivitäten zum Betrieb einer Produktlinie



„Bereitstellung der Maschinerie“

Aufgaben:

- Gültigkeitsbereich abgrenzen
- Zentrale Güter bereitstellen (z.B. Softwarearchitektur, Komponenten)
- Produktionsplan

Benötigte Grundlagen:

- Randbedingungen u. Beschränkungen
- Muster und Stile
- Bereits existierende Komponenten

„Domain Engineering“



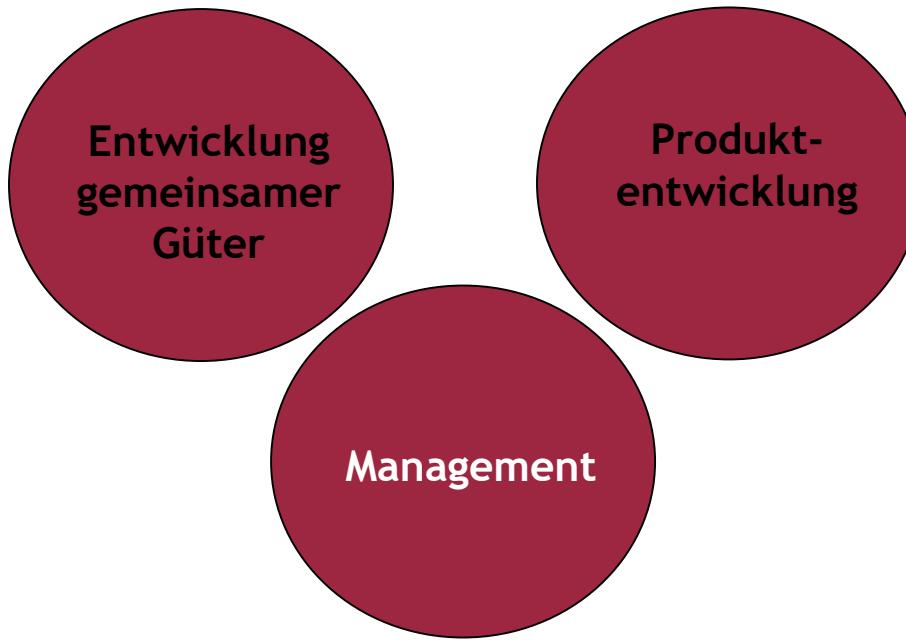


„Systemproduktion“

Aufgabe:

Mit möglichst geringem Aufwand und großem Nutzen auslieferbare Systeme zu produzieren.

„Application Engineering“



„Zuteilung von Ressourcen, Koordination, Überwachung“

- **Technisches Management:** Entwicklung der Architektur und Basiskomponenten, Prozessüberwachung, Kontrolle des Entwicklungsfortschritts
- **Organisatorisches Management:** personelle Besetzung und organisatorische Struktur
- **Produktlinienmanager:** führt die Umsetzung des Produktlinienansatzes an (insbes. bei Einführung wichtig)

„Eine Softwarekomponente ist das Medium der Komposition von Systemen mit explizit bereitgestellten, benötigten und Konfigurationsschnittstellen sowie Qualitätsattributen“

- Von außen betrachtet sieht man
 - › Die Schnittstellen, die eine Komponente von anderen Komponenten benötigt
 - › Die Schnittstellen, die für andere Komponenten bereitgestellt wird
 - › Möglichkeit der Konfiguration und Parametrierung (Variationspunkte)
 - › Qualitätsattribute, z.B. Portierbarkeit
- Drei Arten der Wiederverwendung:
 - › Von Version zu Version des Systems: Komponente unterstützt mehrere Versionen des Systems
 - › Von Produkt zu Produkt im Rahmen einer Linie: Komponente wird in mehreren Produkten und in aufeinander folgenden Versionen dieser Produkte eingesetzt
 - › Von Unternehmen zu Unternehmen (COTS-Komponente): unternehmensübergreifende Verwendung der Komponente

„Domänen oder Einsatzgebiete beschreiben den Wirkungsbereich eines Softwaresystems oder eines einzelnen Bausteins.“

- Bei Produktlinienarchitekturen wird eine Domäne meist durch eine primäre Komponente repräsentiert
 - › Klassen in Komponenten definieren Konzepte in der zugehörigen Domäne
 - › Anhaltspunkt für den Architekten für die Aufteilung der Komponenten
 - › Primäre Komponente wird in der Praxis oft durch kleinere, sekundäre Komponenten an den benötigten Leistungsumfang angepasst
- Hierarchien und Granularität
 - › Kleine Komponenten lassen sich leicht wiederverwenden, ohne dass große Teile der Komponente weggelassen werden müssen
 - › große Komponenten ermöglichen das Zusammensetzen des Systems aus wenigen Bausteinen und sparen dadurch Fixkosten, die bei der Wiederverwendung anfallen

- Beim Einsatz von Komponenten in Produktlinien gibt es zwei Anwendungsfälle:
 - › Komponente ist Bestandteil der Produktlinie
 - » Variationspunkte müssen berücksichtigt werden:

Mechanismus	Zeitpunkt der Spezialisierung	Typ der Variabilität
Vererbung	Bei der Definition der Klasse	Spezialisierung (z.B. neue Funktionalität)
Erweiterung	Erstellung der Anforderung	Neuer Anwendungsfall basiert auf existierendem Anwendungsfall
Verwendung	Erstellung der Anforderungen	Neuer Anwendungsfall verwendet existierenden Anwendungsfall
Konfiguration	Vor der eigentlichen Laufzeit	Zusätzliche Ressource wird von der Komponente geladen
Parametrierung	Implementierung von Komponenten	Definition von Funktionen mit abstrakten Eingangswerten
Templates	Implementierung von Komponenten	Parametrierbare Definition von Datentypen
Generierung	Vor der eigentlichen Laufzeit	Parametrierbares Generierungswerkzeug

- › Komponente ist eine Neuentwicklung für ein bestimmtes Produkt
 - » Einfacher, da keine Variabilität
 - » Erweiterung auf mehrere Produkte und evtl. die gesamte Produktlinie zumindest in Ansätzen einplanen

- Schnittstellen sollten von keiner Komponente abhängen

„Eine Schnittstelle ist ein Vertrag zwischen einer Komponente, die eine bestimmte Funktionalität benötigt und einer Komponente, die diese Funktionalität bereitstellt.“

- Syntax muss definiert sein
- Semantik muss festgelegt sein, z.B. die Aufrufreihenfolge und die gegenseitige Abhängigkeit der Aufrufe

„Ein Protokoll beschreibt die Schnittstellenfunktionen zusammen mit einem bestimmten Aufrufablauf.“

- Restriktionen und Randbedingungen:
 - › Fachliche und Qualitätsanforderungen, einzuhaltende Richtlinien
 - › Maximierung der möglichen Anwendungsszenarien durch Analyse der geforderten Variabilität, dadurch Anforderungen an Entwurf und Schnittstellen der Komponenten
 - › Verwendung von Legacy-Code zur Kostenersparnis
 - › Anforderungen aus der nötigen Flexibilität und den geforderten Variationspunkten
- Einschränkung der Freiheitsgrade und Vereinfach der Arbeit

- Möglichst ohne Eingriff in die Interna einer Komponente (Black-Box-Prinzip)
- Adaptionsschicht soll transparent sein (d.h. kein Unterschied, ob auf die Adaptionsschicht oder auf die Komponente zugegriffen wird)
- Adaptionen sollen beliebig miteinander kombinierbar sein
- Adaptionen sollen Kombinierbarkeit von Komponenten nicht einschränken
- Wiederverwendbarkeit und Konfigurierbarkeit

■ **Copy-Paste**

- › Klonen von Komponenten mit nachträglicher Veränderung des Quellcodes
- › Verletzt das Black-Box-Prinzip, ist aber transparent (erhält Schnittstelle)

■ **Vererbung**

- › Keine große Veränderung des Quellcodes, aber Ingenieur muss Interna der Komponente gut kennen, verletzt damit Black-Box-Prinzip
- › Transparenz ist gegeben

■ **Wrapper**

- › Eine Adaptionsschicht wird um die Komponente herumgelegt
- › Schnittstelle wird verdeckt, d.h. keine Transparenz

„Ein Framework ist eine Menge von Klassen, die ein abstraktes Design für Lösungen zu einer Familie von verwandten Problemen darstellen.“

- Zunehmend mehrere Frameworks gleichzeitig eingesetzt
- Werden als Komponenten mit sehr hohem Grad an Konfigurierbarkeit betrachtet
- Prinzip der Umkehrung der Kontrolle: applikative Teile werden vom Framework aufgerufen und nicht umgekehrt, Framework behält damit Kontrolle
- Einsatz in Produktlinien meist hilfreich, da sie einen sehr hohen Grad an Wiederverwendbarkeit kennen

- **Produktspezifische Erweiterung**
 - › Pro Produkt wird das Framework einmal instantiiert und produktabhängig erweitert
- **Standardspezifische Erweiterungen**
 - › Erweiterung pro benötigtem Standard zu bestimmten Systemaspekten
 - › Vorteil ist die einheitliche Sicht für andere Komponenten
- **Feingranulare Erweiterungen**
 - › Möglichst unabhängige Variationspunkte
 - › Framework mit vielen kleinen, voneinander unabhängigen Klassen
 - › Flexibles Konzept mit hohem Wiederverwendungspotential
- **Generatorbasierter Ansatz**
 - › Erweiterung des Frameworks wird aufgrund einer Spezifikation durch einen Generator automatisch erstellt
 - › Flexibilität, hoher Wiederverwendungsgrad und automatische Sicherstellung der konzeptuellen Integrität der Erweiterungen

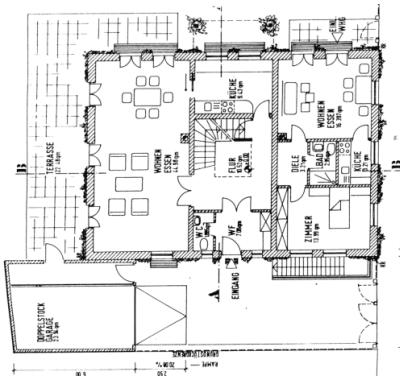
- Einführung in Software-Architekturen und Organisation
- Entwurf von SW-Architekturen
- Dokumentation von Software-Architekturen
- Evaluation / Bewertung von SW-Architekturen
- Toolbox des Softwarearchitekten
- Produktlinien für Software
- **Enterprise Architecture Management**

Bisher: Betrachtung von Softwarearchitekturen

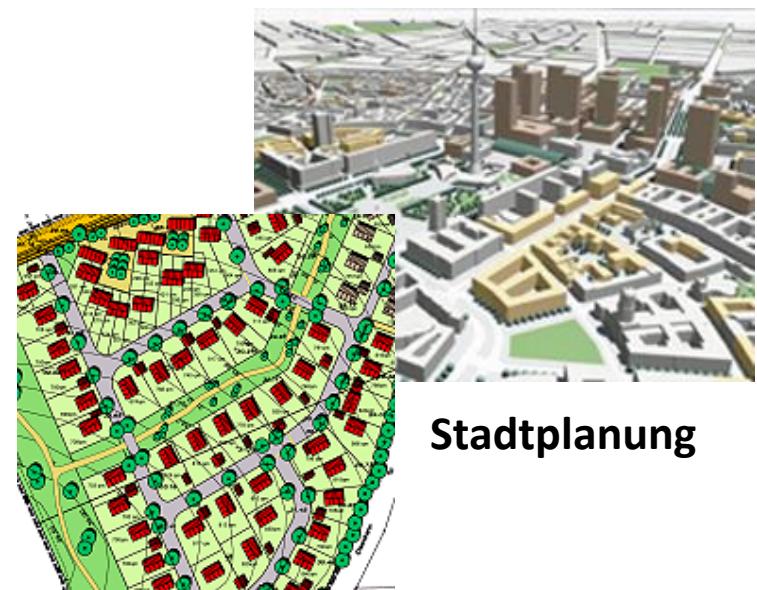
Beschäftigten sich mit dem Aufbau einzelner Softwaresysteme

Jetzt: Betrachtung von Unternehmensarchitekturen

Beziehen sich auf mehrere Softwaresysteme, haben einen geringeren Technikbezug und einen höheren betriebswirtschaftlichen Bezug.



Planung eines Hauses



Stadtplanung

- Motivation und Problemstellung
- Elemente einer EA
- Frameworks
 - › Beispiel: TOGAF
- Modellierung von EA
 - › Beispiel: Archimate
- Tooling

- **Motivation und Problemstellung**
- Elemente einer EA
- Frameworks
 - › Beispiel: TOGAF
- Modellierung von EA
 - › Beispiel: Archimate
- Tooling

- Fehlende Verzahnung von Geschäftsstrategie und IT–Aktivitäten (Business – IT Alignment)
- Erfolg eines Unternehmens wird entscheidend von der Ausgestaltung der IT Landschaft bestimmt
- IT wird nur als Service-Lieferant oder Kostenfaktor gesehen und nicht als Partner bei der Umsetzung der Unternehmensstrategie,
- Redundanzen, mangelnde Transparenz
- Fehlende ganzheitliche Sicht auf die IT
- Kritische Abhängigkeiten in den Geschäftsprozessen und der IT sind nicht bekannt
- Wettbewerbsfähigkeit durch eine koordinierte Anpassung und Veränderung der Strukturen im Unternehmen erhalten

- Prozesse verbleiben im exklusiven Einflussbereich einzelner Geschäftseinheiten.
- Anwendungen werden aus der Sicht einzelner Ressorts entworfen, umgesetzt und betrieben. Dadurch fehlt der übergreifende Blick auf angrenzende Prozesse, und die neu zu entwickelnde Infrastruktur richtet sich an so entstandenen Silos aus.
- Geschäftliche und technologische Silos erzeugen durch Datenduplikation und verteilte Anreicherung unerwünschte Abhängigkeiten zwischen Architekturelementen und eine erhöhte Schnittstellenkomplexität – mit negativen Folgen für die Flexibilität von Unternehmen.

- Was ist die optimale IT-Unterstützung für die Geschäftsstrategie, das mehrere 100 IT-Systeme benötigt, um seine Aufgabe zu erfüllen?
- Wie verwaltet man ein IT-Portfolio von mehreren hundert großen IT-Systemen, von denen jedes eine Investition im hohen zweistelligen bis dreistelligen Millionen-Euro-Bereich sein kann?
- Können neue Produkte und Services mit den existierenden Geschäftsprozessen und IT-Infrastruktur eingeführt werden?
- Wie kontrolliert man die Komplexität der Unternehmens-IT?
- Wie kann man technologische Innovationen in ein Portfolio von Anwendungen einpassen, die auch mal 30 Jahre alt sind?
- Welche Geschäftsbereiche und Nutzer sind von einer Anwendungsmigration betroffen?

Unternehmensarchitekturen „in a nutshell“

“Enterprise architecture is the process of translating business vision and strategy into effective enterprise change by creating, communicating and improving key principles and models that describe the enterprise’s future state and enable its evolution.”

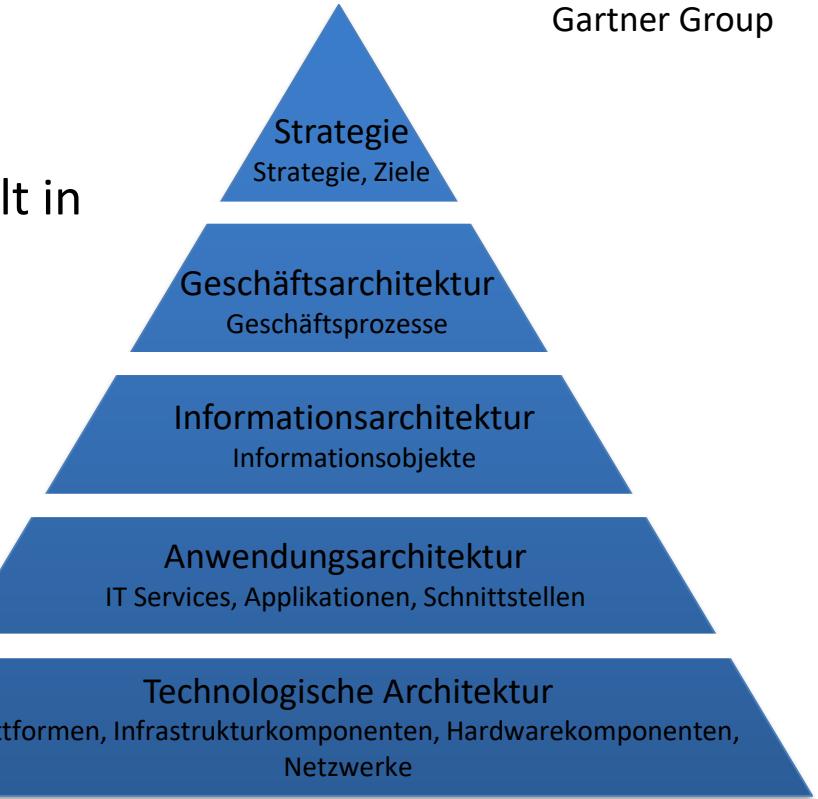
Gartner Group

Statische Sicht:

Unternehmensarchitekturmodell, aufgeteilt in verschiedene Ebenen

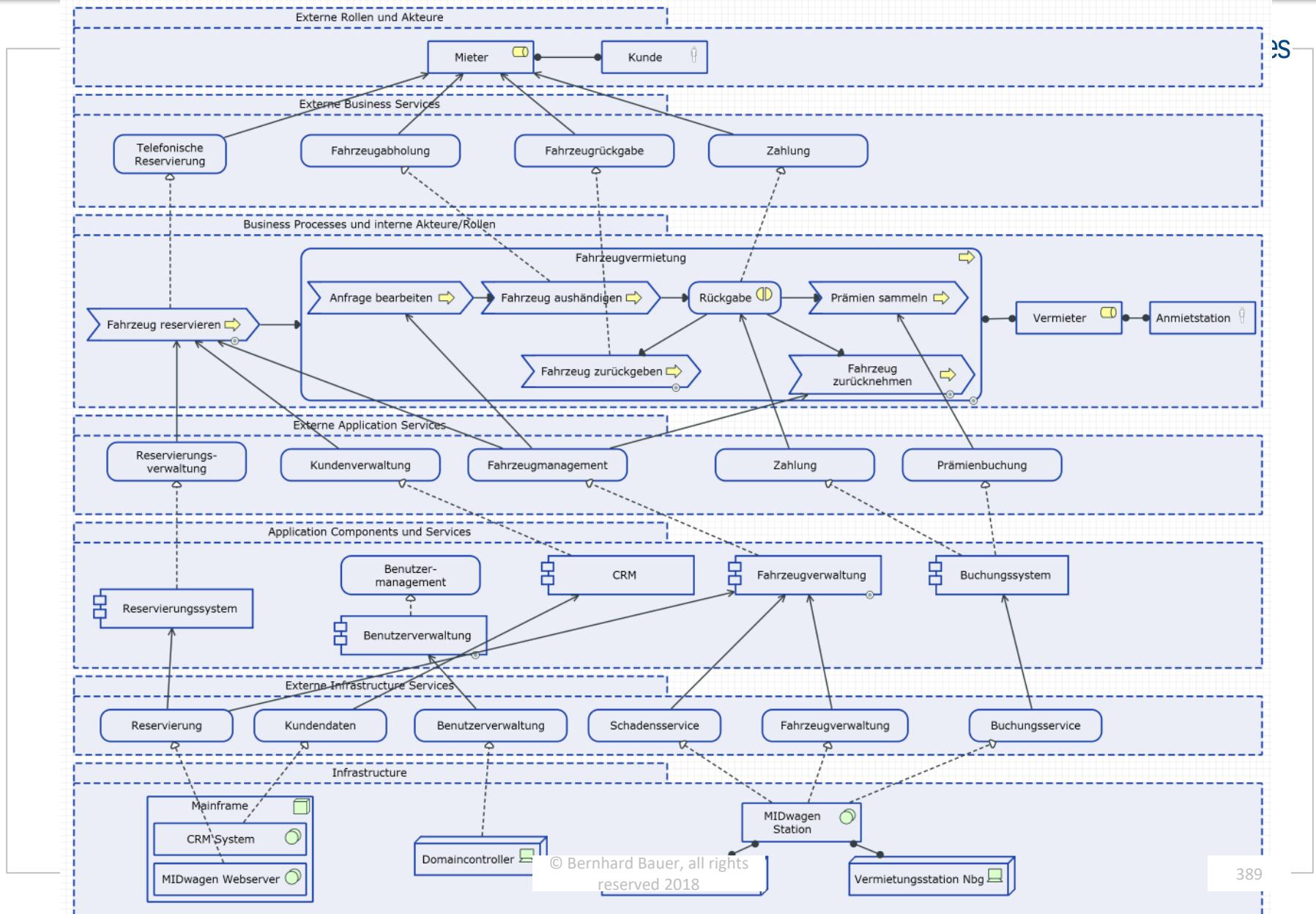
Dynamische Sicht:

Unterstützung der Evolution des aktuellen Modells ausgehend von der Strategie, durch Definition und Kommunikation zukünftiger Prinzipien und Modelle



