Fachhochschule
Dortmund
University of Applied Sciences and Arts

we
focus
on
students

Softwaretechnik 2

Architekturstile

Fachhochschule
Dortmund
University of Applied Sciences and Arts

Architekturstile

Architekturkonzeption

Architekturstile

Prinzipien

Architektursichten

Architektur und Agilität

Fachhochschule
Dortmund
University of Applied Sciences and Arts

Architekturkonzeption

Rückblick

Architekturtreiber

Architektur

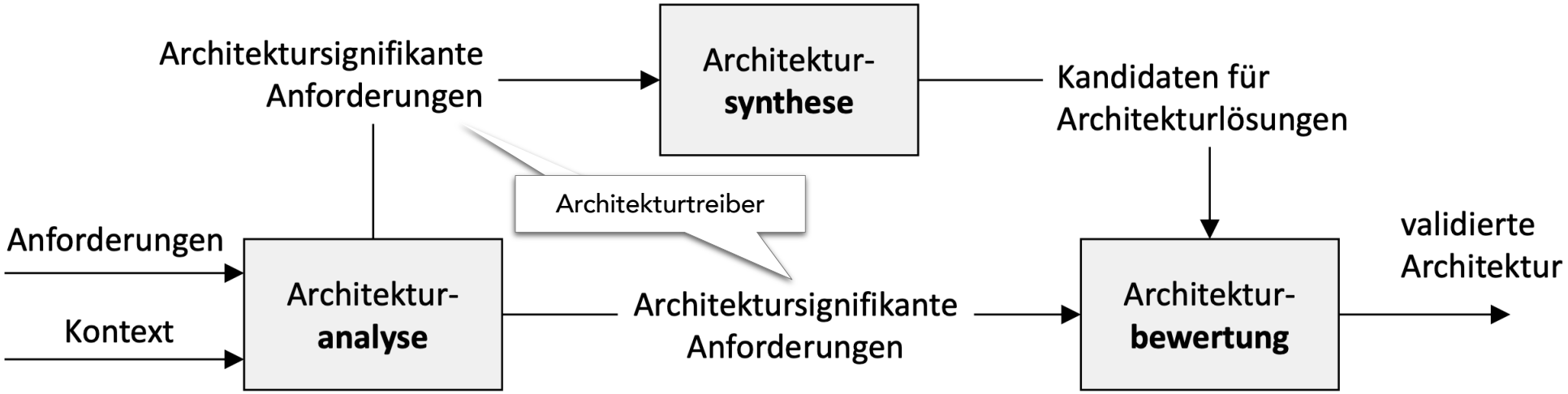
Anforderungen

A (system) **architecture** constitutes
„the fundamental concepts of properties of a system in its environment embodied in its elements, relationships and in the principles of its design and evolution."
(ISO 42010)

Eine (System)**Architektur** stellt
„die grundlegenden Konzepte der Eigenschaften eines Systems in seiner Umgebung umgesetzt durch seine Elemente, Beziehungen und in den Prinzipien seines Designs und seiner Entwicklung" dar.
(ISO 42010)

[ISO/IEC/IEEE. 2011. Systems and software engineering — Architecture description. Standard ISO/IEC/IEEE 42010:2011(E). International Organization for Standardization/International Electrotechnical Commission/Institute of Electrical and Electronics Engineers.]

Architekturkonzeption



[In Anlehnung an: Christine Hofmeister, Philippe Kruchten, Robert L. Nord, Henk Obbink, Alexander Ran, Pierre America: A general model of software architecture design derived from five industrial approaches. The Journal of Systems and Software 80 (2007) 106–126. Aus: T. Weyer: Softwarearchitektur, 19/20]

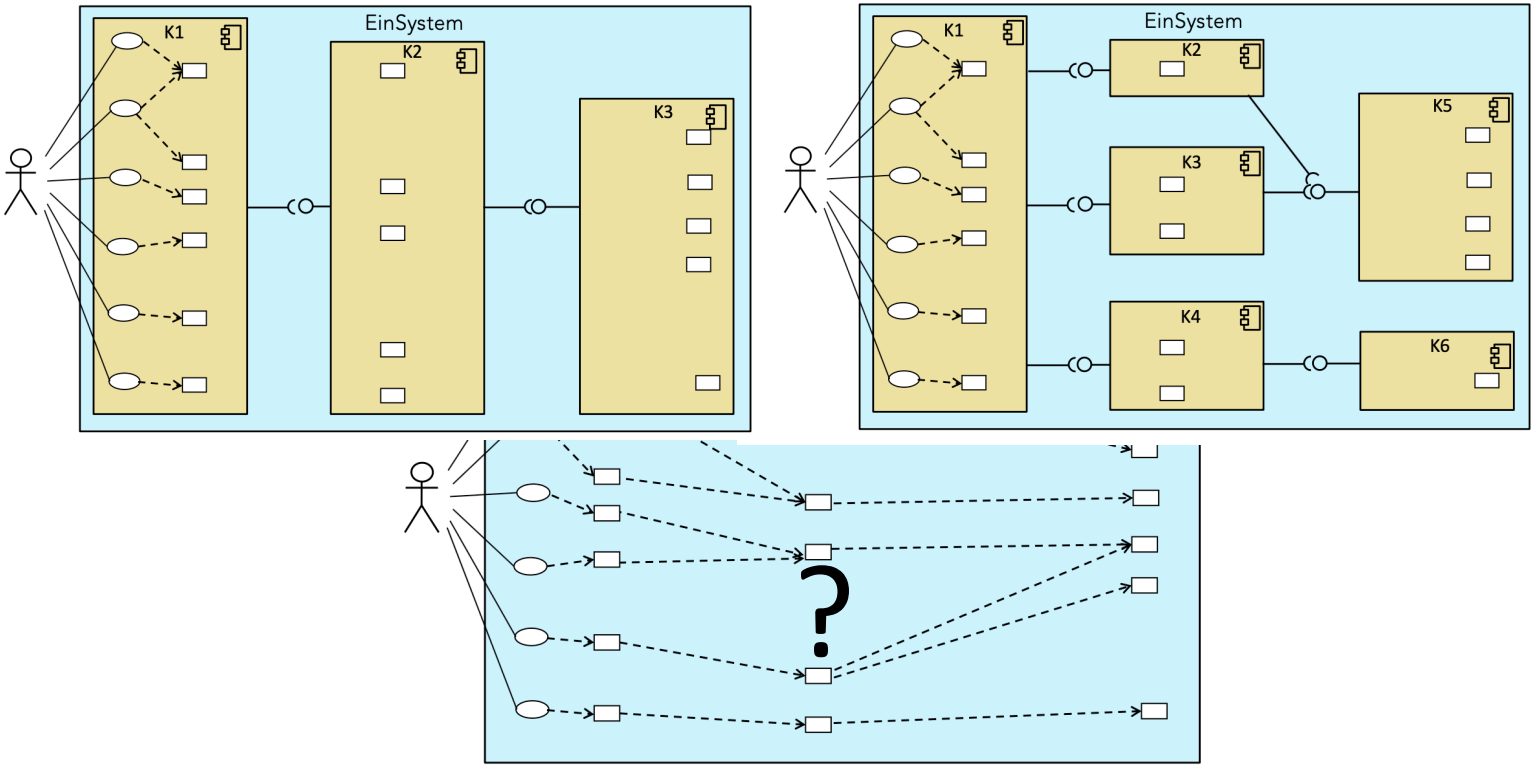
Architekturtreibende Anforderungen

- Architekturtreiber sind eine **Teilmenge der Anforderungen**, die wesentliche Aspekte der Softwarearchitektur bestimmen
  - Funktionale Anforderungen
  - Qualitätsanforderungen
  - Randbedingungen

  } nicht-funktionale Anforderungen
- häufig sind nicht-funktionale Anforderungen darüber hinaus auch ein „Sammelbecken" für **unklare** bzw. **unterspezifizierte Anforderungen**
- Erinnerung: Anforderungen müssen am späteren System objektiv überprüfbar sein! → SWT1

Architekturtreiber – Funktionalen Anforderungen

Wie zerlegt man das System geeignet in Teile?



Anwendungsfall

Angebotene Schnittstelle

Genutzte Schnittstelle

Subsystem

Architekturtreiber – Funktionalen Anforderungen

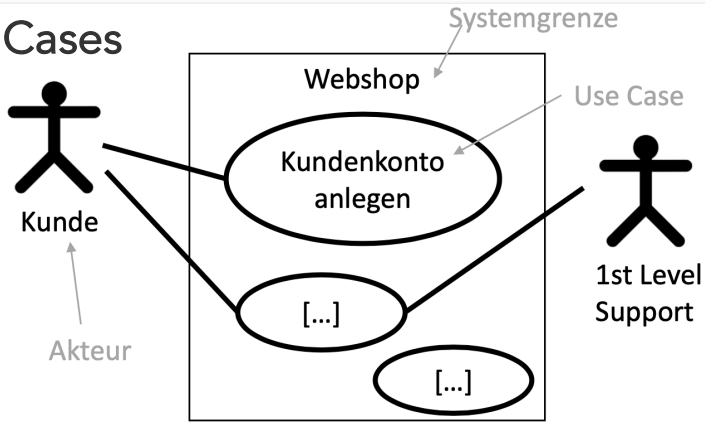
- Funktionale Treiber umfassen typischerweise die **wesentlichen Funktionalitäten**, die das System **an seiner Schnittstelle** anderen Akteuren (Personen, Systeme) zur Verfügung stellen muss, um seinen Zweck zu erfüllen

- **User Stories** (Schema)

  Als **[Wer?]** möchte **ich [Was tun können?]**, um **[Welchen Mehrwert zu erreichen?]**.

  Beispiel:
  Als **Kunde** möchte ich ein **Kundenkonto anlegen**, um **im Webshop Bestellungen tätigen zu können**.

- **Use Cases**



[T. Weyer: Softwarearchitektur 19/20]

Architekturtreiber – Nicht-funktionalen Anforderungen

NFR-12: Das System soll sicher sein.

Ist eine unterspezifizierte Anforderung, die unmittelbar zu Vagheit führt. Was tatsächlich mit "sicher" gemeint ist, kann sich von Stakeholder zu Stakeholder stark unterscheiden.

