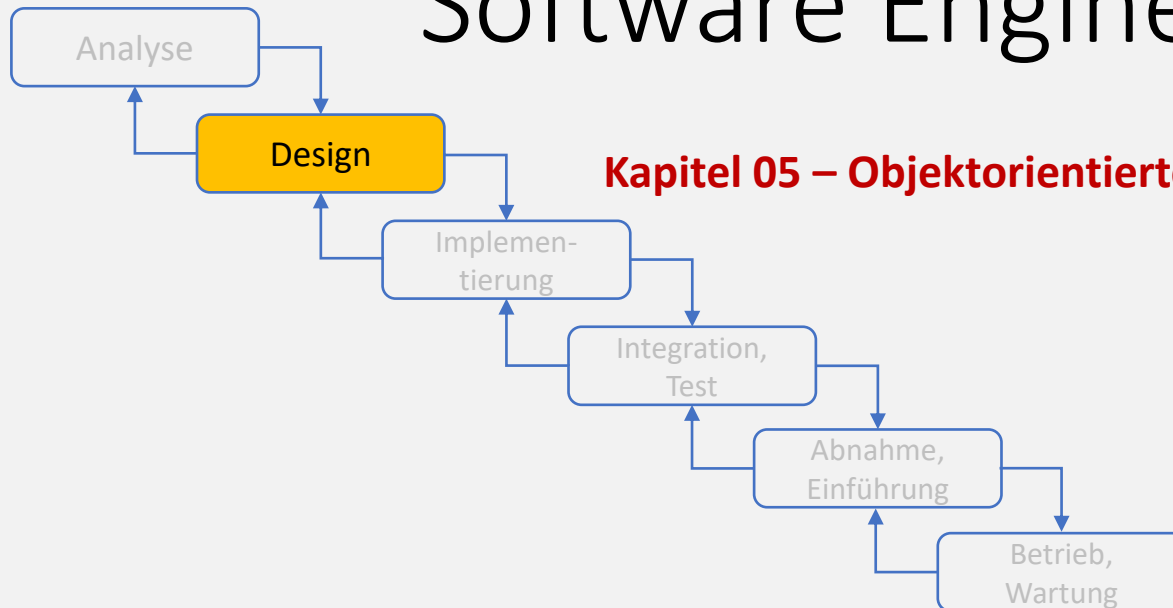




# Software Engineering

## Kapitel 05 – Objektorientiertes Design (OOD)



Softwarearchitektur und  
Architektursichten

# Gliederung

---

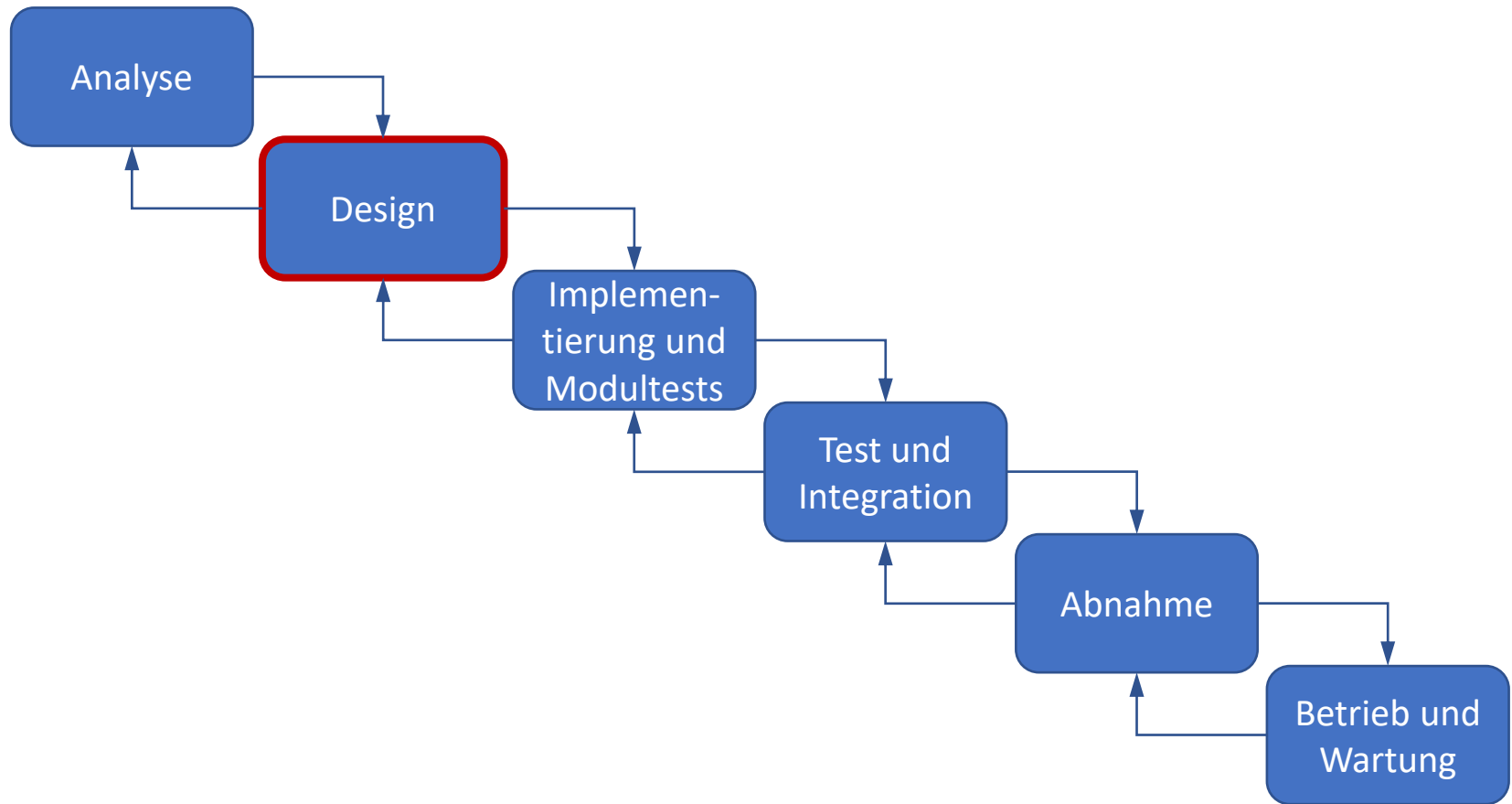
1. Motivation
2. Softwarearchitektur
3. Architektursichten
  - nach Kruchten
  - nach Starke
4. Architekturprinzipien (Kopplung, Kohäsion, DRY, OCP, ...)
5. Architekturmuster (Schichten, Pipes&Filters, Blackboard, ...)

# Lernziele

---

- Was ist Objektorientiertes Design?
- Warum betreiben wir Objektorientiertes Design?
- Welche Methoden unterstützen uns beim OOD?