Unternehmensarchitektur MIDWagen

Übersicht Ist-Architektur



Definition architecture:

„fundamental organisation of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution“
(IEEE 1471:2000)

Unternehmensarchitektur:

Grundlegende Struktur eines Unternehmens als sozio-technisches System

- Geschäftsmodell
- Organisationsstruktur
- Geschäftsprozesse
- Daten und Informationsobjekte
- Anwendungen
- Technologie

Prinzipien zur Steuerung des Designs und der Evolution

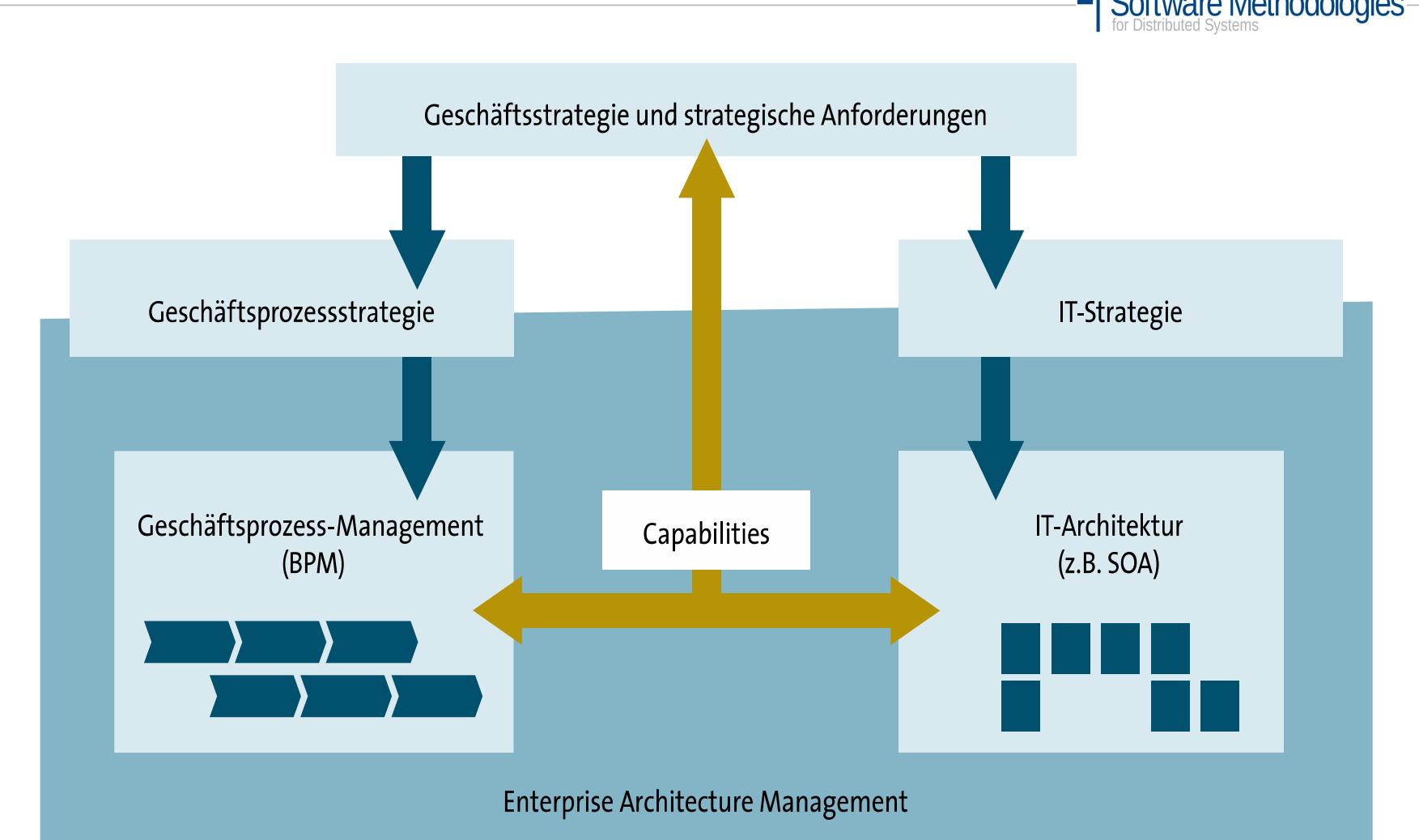
- Dokumentation der aktuellen Architektur (as-is) und Entwicklung einer gewünschten Ziel-Architektur (to-be)
- Entwicklung und Strukturierung der Komponenten
- Sicherstellen der Konsistenz der Abhängigkeiten der einzelnen Komponenten

- Baut auf den dokumentieren Unternehmensarchitekturemodellen (Ist- und Soll-Architektur) auf
- Beinhaltet den Prozess der Erstellung, Realisierung und Umsetzung der Unternehmensarchitektur
- Modellierung der Architektur ist nur ein Teil
- Ziel: Optimierung der Leistungsinfrastruktur in Bezug auf die Geschäftsziele bzw. die angestrebten Geschäftsmodelle.

Verständnis von EAM:

- EAM als **Management-Philosophie**: Verstehen, planen, entwickeln und kontrollieren der Unternehmensarchitektur
- EAM als **Organisationsfunktion**: Etablierung und Verbesserung der Prozesse zur Strategieplanung und -realisierung
- EAM als **Methode**: Verbesserung der Entscheidungsfindung
- EAM als **Kultur**: Schaffung eines gemeinsamen Verständnisses und Vision

(Ahlemann)



Kurzfristig

- Transparenz
- Einschränkung von Redundanz
- Entscheidungsunterstützung
- Identifikation von Handlungsbedarf
- Bewertung des IT-Portfolio
- Unterstützung in der Risikoanalyse

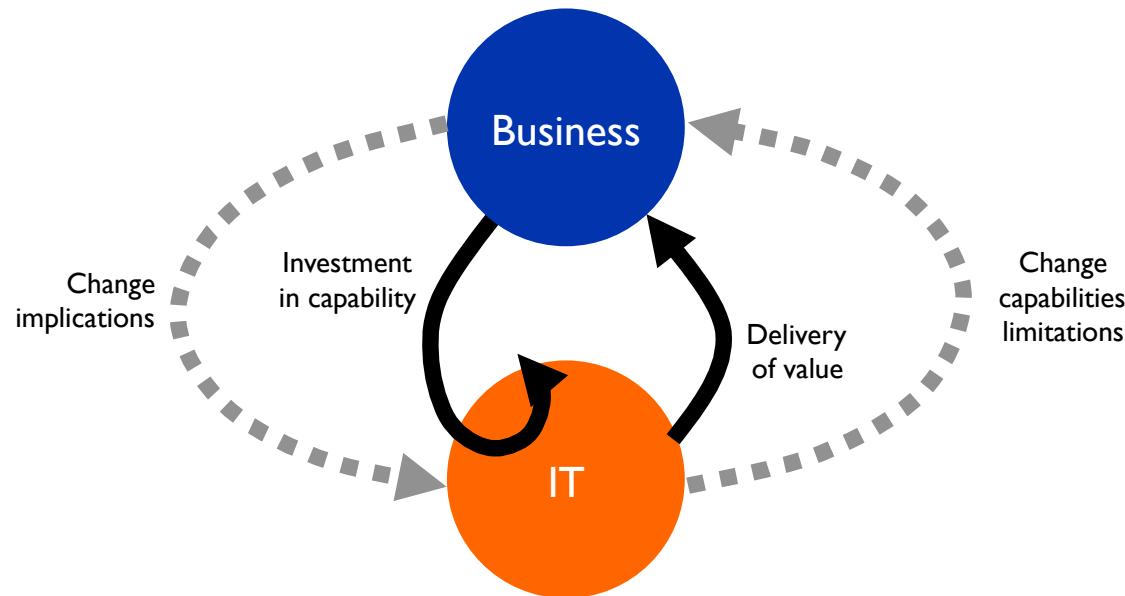
Langfristig

- Management der Komplexität
- Flexibilität der IT
- Synergiepotenziale entdecken
- Capability Management
- Identifikation von wertschöpfenden Innovationen
- Strategische Positionierung des Unternehmens

Grundlegende Elemente einer EA:

- Standardisiertes Vorgehen
 - › Sichert Hohe Qualität, Nachvollziehbarkeit der Entscheidungen, Ergebnisse werden nicht nur einmal verwendet sondern unterstützen nachhaltig den Weiterentwicklung der EA
- Gemeinsame Sprache und gemeinsame Modelle
 - › Einheitliches Verständnis von Konzepten, Begriffen; Verbessert die Kommunikation (v.a. zwischen Fachbereich – IT)
- Definierte Organisationsstruktur mit Prozessen und einem passenden Governance Modell

- Bezeichnet die Ausrichtung der IT an den Geschäftszielen des Unternehmens
- Nutzung von EA Artefakten um eine optimale Ausrichtung zu erreichen



Quelle: Dieter Masak, Wolfgang Keller: plenum Management Briefing Unternehmensarchitektur

- **Kognitives Alignment:** Ausrichtung der Geschäfts- und der IT-Welt im Unternehmen auf ein gemeinsames Gedankengut
- **Strategisches Alignment:** Abstimmung von Geschäftsstrategie und IT-Strategie
- **Architektonisches Alignment:** Unterstützung der aktuellen Geschäftsprozesse und Organisationsstrukturen durch die aktuellen IT Systeme
- **Temporales Alignment:** Wie schnell können sich die Softwaresysteme bzw. die komplette IT eines Unternehmens an Änderungen im Geschäftsmodell anpassen
- **Systemisches Alignment:** Inwieweit sind Informations- und Kommunikationssysteme eines Unternehmens geeignet, das Unternehmen tatsächlich zu steuern

(Keller, Masak (2008): Was jeder CIO über IT Alignment wissen sollte)

- Ableitung von Zielen für EAM
- Auftragsklärung (Vision und Mandat festlegen)
- Scoping: Definition der EA-Services (EAM Einsatzbereich festlegen)
 - › Welche Unternehmensbereiche
 - › Angestrebter Detaillierungsgrad
 - › Zeithorizont und Meilensteine
 - › Grenzen durch Ressourcen und Kompetenzen
 - › Erwarteter Nutzen
- Ist-Analyse (Ausgangsbasis analysieren)
 - › Vorhandenen Prozesse zur strategischen Planung
 - › Prozess der Strategieumsetzung
 - › Betrieb
 - › Bestehende Governance-Strukturen
 - › Verwendbare und vorhandene Methoden und Tools analysieren

(Bitkom)

- Design-To-Be festlegen:
 - › Integration von EAM in die Organisation
 - › Bestehende Organisation um fehlende Aspekte des EAM ergänzen/anpassen
 - › Gestaltungselemente (Tools und Frameworks) festlegen
- Priorisierung der zu implementierenden EA-Services
Ziel: in 3-6 Monaten erste schnelle Erfolge realisieren
- Zeitliche Einführungsplanung
 - › Definition einmaliger Aktivitäten für Start-up (Setup-Aktivitäten)
 - › Umsetzung der EA-Services (Deployment)
 - › Überführung in Regelbetrieb, Erfolgskontrolle
 - › Optimierung des EAM

(Bitkom)

- **Top-Down**
 - › Empfohlene Einführungsstrategie
 - › Initialer Setup, dann sukzessive Roll-outs strukturiert nach
 - » Unternehmensbereichen (organisatorisch, horizontal)
 - » Top-down nach der Ebene/Komplexität (vertikal, strategisch, operativ)
 - » Nach Domänen (fachlich/diagonal, Applikation, GP)
 - › Idealerweise projektbezogene Pilotierung vor dem Roll-out
- **Bottom-up**
 - › Nicht empfohlen
 - › Fehlende Ganzheitlichkeit des Ansatzes
Oft fehlt hierbei die Unterstützung des Managements

(Bitkom)

- Big-Bang Einführung
 - › Durchgängige Einführung in einem Schritt
 - › Theoretischer Ansatz, in gewachsenen Strukturen nicht umsetzbar, da das EAM eine Kultur voraussetzt, die erst wachsen muss
- Werkzeugorientiertes Vorgehen (Tool-driven)
 - › Definition der Prozesse und Governance anhand eines vorher ausgewählten Tools
 - › Herausforderung: Tool Auswahl (Anforderungen müssen zu Beginn sehr detailliert vorhanden sein)
 - › Probleme: Nicht nachhaltig, ignoriert kulturellen Aspekt, vernachlässigt Geschäftsnutzen
 - › Vorteil: Tooleinsatz erhöht Effizienz und Konsistenz der Modellierung

(Bitkom)

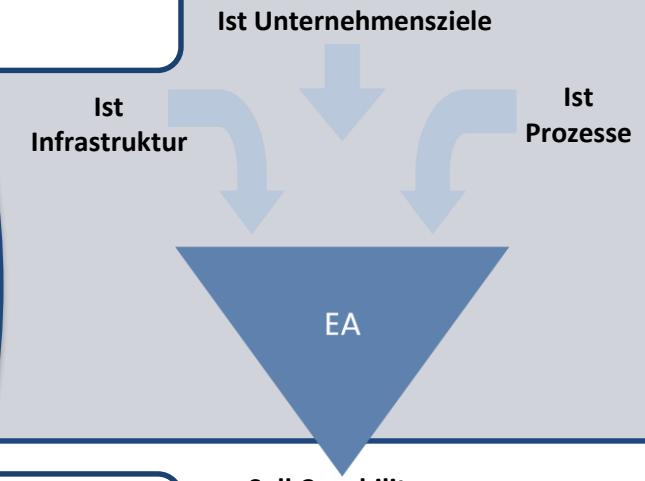
Elemente des Unternehmensarchitekturmanagements

 **Software Methodologies**
for Distributed Systems

- Motivation und Problemstellung
- Elemente einer EA
 - › EA Prozess und seine Einbettung im Unternehmen
 - › Ebenenmodell
 - › Governance
- Frameworks für das EAM
 - › Beispiel: TOGAF
- Modellierung von EA
 - › Beispiel: Archimate
- Tooling

Aufbau einer EA

- Aus Analyse der Ist-Prozesse, Ist-Infrastruktur und Ist-Unternehmensziele EA aufbauen
- Soll-Capabilities identifizieren und daraus Zielarchitektur ableiten
- Entwicklung eines Maßnahmenkataloges aus Vergleich zwischen Ist- und Ziel-Architektur



Implementierung

- Architektur auf die Unternehmensziele ausrichten
- Capabilities entwickeln
- Zielerreichung prüfen und ggf. nachsteuern
- IT-Projektportfolio und Infrastrukturvorhaben werden mit Maßnahmenkatalog abgeglichen und dann umgesetzt

Definitionen

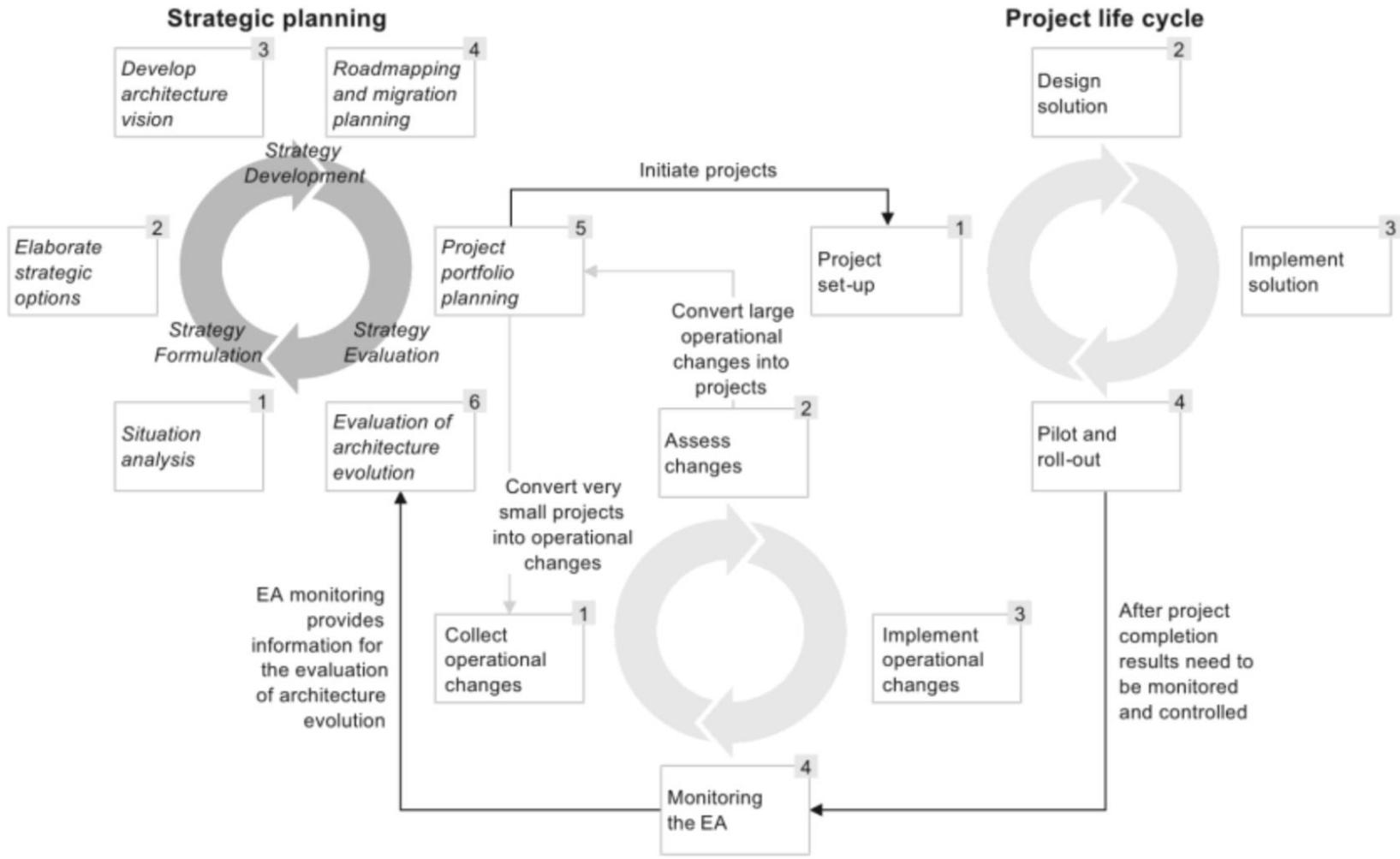
- „**Capability** is the ability to perform actions. As it applies to human capital, capability is the sum of expertise and capacity.“ (Wikipedia)
- A **business capability** defines the organization's capacity to successfully perform a unique business activity. Capabilities:
 - › Are the building blocks of the business,
 - › represent stable business functions,
 - › are unique and independent from each other,
 - › are abstracted from the organizational model,
 - › capture the business interests. (Webinar von Forrester Research 2009)
- „Unter Capability wird [...] eine (Geschäfts-)Fähigkeit verstanden, die eine Organisation, Person oder System besitzt. Capabilities werden typischerweise mit allgemeinen bzw. übergeordneten Begriffen benannt und erfordern eine Kombination von Menschen, Organisation, Prozessen und Technologie, um realisiert werden zu können.“ (Leitfaden EAM Bitkom)

- Ein Portfolio von Business Capabilities kann ähnlich mit Eigenschaften belegt werden, wie ein Applikationsportfolio
- Fragestellungen:
 - › Ist die Geschäftsfähigkeit „strategisch“ wichtig?
 - › Ist die Geschäftsfähigkeit adäquat implementiert?
 - › Wer sind die Kunden, die die Geschäftsfähigkeit nutzen?
 - › Welche Volumina von was werden zu welchen Kosten dort bearbeitet?
 - › Wie wird „Erfolg“ einer Geschäftsfähigkeit gemessen?
 - › Wer ist der „Owner“ einer Geschäftsfähigkeit?