Unterspezifizierte Anforderungen sollten soweit konkretisiert (verfeinert) werden, bis diese eindeutig definiert und objektiv am späteren System überprüfbar sind, z.B.:

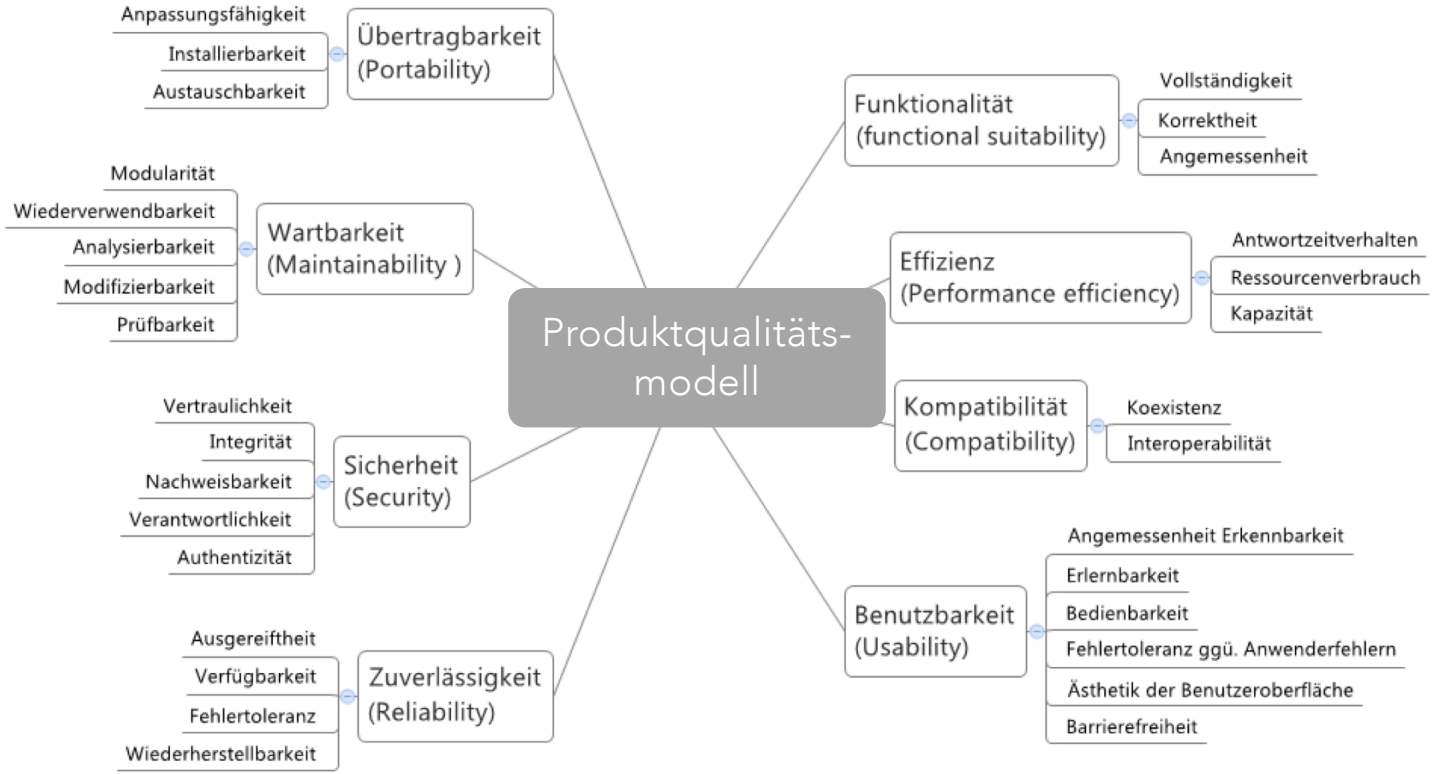
R-12.1: Jeder Benutzer muss sich vor der Verwendung des Systems mit seinem Benutzernamen und seinem Kennwort beim System anmelden.

R-12.2: Das System muss den Benutzer alle vier Wochen daran erinnern, das Passwort zu ändern.

R-12.3: Wenn der Benutzer sein Passwort ändert, muss das System überprüfen, ob das neue Passwort mindestens acht Zeichen lang ist und alphanumerische Zeichen enthält.

R-12.4: Das im System gespeicherte Benutzerpasswort ist gegen Passwortdiebstahl zu schützen.

Nicht-funktionalen Anforderungen – Software-Qualitätseigenschaften nach ISO 25010



[Quelle: ISO/IEC 25010:2011]

Nicht-funktionalen Anforderungen – Software-Qualitätseigenschaften nach ISO 25010

[Quelle: ISO/IEC 25010:2011]

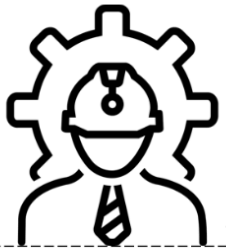
Architekturtreiber – Randbedingungen

- **Randbedingungen** (Constraints) definieren Vorgaben bzw. Einschränkungen
  - **technischer**, **organisatorischer** oder **rechtlicher** Natur
  - Ursprung sind **Stakeholder** (z.B. Kunde, **Gesetzgeber**, Entwicklungsorganisation)
  - schränken die Anzahl möglicher (gültiger) Softwarearchitekturen ein

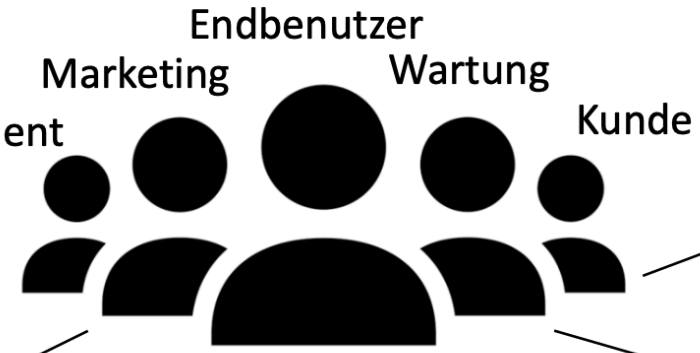
- **Beispiele**

C-08:	Das System soll mit Hilfe einer mehrschichtigen serviceorientierten Architektur realisiert werden.	technische
C-15:	Der Aufwand für die Systementwicklung darf nicht mehr als 480 Personenmonate betragen.	organisatorisch
C-42:	Personenbezogene Daten müssen gemäß der DSGVO verarbeitet werden.	rechtlich

Architekturtreibende Anforderungen



Software-Architekt



Geringe Kosten & Mitarbeiterauslastung

Geringe Kosten, fristgerechte Auslieferung, geringe Änderungsrate

Benötigte Features, kurze Markteinführungszeit, geringe Kosten, Wettbewerbsfähigkeit des Produktes

Funktionsumfang, Verhalten, Performance, Sicherheit (Safety, Security), Benutzbarkeit

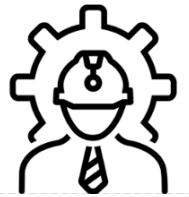
Änderbarkeit & Erweiterbarkeit

[T. Weyer: Softwarearchitektur 19/20]

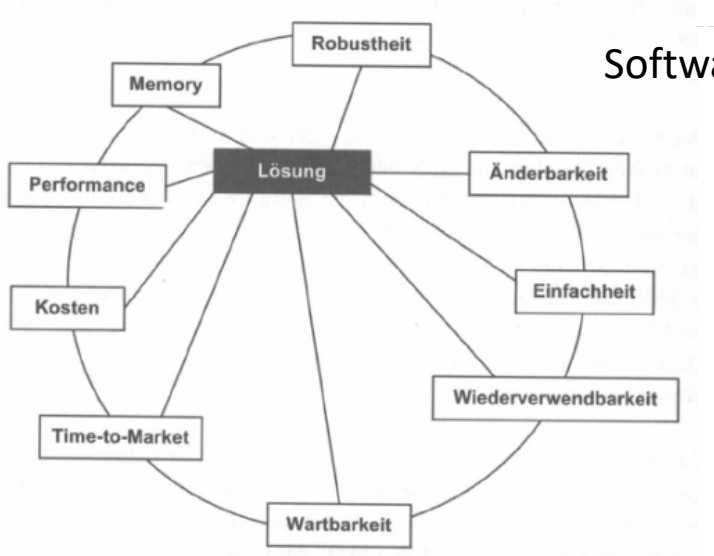
Architekturtreibende Anforderungen

Welche Systemstruktur garantiert die gewünschten (globalen) Eigenschaften?

- Architekturtreibende Anforderungen/ globale Eigenschaften
  - können sich **wechselseitig** ausschließen oder **beeinflussen** - behindern oder stärken
  - sie sind meist **emergent**, d.h. sie lassen sich nicht Einzelteilen zuweisen und aggregieren, sondern entstehen erst aus dem Zusammenwirken der Teile
  - emergente Eigenschaften lassen sich **schlecht** im Voraus **abschätzen**