# Phasen eines Software-Projektes



# Designphase

Nach der **Analysephase** (Analyse und Modellierung) der in Software abzubildenden Prozesse und

Vor der **eigentlichen Implementierung** werden weitere vorbereitende Schritte unternommen, um eine passende Software zu erstellen:

- Grobentwurf: **Softwarearchitektur**
  - Feinentwurf:
    - Strukturiertes Design (SD)
    - **Objektorientiertes Design (OOD)**
- 
- Themen dieser Vorlesung

# Designphase: Motivation

---

## Eigenschaften eines typischen IT-Großprojektes:

- 60-70 Anwendungskernobjekte
- 100-200 Dialoge
- Projektdauer ca. 2-3 Jahre
- Teamstärke variiert von 2 – 20 oder mehr
- Hohe Investitionskosten
- Potentiell hunderte Benutzer im Dialogbetrieb
- Entscheidend für das Kerngeschäft der Organisation, die das Projekt beauftragt hat

# Designphase: Motivation

---

## Typische Ausgangssituation im IT-Großprojekt:

- 15 Entwickler starten gleichzeitig mit der Implementierung
- 12 der 15 Entwickler haben nur geringe Erfahrung bei der Entwicklung von Großprojekten
- 3 der 15 Entwickler haben Projekterfahrung in mehreren Projekten
- Trotzdem wird das Projekt ein Erfolg! Warum?

# Designphase: Motivation

## Der Entwurf:

I am more convinced than ever. Conceptual integrity is central to product quality. **Having a system architect** is the **most important** single step toward conceptual integrity... After teaching a software engineering laboratory more than 20 times, I came to insist that **student teams** as small as **four people choose a manager,** and **a separate architect.**

[Fred Brooks, The Mythical Man-Month (20th Anniversary Edition. 1995)]



# Designphase: Ziele, Aktionen und Ergebnisse

Ziel:	Wie und womit erfolgt die Realisierung?
Aktionen:	<ul style="list-style-type: none"><li>• Ermitteln und/oder Festlegen von Umgebungs- und Randbedingungen</li><li>• Grundsatzentscheidungen treffen</li><li>• Spezifikation der Systemkomponenten</li><li>• Programmierung im Großen</li></ul>
Ergebnis:	<ul style="list-style-type: none"><li>• Softwarearchitekturmodell</li><li>• Systemkomponenten und Beziehung untereinander</li><li>• Schnittstellen zwischen Komponenten</li><li>• Schnittstellen zur Umgebung des Produkts</li></ul>

# Designphase: Ziele

- **Motivation:**

- Übergang von fachlichen Anforderungen hin zu einer Realisierung **(Produktdefinition: WAS?)**  
**(Produktentwurf: WIE?)**
- Um Problemkomplexität beherrschbar zu machen ist Strukturierung und Dekomposition notwendig

- **Ziele des Entwurfs:**

- Gliederung des Systems in **überschaubare Einheiten** (Systemkomponenten)
- Festlegung der **Lösungsstruktur** (Wie soll das Produktmodell realisiert werden)
- **Hierarchische Gliederung**
- Beschreibung der **Beziehungen zwischen Systemkomponenten**
- Spezifikation des Funktions- und Leistungsumfangs sowie des Verhaltens der Systemkomponenten (informell, semiformal oder formal)
- Festlegung der **Schnittstellen**, über die die Systemkomponenten kommunizieren

- **Hilfsmittel:**

- Standardstrukturen, Muster (Patterns)

# Designphase: Ausgangspunkt

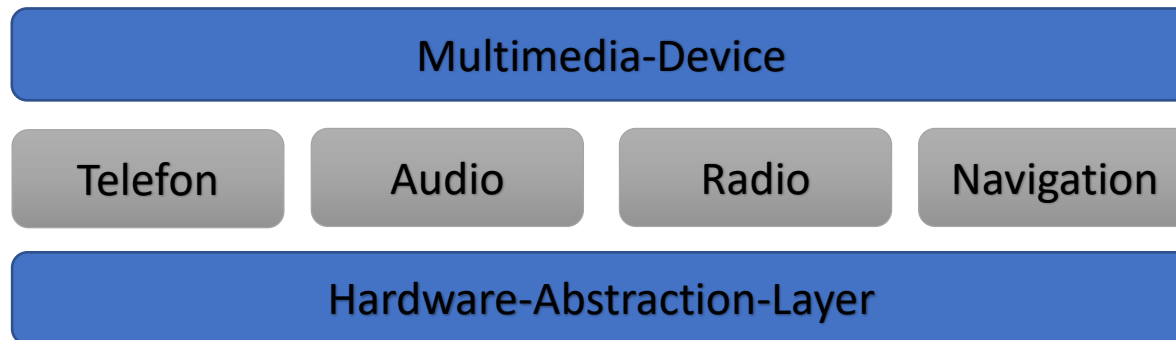
- **Ausgangspunkt:**
  - Anforderungsspezifikation (Pflichtenheft)
  - Produktmodell (Klassen-, Sequenz-, Aktivitäts-, Zustandsdiagramme, etc.)
- **Ziel:**
  - Vom “WAS” zum “Wie”: als Vorgabe für die darauffolgende Implementierung
- **Achtung:**
  - Es gibt keine generell gute oder schlechte Architektur
  - Wichtig ist immer der Kontext der zu definierenden Ziele und die Flexibilität gegenüber künftiger Änderungen

# Designphase: Gliederung (Grobentwurf)

## Software-Design wird häufig in Grob- und Feinentwurf unterteilt

- **Grobentwurf:**

- Architekturentwurf
  - Subsystem-Spezifikation
  - Schnittstellen-Spezifikation
  - Möglichst unabhängig von der Implementierungssprache
- Beispiel: Auto-Entertainment-System