Verwendung von Capabilities

- Capabilities entwickeln sich zunehmend als Planungsinstrument
- Abbildung aller Capabilities in einer Capability Map und Verknüpfen dieser mit den entsprechenden Architekturelementen zur Realisierung: Prozesse, Personen, Informationen und IT Systeme
- Capabilities beschreiben die Wettbewerbsvorteile eines Unternehmens, also diejenigen Prozesse und Produkte wo das Unternehmen besser, kostengünstiger oder einfach anders als seine Mitwerber ist
- Herausstellen der Capabilities ermöglicht zielgerichtetes optimieren dieser und hilft Architekturelemente nach ihrem strategischen Wert zu bewerben
 - › Z.B. Optimalen Gestaltung von Anwendungslandschaften im Hinblick auf Sourcing
 - » Man kann beobachten, dass in Ist-Anwendungen strategische und nicht strategischen Geschäftsfähigkeiten gemischt sind
 - » Ein solcher Zustand behindert Sourcing

Prozesszyklen im Kontext des EAM



- Jede Ebene beschreibt die wesentlichen Komponenten der Teilarchitektur
 - › Beispiel Prozesse: EA-Modell enthält die Prozesskomponenten mit ihren Abhängigkeiten, die konkrete Ausprägung/Ablauf der Prozesse ist nicht Teil der EA sondern des BPM
- Elemente und Komponenten der einzelnen Architekturen lassen sich jedoch auch immer ebenen übergreifend betrachten
 - › Z.B zur Analyse der Auswirkungen von Architekturänderungen
- Keine klar definierten Ebenen, überall ein bisschen anders

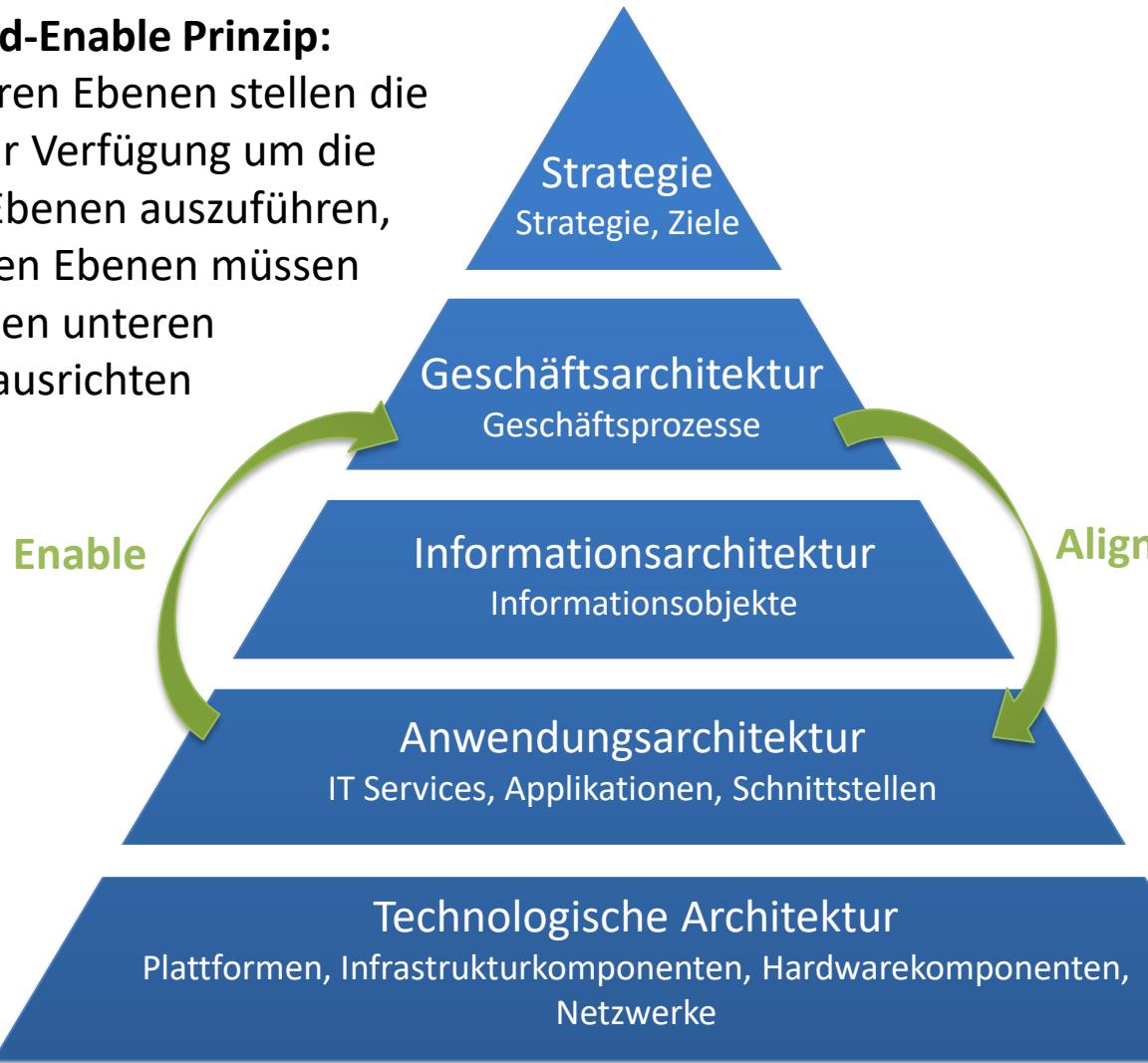
Ziele:

- Auflösen von Komplexität: Aufteilung der Gesamtarchitektur in verschiedene Ebenen
- Trennung von verschiedenen Themenbereichen
- Optimale Ausrichtung der Ebenen aufeinander (Business-IT Alignment)

Typische Ebenen

Align-und-Enable Prinzip:

Die unteren Ebenen stellen die Mittel zur Verfügung um die oberen Ebenen auszuführen, die oberen Ebenen müssen sich an den unteren Ebenen ausrichten

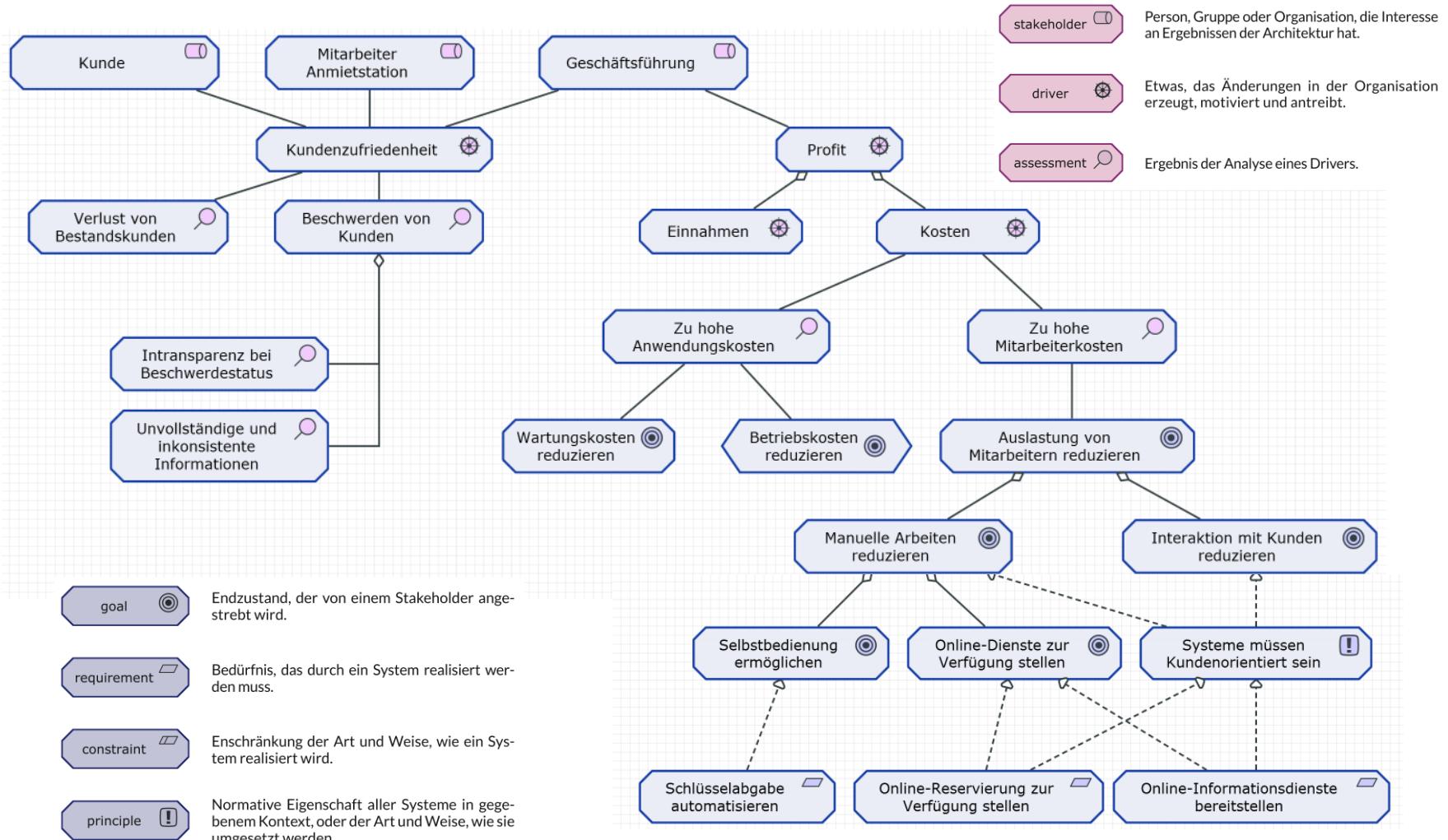




- Typische Elemente: Value Networks, Kunden, Märkte, Produkte, Ziele und dazugehörige Metriken
- **Geschäftsstrategie:** Ein (meist nicht vorhandenes) Dokument, das von der Vorstandsebene eines Unternehmens inhaltlich erarbeitet werden sollte
- Geschäftsstrategie beschreibt die Entwicklung des Unternehmens in der Zukunft
- Klar definierte Geschäftsstrategie erlaubt zielgerichtete Aktivitäten der IT
- Wer nicht weiß was wer will kann auch nicht wissen wohin er gehen muss!

Unternehmensarchitektur MIDWagen

Ziele und Anforderungen





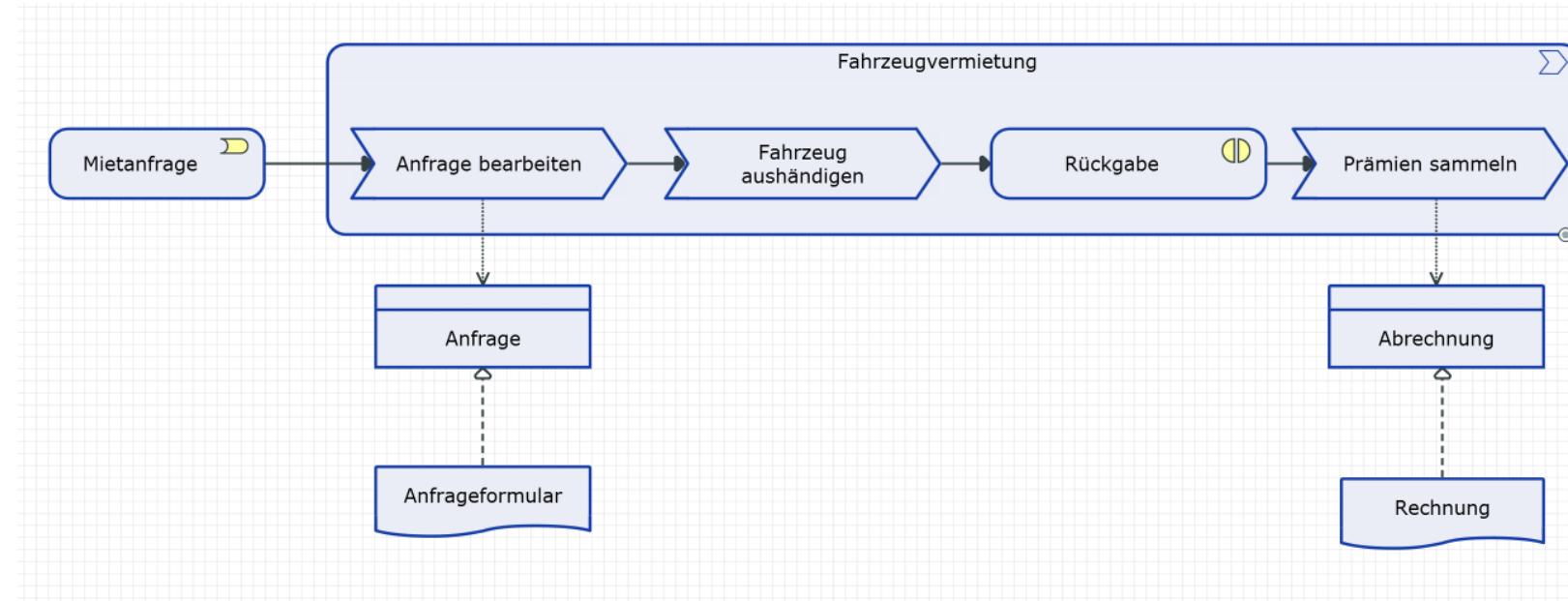
- Summe aller Beschreibungen der Geschäftsprozesse eines Unternehmens und seiner Organisationsstruktur
- Beschreibt die Aufgabenebene des Unternehmens ausgehend von der Strategie
- **Typische Elemente:** Prozesse, fachliche Funktionen, Organisationseinheiten, Rollen

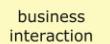
Modell und Diagramme:

- Prozesslandkarten: Bietet in mehrere Ebenen Orientierung (Hauptprozess bis Arbeitsanweisung)
- Organigramme
- Liste der angestrebten Capabilites und deren Verbindung zu Prozessen

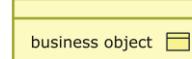
Unternehmensexarchitektur MIDWagen

Prozesssicht

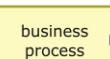


 business interaction

Beschreibt Verhalten einer Business Collaboration.

 business object

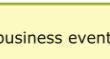
Passives Element, das aus Business Sicht relevant ist.

 business process

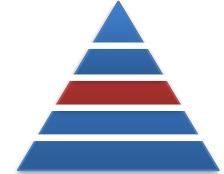
Gruppiert Verhalten basierend auf Reihenfolgen von Aktivitäten.

 representation

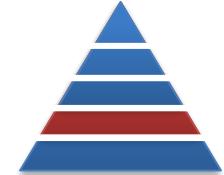
Wahrnehmbare Form der Informationen eines Business Objects.

 business event

Ein Ereignis (intern oder extern) tritt ein und beeinflusst Verhalten.



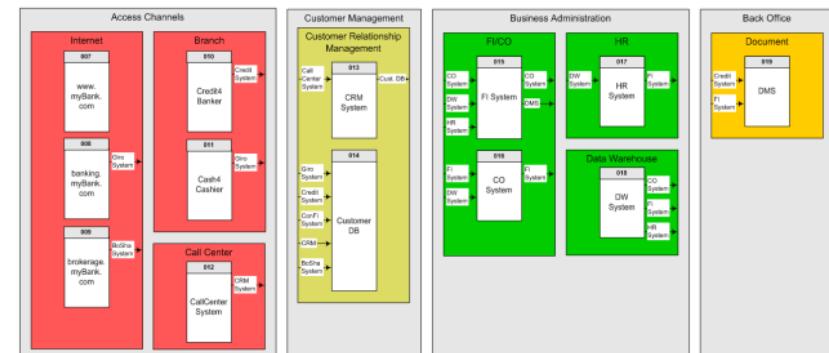
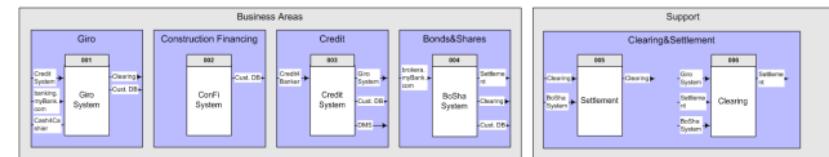
- **Typische Elemente:** Datenobjekte mit deren Beziehungen
- Fachliches Referenzmodell, damit Anwendungen nach einheitlichen Grundsätzen entwickelt werden und heterogene Anwendungen zu einem sinnvollen, in der Architektur definierten Ganzen zusammengefügt werden können
- Sollte mit Geschäftsprozessen und der Anwendungsarchitektur verknüpft werden
 - › Beschreibt wo welche Daten generiert, gelesen oder geändert werden
- **Modelle und Diagramme:**
 - › Datenmodell



- **Typische Elemente:** Anwendungen, IT Services, Realisierungs- und Nutzungsbeziehungen
- Betrachtet die gewählte/beabsichtigte Systemlandschaft
 - › Welche Einheit löst welche Aufgabe?
- Beschreibt, meist frei von fachlichen Fakten, mit welchen Arten von technischen Komponenten ein Softwaresystem/die Unternehmensarchitektur aufgebaut ist
- Stellt Bezug zu Prozessen her:
Welches System führt welchen Prozess aus?

Modelle und Diagramme:

- Clusterkarten
- Anwendungsportfolios
- Prozessunterstützungskarten

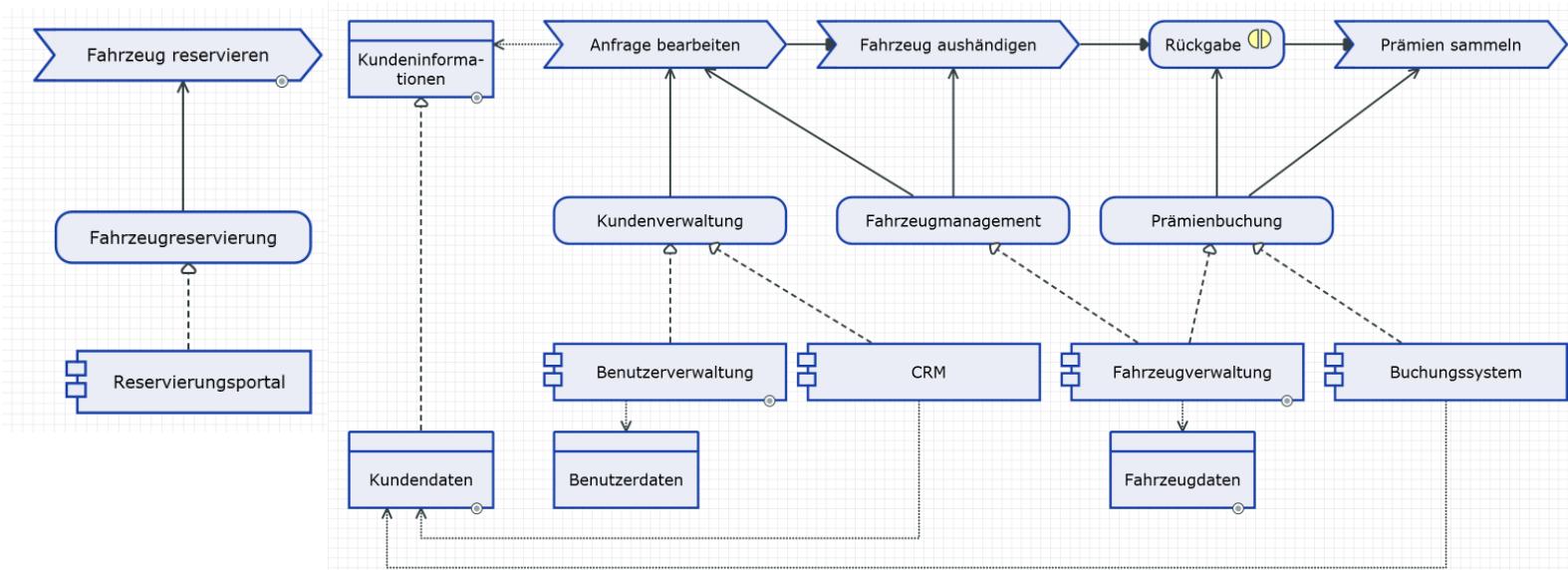


Clusterkarte

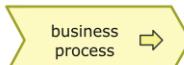
Unternehmensarchitektur MIDWagen

Application Usage Viewpoint

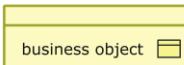
- Beschreibt wie die Anwendungen im Geschäftsumfeld verwendet werden
- Anwendungen, Abhängigkeiten zwischen Anwendungen, Services, Geschäftsprozesse, Datenobjekte



Beschreibt Verhalten einer Business Collaboration.



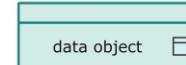
Gruppert Verhalten basierend auf Reihenfolgen von Aktivitäten.



Passives Element, das aus Business Sicht relevant ist.



Modularer Teil eines Software Systems. Kapselt Verhalten und Daten und stellt diese über Schnittstellen zur Verfügung.