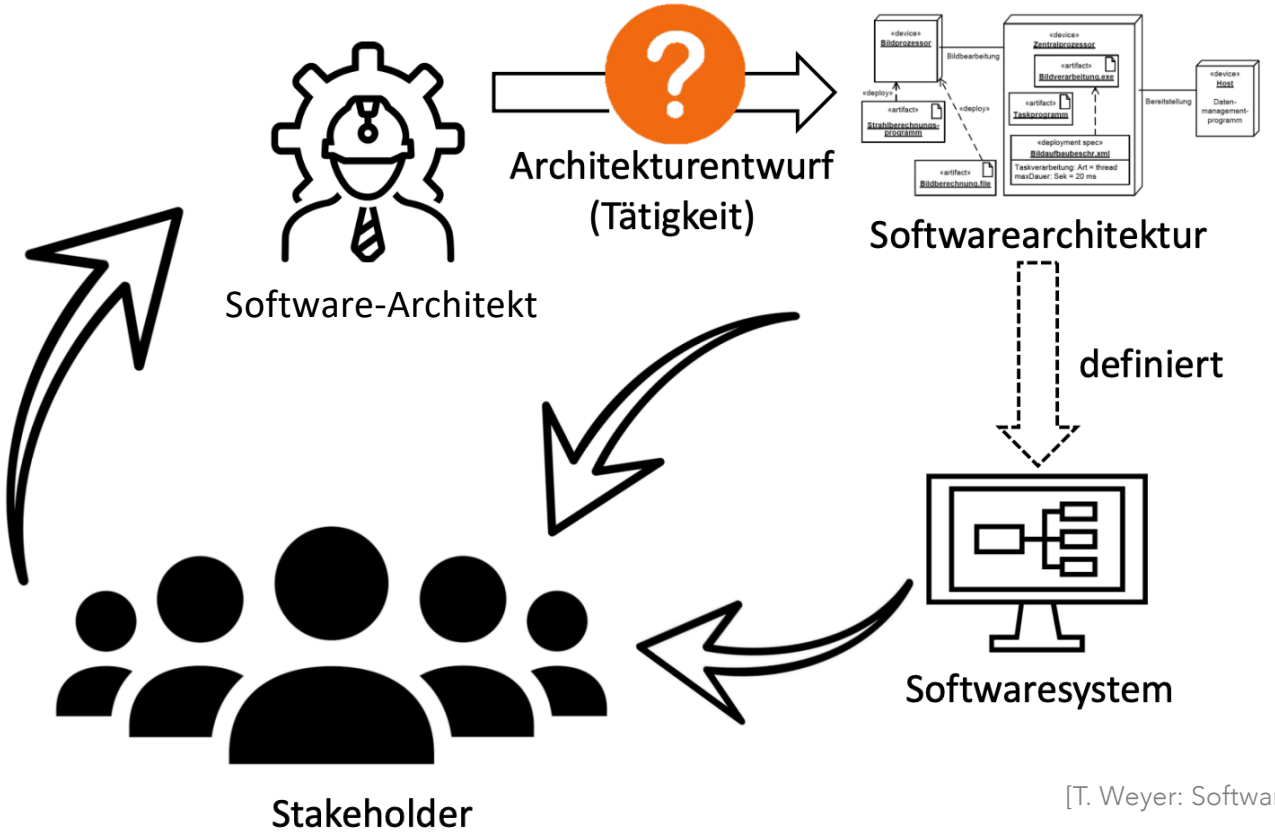


Software-Architekt



Architekturtreibende Anforderungen

Welche Systemstruktur garantiert die gewünschten (globalen) Eigenschaften?

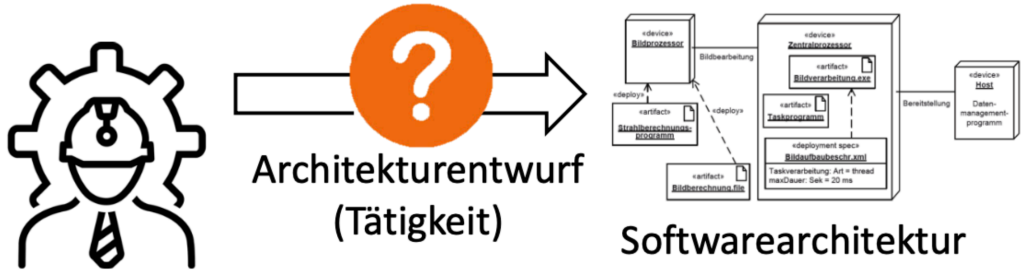


Legende:
beeinflusst

[T. Weyer: Softwarearchitektur 19/20]

Architekturtreibende Anforderungen

Welche Systemstruktur garantiert die gewünschten (globalen) Eigenschaften?

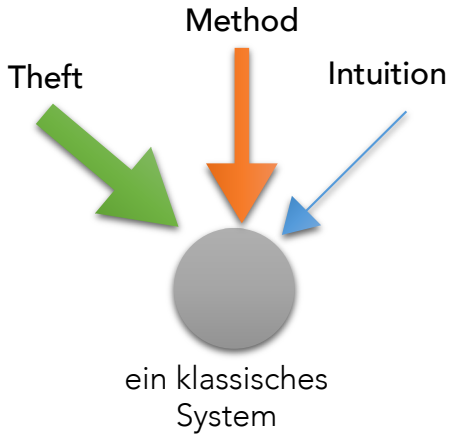


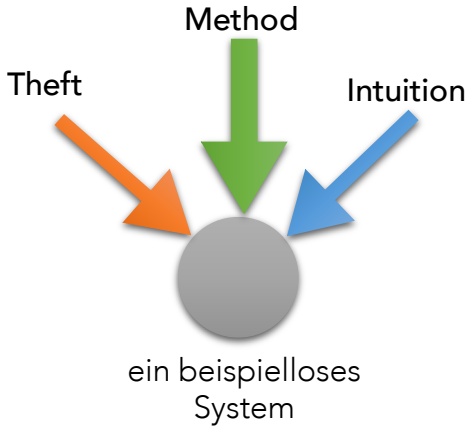
- Architekturkonzepte und globale Eigenschaften
  - globale **Eigenschaften** verschiedener Systeme **ähneln** einander
  - **keine direkte Ableitung** eines Architekturkonzepts aus den globalen Eigenschaften
  - durch **Erfahrungswissen** haben sich gewisse Arten von Gesamtstruktur bewährt

Architekturtreibende Anforderungen

Welche Systemstruktur garantiert die gewünschten (globalen) Eigenschaften?

- Architekturquellen
  - **Theft:** Übernahme von einem früheren oder bestehenden System oder aus technischer Literatur.
  - **Methode:** Versuch der Ableitung der Architektur aus den Anforderungen
  - **Intuiton:** Erfahrungswissen des Architekten





[P. Kruchten: „Mommy, Where Do Software Architectures Come From?, In: 1st International Workshop on Architektures for Software Systems, Seattle, 1995]

Fachhochschule
Dortmund
University of Applied Sciences and Arts

Architekturstile

Architekturstil

Definition nach Shaw

[**An architectural style**] defines a **family of systems** in terms of a **pattern of structural organisation**. more specifically, an architectural style defines a **vocabulary of components** and **connector types**, and a set of **constraints** on how they can be combines.[Shaw 96]

[**Ein Architekturstil**] definiert eine **Systemfamilie** im Sinne eines **Musters der strukturellen Organisation**. Insbesondere definiert ein Architekturstil ein **Vokabular von Komponenten** und **Verbindungstypen** sowie eine Reihe von **Einschränkungen**, wie sie kombiniert werden können. [Shaw96]

Architekturstil versus Architekturmuster

- Ein **Architekturstil** ist keine komplette Architektur, sondern mehr ein **Prinzip oder Grobkonzept**, wie man ein System (oder Teile davon) organisieren kann.
  → Basisarchitekturen
- Architektur-Stile können genutzt werden, um **Architekturen zu kategorisieren**.
- Ein **Architekturmuster** stellt ein **Lösungsschema für ein Problem** im Architekturentwurf dar. Es umfasst Strukturen und Verhalten, die durch Teildefinitionen von Komponenten, Klassen, Methoden, Vererbungs- und Komponentenbeziehungen sowie erklärenden Text beschrieben werden.
- Die **Abgrenzung** zwischen Architekturstil und –muster ist **schwer** oft werden die Begriffe auch synonym verwandt. Muster sind aber in der Regel detaillierter.

Fachhochschule Dortmund
University of Applied Sciences and Arts

Architekturstile

we focus on students

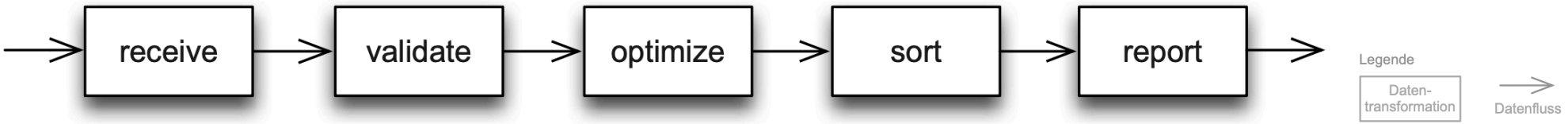
nach Gernot Starke 2017

Stil	Intention	Beispiele
Datenflusssysteme	Besteht aus Folge (Sequenz) von Operationen auf Daten	Batch-Sequentiell Pipes und Filter Prozesssteuerung
Datenzentrische Systeme	gemeinsam verwendeter, zentraler Datenbestand	Repository Blackboard
Hierarchische Systeme	Bestehen aus Bausteinen in unterschiedlichen Ebenen einer Hierarchie	Haupt- und Unterprogramm, Master-Slave, Schichten Ports und Adapter virtuelle Maschine
Verteilte Systeme	Bestehen aus Speicher- und Verarbeitungsbausteinen, die über Kommunikationsnetze interagieren	Client-Server, CQRS, Broker-Architekturen, serviceorientierte Architekturen Peer-to-Peer
Ereignisbasierte Systeme	Bestehen aus voneinander unabhängigen Bausteinen, die über Ereignisse (Events) kommunizieren, sich implizit aufrufen.	Ungepufferte Kommunikation: Broadcast, Publisher-Subscriber Gepufferte Kommunikation: Message-Queue, Message-Service
Interaktionsorientierte Systeme	Systeme mit (grafischer) Bedienoberfläche	Model-View-Controller u.a.
Heterogene Systeme	Verwenden mehrere Architekturstile parallel	beispielsweise REST

Datenflusssysteme

- **Batch-Sequentiell**
- Pipes und Filter
- Prozesssteuerung

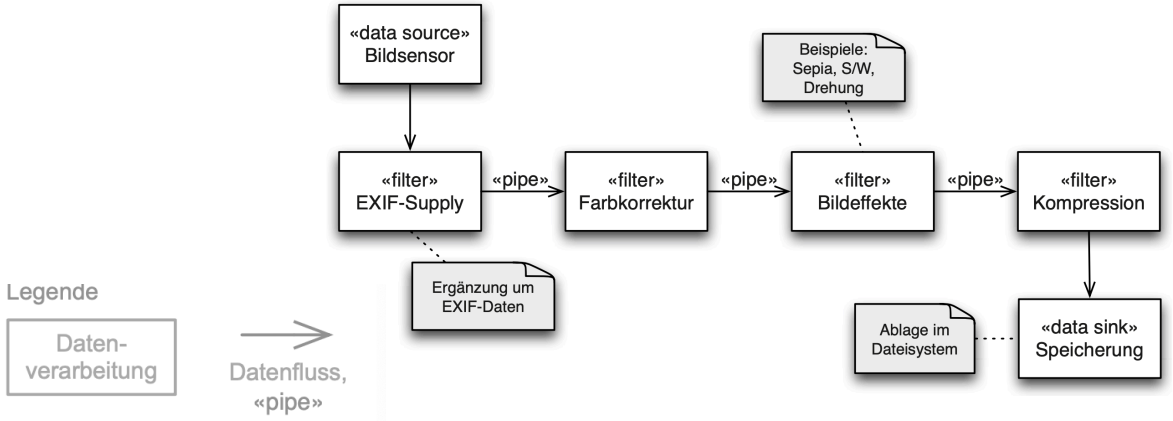
- Batchsysteme stammen aus den Zeiten von COBOL und Mainframes.
- Eingangsdaten werden meist strikt sequentiell verarbeitet
- Daten werden in „Stapeln“ (engl. Batch) von einem Verarbeitungsbaustein zum nächsten weitergereicht
- Verbindungen zwischen den Bausteinen sind oft Dateien
  Schnittstellen: Ein- und Ausgabedateien
- Ablaufsteuerung und Koordination erfolgt durch externe Bausteine (bspw. Skripte)