# Designphase: Gliederung (Feinentwurf)

## Software-Design wird häufig in Grob- und Feinentwurf unterteilt

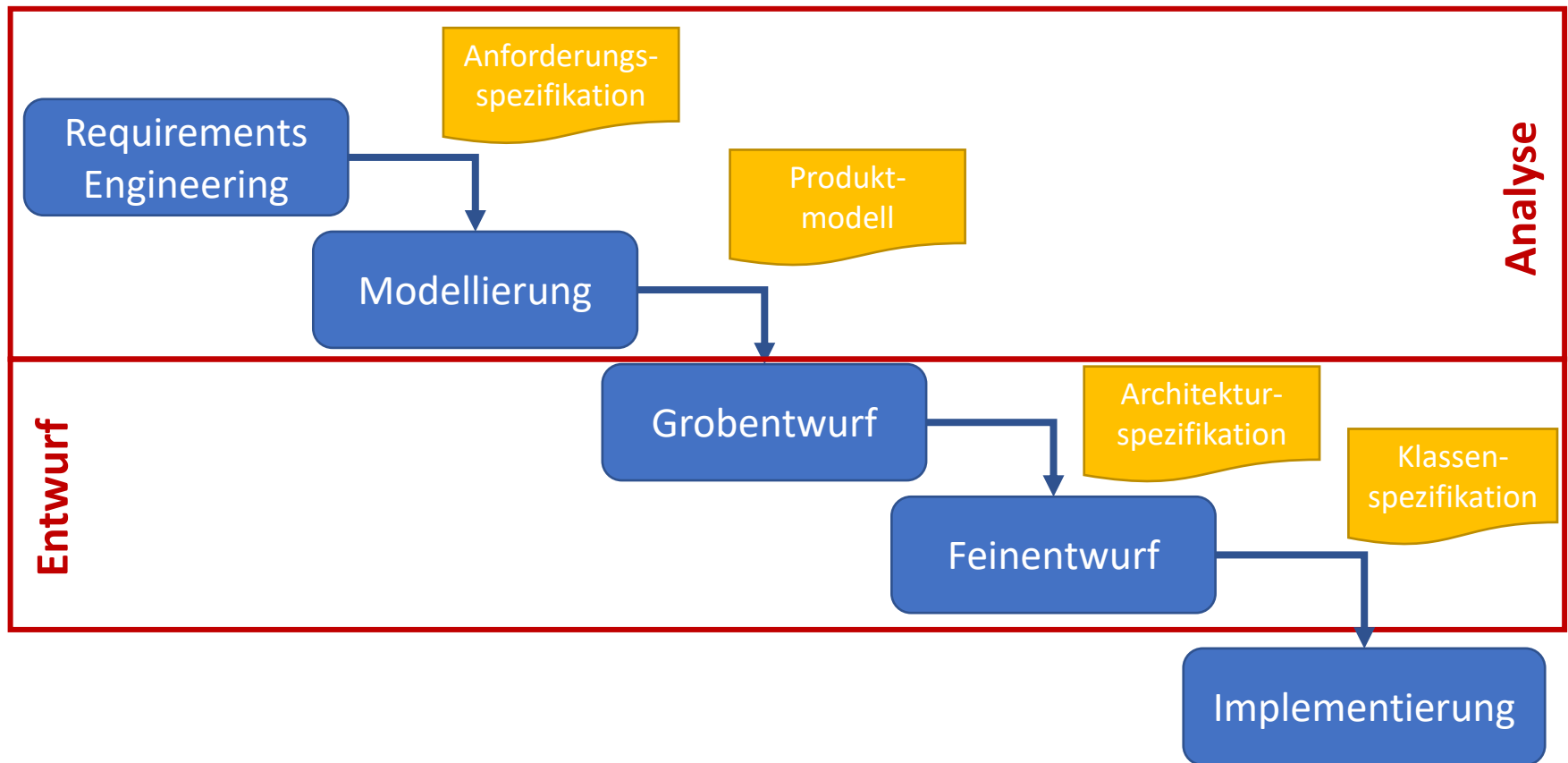
- **Feinentwurf:**

- Komponentenentwurf
- Datenstrukturentwurf
- Algorithmen
- Angepasst an Implementierungssprache und Plattform

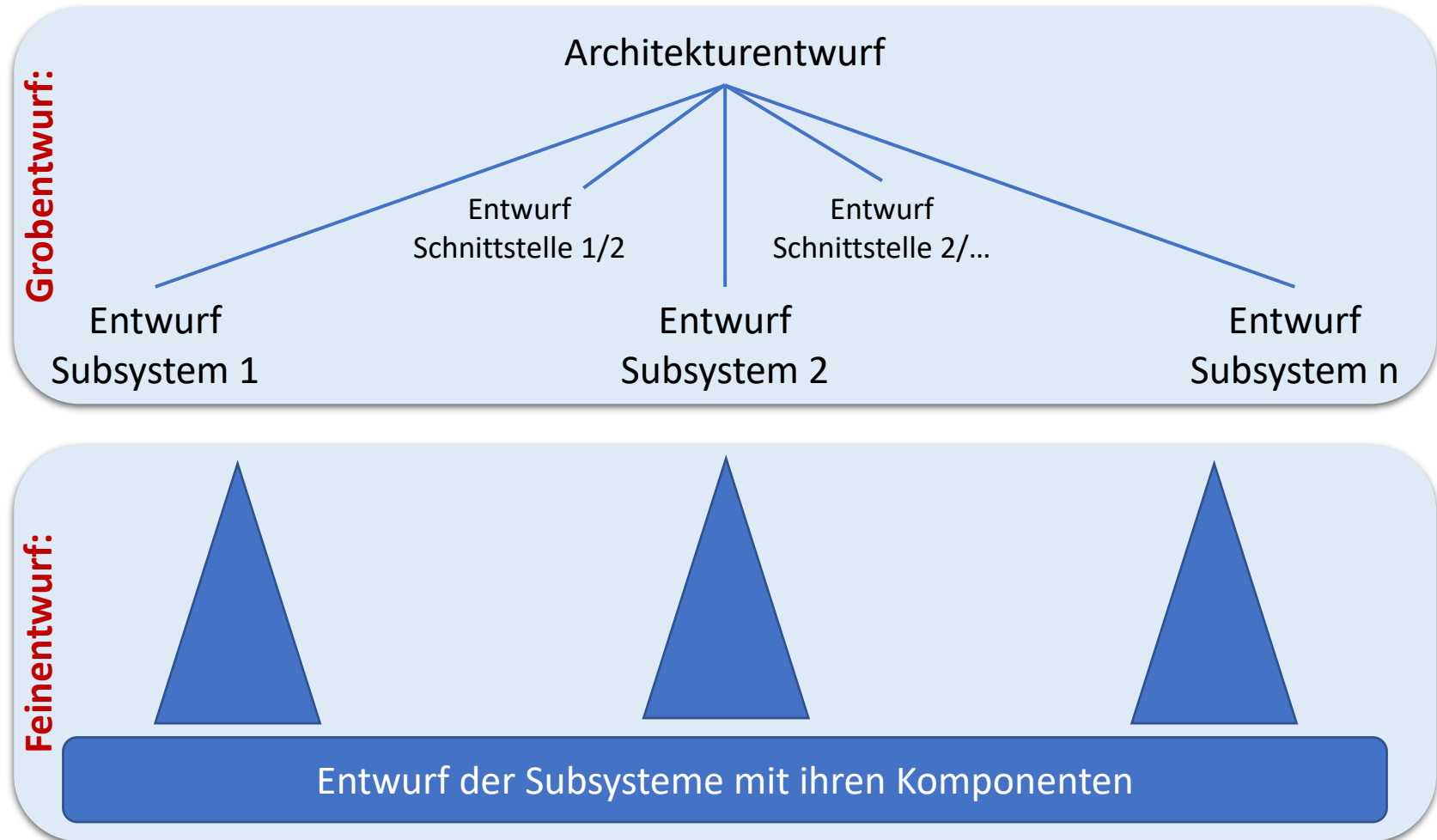


# Designphase: Architektur

## Von der Anforderung zur Architektur:



# Designphase: Arbeitsteilung beim Entwurf



# Designphase: Kriterien für einen guten Entwurf

## Korrektheit:

- Erfüllung der Anforderungen
- Wiedergabe aller Funktionen des Systemmodells
- Sicherstellung der nicht-funktionalen Anforderungen

## Wiederverwendung:

- Gleichartige Aufgaben sollten nicht mehrfach realisiert werden
- Ansonsten: Probleme in der Weiterentwicklung und Wartung

## Verständlichkeit & Präzision:

- Gute Dokumentation
- Programmierstil, Logische Struktur

## Anpassbarkeit:

- Einfache Erweiterbarkeit

### Hinweis:

Kriterien gelten auf allen Ebenen des Entwurfs (Architektur-, Subsystem- und Komponentenebene)



# UML als Beschreibungsmittel

## UML-Beschreibungsmittel für die Entwurfsphase:

- für den Architekturentwurf:
  - Logische Strukturen (Pakete, Paketdiagramme, Subsysteme, Schnittstellen)
  - Physische Strukturen (Komponenten, Komponentendiagramme, Einsatzdiag.)
- für den Strukturentwurf:
  - Klassendiagramme, Klassen
- für den Verhaltensentwurf:
  - Interaktionsdiagramme, Zustandsdiagramme/Zustandsautomaten

# Warum Software-Architektur?



- Strukturiertes Vorgehen bei SW-Entwicklung
- Solides Fundament für Software (“Statik”)
- Wohldefinierte Punkte für Erweiterungen
- Keine unkontrollierten “Balkonanbauten”

# Softwarearchitektur

# Softwarearchitektur: Definition

---

## **Definition: Softwarearchitektur**

A software architecture provides a model of a whole software system that

- Is composed of internal behavioral units (i.e. components) and
- Their interaction, at a certain level of abstraction

All postulated requirements that are relevant to the later construction of the system have to be incorporated in this model.