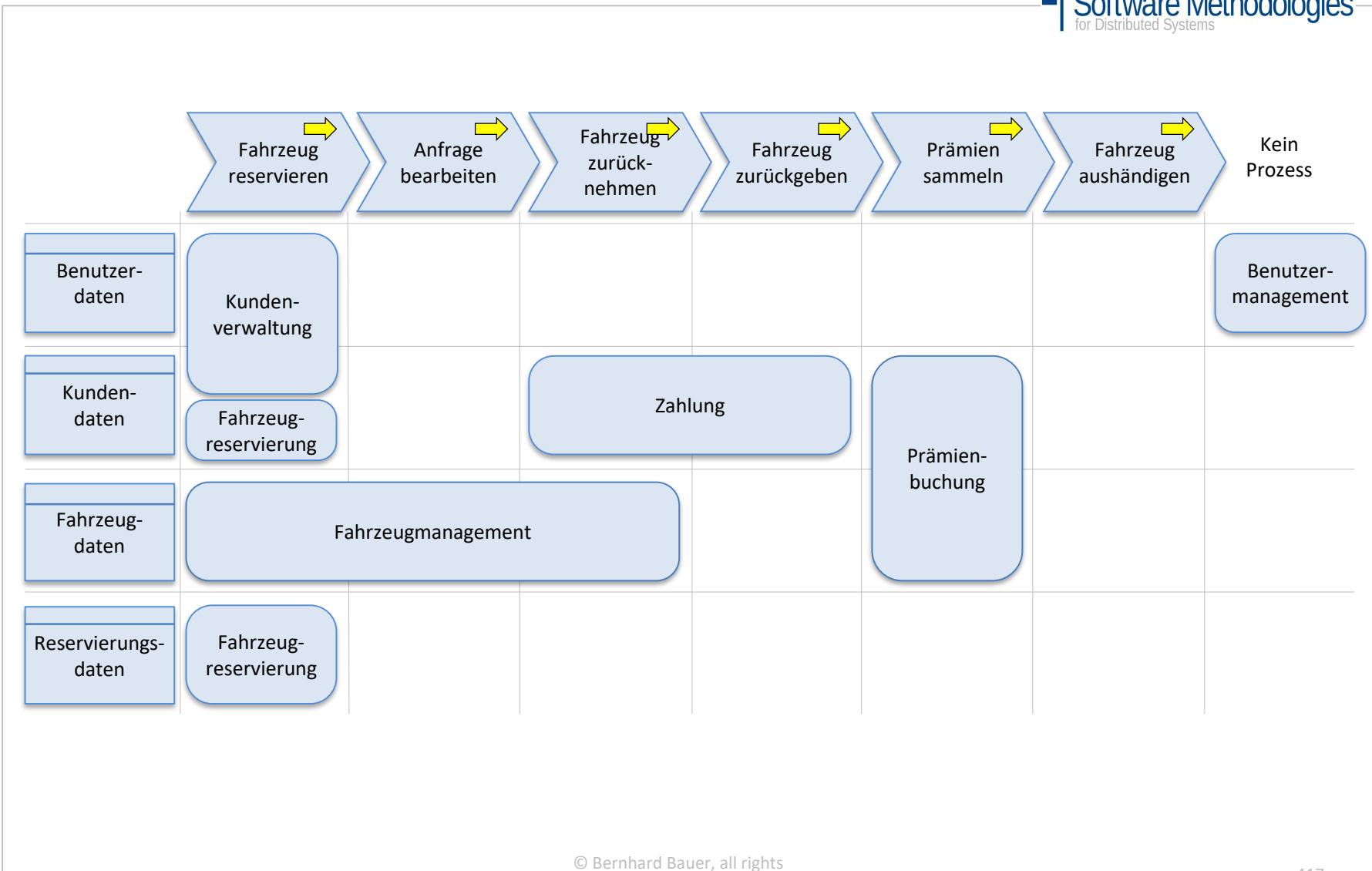
Passives Element, das automatisiert verarbeitet werden kann.



Service, der automatisiertes Verhalten zur Verfügung stellt.

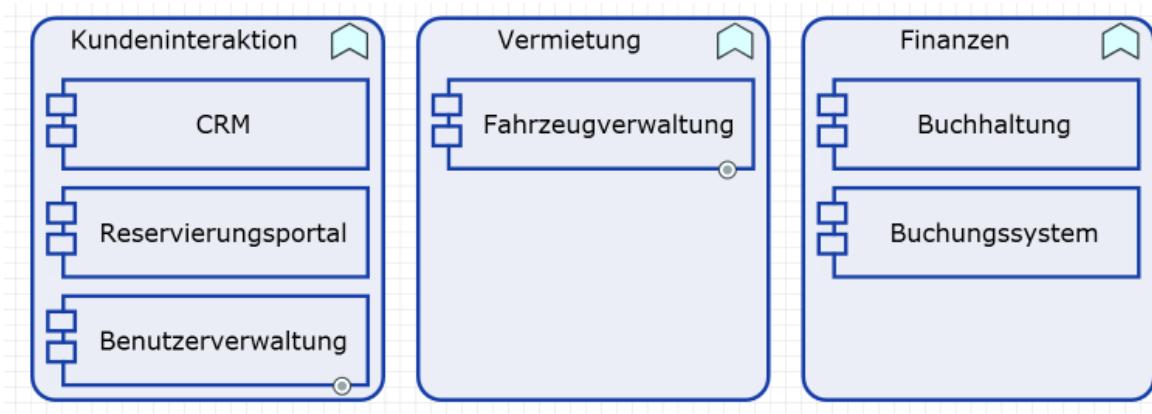
Unternehmensarchitektur MIDWagen

Prozessunterstützungskarte



Unternehmensarchitektur MIDWagen

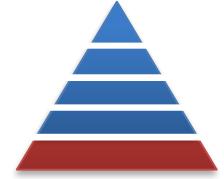
Anwendungsübersicht nach Funktionen



Gruppiert automatisiertes Verhalten, das durch eine Application Component ausgeführt werden kann.



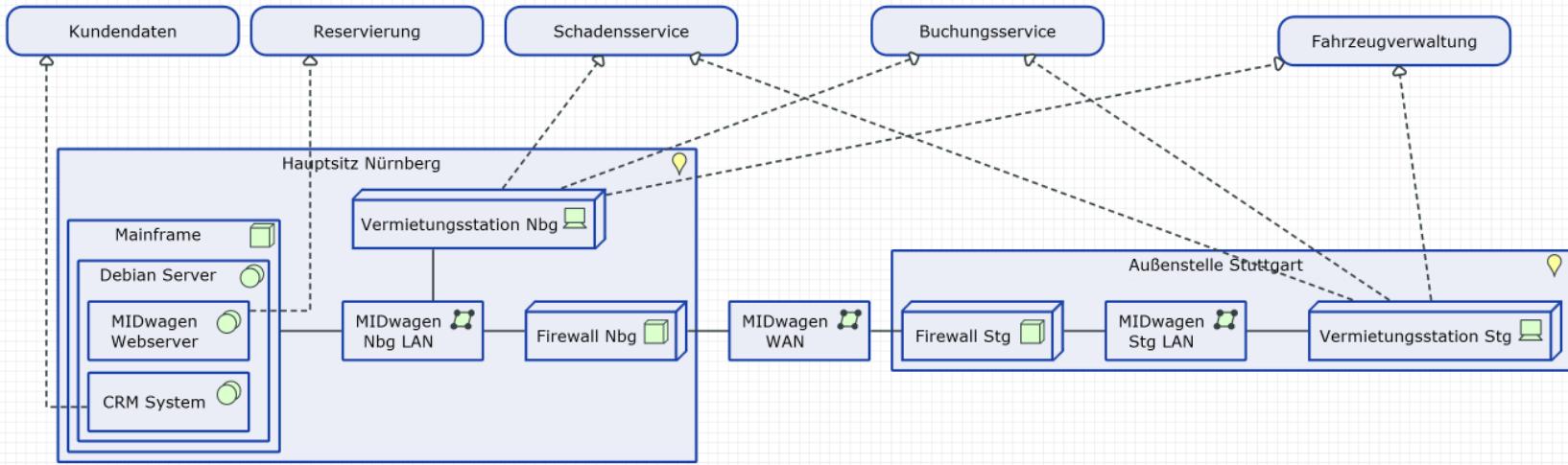
Modularer Teil eines Software Systems. Kapselt Verhalten und Daten und stellt diese über Schnittstellen zur Verfügung.



- Beschreibt die Sicht des operativen Betriebes
- Abbildung der Anwendungen auf
 - › Physische Komponenten: Rechner, Netzwerke
 - › Logische Komponenten: virtualisierte Datenspeicher und Rechner
- Bildet auch Protokolle, Service Level und Service Provider sowie deren regionale Verteilung ab
- **Typische Elemente:** Server, Netzwerke, Datenspeicher, Deployment-Beziehungen zu Anwendungen (IT Infrastrukturkomponenten und –services)
- **Diagramme und Modelle:**
 - › Clustermaps
 - › Lebenszykluskarten von Infrasktrukturkomponenten

Unternehmensarchitektur MIDWagen

Infrastruktursicht



infrastructure service 

Extern sichtbare Funktionalität, die von einem oder mehreren Nodes über wohldefinierte Interfaces zur Verfügung gestellt werden.

network 

Kommunikationsmedium zwischen zwei oder mehreren Devices.

node 

Recheneinheit auf der Artifacts eingesetzt oder gespeichert werden.

location 

Konzeptionelle Position im Raum.

device 

Hardware, auf der Artifacts eingesetzt oder gespeichert werden.

system software 

Softwareumgebung für bestimmte Arten von Komponenten und Objekten, die in Form von Artifacts eingesetzt werden.

Governance Modell ist grundlegend für den Erfolg des EAM

■ Ziele einer EA Governance

- › Sichert die fachliche Beteiligung und dass Erreichen von realen Geschäftsnutzen
- › Fördert das andauernde Business-IT Alignment
- › Steuert die Einführung von Strategien und Standards

■ Aufgaben

- › Festlegung und Betrieb von entsprechenden Gremien (mit ihren Rollen, Verantwortlichkeiten und Entscheidungsrechten um den Betrieb und die Evolution der EA sicherzustellen)
- › Aufsetzen von Guidelines und Standards zum Sicherstellen dass die richtigen Dinge zum richtigen Zeitpunkt getan werden
- › Integration mit Projektlebenszyklen und anderen Organisationsprozessen

Besteht aus Entscheidungs- und Arbeitsgremien auf verschiedenen Ebenen

- › Kontrollieren die Weiterentwicklung der EA im Sinne der Geschäftsstrategie
- › Auch Eskalationsinstanz, wenn z.B. Anforderungen keinem strategischen Unternehmensziel dienen
- › Sollten Projekt freigabeveto haben
- › Sichern Konformität der Architektur mit den definierten Architektur-Richtlinien
 - » Richtlinien bestimmen z.B. Verantwortlichkeiten für Architekturbausteine
- › Aktive Einbindung aller relevanten Prozessteilnehmer (Manager, IT-Architekturen und –Entwickler auf Fach- und IT-Seite)
- › Definition von Rollen und Verantwortlichkeiten
- › Kommunikation der Organisationstrukturen und Standards

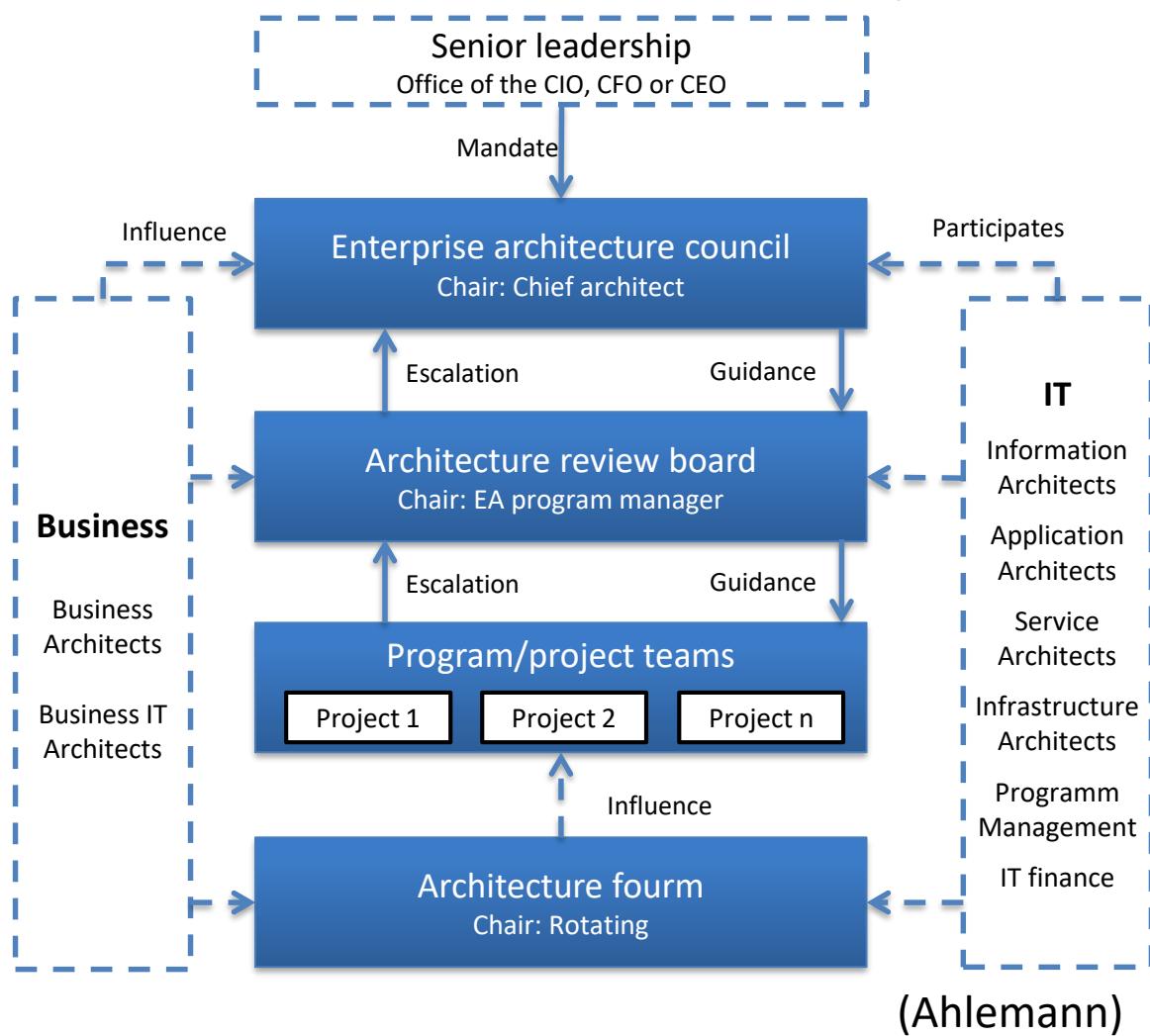
(Bitkom)

Hauptgremien einer Governance:

- Enterprise architecture council (EAC)
- Architecture Review Board (ARB)
- Architecture Forum

Abhängig von den EAM Zielen, dem Reifegrad und der Organisationsstruktur zweckmäßige Auswahl und Zusammensetzung der Gremien

- Zu Beginn z.B. nur Architecture Forum
- Anschließend ein Gremium als Kombination aus EAC und ARB



- Enterprise Archicture Council
 - › Festsetzen von Best Practices, leitenden Prinzipien, Standards, Referenzarchitetturen und anderen architektonischen Richtlinien
 - › Einsetzen von Arbeitsgruppen zur Entwicklung und Wartung dieser
- Architecture Review Board
 - › Kontakt zu den Projektteams
 - › Sicherstellen der Einhaltung von Architekturstandards
- Architecture Forum
 - › Weniger Formale Struktur
 - › Austausch zwischen Interessensparteien

- Motivation und Problemstellung
- Elemente einer EA
- **Frameworks für das EAM**
 - › Beispiel: TOGAF
- Modellierung von EA
 - › Beispiel: Archimate
- Tooling

Framworks für das EAM

■ | **Software Methodologies**
for Distributed Systems

- Stellen Referenzarchitekturen, Methoden, Checklisten und Best Practice Prozesse bereit
- Frameworks strukturieren die Architekturbeschreibung durch Identifikation und Verknüpfung verschiedener Architektur-Viewpoints sowie Vorschlägen für Modellierungstechniken
- Praxisnaher Startpunkt für das EAM

EA Modeling: Techniken um die Architektur zu beschreiben

EA Tools: Management der EA

Nicht verwechseln mit Programmierframeworks aus der Softwareentwicklung.

- Komponenten eines Frameworks:
 - › Referenzunternehmensarchitektur
 - › Gemeinsames Vokabular bzw. Glossar
 - › Anleitung und Werkzeuge für die Konzeptualisierung und Dokumentation der EA
 - › Methoden für die Planung und Realisierung der EA bzw. für die Einführung eines EAM

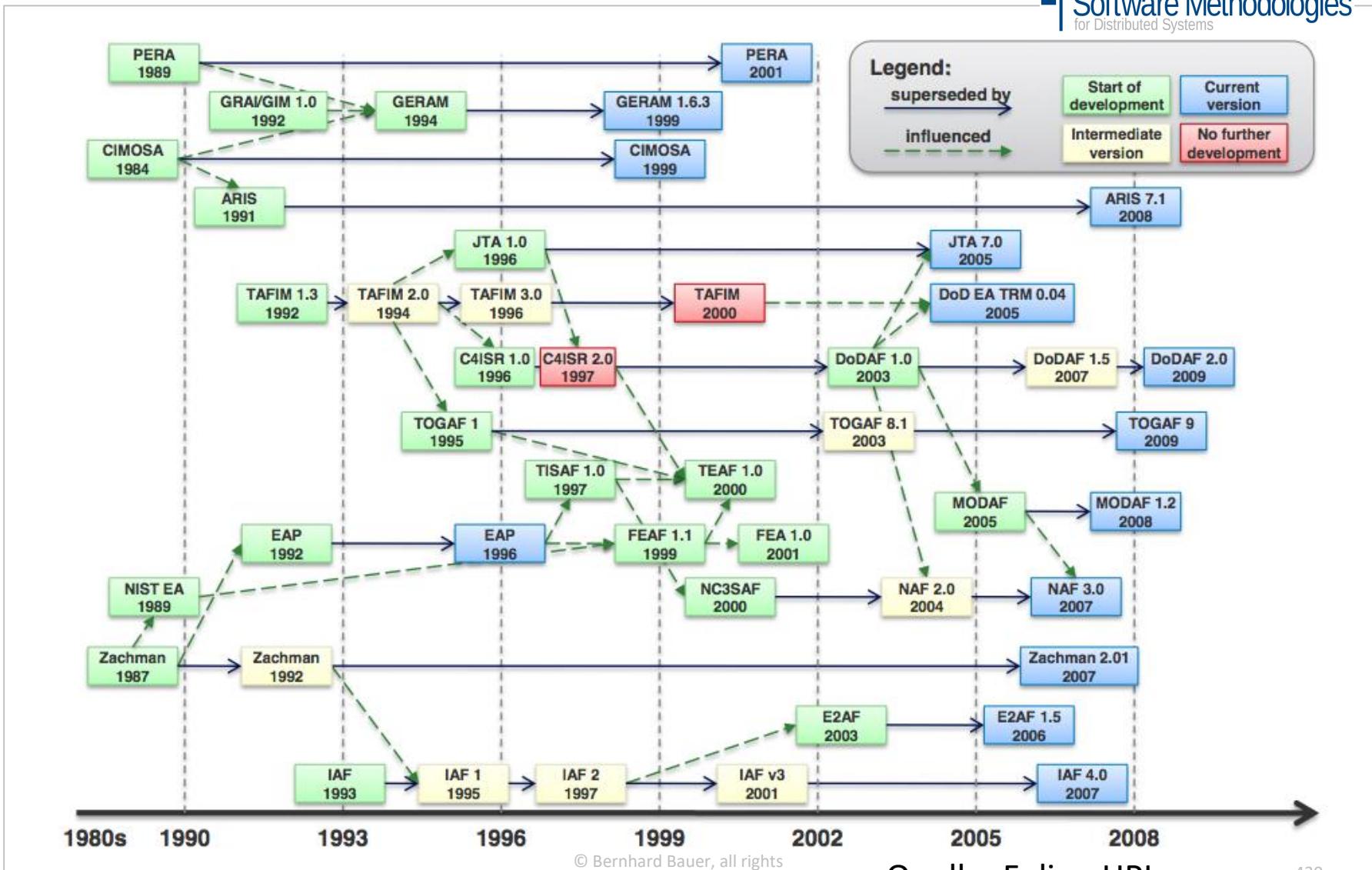
Alternativen:

- Auswahl eines bereits existierenden Frameworks
- Anpassung eines existierenden Frameworks
- Erstellung eines komplett neuen Frameworks

Relevante Dimensionen zur Auswahl (nach Ahleman):

- › Taxonomy completeness
- › Process completeness
- › Scope
- › Level of detail
- › Addressed stakeholders
- › Reference model guidance
- › Practice guidance
- › Maturity model
- › Business focus
- › Governance guidance
- › Partitioning guidance
- › Prescriptive catalogue
- › Representation
- › Vendor neutrality
- › Information availability
- › Time to value
- › Transformation

Übersicht EAM Frameworks



- 1987 als eines der ersten Frameworks für EA von seinem Namensgeber John Zachman eingeführt
- Konzepte werden in Frameworks wie FEAf, DoDAF wieder verwendet
- Keine Methode, beschreibt sich selber als Ontologie und Metamodell
- Kann mit jedem Tool bzw. jeder Methodik verwendet werden, da es hiervon unabhängig ist
- Zwei-dimensionales Klassifizierungsschema für die Beschreibung eines Unternehmens
- Beschreibt ein ganzheitliches Modell des Unternehmens aus 6 Perspektiven
 - Executive, Business Management, Architect, Engineer, Technician, Enterprise
- Berücksichtig in jeder Perspektive 6 Dimensionen
 - What?, How?, Where?, Who?, When?, Why?
- Keine methodische Unterstützung bei der Etablierung des Frameworks
- Keine methodische Unterstützung bei EA-Prozessen (Einführung und Evolution)

Zachman Framework

The Zachman Framework for Enterprise Architecture™ The Enterprise Ontology™



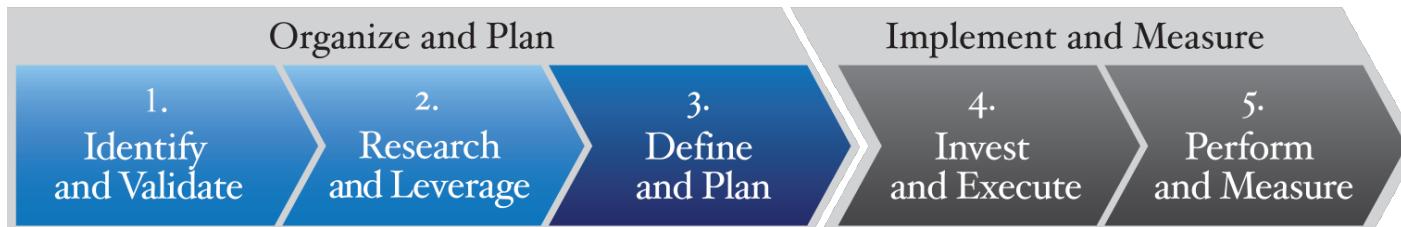
© 1987-2011 John A. Zachman, all rights reserved. Zachman® and Zachman International® are registered trademarks of John A. Zachman.