Datenflusssysteme

- Batch-Sequentiell
- **Pipes und Filter**
- Prozesssteuerung

- Filter
  - sind aktive Verarbeitungs- oder Transformationsbausteine, die Eingabedaten über Eingangs-Pipe(s) erhalten, prozessieren und über Ausgangs-Pipe(s) weitergeben
  - haben in diesem Muster keinerlei direkte Abhängigkeiten zu anderen Filtern
  - Filter können ihre Verarbeitungsergebnisse an eine oder mehrere Ausgangs-Pipes weiterreichen. Sie können damit Verzweigungen innerhalb der Datenflüsse realisieren
- Pipes
  - puffern und transportieren Daten
- Beispiele
  - Pipe-Symbol in Unix
  - Digitalkamera (vereinfacht s.r.)

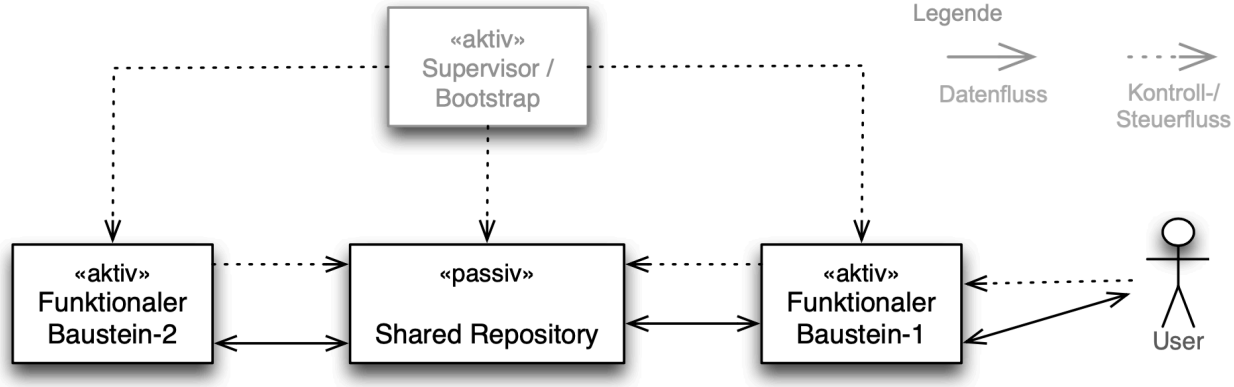


Datenzentrische Systeme

- **Repository**
- Blackboard

- Repository-Systeme gruppieren eine Reihe **aktiver Bausteine** (Agenten) um einen **passiven Datenspeicher**
- die Agenten realisieren **Verarbeitung und Ablaufsteuerung** innerhalb des Systems

- **Beispiele**
  - DB-basierte Systeme
  - Versionsverwaltungssystem
  - Repository in Data-Warehouse- oder Business-Intelligence-Systeme
  - Registry von Microsoft Windows



(vereinfachte) Bausteinstruktur eines Repository-Systems

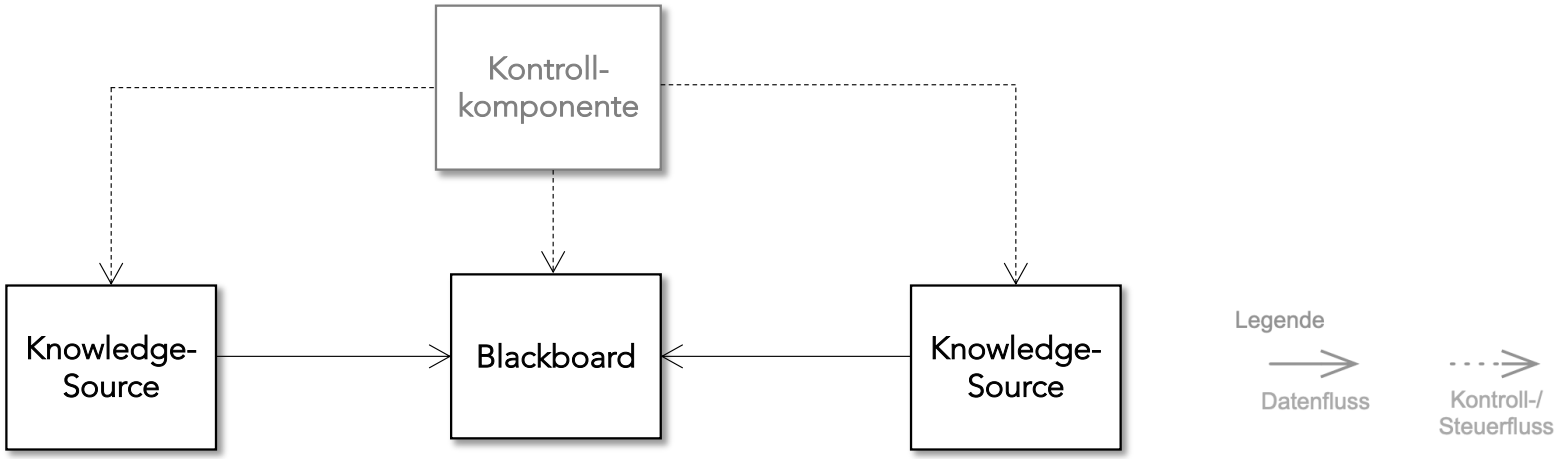
Datenzentrische Systeme

- Repository
- **Blackboard**

- Blackboard-Systeme werden in Bereichen eingesetzt, die **schlecht strukturiert** oder (noch) **wenig erschlossen** sind bzw. in denen geschlossene Lösungen nicht bekannt oder möglich sind.
- Sie wurden beispielsweise im Bereich der künstlichen Intelligenz für **Expertensysteme** eingesetzt, etwa zur Sprach- oder Mustererkennung mit den folgenden Bestandteilen:
  - eine oder mehrere **Knowledge-Sources**, die das Problem aus ihrer spezifischen Sicht untersuchen und Lösungsvorschläge an das Blackboard weitergeben
  - das zentrale „**Blackboard**“, das die Lösungsansätze oder -bestandteile der Knowledge-Sources verwaltet, vergleichbar mit einer **Sammelstelle** für Zwischenergebnisse und endgültige Resultate der Verarbeitung
  - eine **Kontrollkomponente**, die das Blackboard beobachtet und bei Bedarf die Ausführung der Knowledge-Sources steuert

Datenzentrische Systeme

- Repository
- **Blackboard**



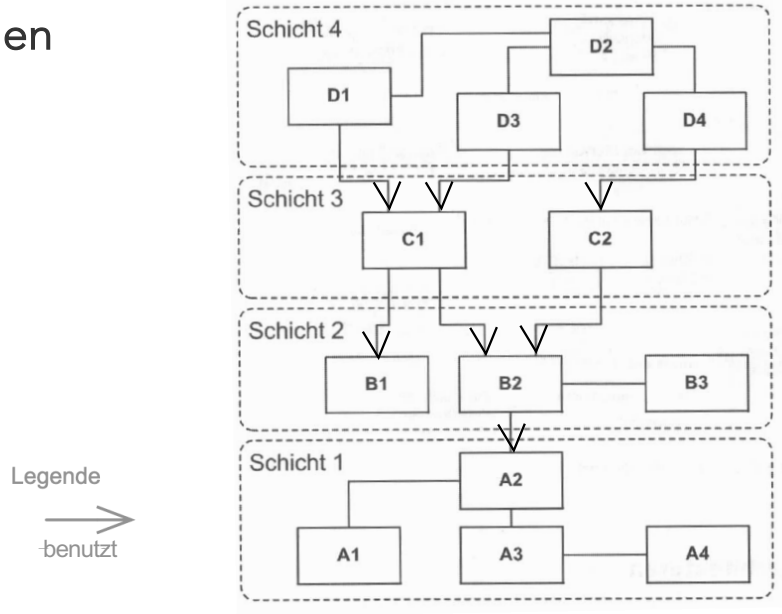
Hierarchische Systeme

- bestehen aus Elementen, die einander **über- bzw. untergeordnet** sind
- untergeordnete Bausteine bieten dabei ihren „Oberen“ **feste Dienste** an
- **Aufruf- oder Nutzungsbeziehungen** verlaufen strikt in Richtung dieser Hierarchie
- Hierarchische Strukturen werden in der Praxis **häufig** mit anderen Architekturstilen **kombiniert**.
- **Wichtige Vertreter**
  - Haupt-/Unterprogramm-Struktur → imperativen Programmiersprachen
  - Master-Slave, bei dem mehrere (redundante) Slave-Bausteine identische Dienste einem Master anbieten
  - Schichten (Layer) → sehr weit verbreitet
  - Ports und Adapter → spezielle Ausprägung des Schichtenmodells
  - virtuelle Maschine → auch Spezialisierung des Schichtenmodells

Hierarchische Systeme

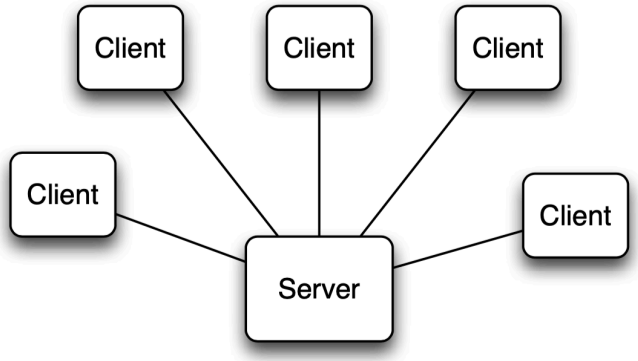
- **Schichten**
- ...