# Softwarearchitektur: Definition

## Definition: Softwarearchitektur

A software architecture provides a **model** of a whole software **system** that

- Is **composed of** internal behavioral units (i.e. **components**) and
- Their **interaction**, at a certain level of **abstraction**

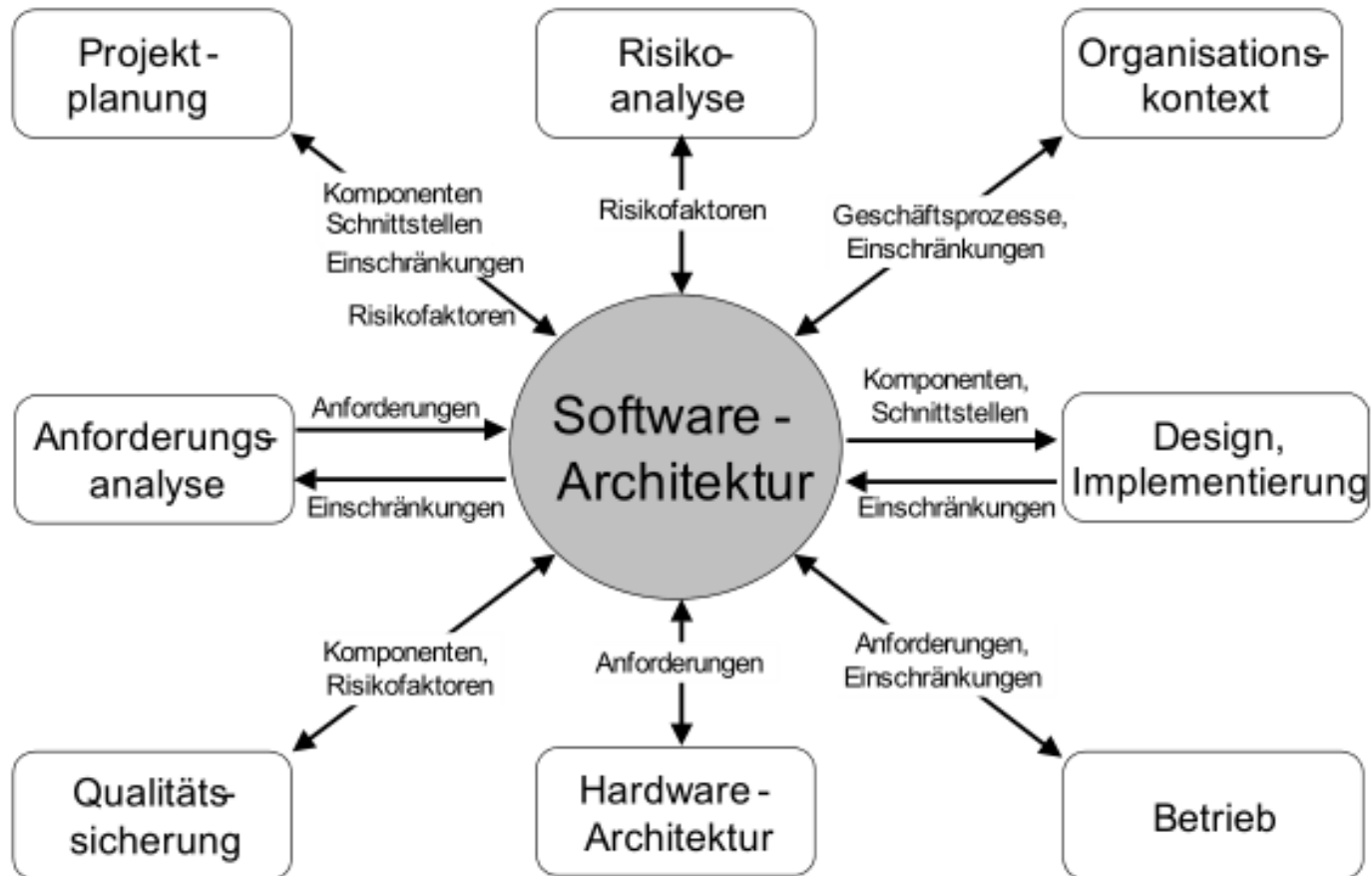
**All** postulated **requirements** that are **relevant** to the later construction of the system have to be **incorporated** in this model.

# Softwarearchitektur

## Softwarearchitektur:

- Früheste Softwaredesign-Entscheidung der Systementwicklung (Architekturentwurf)
- Ergebnis ist Architekturmodell (klärt: **wie ist System** als Reihe miteinander kommunizierender Komponenten **organisiert**)
- **Essentielle Eigenschaften** wie Modifizierbarkeit, Wartbarkeit, Sicherheit oder Performanz sind **von diesem Entwurf abhängig**
- Einmal eingerichtete Softwarearchitektur ist später **nur mit viel Aufwand** (Kosten!) **abänderbar**
  - Entscheidung über dieses Design ist einer der kritischsten Punkte im SW-Entwicklungsprozess
- Es besteht immer die **Gefahr**, dass **ein in der Theorie sehr gut abgestimmtes** Architekturkonzept in der Implementierungsphase nicht optimal oder sogar gar **nicht umgesetzt werden kann** (Gründe: technischer Natur oder wegen nicht einkalkuliertem Zusatzaufwand)