To request Permission Use of Copyright, please contact: Zachman.com

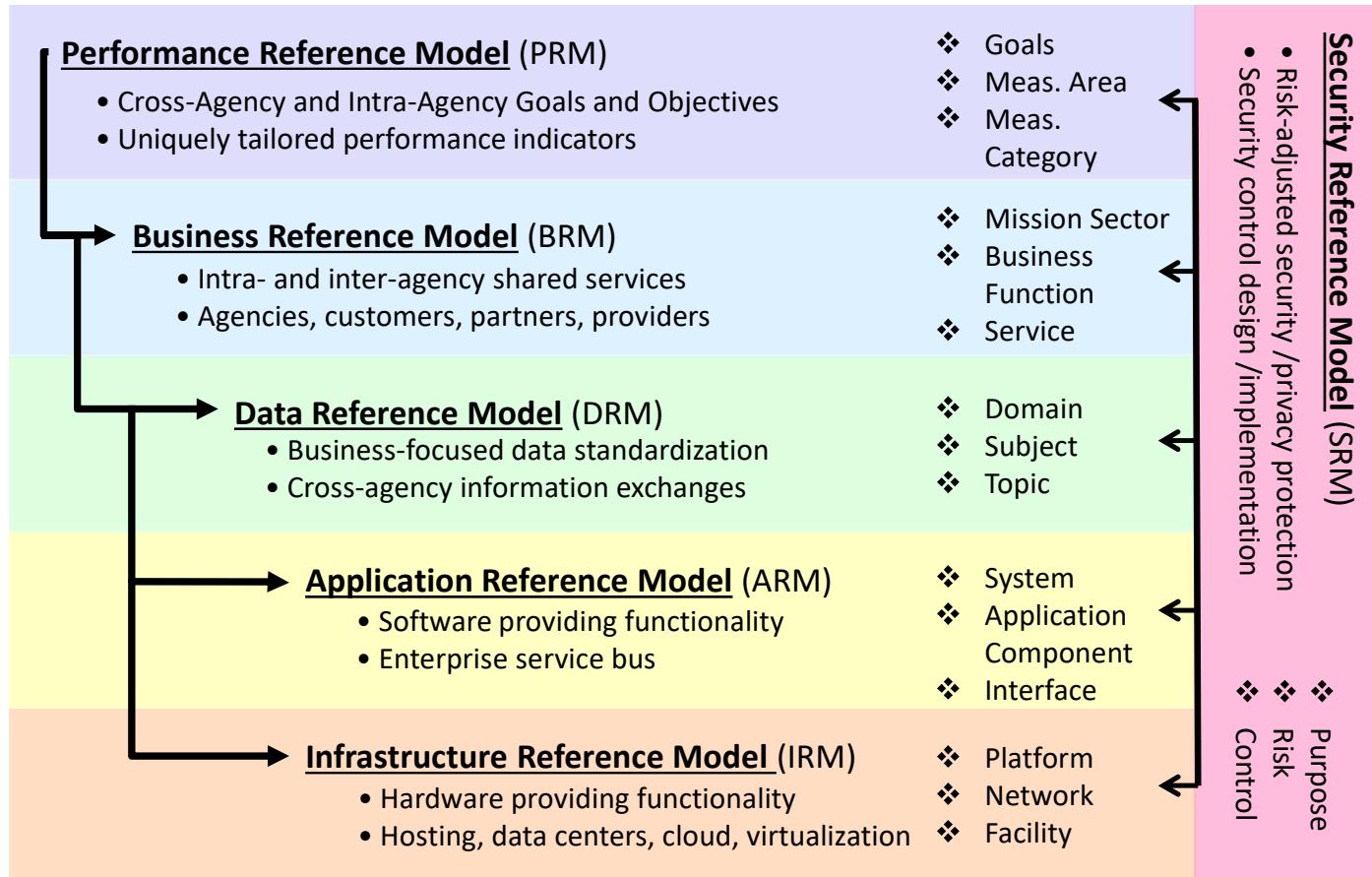
© Bernhard Bauer, all rights
reserved 2018

- Die Spalten zielen auf die W-Fragen ab: „Was?“, „Wie?“, „Wo?“, „Wer?“, „Wann?“, „Warum?“
- Jede Zeile beantwortet damit die Fragestellung „Was wird wo, wann, wieso, von wem und wie gemacht?“ aus einer bestimmten Perspektive
- Jede Zelle ist einzigartig und unterscheidet sich von jeder anderen Zelle.
- Inhalt einer Zelle umfasst eine Architektureigenschaft bezüglich eines Aspekts des Systems für eine bestimmte Personengruppe
- Beziehungen zwischen den Zellen sind nicht im Detail im Framework beschrieben
- Alle Zellen zusammen stellen die Architektur dar

- Beschreibt den Aufbau und die Nutzung einer Unternehmensarchitektur für Regierungsorganisationen
- Herausgegeben vom Weißen Haus
- Ziele des Frameworks:
 - Behörden-übergreifende Analysen
 - Identifizierung von doppelten Investitionsmaßnahmen
 - Lücken und Chancen für Zusammenarbeit innerhalb einer und zwischen verschiedenen Behörden
 - Planungsunterstützung bei der Transformation der Regierung zu Erreichung eines definierten Zielzustands
- **Consolidated Reference Model (CRM)**
 - Definiert eine gemeinsame Sprache
 - Besteht aus einer Menge von zusammenhängenden Referenzmodellen, welche die folgenden 6 Domänenarchitekturen des Frameworks beschreiben
 - » Strategy, Business, Data, Applications, Infrastructure, Security
- **Collaborative Planning Methodology:** Beschreibt die Schritte zum Erarbeiten einer Transitionsstrategie, die es ermöglicht den gewünschten Soll-Zustand zu erreichen



Consolidated Reference Model (CRM)



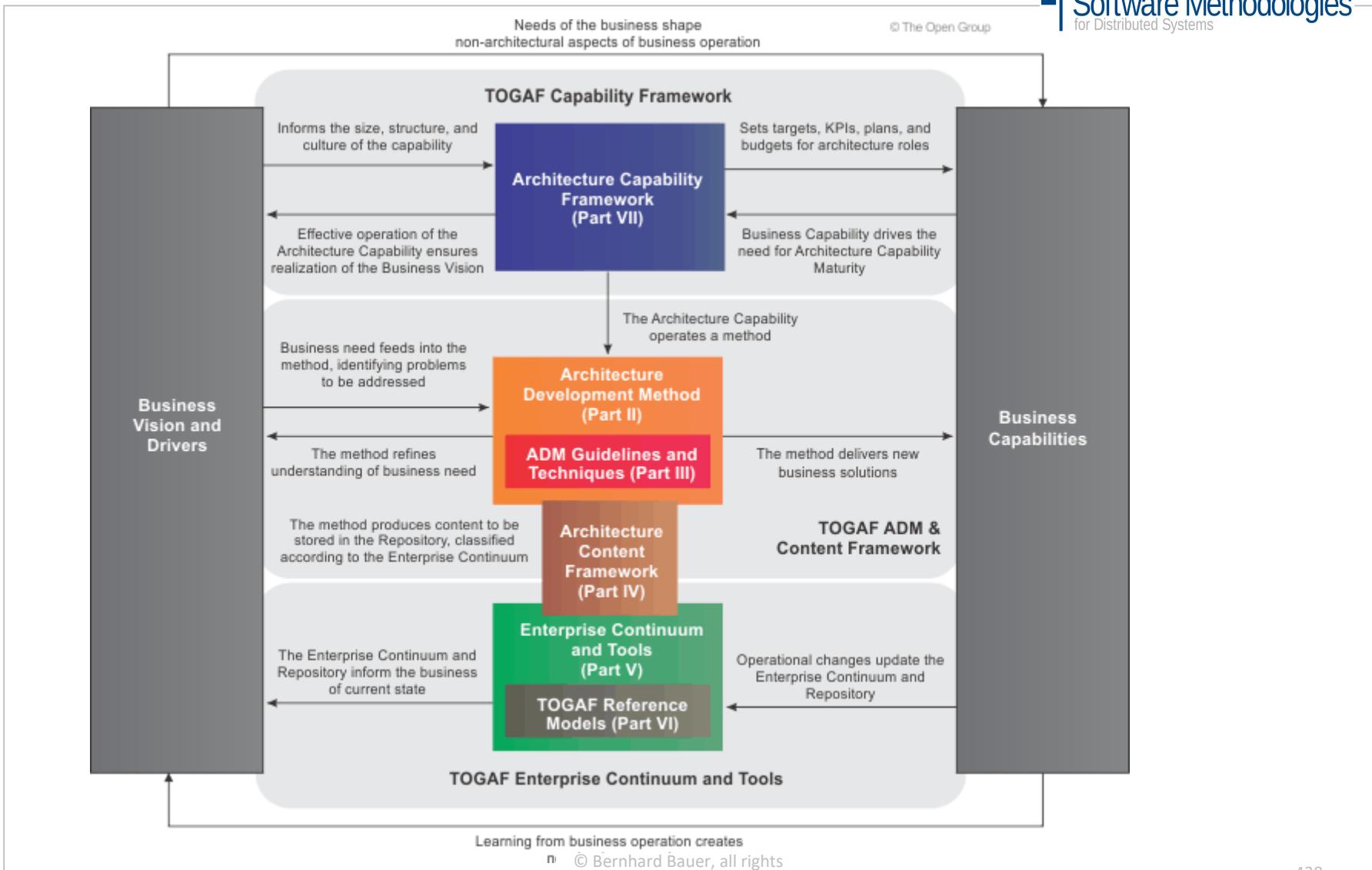


- 1995 im Rahmen des Architekturforums der Open Group, mit der Zielsetzung eines allgemein anerkannten und unabhängigen Standards, entwickelt
- Entstand aus den Best-Practices von über 300 Unternehmen weltweit und wird kontinuierlich von diesen weiterentwickelt

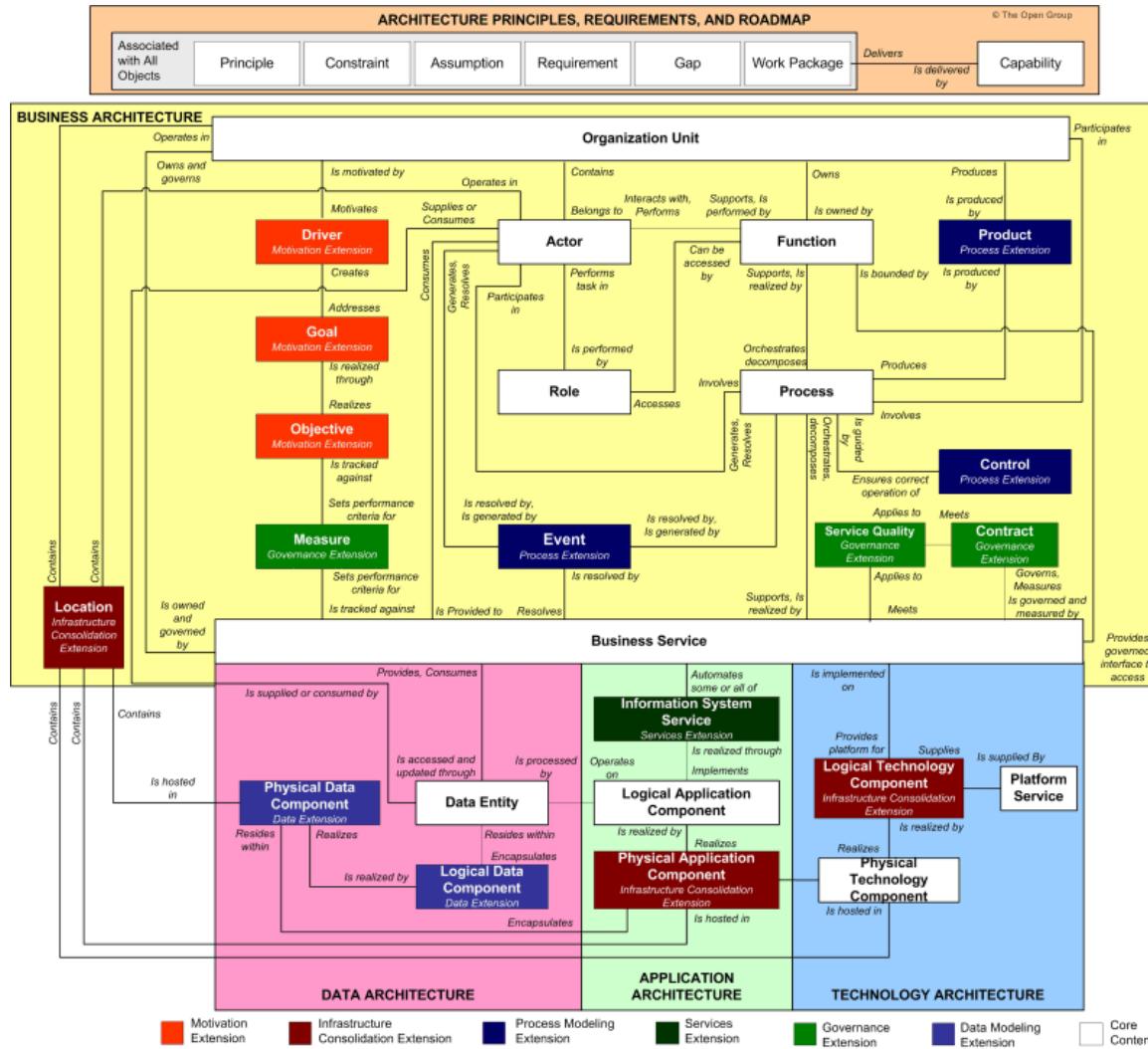
Key Facts:

- Vier Architekturebenen
 - › Geschäftsarchitektur, Daten, Anwendungen, Technologie
- Richtlinien für die Auswahl und Entwicklung bestimmter Sichten auf die einzelnen Ebenen, schreibt explizit jedoch keine Sichten fest vor
- Zentraler Bestandteil ist die Architecture Development Method ADM
 - › Schrittweises Vorgehen zur Entwicklung einer EA
 - › Iterativen Prozessablauf, um die Architektur kontinuierlich aufzubauen und zu verbessern

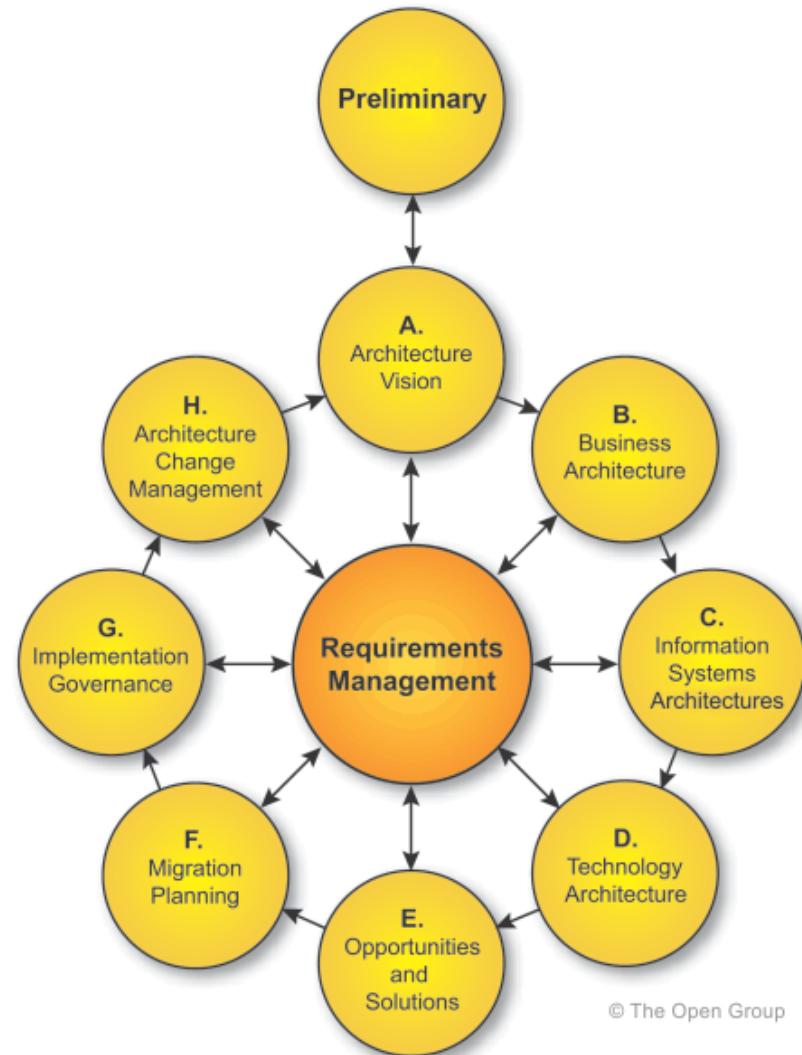
- **Part I: Introduction**
 - › Allgemeine Einführung der wichtigsten Konzepte von TOGAF
 - › Definition von Grundbegriffen
- **Part II: Architecture Development Method**
 - › Kern von TOGAF: Beschreibt ein Vorgehensmodell zur Entwicklung einer Unternehmensarchitektur
 - › Phasenbasierter, iterativer Ansatz der sowohl Planungs- als auch Realisierungsaufgaben umfasst
- **Part III: ADM Guidelines and Techniques**
 - › Sammlung von Richtlinien und Techniken für TOGAF und ADM
- **Part IV: Architecture Content Framework**
 - › Metamodel zur Beschreibung von Architekturartefakten und wiederverwendbaren „Building Blocks“
 - › Beschreibung von typischen Architektur-Deliverables
- **Part V: Enterprise Continuum & Tools**
 - › Enterprise Continuum bietet Strukturierungsmöglichkeiten um die produzierten Ergebnisse zu kategorisieren und zu speichern
 - › Beschreibt ein Repository, in dem Architekturreferenzen, -beschreibungen und Muster abgelegt werden, um diese zu einem späteren Zeitpunkt erneut wiederverwenden zu können
- **Part VI: TOGAF Reference Models**
 - › Technical Reference Model (TRM): Referenzmodell zur Umsetzung beliebiger Systemarchitekturen
 - › Integrated Information Infrastructure Reference Model (III-RM): Schwerpunkt liegt auf Anwendungskomponenten und -software
- **Part VII: Architecture Capability Framework**
 - › Beschreibt die benötigte Organisationsstruktur, Prozesse, Fähigkeiten, Rollen und Verantwortlichkeiten um eine Architekturfunktion innerhalb eines Unternehmens aufzubauen und umzusetzen und in allen Phasen der Architekturentwicklung eine Verbindung zwischen den strategischen und den geschäftlichen Zielen des Unternehmens herzustellen



Part IV Architecture Content Framework: TOGAF Content Metamodel



- Methode für die Entwicklung und das Management des Lebenszyklus einer Unternehmensarchitektur
- Iterativer Ansatz: der ganze Prozess, zwischen den Phasen und während einer Phase
- Detailgrad der Betrachtungen und der zeitliche Horizont bleiben offen
- Keine Festlegung von konkreten Modellierungssprachen oder Artefakten für die einzelnen Phasen
- Guidelines, Templates und Checklisten zur Anwendung des TOGAF ADM werden in Part III vorgeschlagen
- Möglichkeit der Anpassung an unternehmenspezifische Besonderheiten oder andere EA Frameworks
- Umfang der Architekturinitiative muss vom Unternehmen selber festgelegt werden
- Breite und Tiefe können mit jeder Iteration verfeinert bzw. weitere Architekturendokumente ergänzt werden



Preliminary

- Vorbereitende Aktivitäten zur Erstellung einer neuen Unternehmensarchitektur
- Aufbau der Architekturfunktion im Unternehmen: Organisationsmodell, Festlegung von Prozessen und Ressourcen für die Steuerung, Auswahl von Tools, Anpassung des Frameworks an das Unternehmen

Ergebnisse: Organisationsmodell, angepasster Architekturframework, initiales Architekturepository, Geschäftsprinzipien, Ziele, Architektur Governance Framework

A: Architecture Vision

- Entwicklung einer High-Level Architektur Vision sowie Fähigkeiten und Geschäftsnutzen die durch die EA erreicht werden sollen
- Betrachtung bereits existierende Architekturen hinsichtlich ihrer Wiederverwendbarkeit
- „Statement of Architecture Work“ beinhaltet die Ergebnisse dieser Phase