- Bausteine **innerhalb** einer Schicht können untereinander **frei** interagieren
- **zwischen zwei Schichten** kann nur durch die **vorgegebenen Schnittstellen** kommuniziert werden
- Empfehlung: strikte Schichtenarchitektur
  ⇔
  keine Überbrückung von Schichten
- **Hauptziele**
  - erhöhte **Veränderbarkeit** des Systems
  - erhöhte **Portierbarkeit** des Systems und/oder
  - erhöhte **Wiederverwendbarkeit** der Schichten



Verteilte Systeme

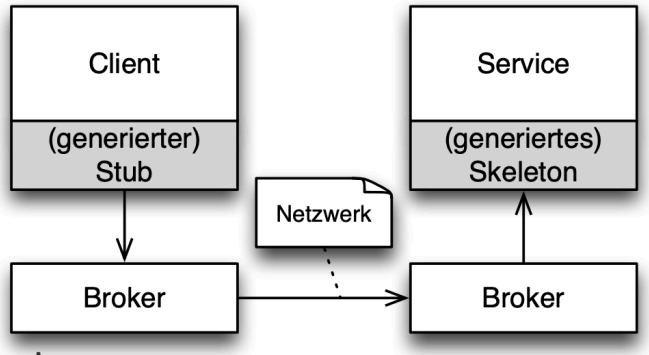
- **Client-Server**
- Broker-Architekturen
- serviceorientierte Architekturen
- Peer-to-Peer
- .....



- Client-Server-Systeme bestehen aus mindestens zwei unabhängigen aber **miteinander kommunizierenden Prozessen**:
  - **Clients benötigen** bestimmte Dienste oder Services, die sie bei einem ihnen bekannten Server erfragen.
  - **Server stellen** Clients bestimmte Dienste **bereit**. Auf entsprechende Anfragen hin erfüllen sie diese Aufgaben und schicken die Ergebnisse an den aufrufenden Client zurück.
- Die Kommunikation zwischen Client und Server erfolgt **synchron** – ansonsten klassifiziert man diese Systeme als ereignis- oder nachrichtenbasiert

Verteilte Systeme

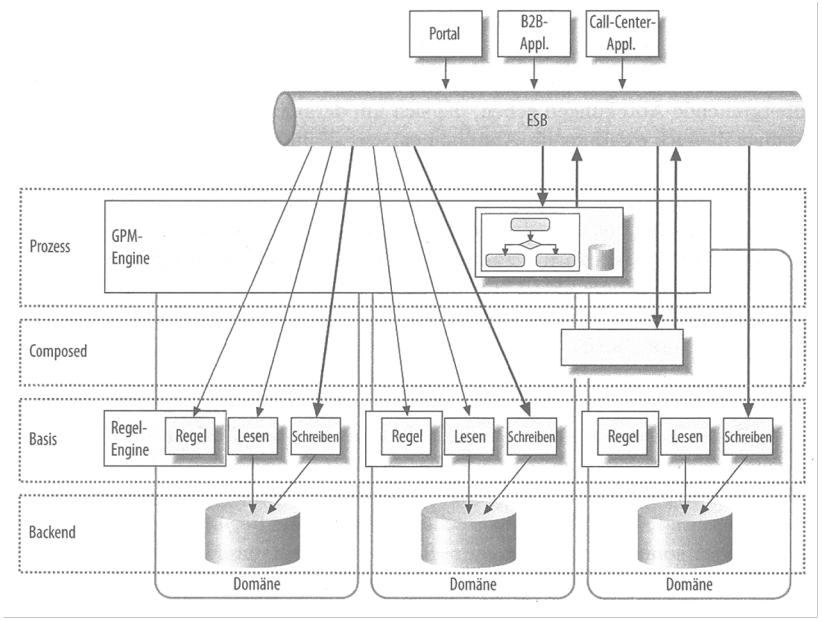
- Client-Server
- **Broker-Architekturen**
- serviceorientierte Architekturen
- Peer-to-Peer
- .....



- Architekturstil, der in den 80gern durch CORBA (Common Object Request Broker Architecture) bekannt geworden ist
- realisiert **Ortstransparenz**, sowie **Unabhängigkeit von Programmiersprache und Netzwerktopologie** durch folgende Komponenten:
  - **Client:** der einen bestimmten Dienst verwenden möchte, aber nur die Schnittstellendefinition kennt
  - **Service**: erbringt bestimmten Dienst ohne seine Clients zu kennen
  - **Broker**: koordiniert Aufrufe eines Clients mit angebotenen Diensten eines Service; regelt Parameter- und Ergebnisübergabe und findet die konkreten Implementierungen/Ausführungsorte von Services
  - Weitere Konzepte: Stubs, Skeletons und Bridges (optional)

Verteilte Systeme

- Client-Server
- Broker-Architekturen
- **serviceorientierte Architekturen**
- Peer-to-Peer
- .....



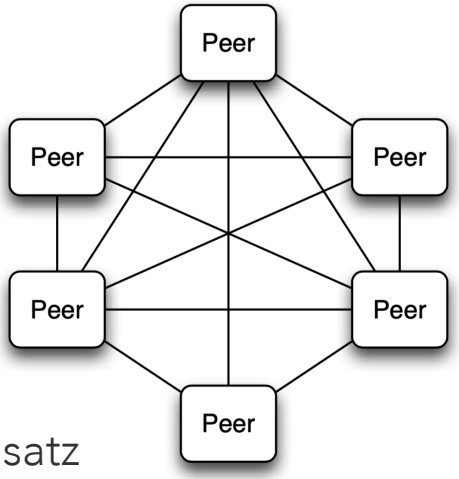
- Serviceorientierte Architekturen (**SOA**) zielen auf eine Entkopplung von Geschäftsprozessen und Funktionen bereitstellenden Software-Bausteinen ab. (ca. 2005)

- fachliche Funktionen werden als wiederverwendbare, verteilte, lose gekoppelte und standardisiert zugreifbare Dienste (**Services**) repräsentiert

- Spezielle Infrastrukturen bieten dazu ggf. folgende Komponenten an:
  - **Orchestrierung**: (Standard: Business Process Execution Language (BPEL)) → Process-Engine
  - Regelbasierte Steuerung → Rule-Engine
  - Unterstützung von Internet-Standards wie Web Services, SOAP und XML

Verteilte Systeme

- Client-Server
- Broker-Architekturen
- serviceorientierte Architekturen
- **Peer-to-Peer**
- .....

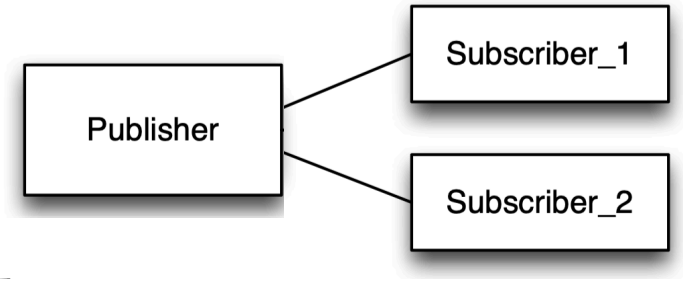


- Peer-to-Peer (P2P)-Architekturen kommen im **Internet** zum Einsatz
- (P2P)-Architekturen bestehen aus gleichberechtigten und über Netzwerke verbundenen Komponenten (**Peers**), die im Netz gleichzeitig die Rolle von **Clients und Servern** wahrnehmen
- Peers **teilen sich Ressourcen** wie etwa CPU-Zeit, Speicherplatz, aber auch Dateien
- (P2P)-Architekturen zeichnen sich durch **hohe Ausfallsicherheit** aus, weil es **keinen Flaschenhals** für Verfügbarkeit gibt (keinen single point of failure)
- das **Auffinden und Erkennen von Peers ist in großen Netzen schwierig**, ebenso die Fehler- und Ausnahmebehandlung
- Ein wichtiger Einsatzzweck von P2P-Netzen ist die **ausfallsichere Datenverteilung**. Heute werden u.a. digitale Telefonate oder Instant-Messaging-Daten über P2P-Netze geleitet.

Ereignisbasierte Systeme

- **Ungepufferte Kommunikation**: Broadcast, Publisher-Subscriber
- Gepufferte Kommunikation: Message-Queue / Message-Service

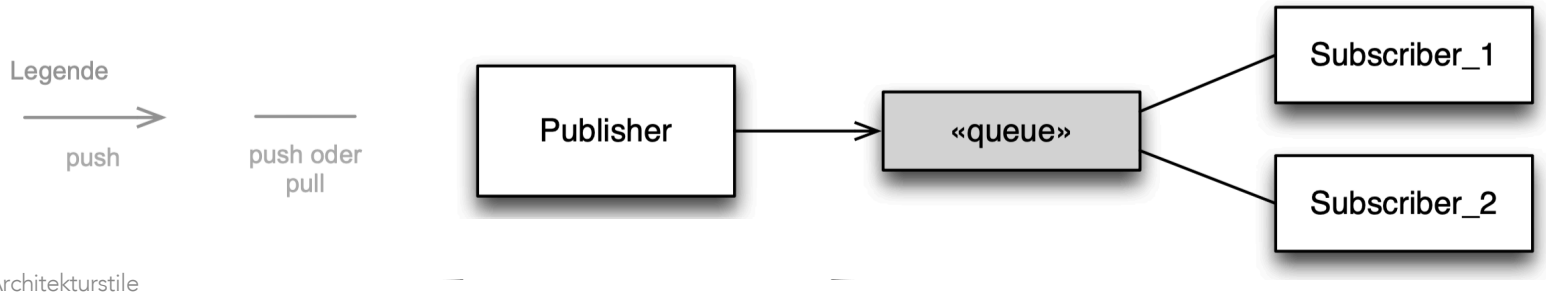
- In der **Broadcast-Variante** ereignisbasierter Systeme publiziert eine Event-Source Ereignisse auf einem lokalen Netzwerk. Alle verbundenen Empfänger prüfen jedes Ereignis, ob es für den jeweiligen Empfänger interessant ist: Falls ja, wird das Ereignis bearbeitet, falls nein, wird es ignoriert.
- Alternativ können sich Empfänger von Ereignissen (**Subscriber**) bei den Sendern (**Publisher**) registrieren. Sender übermitteln Ereignisse in diesem Fall direkt an Subscriber.