# Kontext der Softwarearchitekturarbeit

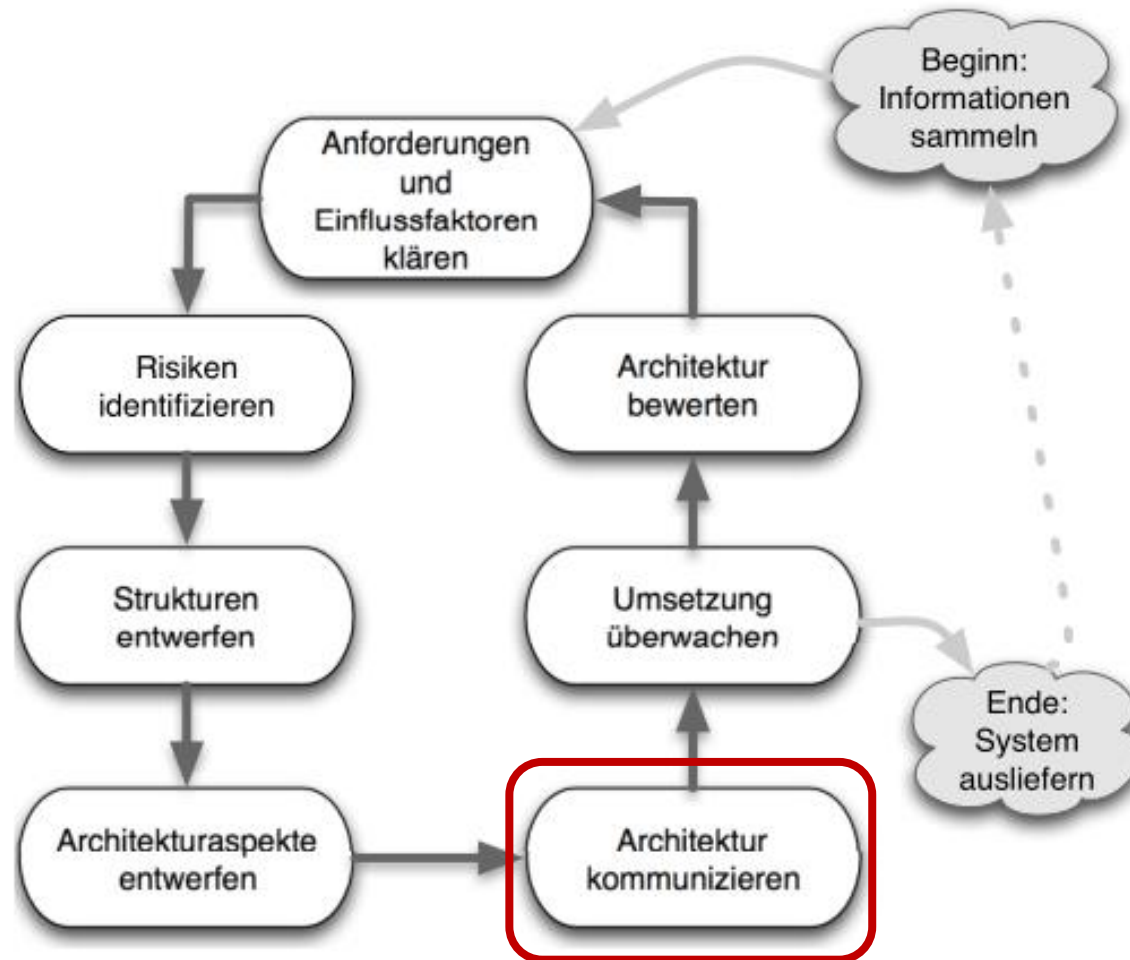


# Vorteile eines expliziten Entwurfs und eindeutiger Dokumentation der SW-Architektur

- **Kommunikation** (zwischen Projektbeteiligten):
  - Architektur ist stark vereinfachte Darstellung des Systems
  - Damit: Diskussionsgrundlage für verschiedene Projektbeteiligte
- **Systemanalyse**:
  - Um Systemarchitektur in frühem Stadium der Systementwicklung klar darzustellen sind verschiedene Analysen notwendig
  - Entscheidungen haben tiefgreifenden Einfluss auf kritische Anforderungen des Systems (Leistungsfähigkeit, Zuverlässigkeit, Wartbarkeit, etc.)
- **Wiederverwendung**:
  - Als kompakte Darstellung des Systems im Hinblick auf den Systemaufbau und die Interaktion der Komponenten in Systemen mit ähnlichen Anforderungen wiederverwendbar



# Entwurf von Softwarearchitekturen



[Quelle: Effektive Softwarearchitekturen, G. Starke]

# Architektur: Kommunikation - Empfehlungen

- Aktiv **Rückmeldungen** von Stakeholdern **einholen**:
  - Das hilft, Defizite in Architekturkommunikation frühzeitig zu erkennen
- **Sichten** zur getrennten Beschreibung unterschiedlicher Strukturen verwenden
- **Top-down kommunizieren und dokumentieren**:
  - Mit Vogelperspektiven beginnen und schrittweise Details hinzufügen
- **Vorlagen** für Gliederung Ihrer Dokumentation benutzen

# Architektur: Dokumentation

**Beobachtung: Die Lebensdauer von IT-Systemen übersteigt i.d.R. die initiale Erstellungszeit bei weitem**

**Verständliche, aktuelle, redundanzfreie Dokumentation** ermöglicht auch über einen langen Zeitraum:

- Überblick zu wahren
- Auftretende Probleme und Fehler zeitnah zu beseitigen
- Geänderte Anforderungen mit angemessenem Aufwand zu erfüllen
- Auf Änderungen im technischen Umfeld zu reagieren  
(z.B. Änderung von Hardware, Betriebssystemen, Betriebssystemversionen, Middleware, Fremdsystemen, Datenbanken etc.)

# Sichten und Softwarearchitektur

## Begriff Sicht (engl. view):

Eine Sicht ist eine Repräsentation eines Gesamtsystems aus einer festgelegten Perspektive

### • Eigenschaften einer Sicht:

- Beschreibt **nur gewisse Eigenschaften** eines Gesamtsystems (Selektivität)
- Braucht eine **Beschreibungstechnik** (Plan oder Modell)
- Ist meist **nicht vollst. unabhängig** von anderen Sichten (Idealfall: Orthogonalität)
- Sichten sollten parallel bearbeitet werden können (Schnittstellenproblematik)
- Zwei Sichten eines Systems sollten sich **nicht widersprechen** (Konsistenz)
- Summe aller Sichten sollte in eine Gesamtsicht münden

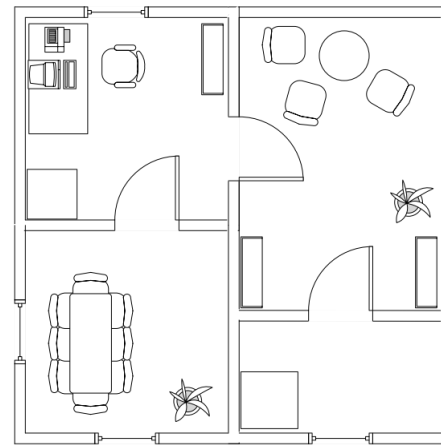
### • Beispiel-Ansätze:

- Das “4+1-Sichtenmodell” von Kruchten
- Die 4 Sichten von Starke
- Die 4 Sichten von Hofmeister

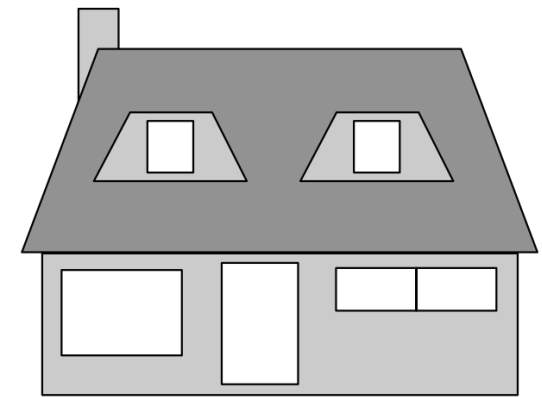
# Sichtenkonzept: Analogie zum Hausbau

Plan/Sicht	Bedeutung	Format	Nutzer
Grund- und Aufriss	Lage und Beschaffenheit von Mauern, Maueröffnungen (Türen, Fenstern, Durchgängen), Böden, Decken	Normiert nach DIN	Maurer, Käufer
Elektroplan	Lage von spannungsführenden Leitungen, Schaltern, Steckdosen, Verteilern, Sicherungen sowie sonstiger Elektroinstallation	Normiert nach DIN	Architekt, Käufer, Elektriker, Küchenbauer, Verwaltung (wegen Stromversorgung)
Heizungs-, Wasser- und Sanitärplan	Lage von Wasser- und Abwasserleitungen, Heizungsrohren sowie Gasleitungen	Normiert nach DIN	Architekt, Heizungs- und Sanitärinstallateur, Käufer, Küchenbauer, Verwaltung (wegen Abwasseranschluss)
3D-Modell	Dreidimensionale Darstellung des Gebäudes im Ganzen oder in Teilen	Beliebig, Bilder oder Filme („virtuelle Begehung“)	Käufer, Verkäufer
Raumplan	Zweidimensionale Darstellung von Zimmern und Einrichtung	Beliebig, angelehnt an DIN	Käufer, Innenarchitekt, Küchenbauer