Ergebnisse: Verfeinerte Prinzipien und Ziele, Assessment der Fähigkeiten, Architektur Vision, Entwurfsdokument zur Definition der der Ist- und Zielarchitektur

B: Business Architecture (Geschäftsarchitektur)

- Geschäftsarchitektur beschreibt Produkt- und Servicestrategien, organisatorische, funktionelle und geographische Aspekte, sowie Informationen und Prozesse der Geschäftsumwelt.
- Aufnahme der aktuellen Ist-Geschäftsarchitektur
- Entwicklung der Ziel-Geschäftsarchitektur, welche die Anforderungen an das Unternehmen beschreibt um die Ziele und Strategien der Vision zu erreichen
- Basierend auf den Unterschieden zwischen den beiden Architekturen (Gap Analyse) werden Komponenten für eine Roadmap vom Ist zum Ziel identifiziert

Ergebnisse: Verfeinerung bestehender Artefakte, detaillierte Ist- und Ziel-Geschäftsarchitektur, Entwurfsspezifikation für Architekturanforderungen (mit Ergebnissen der Gap Analyse, technischen Anforderungen und aktualisierten Geschäftsanforderungen), Komponenten der Geschäftsarchitektur für die Roadmap

C: Information System Architecture (Informationssystemarchitektur)

- Beschreibt die Daten- und Anwendungsarchitektur
- Aufnahme der aktuellen Ist-Daten- und Anwendungsarchitektur
- Entwicklung der Ziel-Daten- und Anwendungsarchitektur, welche beschreibt wie die Geschäftsarchitektur und die Architekturvision ermöglicht werden
- Basierend auf den Unterschieden zwischen den beiden Architekturen (Gap Analyse) werden Komponenten für eine Roadmap vom Ist zum Ziel identifiziert

Ergebnisse: Verfeinerung bestehender Artefakte, detaillierte Ist- und Ziel-Informationssystemarchitektur, Entwurfsspezifikation für Architekturanforderungen (mit Ergebnissen der Gap Analyse, technischen Anforderungen, aktualisierten Geschäftsanforderungen und Einschränkungen für die Technologiearchitektur), Komponenten der Daten- und Anwendungsarchitektur für die Roadmap

D: Technology Architecture (Technologiearchitektur)

- Beschreibt Technologiekomponenten und Plattformen, ihre Verortung, die physische Netzwerkkommunikation sowie Hardware und Netzwerk Spezifikationen
- Aufnahme der aktuellen Ist-Technologiearchitektur
- Entwicklung der Ziel-Technologiearchitektur, welche die logischen und physischen Anwendungen und Datenkomponenten realisiert und damit die Architekturvision und Interessen der Stakeholder erfüllt

Ergebnisse: Verfeinerung bestehender Artefakte, detaillierte Ist- und Ziel-Technologiearchitektur, Entwurfsspezifikation für Architekturanforderungen (mit Ergebnissen der Gap Analyse, aktualisierten technischen Anforderungen), Komponenten der Technologiearchitektur für die Roadmap

E: Opportunities and Solutions

- Zusammenführung und Konsolidierung der in den Phasen B, C und D identifizieren Roadmap Komponenten
- Definition von Transtitionsarchitekturen (Zwischenarchitekturen) für eine inkrementellen Entwicklung der Ziel-Architektur

Ergebnisse: ggf. Verfeinerung bestehender Artefakte, konsolidierte Unterschiede zwischen Ist und Ziel, Bewertung der Lösungen und entwickelten Fähigkeiten, Architektur Roadmap (Portfolio der Arbeitspakete, Transitionsarchitekturen, Empfehlungen zur Umsetzung)

F: Migration Planning

- Abstimmung der geplanten Änderungen mit anderen Projekten und Integration in das Change Portfolio Sicherstellen das der Geschäftsnutzen und die Kosten der Arbeitspakete von den Haupt-Stakeholdern verstanden wurden
- Erstellung eines Realisierungs- und Migrationsplans in Kooperation mit den Portfolio- und Projektmanagern

Ergebnisse: Realisierungs- und Migrationsplan, finalisierte Architektdokumentation und Roadmap

G: Implementation Governance

- Steuerung der einzelnen Realisierungsprojekte
- Sicherstellen der Konformität der Projekte mit der Ziel-Architektur
- Steuerung von Änderungsanforderungen

H: Architecture Change Management

- Sicherstellen das die Steuerungsfunktionen ausgeführt werden und die Architektur gepflegt wird
- Sicherstellen dass die Architektur den Geschäftswert erfüllt
- Kohäsives und architekturbasiertes Änderungsmanagement
- Überwachung der Ergebnisse der Steuerungsfunktion, von neuen Entwicklungen in der Technologie und von Veränderungen im Geschäftsumfeld
- Entscheidung über die Initiierung eines neuen Architektur-Evolutionszykluses bei Änderungen

Requirements Management

- Dynamischer Prozess zur Identifizierung und Speicherung von Anforderungen an die Unternehmensarchitektur sowie Änderungen dieser begleitend zu allen Phasen des ADM-Zykluses
- Bereitstellen der relevanten Anforderungen in einer Phase des ADM Zyklus
- Verwaltung der Anforderungen ist zyklus-übergreifend
- Sicherstellen dass alle Änderungen an Anforderungen durch entsprechende Steuerungsprozesse abgestimmt werden und damit mit allen anderen Phasen berücksichtigt sind
- Sicherstellen das architekturelle Anforderungen für eine Phase rechtzeitig vorhanden sind
- Kein Verwerfen, Zuweisen oder Priorisieren von Anforderung. Dies geschieht in den entsprechenden Phasen des ADM

Die einzelnen Phasen des ADM sind weiter in Schritte unterteilt.

Schritte für die Architekturentwicklungsphasen (B, C und D):

- › Auswahl von Referenzmodellen, Viewpoints und Tools
- › Aufnahme und Beschreibung der Ist-Architektur
- › Entwicklung und Beschreibung der Ziel-Architektur
- › Durchführen einer Gap Analyse zur Identifizierung der Unterschiede zwischen Ist und Ziel
- › Definition von möglichen Roadmap-Komponenten
- › Auflösung von Auswirkungen auf die restliche Architektur
- › Durchführen eines Stakeholder-Review
- › Finalisieren der Architekturbeschreibungen
- › Erstellung des Dokuments zur Architekturdefinition

Gründe für Einschränkungen einer Architekturintiative:

- Zuständigkeit/Einfluss des Architekturteams
- Ziele und Interessen der Stakeholder an die Architektur
- Verfügbarkeit von Personal, Geld und anderen Ressourcen

Dimensionen zur Einschränkungen des Umfangs:

- **Breite:** Was gehört zum Unternehmen und welche Teile des Unternehmens sollen in der Architektur abgebildet werden?
 - › Sehr große Unternehmen bestehen oft aus mehreren Teilbereichen, die als eigenes „Unternehmen“ betrachtet werden können
 - › Unternehmensgrenzen heutiger Unternehmen können nicht mehr klar definiert werden, durch intensive Zusammenarbeit mit Zulieferern, Kunden und Partnern
- **Tiefe:** Bis zu welchem Detaillevel soll die Architektur erfasst werden? Wie viel Architektur ist genug?
 - › Abwägung von Aufwand und Nutzen
 - › Abgrenzung zu anderen Aktivitäten wie Prozessmanagement und Software Architektur
- **Zeitliche Einschränkung:** Was ist der Zeithorizont für die Architekturvission?
 - › Detailgrad der Architekturbeschreibung für diese Vision hinsichtlich Machbarkeit und Ressourcen
 - › Ggf. Definition von Zwischenarchitekturen für die nähere Zukunft
- **Architekturdomänen:** Umfang der Architekturbeschreibung
 - › Zeitliche und finanzielle Rahmenbedingungen, sowie mangelnde Ressourcen, erschweren die „all-inclusive“ Erstellung einer Unternehmensarchitektur (Volle Berücksichtigung von Geschäftsbereich, Daten, Anwendungen und Technologie)
 - › Vorher festlegen, wo der Umfang der Architekturdomänen eingeschränkt werden kann (Aufwand-Nutzen Verhältnis beachten)

- Referenzmaterial zum Aufbau einer Architekturfunktion im Unternehmen
 - › Benötigte Organisationsstrukturen, Prozesse, Rollen, Verantwortlichkeiten und Fähigkeiten
- **Referenzprozess** zum Aufbau einer Architekturfunktion im Unternehmen (orientiert an den Phasen des ADM):

A: Architecture Vision	Identifizierung von Stakeholder, Prinzipien, Zielen und der Architekturvision, Festlegung von Architekturprinzipien
B: Business Architecture	Definition der Architekturbegriffe, des Architektur-Frameworks und eines Architektur-Prozesses, Auswahl von Sichten und Performanzmetriken sowie einem Governance-Framework
C: Data Architecture	Spezifikation und Steuerung des Enterprise Continuum und Architekturrepository
C: Application Architecture	Auswahl eines Toolsets zum Erstellen, Warten, Veröffentlichen, Verteilen und Steuern der Architekturdokumente

Part VII: Architecture Capability Framework

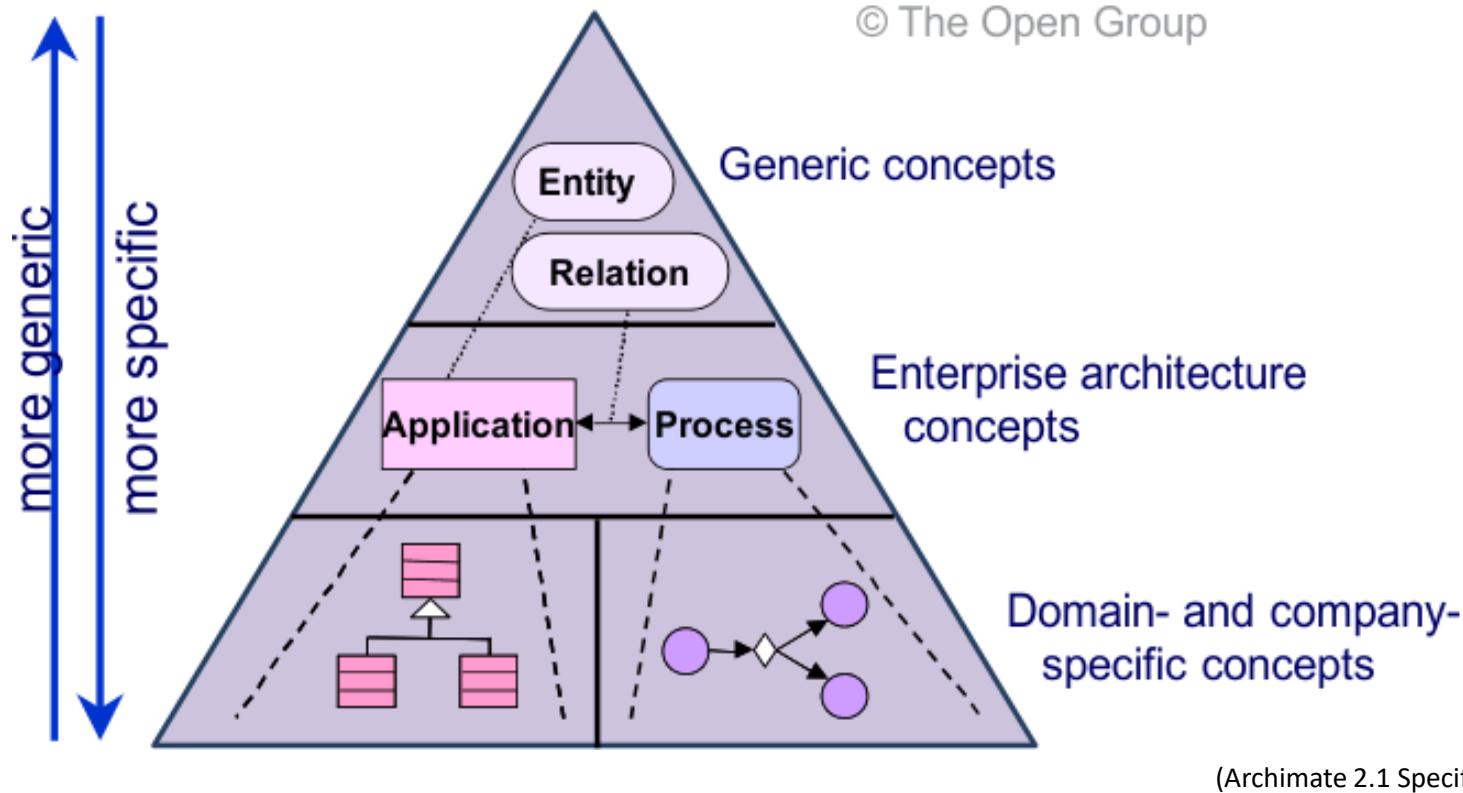
D: Technology Architecture	Definition der benötigten technischen Infrastruktur
E: Opportunities & Solutions	Definition der Änderungen in der Unternehmensorganisation zur Realisierung der Architekturfunktion
F: Migration Planning	Aufsetzen der benötigten Tools, Umsetzungsplanung der definierten Architekturprozesse und Frameworks im Unternehmen
G: Implementation Governance	V.A. Steuerung der Umsetzung der definierten Business Architecture über die definierten Änderungsprozesse im Unternehmen
H: Architecture Change Management	Management von Änderungen an die Architekturfunktion, zum Beispiel die Notwendigkeit für ein neues Architekturdokument
Requirements Management	Management der Anforderungen an die Architekturfunktion

- Motivation und Problemstellung
- Elemente einer EA
- Frameworks
 - › Beispiel: TOGAF
- **Modellierung von EA**
 - › Beispiel: Archimate
- Tooling

- EA Modelle veranschaulichen Architekturbeschreibungen
- Repräsentieren verschiedene Sichten auf die Architektur
- Verschiedene Modellierungstechniken die sich hinsichtlich Abstraktionsgrad, berücksichtigten Schichten, graphischer Repräsentation, ... unterscheiden
- EA Modellierung: Erstellung einer abstrakten Abbildung des Unternehmens
- Abhängig vom Ziel und Fokus des EAM entscheidet der Architektur welche Teile des Unternehmens relevant sind abgebildet werden sollen
- Wiederverwendung existierender Modellierungstechniken <-> Entwicklung einer eigenen
- Wiederverwendung:
 - › Kombination verschiedener Modellierungssprachen
 - » z.B. Kombination von UML Modellen
 - › Verwendung einer Sprache, die die gesamte EA abdeckt
 - » z.B. Archimate, UML-Profile UML4ODP für RM-ODP oder UPDM für DoDAF/MODAF

- Stellt eine einheitliche Darstellung für Diagramme zur Beschreibung von Unternehmensarchitekturen bereit
- Bietet einen integrierten Ansatz zur Beschreibung und Visualisierung der verschiedenen Architekturdomänen und ihrer Beziehungen und Abhängigkeiten
- Baut auf dem Konzept der „Service-Orientierung“ auf
- Berücksichtigung der Entwicklung einer Architektur über die Zeit (Transformations- und Migrationsplanung)
- Entwicklung des Archimate Standards ist eng an die Entwicklung des TOGAF Standards gekoppelt
- Inhalt der Archimate Spezifikation
 - › Visuelle Beschreibungssprache mit entsprechenden Konzepten zur Beschreibung von Abhängigkeiten zwischen den Domänenarchitekturen
 - › Definition von Viewpoints für Stakeholder

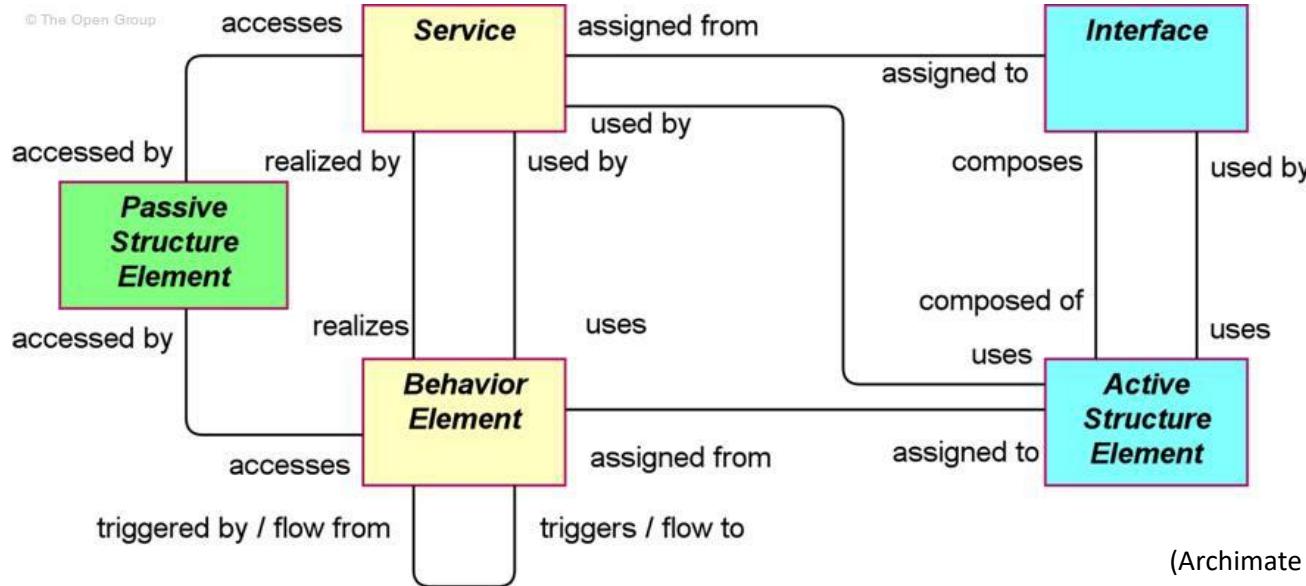
Abstraktionsgrad von EA Konzepten



Eine Beschreibungssprache für Unternehmensarchitekturen muss die Mitte bilden zwischen den spezifischen Sprachen der einzelnen Domänen, z.B. UML oder BPMN, und den sehr allgemeinen Architekturkonzepten („Entity“ und „Relation“).

- Schichten der ArchiMate Sprache:
 - › **Business Layer:** Beschreibt die angebotenen Produkte und Service, welche im Unternehmen durch Geschäftsprozesse, durchgeführt von Akteuren, realisiert werden.
 - › **Application Layer:** Unterstützt die Business Layer mit Anwendungsservices, realisiert durch Anwendungen
 - › **Technology Layer:** Beschreibt die angebotenen Infrastrukturservices (z.B. Bearbeitung, Speicherung, Kommunikationsservices), welche zum Betrieb der Anwendungen notwendig sind sowie die entsprechende Systemsoftware und Hardware die zur Realisierung gebraucht wird.
- Aspekte:
 - › Active Structure Aspect
 - › Behavior Aspect
 - › Passive Structure Aspect
- Erweiterungen für ArchiMate:
 - › Motivation Extension: Erweitert ArchiMate um Konzepte wie Ziele, Prinzipien und Anforderungen. Entspricht der „Why?“-Spalte im Zachman Framework.
 - › Implementation and Migration Extension: Erweitert ArchiMate um Konzepte zur Unterstützung der späten Phasen des ADM Zyklus, welche die Implementierung und Migration der Architektur betreffen.

ArchiMate: Core Concepts



(Archimate 2.1 Specification)

Drei Hauptkategorien von Architekturelementen:

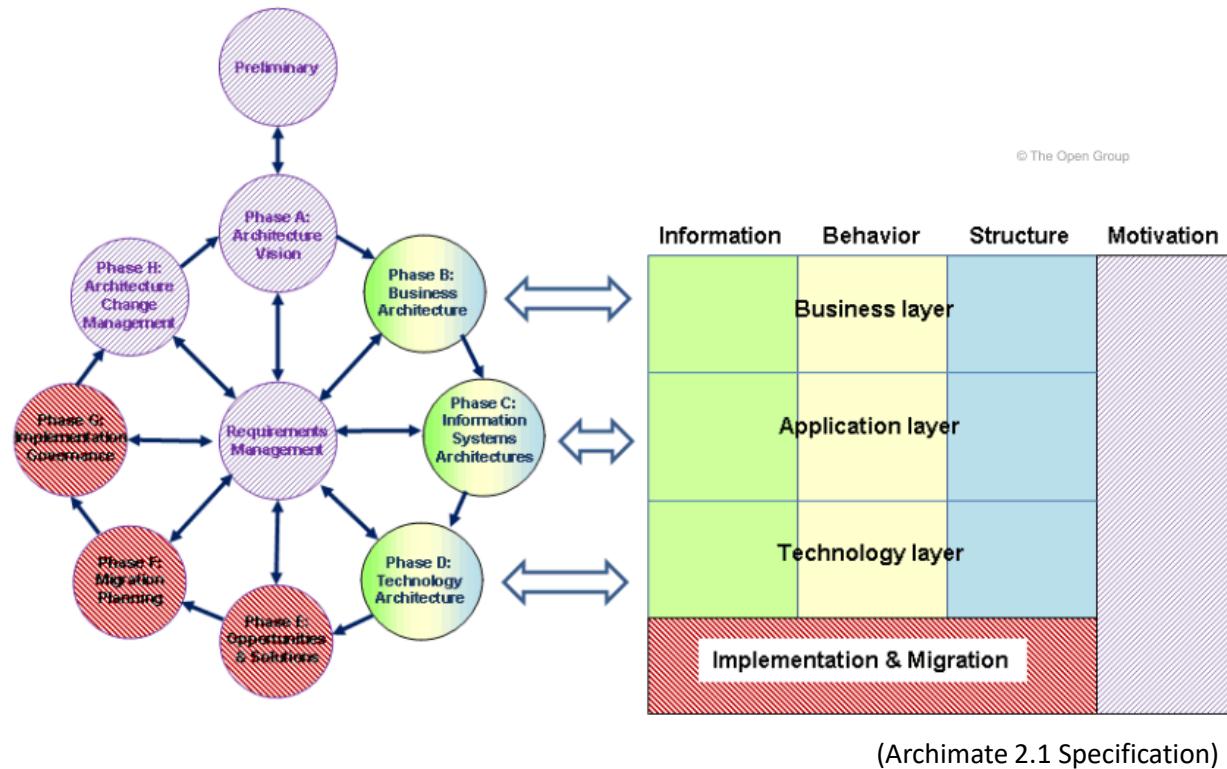
- Die *Active Structure Elements* sind Akteure, Anwendungen und Geräte, also die „Subjekte“ einer Aktivität (blau)
- Behavioral Structure Elements* bilden Verhalten ab (Prozesse, Funktionen, Services). Diese Elemente werden *Active Structure Elements* zugewiesen, um zu zeigen wer oder was das entsprechenden Verhalten ausführt (gelb)
- Passive Structure Elements* sind Objekte auf denen Verhalten ausgeführt wird. In der Facharchitektur sind das typischerweise Informationsobjekte in der Anwendungsarchitektur Datenobjekte.