Ereignisbasierte Systeme

- Ungepufferte Kommunikation: Broadcast, Publisher-Subscriber
- **Gepufferte Kommunikation:** Message-Queue / Message-Service

- **Message-** oder **Event-Queue**-Architekturen ähnelt dem Publisher-Subscriber-Stil, wobei jedoch die Nachrichten oder Events (in einer Warteschlange) **gepuffert** werden.
- Ein Publisher schickt Nachrichten an eine **Queue**, aus der Empfänger sie entweder **synchron oder asynchron** abholen →fire-and-forget-Prinzip
- **Message-Service-Architekturen** bieten darüber hinaus noch **weitere Services** an, wie bspw. Routing, Prioritäten, Archivierung, Verschlüsselung, Protokollierung, etc.



Fachhochschule Dortmund
University of Applied Sciences and Arts

we focus on students

Architekturstile

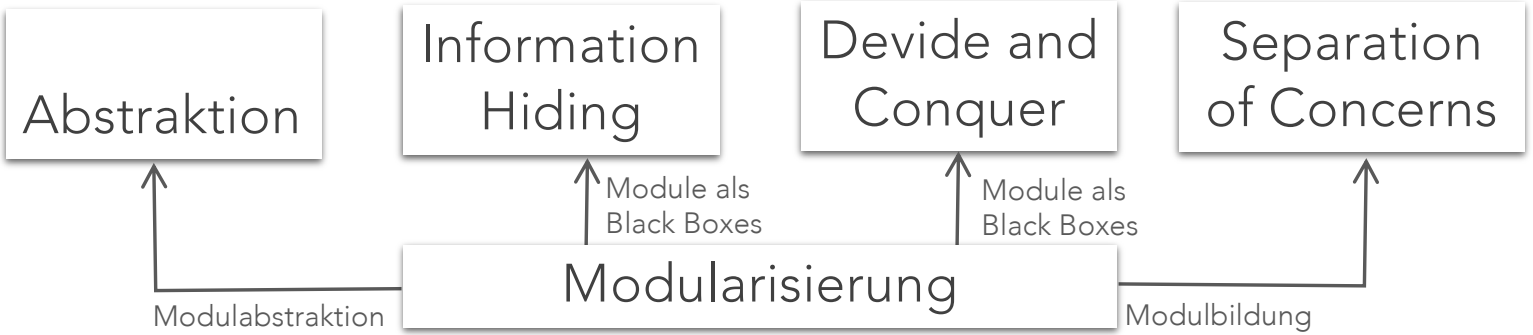
nach Gernot Starke 2017

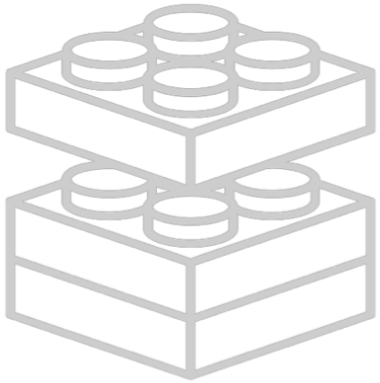
Stil	Intention	Beispiele
Datenflusssysteme	Besteht aus Folge (Sequenz) von Operationen auf Daten	Batch-Sequentiell Pipes und Filter Prozesssteuerung
Datenzentrische Systeme	gemeinsam verwendeter, zentraler Datenbestand	Repository Blackboard
Hierarchische Systeme	Bestehen aus Bausteinen in unterschiedlichen Ebenen einer Hierarchie	Haupt- und Unterprogramm, Master-Slave, Schichten Ports und Adapter virtuelle Maschine
Verteilte Systeme	Bestehen aus Speicher- und Verarbeitungsbausteinen, die über Kommunikationsnetze interagieren	Client-Server, CQRS, Broker-Architekturen, serviceorientierte Architekturen Peer-to-Peer
Ereignisbasierte Systeme	Bestehen aus voneinander unabhängigen Bausteinen, die über Ereignisse (Events) kommunizieren, sich implizit aufrufen.	Ungepufferte Kommunikation: Broadcast, Publisher-Subscriber Gepufferte Kommunikation: Message-Queue, Message-Service
Interaktionsorientierte Systeme	Systeme mit (grafischer) Bedienoberfläche	Model-View-Controller u.a.
Heterogene Systeme	Verwenden mehrere Architekturstile parallel	beispielsweise REST

Fachhochschule
Dortmund
University of Applied Sciences and Arts

Prinzipien

Elementare (Architektur-)Entwurfsprinzipien





Modularisierung

- Abstraktion
- Geheimnisprinzip („Information Hiding))
- Teile-und-herrsche („Divide and Conquer“)
- Getrennte Verantwortlichkeiten („Separation of Concerns“)

Fachhochschule Dortmund
University of Applied Sciences and Arts

KISS

Eine Lebensweissheit, die auch für Architekturen gilt

we focus on students

Einfachheit gewinnt (Keep It Small and Simple, KISS)

Perfektion ist nicht dann erreicht, wenn es nichts mehr hinzuzufügen gibt,
sondern wenn man nichts mehr weglassen kann.

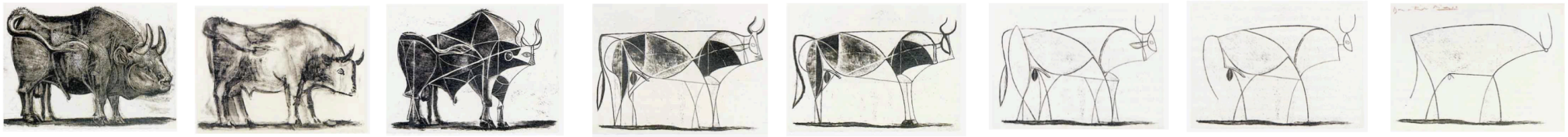
Antoine de Saint-Exupéry

Abstraktion

Grundidee: Zweckgerichtete Reduktion des Informationsgehaltes des Abbilds

- Abstraktionen (Reduktion, Komposition, Generalisierung, Benutzung)
- Systematische Vergröberung und Verfeinerung nach verschiedenen Kriterien
- Verständnis großer Zusammenhänge unter Weglassung der Details
- Verständnis eines Details unter Weglassung/starker Vergröberung des Rests
- Systematischer Zusammenhang zwischen Überblickssichten und Detailsichten

Beispiel: **„El toro" , Pablo Picasso 1945**



[T. Weyer, WS19/20]

Geheimnisprinzip

Grundidee: Verberge die innere Komplexität nach außen (Parnas 72)

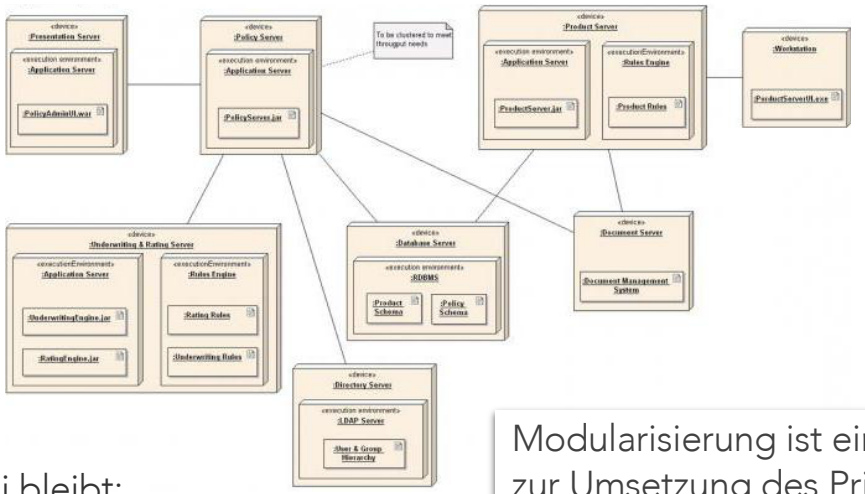
- Zerlegung eines Systems in eine Menge jeweils in sich geschlossener Bausteine (Module).
- Jedes Modul sollte bestimmte Verantwortlichkeiten kapseln (Kohäsion) , die über wohldefinierte Schnittstellen zugänglich sind.
- Die Bausteine sollten Blackboxes sein, deren Innenleben möglichst verborgen bleibt: Es genügt, die Schnittstelle von Bausteinen zu kennen.
- Vorteile: Modulare Bausteine
  - können getrennt voneinander entwickelt,
  - unabhängig voneinander geändert und
  - ohne Nebenwirkungen durch andere Bausteine mit identischer Schnittstelle ausgetauscht werden.

Modularisierung ist eine Spezialisierung des Prinzips der „Getrennten Verantwortlichkeiten“!

Teile-und-herrsche

Grundidee: Zerlegung des Gesamtentwurfsproblems in Teilprobleme, die getrennt betrachtet werden

- ein Modul ist in der Softwaretechnik ein Baustein eines Softwaresystems, der bei der Modularisierung entsteht und
  - der eine funktional geschlossene Einheit darstellt und einen bestimmten Dienst bereitstellt;
  - dessen Verwendung keine keine Kenntnisse über den inneren Aufbau erfordert;
  - der mit der Umgebung ausschließlich über Schnitt-stellen kommuniziert ;
  - der Entwurfsentscheidungen kapselt;
  - dessen innere Veränderung nach außen rückwirkungsfrei bleibt;
  - dessen Korrektheit ohne Kenntnis der Einbettung prüfbar ist.



Modularisierung ist ein Mittel zur Umsetzung des Prinzips: „Teile-und-herrsche"!