# Sichtenkonzept: Beispiel Hausbau



**Raumplan  
(OG)**

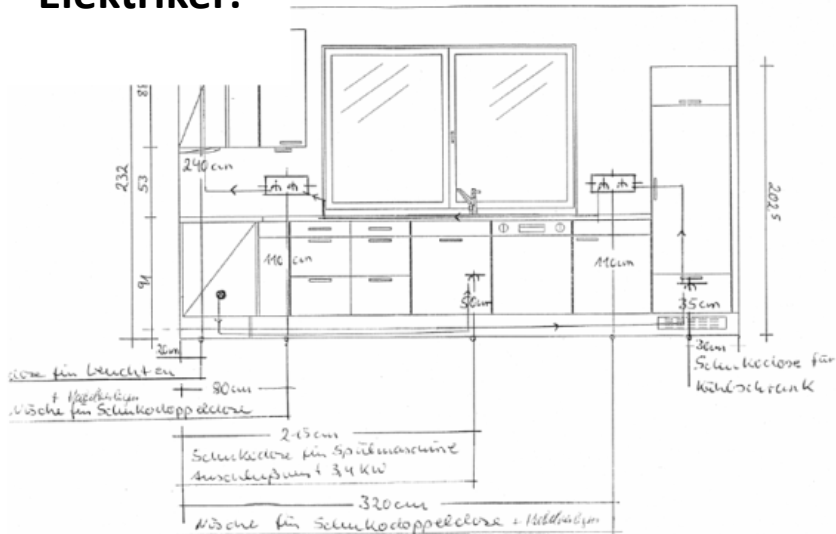


**2D-  
Vorderansicht**

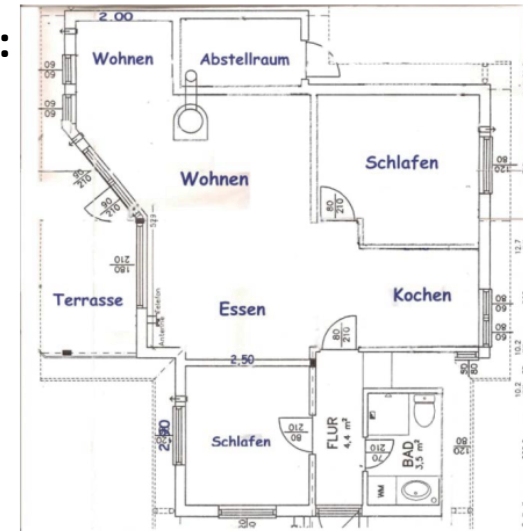
[Quelle: Effektive Software-Architekturen, G. Starke]

# Sichtenkonzept: Beispiel Hausbau

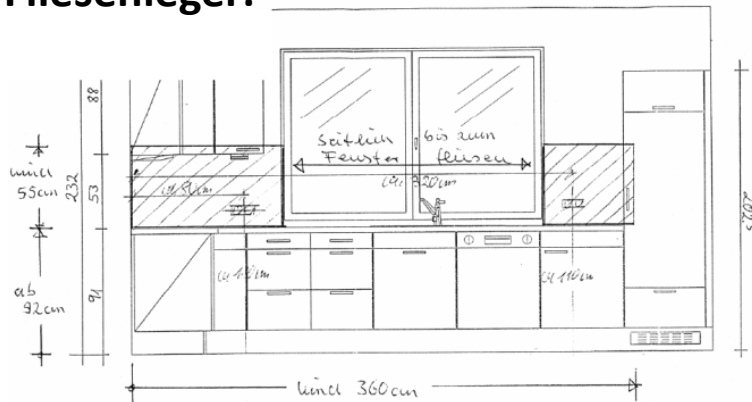
## Elektriker:



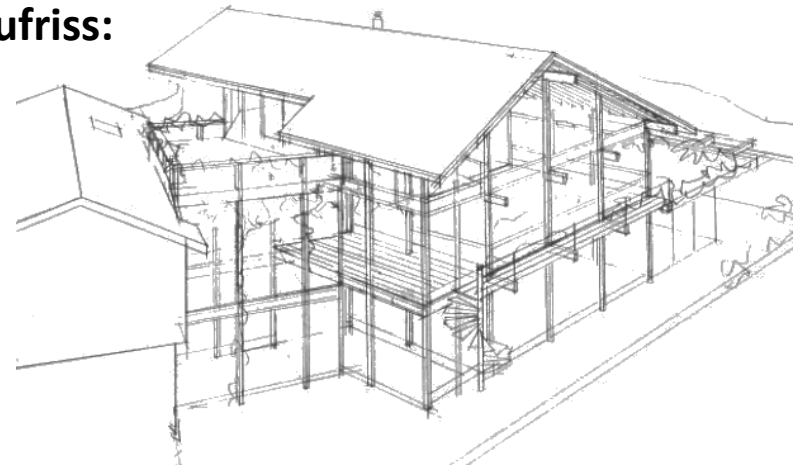
## Grundriss:



## Fliesenleger:



## Aufriss:



# Sichten in der Softwarearchitektur

- **Problem:**

- Softwarearchitekturen i.d.R. **komplex**
- **Einzelne Darstellung** kann Vielschichtigkeit und **Komplexität nicht ausdrücken**
- Architekturbeschreibung für verschiedene Stakeholder mit **unterschiedlichsten Informationsbedürfnissen** wichtig

- **Beispiel:**

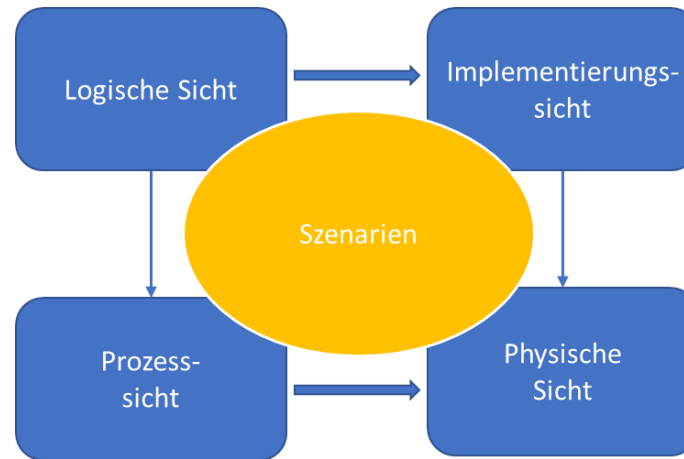
- Auftraggeber oder Projektmanager benötigen andere Informationen als Entwickler, Qualitätsmanagement oder Betreiber der Software

- **Lösung:**     Sichten

- Unterschiedliche Sichten ermöglichen Konzentration auf das jeweils Wesentliche
- Reduzieren Darstellungskomplexität (s. Beispiel Hausbau)

Welche Sichten fallen Ihnen ein, die bei der Architektur eines SW-Systems nützlich sind?