Aktive und passive Elemente werden weiterhin in den Sichten *extern* und *intern* unterschieden.

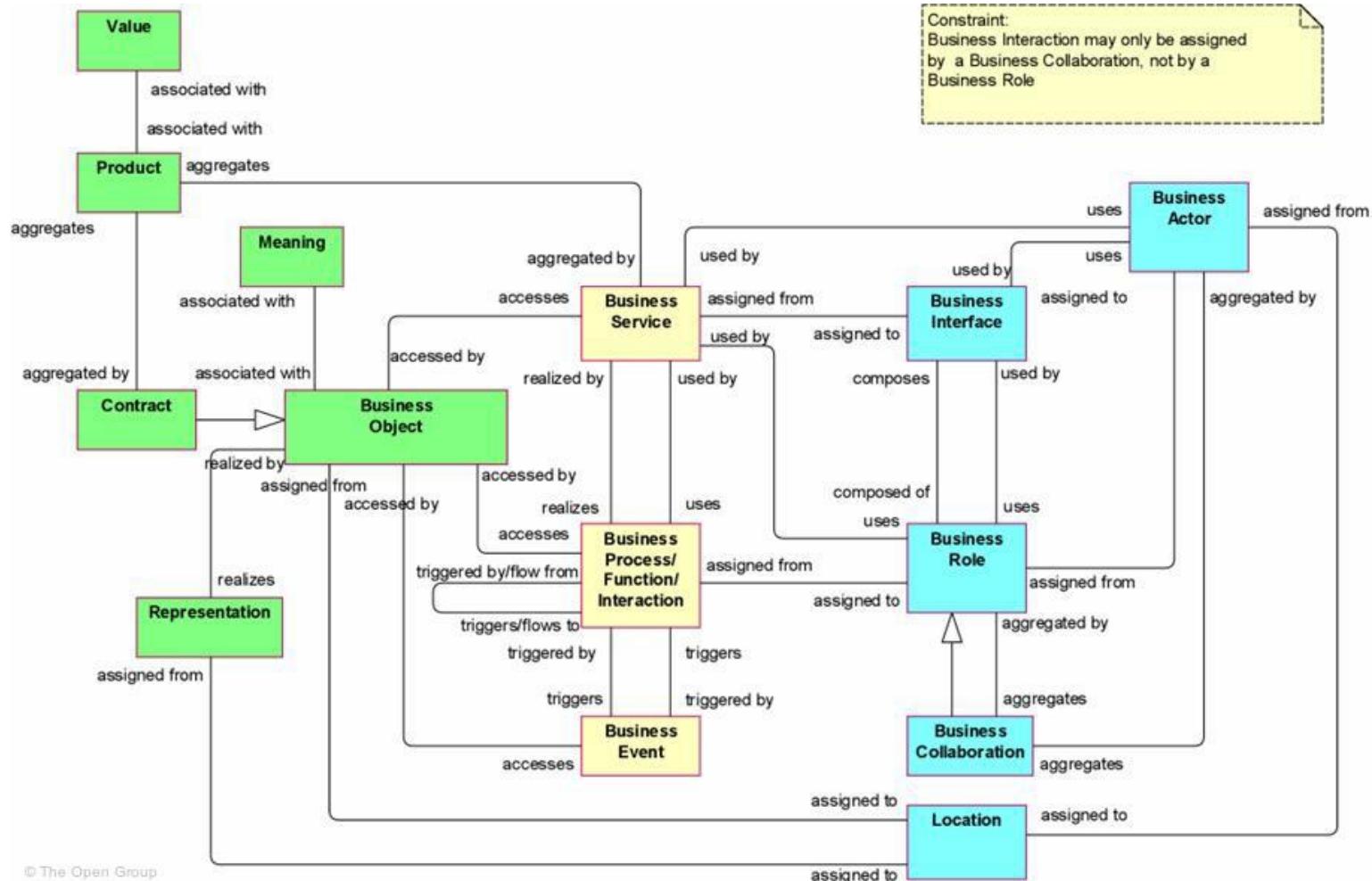
Für den Verhaltensaspekten repräsentiert dies die Prinzipien der Service-Orientierung. Aktive-Strukturelemente haben als externe Komponente ein Interface.

Archimate and Togaf

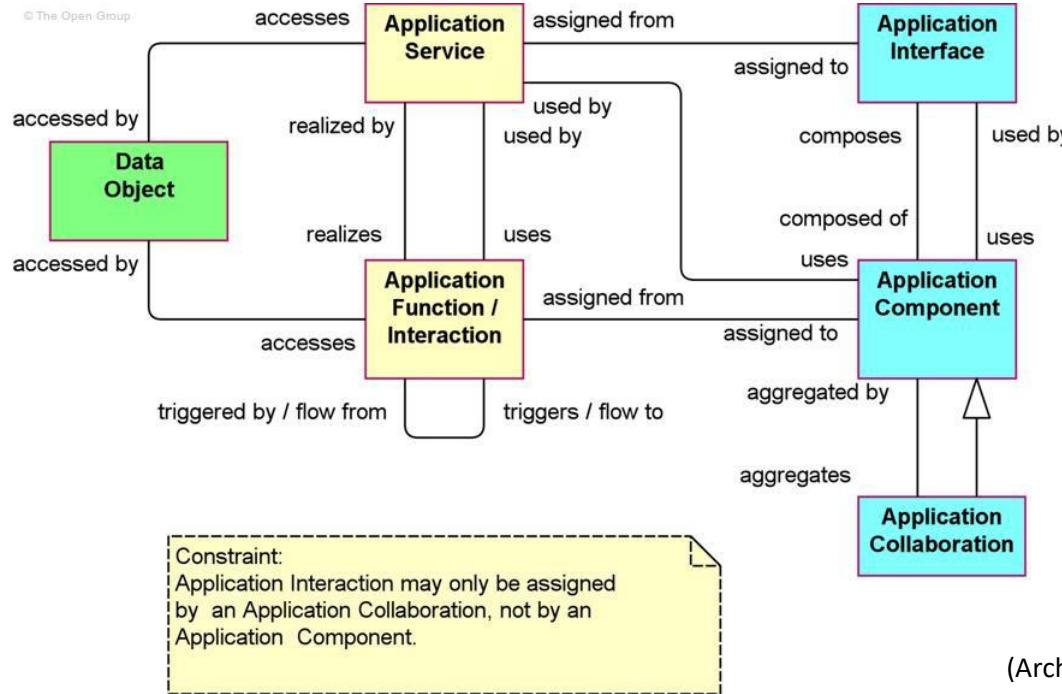
- ArchiMate ergänzt TOGAF in der Hinsicht, dass es eine Hersteller-unabhängige Menge von Konzepten und ihrer graphischen Repräsentation definiert, welche die Erstellung eines konsistenten und integrierten Modells ermöglicht
- TOGAF Viewpoints können nicht immer 1:1 auch ArchiMate Viewpoints abgebildet werden, die notwendigen Konzepte dafür werden aber von ArchiMate bereit gestellt



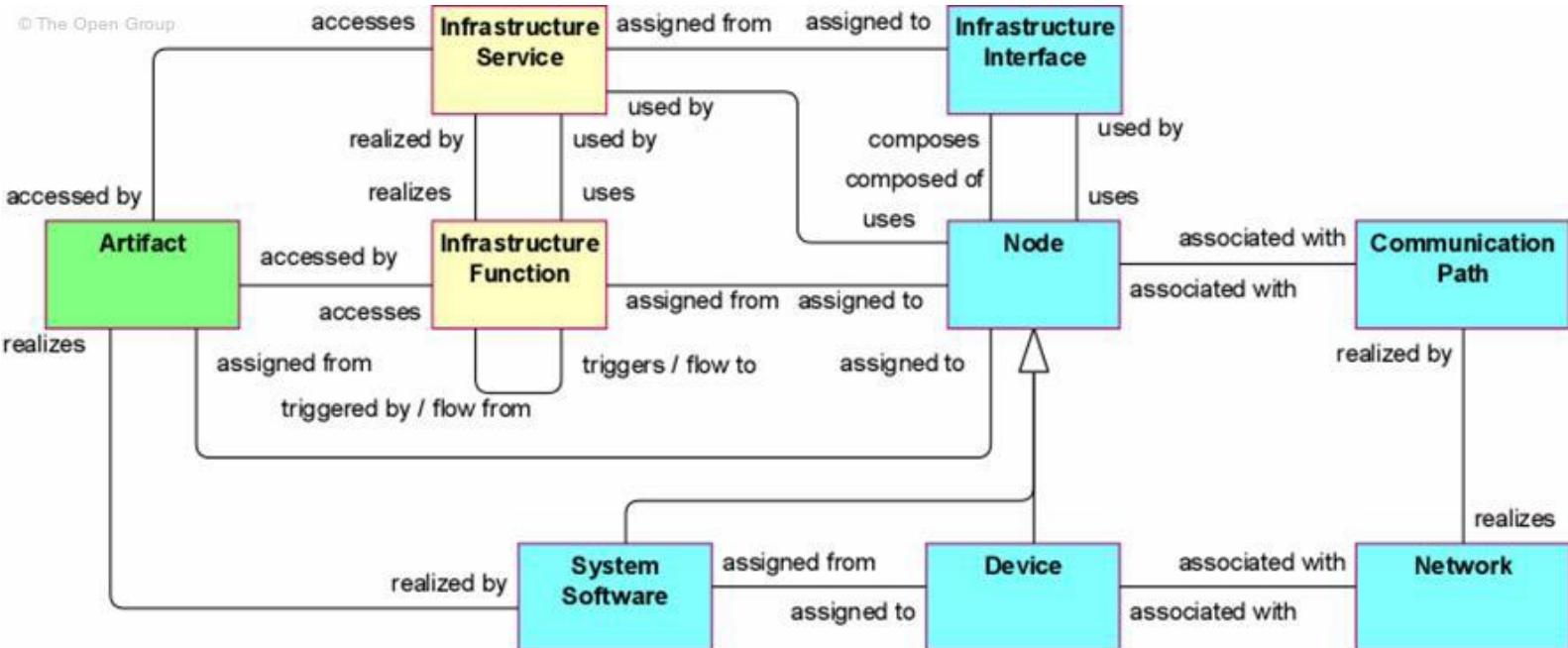
Archimate Business Layer



Archimate Application Layer

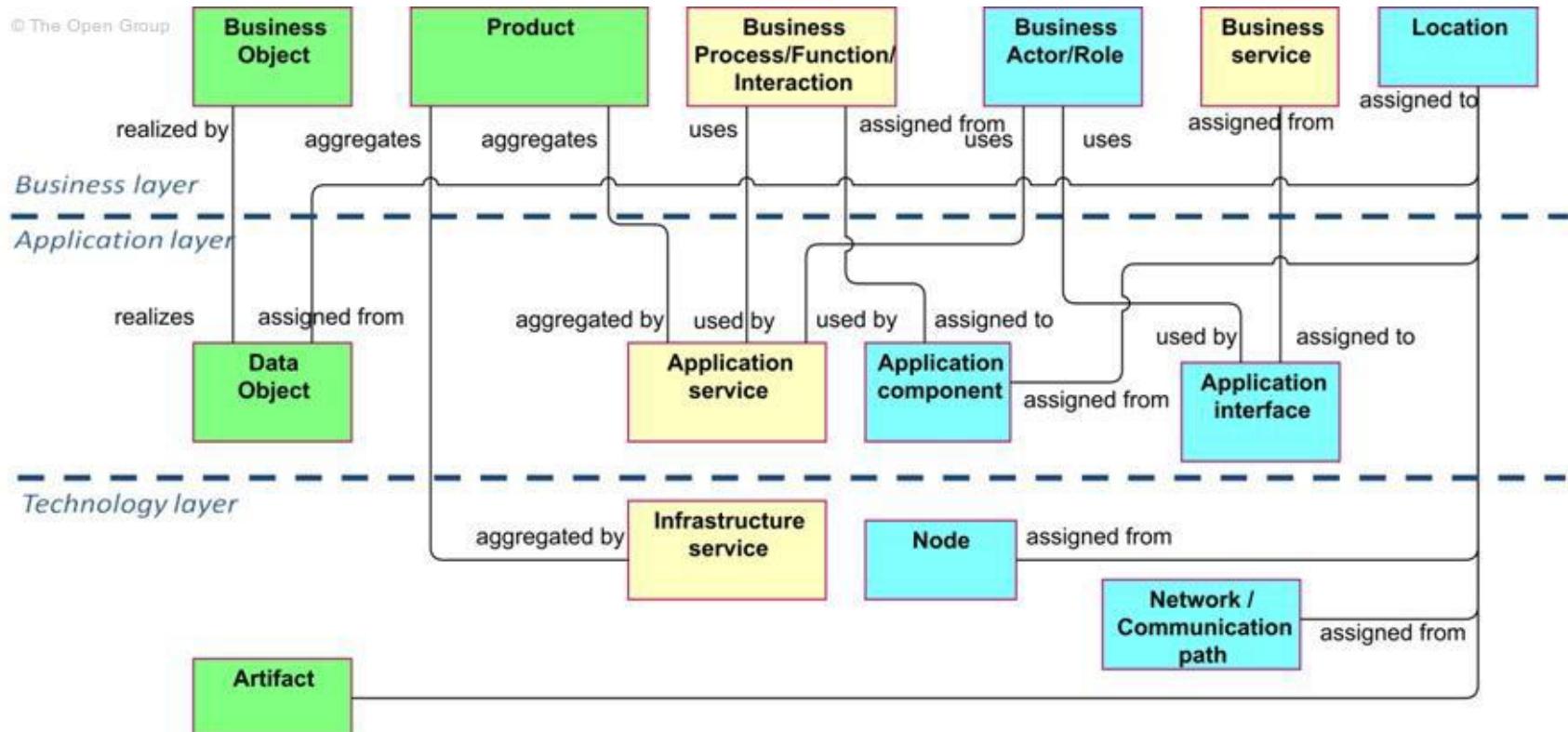


Archimate Technology Layer



(Archimate 2.1 Specification)

Abhängigkeiten des Business Layer zu anderen Schichten

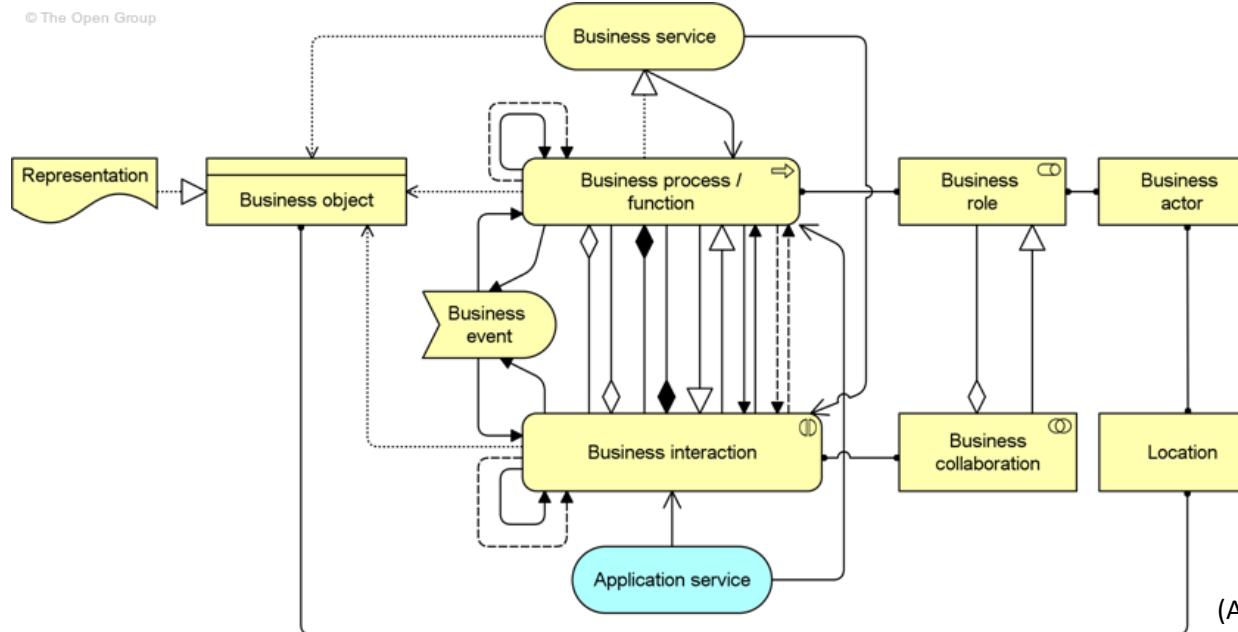


(Archimate 2.1 Specification)

- Viewpoints definieren Abstraktionen auf dem Unternehmensarchitekturmodell
- Jeder Viewpoint zielt auf bestimmte Stakeholder ab und adressiert eine bestimmte Menge von Interessen
- Viewpoints können benutzt werden um bestimmte Aspekte isoliert zu betrachten oder die Beziehung zwischen zwei oder mehreren Aspekten zu verdeutlichen
- Viewpoint in Archimate ist definiert durch eine Teilmenge von Archimate Konzepten (und ihren Beziehungen) sowie die Repräsentation dieser Teilarchitektur in Diagrammen
- Viewpoint im Standard wurden basierend auf Erfahrungen aus der Praxis definiert

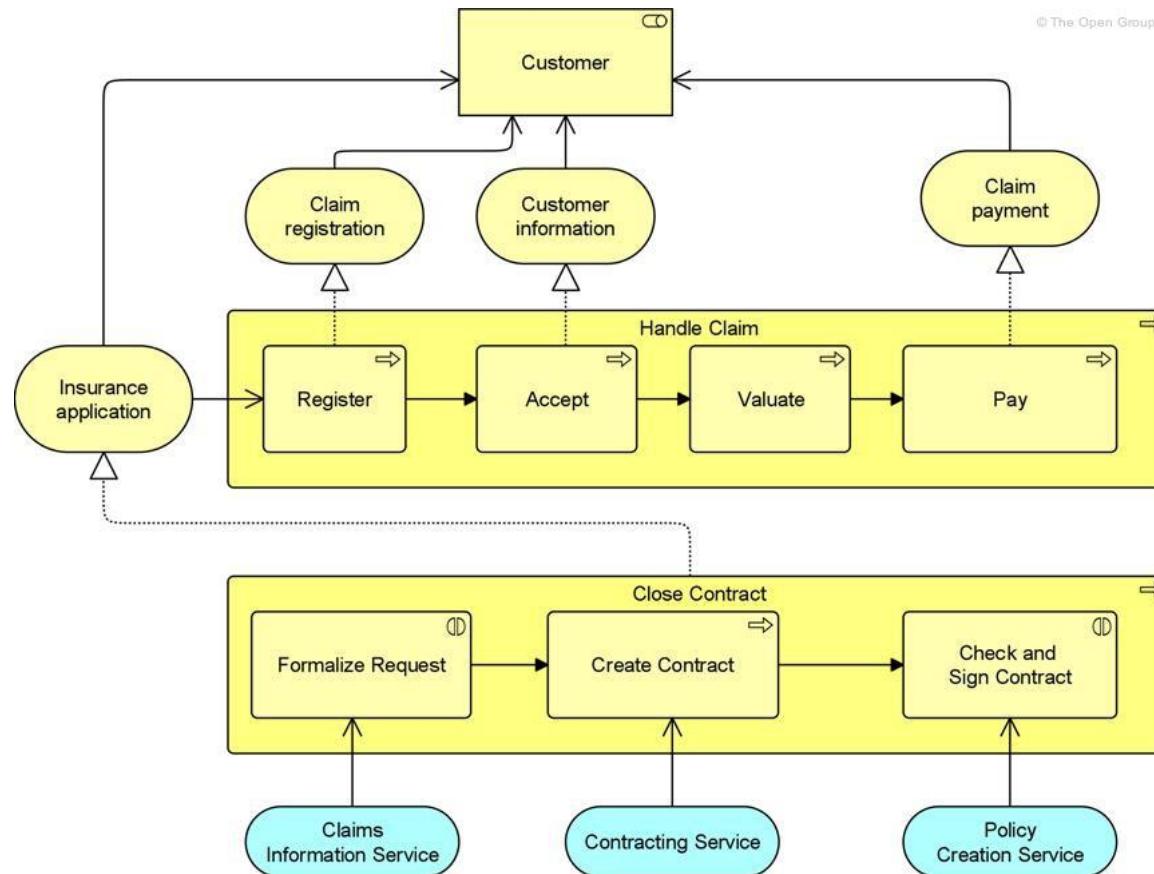
Business Process Co-operation Viewpoint

- The Business Process Co-operation viewpoint is used to show the relationships of one or more business processes with each other and/or with their environment. It can both be used to create a high-level design of business processes within their context and to provide an operational manager responsible for one or more such processes with insight into their dependencies. Important aspects of business process co-operation are:
 - > Causal relationships between the main business processes of the enterprise
 - > Mapping of business processes onto business functions
 - > Realization of services by business processes
 - > Use of shared data