[T. Weyer, WS19/20]

Getrennte Verantwortlichkeiten

Grundidee: Perspektivwechsel

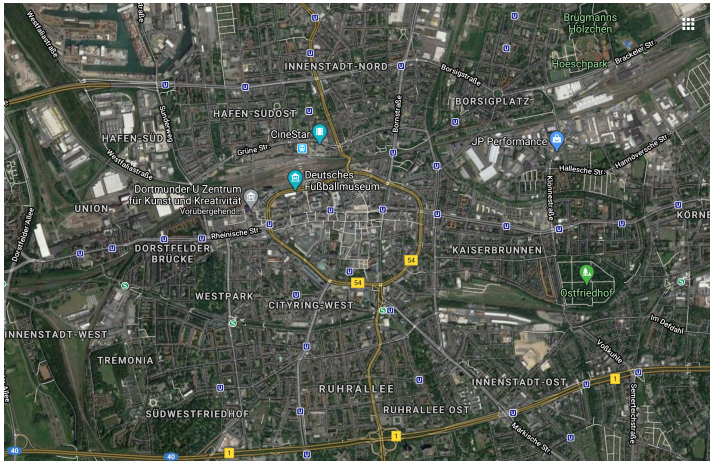
- kritische Situationen lassen sich durch einen Wechsel der Perspektive manchmal besser bewältigen
- eine wirkungsvolle Methode beim Entwurf, da Entwurfsentscheidungen sich besser einschätzen lassen
- Einige Beispiele für solche Perspektiven:
  - Die Kontextabgrenzung, die das gesamte System als eine einzige Blackbox ansieht (und damit von Lösungsdetails abstrahiert)
  - Technische und fachliche Perspektiven
  - Die klassischen Architektursichten (Baustein-, Laufzeit- und Verteilungssicht)

Fachhochschule
Dortmund
University of Applied Sciences and Arts

Architektursichten

Getrennte Verantwortlichkeiten → Dijkstra

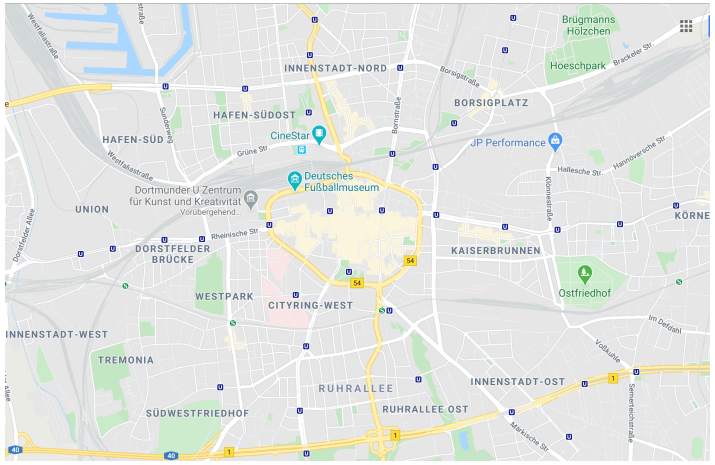
Grundidee: Betrachtung aus spezifischen Gesichtspunkten heraus

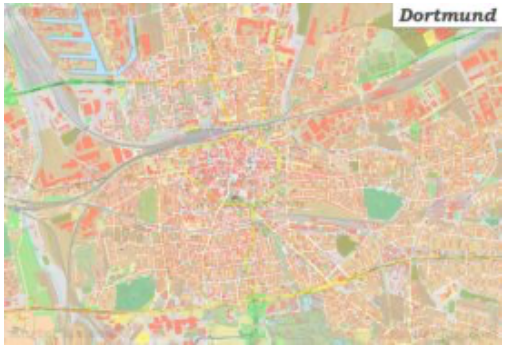


Gelände

Sehenswürdigkeiten

Verkehr

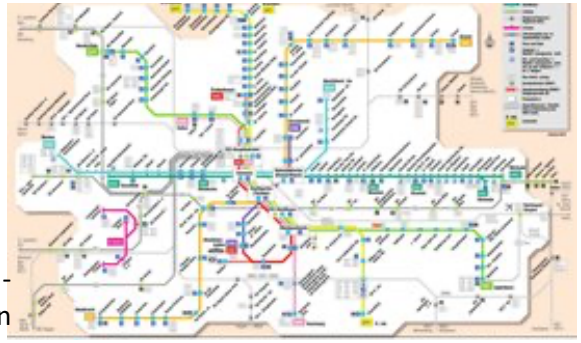




großer Stadtplan

S-Bahn und U-Bahn

Schienen-netzplan

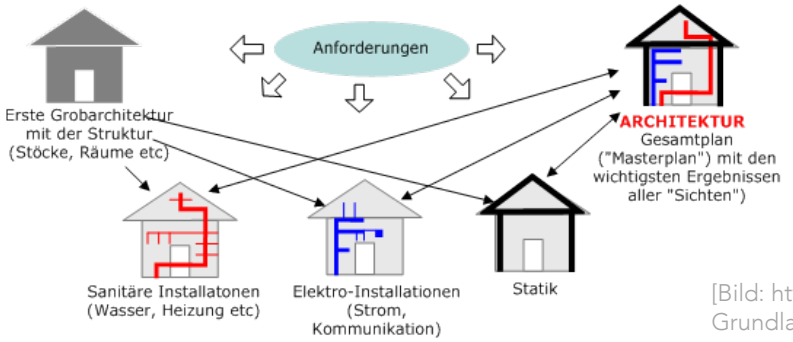


Hilfreiche Strukturen für Architekturen

	Software Structure	Element Types	Relations	Useful For	Quality Attributes Affected
Module Structures	Decomposition	Module	Is a submodule of	Resource allocation and project structuring and planning; information hiding, encapsulation; configuration control	Modifiability
	Uses	Module	Uses (i.e., requires the correct presence of)	Engineering subsets, engineering extensions	"Subsetability," extensibility
	Layers	Layer	Requires the correct presence of, uses the services of, provides abstraction to	Incremental development, implementing systems on top of "virtual machines"	Portability
	Class	Class, object	Is an instance of, shares access methods of	In object-oriented design systems, factoring out commonality; planning extensions of functionality	Modifiability, extensibility
	Data model	Data entity	{one, many}-to-{one, many}, generalizes, specializes	Engineering global data structures for consistency and performance	Modifiability, performance
Component-and-connector Structures C&C Structures	Service	Service, ESB, registry, others	Runs concurrently with, may run concurrently with, excludes, precedes, etc.	Scheduling analysis, performance analysis	Interoperability, modifiability
	Concurrency	Processes, threads	Can run in parallel	Identifying locations where resource contention exists, or where threads may fork, join, be created, or be killed	Performance, availability
Allocation Structures	Deployment	Components, hardware elements	Allocated to, migrates to	Performance, availability, security analysis	Performance, security, availability
	Implementation	Modules, file structure	Stored in	Configuration control, integration, test activities	Development efficiency
	Work assignment	Modules, organizational units	Assigned to	Project management, best use of expertise and available resources, management of commonality	Development efficiency

[Bass et al: Software Architecture in Practice, 3. Auflage, S.15, Addison Wesley, 2012.]

Sichten (Views)



[Bild: https://adenetvm.m-s.ch/Doc/Grundlagen/Systemarchitektur.htm]

- Begriffsklärung
  - Eine Sicht (engl. view) ist eine Repräsentation eines Gesamtsystems aus der Perspektive einer Menge von zusammenhängenden Anliegen (Concerns).
- Eigenschaften von Sichten
  - Jede Sicht beschreibt nur gewisse Eigenschaften des Gesamtsystems (**Selektivität**).
  - Jede Sicht braucht eine Beschreibungstechnik (**Plan oder Modell**).
  - Sichten sind meist nicht komplett unabhängig voneinander (Idealfall: **Orthogonalität**).
  - Sichten sollten möglichst parallel bearbeitet werden können (**Schnittstellenproblematik**).
  - Zwei Sichten eines Systems sollten sich nicht widersprechen (**Konsistenz**).
  - Die verschiedenen Sichten sollten in eine **Gesamtsicht** einmünden.

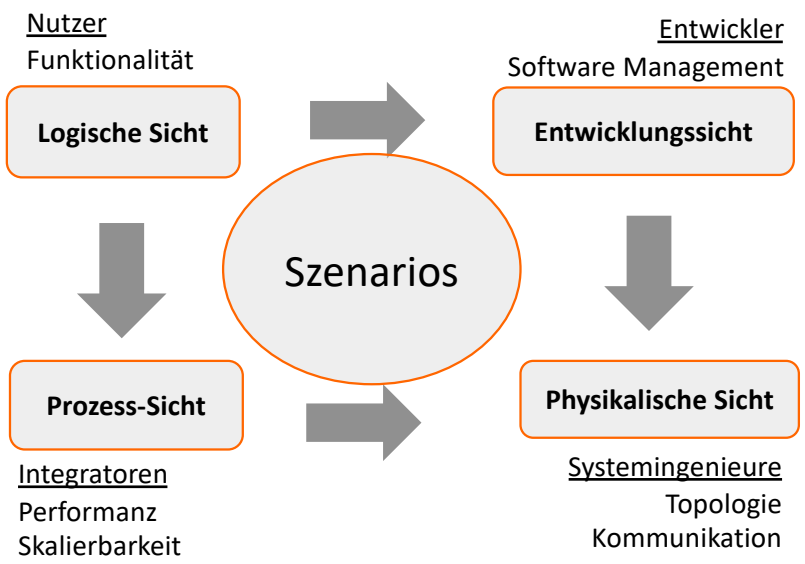
Zachmann-Framework

[J. A. Zachman: A framework for information systems architecture. In: IBM Systems Journal. Band 26, Nr. 3, 1987, S. 277–293.]

	DATEN ***Was***	**FUNKTION** ***Wie***	**NETZWERK** ***Wo***	**PERSONEN** ***Wer***	**ZEIT** ***Wann***	**MOTIVATION** ***Warum***
Zielsetzung/Bereich (*Kontextabhängig*) → *Rolle: Planer*	Liste von wichtigen Faktoren im Geschäft	Liste von Kernprozessen	Liste von Geschäftsstellen	Liste von wichtigen Organisationen	Liste von Ereignissen	Liste von Geschäftszielen/Strategien
Unternehmensmodell (*Konzeptionell*) → *Rolle: Besitzer*	Konzeptionell Datenmodell/ Objektmodell	Geschäftsprozessmodell	Geschäftslogistiksystem	Arbeitsablaufmodell	Ablaufplan	Geschäftsplan
Systemmodell (*Logisch*) → *Rolle: Designer*	Logisches Datenmodell	Systemarchitekturmodell	Distributed Systems Architecture	Human-Interface-Architektur	Prozessstruktur	Geschäftsregelmodell
Technologiemodell (*Physisch*) → *Rolle: Builder*	Physische Daten/ Klassenmodell	Technologie-designmodell	Technologiearchitektur	Darstellungsarchitektur	Kontrollstruktur	Regeldesign
Detaillierte Darstellung (*Aus dem Kontext heraus*) → *Rolle: Programmierer*	Datendefinitionen	Programm	Netzwerkarchitektur	Sicherheitsarchitektur	Zeitplan	Regelspezifizierung
Unternehmen → *Rolle: Nutzer*	Nutzbare Daten	Anwendungszweck	Nutzbares Netzwerk	Arbeitsorganisation	Eingeschlossener Zeitplan	Arbeitsweise

4+1 Sichtenmodell

[Kruchten, Phillipe: „Architectural Blueprints. The "4+1" View Model of Software Architecture", IEEE Software 12 (6), 1995, S. 42-50]



Logische Sicht
- Datenmodell
- Nutzersicht / Fachsicht

→ Klassendiagramm/ Kommunikationsdiagramm/ Sequenzdiagramm

Prozess-Sicht
- Abläufe

→ Aktivitätsdiagramm / Sequenzdiagramm

Physikalische Sicht
- Zusammenhang zwischen Hard- und Software

→ Verteilungsdiagramm

Entwicklungssicht
- die statische Organisation der Software

→ Komponentendiagramm/ Code

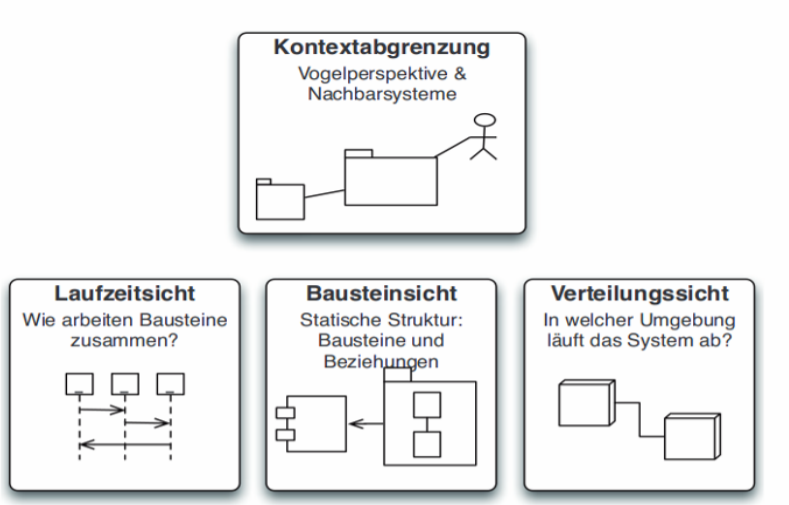
Szenarios
- Benutzungsszenarien

→ Use Cases

Vier Arten von Sichten von Starke

[Starke, Gernot: „Effektive Software-Architekturen. Ein praktischer Leitfaden“, 8. Auflage, Carl Hanser Verlag 2017]

- Kontextabgrenzung
  - Einbettung des Systems in seine Umgebung (Nachbarsysteme, Stakeholder, Infrastruktur)
  - System als Blackbox
  - Sehr abstraktes Modell
- Bausteinsicht
  - Aufbau des Systems aus Subsystemen, Komponenten,
  - Teilpaketen Frameworks, Konfigurationen, ...
  - Zusammenwirken der Bausteine (Schnittstellen)
  - Top-Down mit Blackboxen und Whiteboxen
- Laufzeitsicht
  - Dynamische Struktur
  - Interaktion von Laufzeitinstanzen
- Verteilungssichten (Infrastruktur-)
  - Technische Ablaufumgebung
  - Hardware-Komponenten und ihr Zusammenspiel
  - Deployment-Einheiten

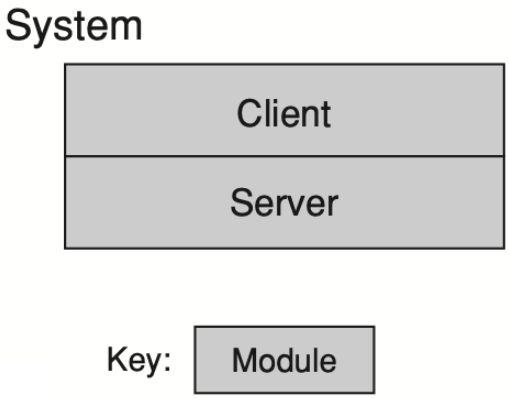


Ergänzende Sichten

Zwei Sichten eines Client-Server-Systems

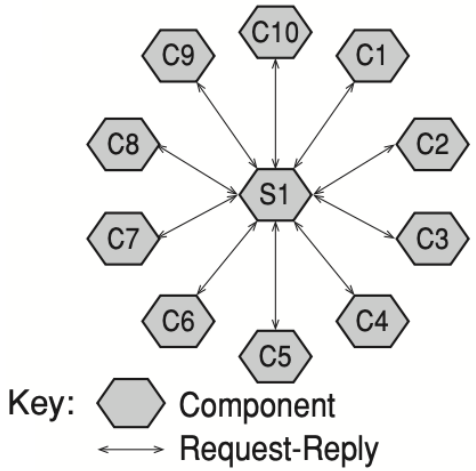
- Modulzerlegungsansicht
  - die Client-Software und
  - die Server-Software.

  => zwei Module



- Komponenten- und Anschlussansicht
  - zur Laufzeit laufen **zehn Clients**,
  - die auf **den Server** zu greifen

  => elf Komponenten



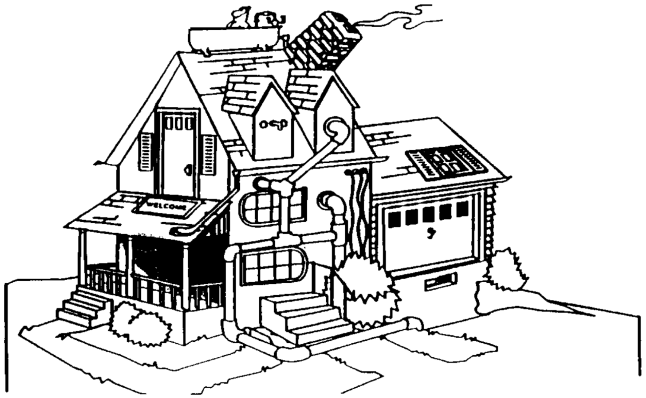
[Bass et al: Software Architecture in Practice, 3. Auflage, S.16, Addison Wesley, 2012.]

Sichten

Take Aways

- verschiedenste Sichtenmodelle in der Literatur, die Architektursichten beschreiben

- wichtige Sichten (Blickwinkel) für Softwarearchitekten
  - Fachliche Sicht
  - Technische Sicht
  - Implementierungssicht
  - Testsicht
  - Verteilungssicht
  - Übergeordnete Sicht (Zusammenhang, Wiederverwendung, Regeln)



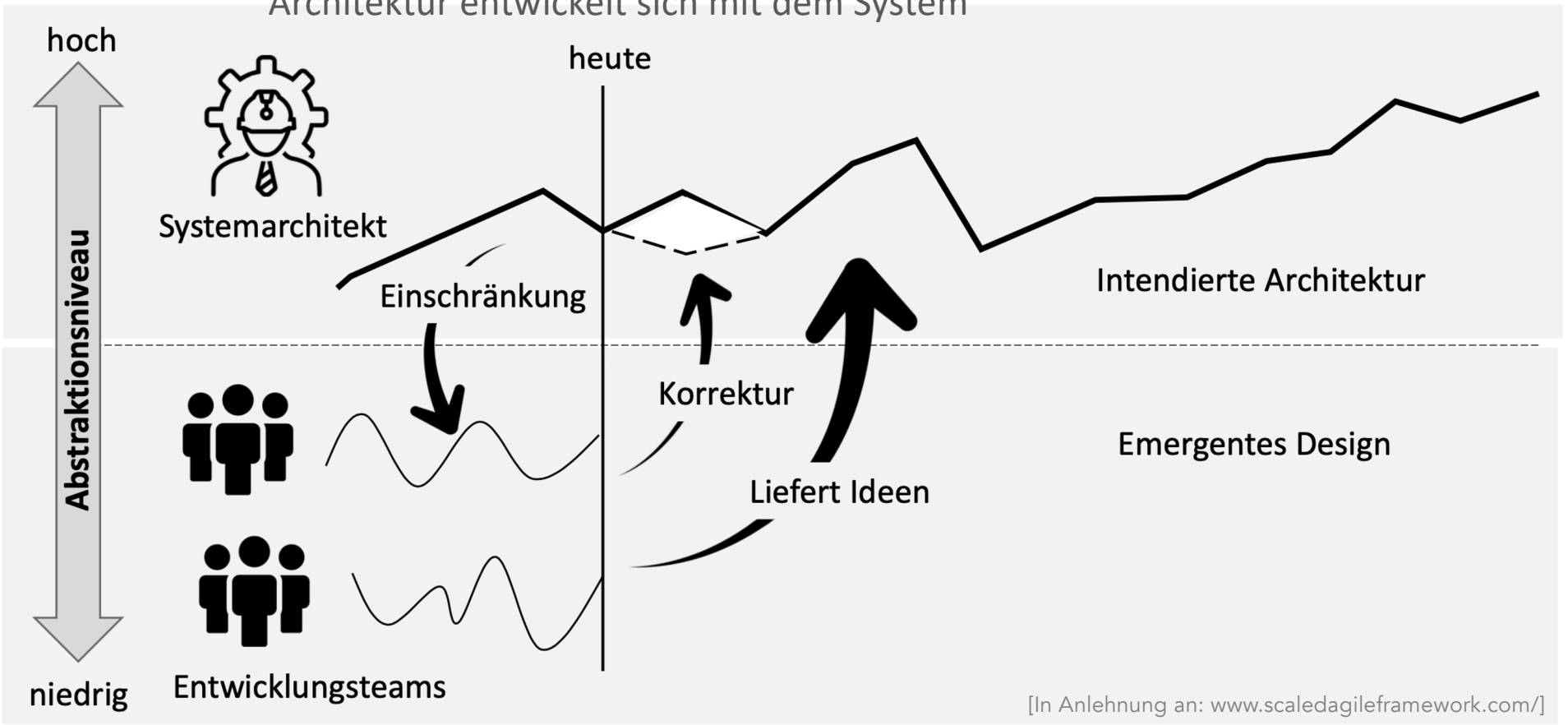
[Bild: Unbekannter Künstler]

Fachhochschule
Dortmund
University of Applied Sciences and Arts

Architektur und Agilität

Architektur in der agilen Entwicklung

Architektur entwickelt sich mit dem System



[In Anlehnung an: www.scaledagileframework.com/]

Fachhochschule
Dortmund
University of Applied Sciences and Arts

Architekturstile