(Archimate 2.1 Specification)

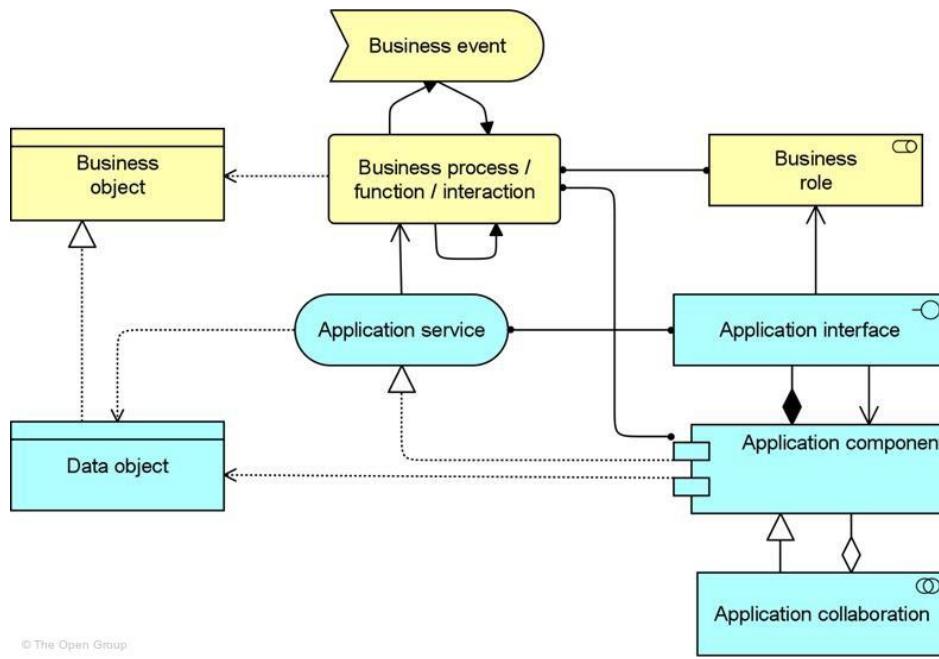
Business Process Co-operation Viewpoint: Example



(Archimate 2.1 Specification)

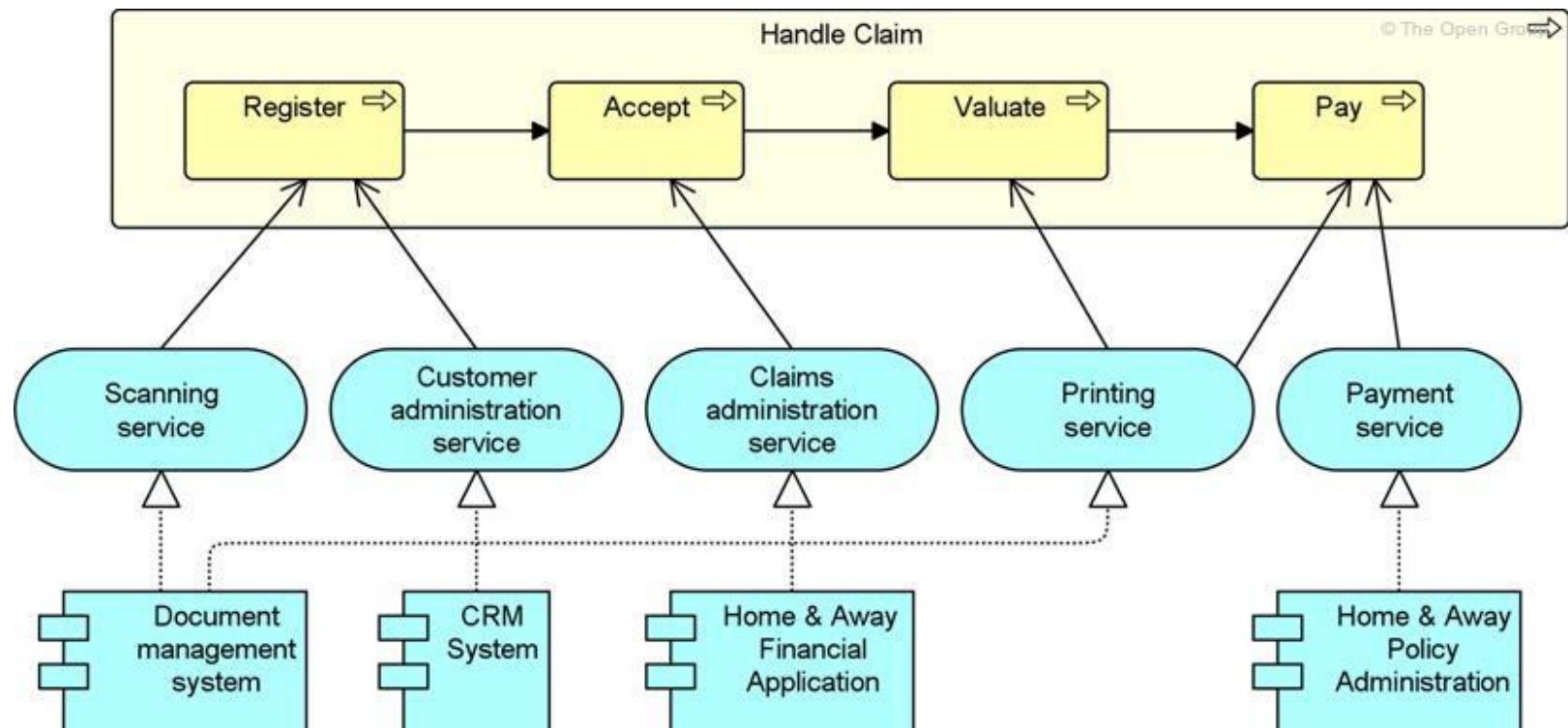
Application Usage Viewpoint

The Application Usage viewpoint describes how applications are used to support one or more business processes, and how they are used by other applications. It can be used in designing an application by identifying the services needed by business processes and other applications, or in designing business processes by describing the services that are available. Furthermore, since it identifies the dependencies of business processes upon applications, it may be useful to operational managers responsible for these processes.



(Archimate 2.1 Specification)

Application Usage Viewpoint: Example



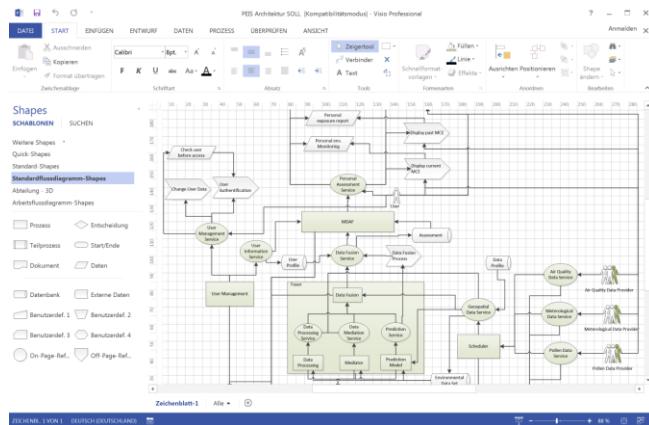
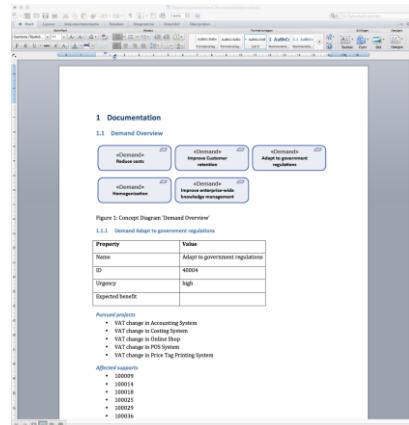
(Archimate 2.1 Specification)

- Motivation und Problemstellung
- Elemente einer EA
- Frameworks
 - › Beispiel: TOGAF
- Modellierung von EA
 - › Beispiel: Archimate
- **Tooling**

EAM Initiativen starten oft mit einfachen Lösungen. Mit zunehmenden Umfang kommen solche Lösungen an ihre Grenzen hinsichtlich Konsistenz, gemeinsames Erstellen von Dokumenten, Analysen und Berichterstattung.

Beispiel für die MS Office Tool-Chain:

- Beschreibung der Architektur in MS Word
 - Verwaltung der EA Elemente in MS Excel
 - Erstellen der Diagramme in MS Visio oder MS Power Point



- Unterstützung bei der Standardisierung der Semantik und Notation der Architekturmodelle (wenn im ganzen Unternehmen eingeführt und genutzt)
- Ermöglichen den Entwurf von konsistenten Modellen (Automatisches Überprüfen von Vorgaben und Architekturprinzipien)
- Unterstützen den Architektur bei der Anwendung von Patterns, der Wiederverwendung von bestehenden Lösungsbausteinen und Komponenten
- Vergleich von Ist- und Soll-Architektur, sowie verschiedenen Alternativ-Architekturen
- Ermöglichen automatisierte Analysen auf den Modellen zur Entscheidungsunterstützung
 - › Auswirkungsanalysen
 - › Quantitative Analysen
- Verwaltung der zunehmenden EA Artefakte bei zunehmenden Reifegrad des EAM
- Mangelndes Verständnis über die vorhandene IT Infrastruktur erschwert Rationalisierungen und Kostenoptimierungen. Application Portfolio Management im Rahmen eines EAM soll hier helfen

(Gartner, Lankhorst)

- Werkezuge sind für die Umsetzung eines EAM sehr wichtig
- Tools sollen Informationen aus verschiedenen Quellen sammeln können, sinnvoll verknüpfen und für die entsprechenden Zielgruppen aufbereiten
- EA Tools erfassen, speichern, strukturieren und analysieren die Daten zu einer EA
- Kategorien die ein Tool unterstützen sollte:
 - › Modellierung und Design
 - › Berichterstattung und Publikation
 - › Speichern und Abfragen der Daten

- Toolauswahl mit Hilfe eines gewichteten Kriterienkatalogs, ergänzt durch eine Marktübersicht
- Fragestellungen
 - › Welche Funktionalität wird gebraucht?
 - › Welche Standardpakete gibt es und wie unterscheiden sie sich?
 - › Was sind die Lizenz-, Wartungs- und Betriebskosten?
 - › Wie viele arbeiten mit dem Tool und welche Fähigkeiten werden gebraucht?
 - › Wie passt das Tool in die bestehende Softwarelandschaft?
- Auswahlkriterien
 - › Funktionalität des Tools
 - › Schnittstellen zu anderen Tools
 - › Marktpräsenz, Strategie und Viabilität des Herstellers
 - › Erfahrungen anderer Kunden

Tool Capabilities



„Do-it-yourself“ (xls, ...)

High flexibility ✓

Reinvent the wheel ✗

Not user friendly ✗



Big Tool Suites

Rich functionality ✓

Deep analysis & modeling ✓

Made for Corp. HQ & skilled experts ✗

High Costs (50 – 300 k€) ✗

Long Projects (6 – 12 month) ✗

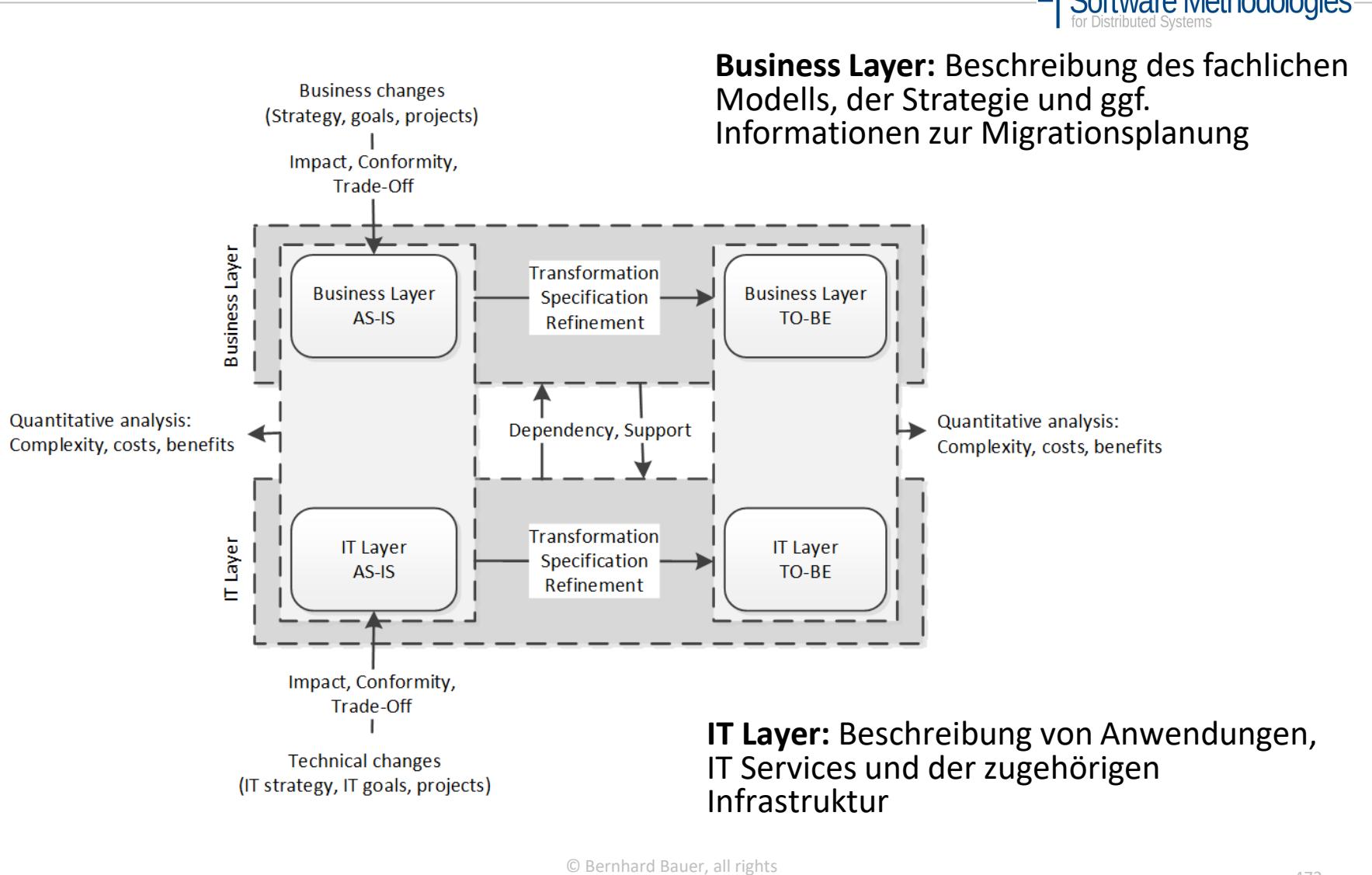
Enterprise Architecture Maturity

5

Funktionale Anforderungen an ein Tool

- Erfassen und Speichern der entsprechenden EA Daten
- Lese- und Schreibzugriff auf die Daten
- Sicherstellen von Konsistenz und Korrektheit der Modelle
 - › Unterstützung bei der Integration von Änderungen, z.B. durch Auswirkungsanalysen
- Strukturierung der Daten durch Unterstützung von verschiedenen Architekturebenen und den Beziehungen dazwischen
- Wiederverwendung von Architekturmustern und Komponenten unterstützen
- Unterstützung der Architekturplanung
 - › Unterstützung der Definition einer Soll Architektur durch Transformation, Spezifizierung und Verfeinerungsmöglichkeiten
 - › Unterscheidung zwischen Architekturen zu verschiedenen Zeitpunkten
 - › Abbildung der Zusammenhänge zwischen den verschiedenen Architekturzuständen um Konsistenz und Entscheidungsunterstützung zu fördern
- Prozessunterstützung der EAM Prozesse
- Entscheidungsunterstützung
 - › Erstellen von Berichten und Visualisierungen
 - › Konformitäts- und Trade-Off Analysen zur Unterstützung der Definition von Strategien und Zielen sowie des Projektmanagements
 - › Quantitative Analysen für Ist- und Soll-Architektur

Funktionale Anforderungen an ein Tool



- Administrative Fähigkeiten (z.B. Sicherheit, Nutzermanagement)
- Unterstützung für existierende Frameworks und Standards mit der Möglichkeit diese an spezifische Anforderungen anzupassen
- Nutzbarkeit: Intuitive, flexible und schnell erlernbare Nutzerschnittstellen
- Erweiterbarkeit und Anpassbarkeit des Tools (hinsichtlich Funktionalität und Metamodell)

Magic Quadrant for Enterprise Architectures (Gartner 2014)



- Ability to Execute: „The ability to compete effectively, impact revenue to a positive degree, and deliver solutions to clients that create vendor-client win-win relationships.“
- Completeness of Vision: „Ability to articulate logical statements about current and future market direction, innovation, customer needs and competitive forces; how consistently they map their strategies and plans to their stated vision; and the practicality of that vision.“
- Hier geht es weniger um Funktionalität des Produktes als um die Platzierung des Herstellers im Markt

- EAM-Tools have different approaches
 - › *Flexibility vs. Guidance* regarding process, method, and information model for supporting EA management
 - › *Preconfigured vs. Customization* regarding the functionality provided by the tool out of the box – two approaches exist: EA management *solution* vs. EA management *platform*
 - › *Integration vs. Single-Point-of-Truth* regarding the information base of the tool, which in the one approach is collected from a variety of sources, while in the other approach being under data sovereignty of the tool itself
 - › (Framework-driven)
- These approaches are not disjoint!
 - › Combinations of different approaches are possible
 - › Tools follow partially several approaches with variable degree of coverage
- Attention: Mostly no exact matching between tools and approaches is possible!

(HPI-Vorlesung/EAM Tool Survey Matthes)

- Meta model driven approach:
 - › Customers can adapt the information model to their needs
 - › Reports and visualizations have to be adapted to the changed information model
 - › Mightiness of the tools at changing the information model is heavily variable; From small proprietary solutions up to MOF compliant solutions
- Methodology driven approach:
 - › Predefined and documented methodology (methodology manual)
 - » How to use which models?
 - » Which elements belong to which models?
 - › Only small or no changes to the information model, methodology remains
 - › Reports and visualizations are coupled to the information model
- Process driven approach:
 - › Methodology is expanded with a management processs
 - » The “what” and “how” of the methodology ist extended by the “when”
 - › Process connects different modules in a process model

Approaches of the tools: Preconfigured vs. Customization

- EA Management Solutions (Preconfigured)
 - › Preconfigured functionality for typical EA Management tasks are provided by delivery
 - › “Misuse” is aggravated
 - › Rampant learning curve (Training, Consulting necessary)
- EA Management Platforms (Customization)
 - › At delivery only basic functionality is provided
 - › Implementation of a company specific EA Management approach is possible
 - › At the beginning of the implementation of the tool a customer specific adaption is necessary

- **Single-point-of-truth**
 - › Data of EA are stored centrally in the EAM Tool
 - › Replication is done „manually“ via imports -> conflict resolution strategy is necessary
 - › High data consistency, clear data sovereignty
- **Integration**
 - › EAM-Tool acts as „Data Warehouse“
 - › Main target of these EAM-Tools is the maintenance of the relation information
 - › Reuse of different data sources
 - › Linking, integration and aggregation of different sources in one model
 - › Demands sophisticated transformation possibilities
 - › Is also called „Metadata Integration“
 - » Data consistency and data sovereignty may be problematic

- Vorlesung HPI
- Strategic Enterprise Architecture Management (Ahlemann)
- Enterprise Architecture at Work (Lankhorst et al.)
- IT Unternehmensarchitektur (Keller)

Danke für Ihre Aufmerksamkeit!