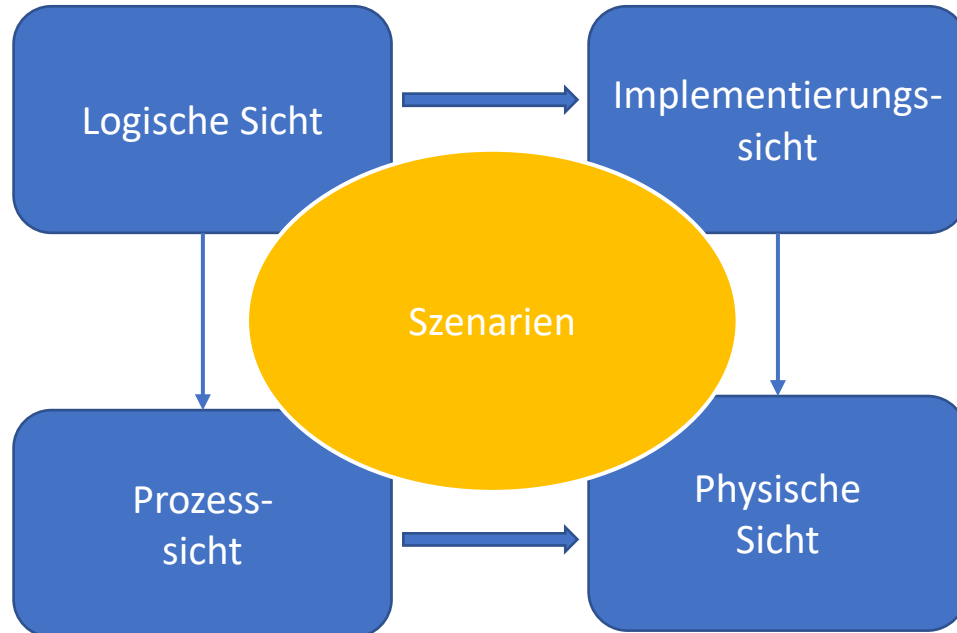


# 4+1 Sichten nach Kruchten

# Softwarearchitektur: die 4+1 Sichten [PK95]

## Es gibt keine allumfassende Architekturdarstellung!

- Aufteilung der Architekturdarstellung in unterschiedliche Sichten nach P. Kruchten:



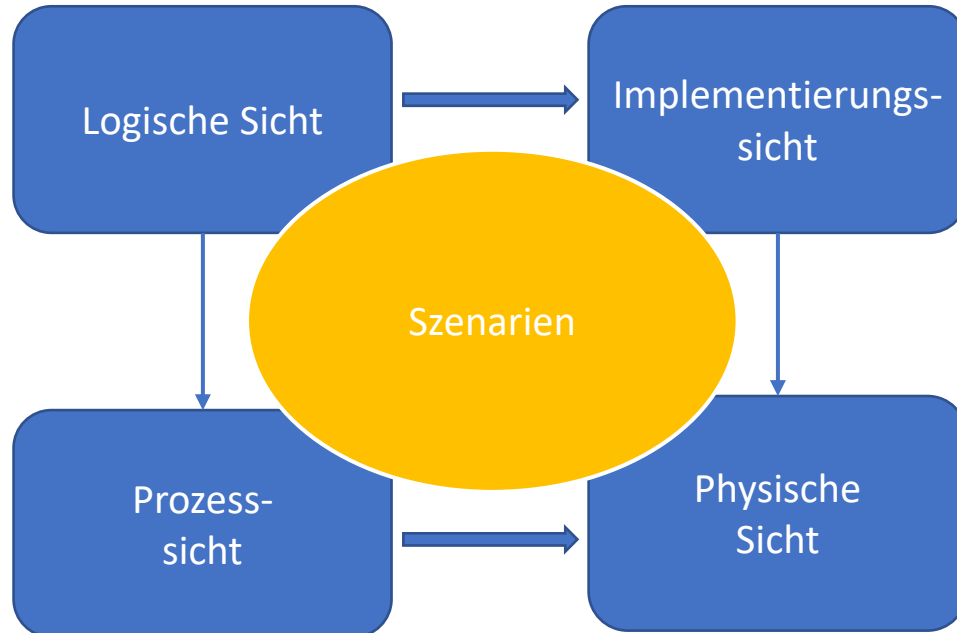
### Logische Sicht:

- **Adressat:** Endanwender
- **Fokus:**
  - Funktionalität für Endanwender
  - Funktionale Anforderungen
  - Welche Dienste werden dem Nutzer vom System bereitgestellt
- **Hilfsmittel u.a.:**
  - Klassendiagramme
  - Sequenzdiagramme
  - Kommunikationsdiagramme

# Softwarearchitektur: die 4+1 Sichten [PK95]

## Es gibt keine allumfassende Architekturdarstellung!

- Aufteilung der Architekturdarstellung in unterschiedliche Sichten nach P. Kruchten:



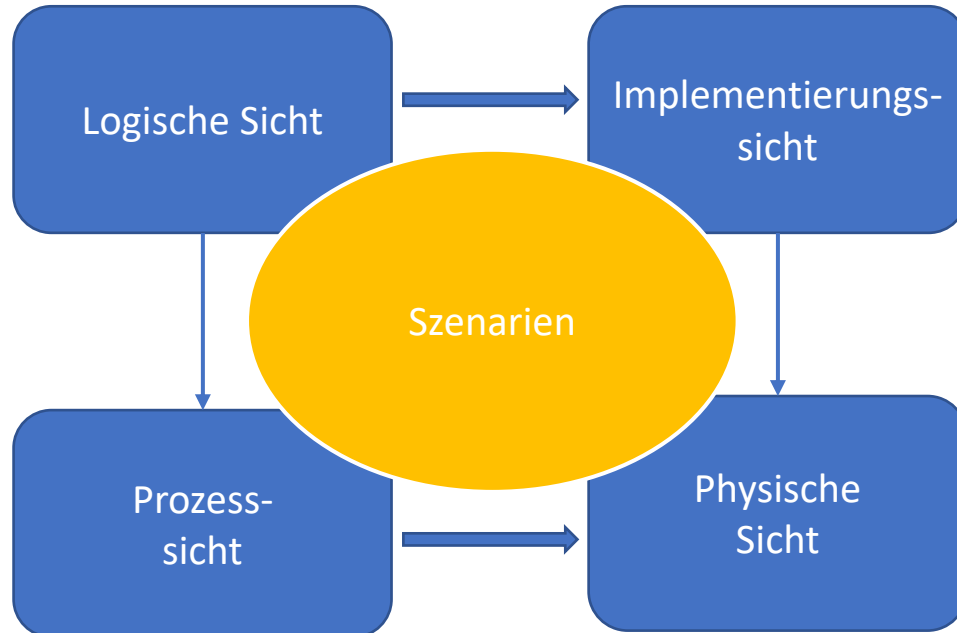
## Implementierungssicht:

- **Adressat:** Entwickler, SW-Manager
- **Fokus:**
  - Systembeschreibung aus Entwicklersicht
  - Statische Organisation der SW
  - Software-Management
- **Hilfsmittel:**
  - Komponentendiagramm oder
  - Paketdiagramm

# Softwarearchitektur: die 4+1 Sichten [PK95]

## Es gibt keine allumfassende Architekturdarstellung!

- Aufteilung der Architekturdarstellung in unterschiedliche Sichten nach P. Kruchten:



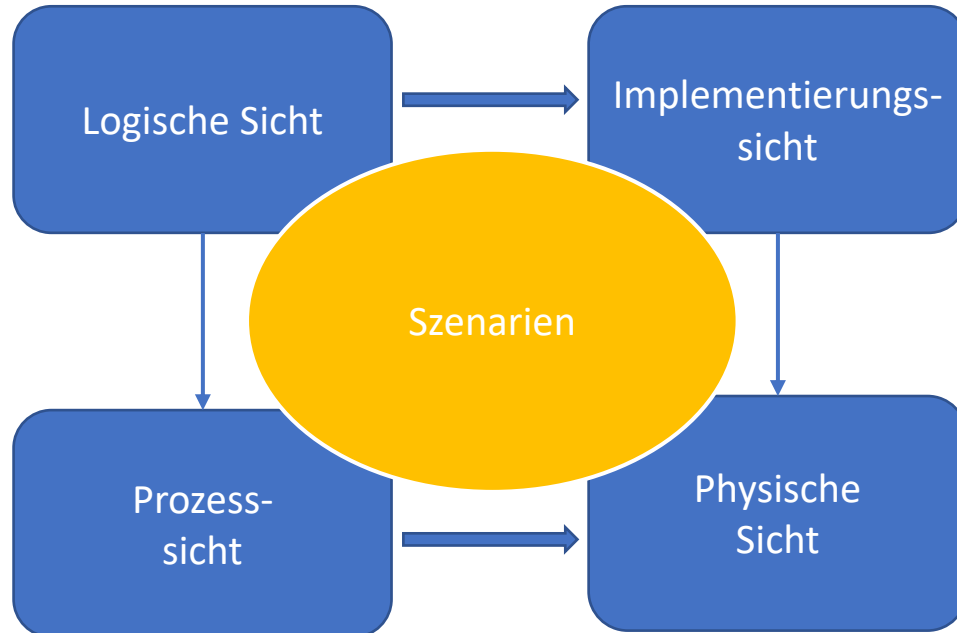
### Prozesssicht:

- **Adressat:** Integrierer
- **Fokus:**
  - Dynamische Aspekte des Systems
  - Nicht-funktionale Anforderungen (Skalierbarkeit, Parallelität, Verteilung, Performanz)
- **Hilfsmittel:**
  - Aktivitätsdiagramm

# Softwarearchitektur: die 4+1 Sichten [PK95]

## Es gibt keine allumfassende Architekturdarstellung!

- Aufteilung der Architekturdarstellung in unterschiedliche Sichten nach P. Kruchten:



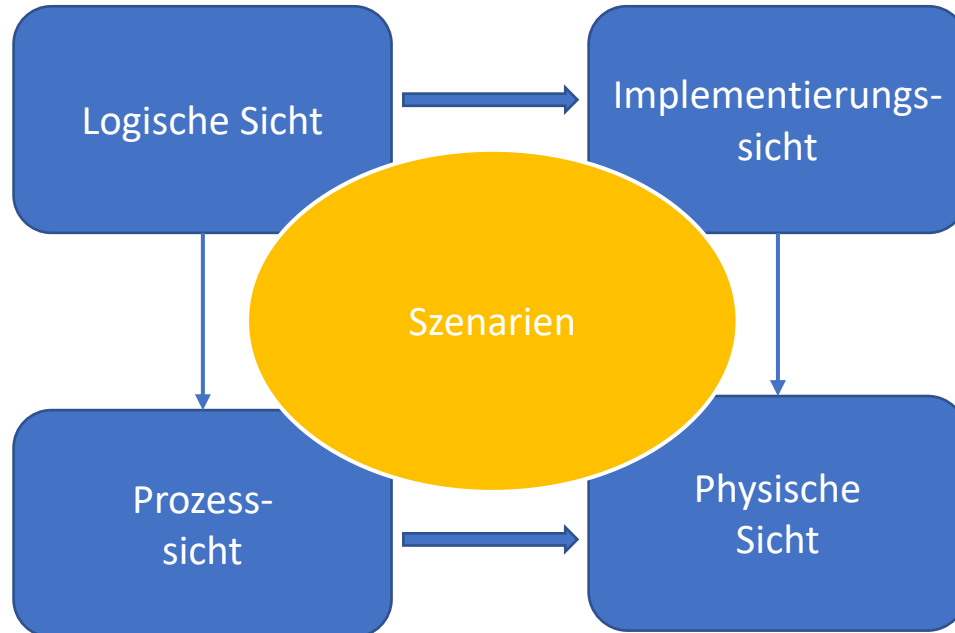
## Physische Sicht:

- **Adressat:** System-Engineers
- **Fokus:**
  - Nicht-funktionale Anforderungen mit Hinblick auf Hardware (Zuverlässigkeit, Erreichbarkeit, Performanz)
  - Verteilung und Kommunikation der HW-Komponenten (physische Ebene)
- **Hilfsmittel:**
  - Verteilungsdiagramm

# Softwarearchitektur: die 4+1 Sichten [PK95]

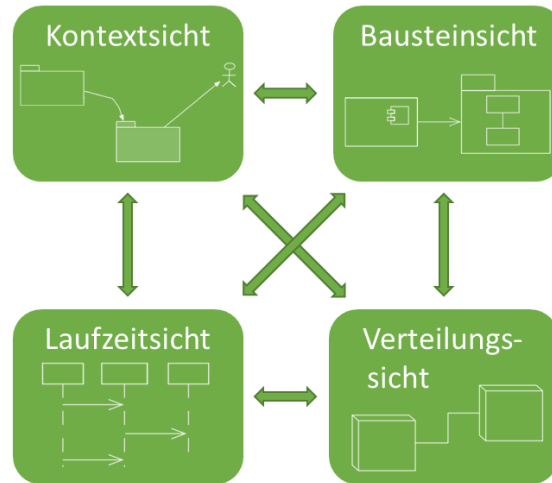
## Es gibt keine allumfassende Architekturdarstellung!

- Aufteilung der Architekturdarstellung in unterschiedliche Sichten nach P. Kruchten:



## Szenarien: (wichtige Anw.fälle)

- **Adressat:** alle Stakeholder
- **Fokus:**
  - Systemkonsistenz
  - Ablaufbeschreibung zwischen Komponenten
  - Architektur überprüfen
- **Hilfsmittel:**
  - Use-Case-Diagramm
  - Sequenzdiagramm



# 4 Sichten nach Starke

# 4 Sichten nach Starke

## 1. Kontextabgrenzung

- Einbettung des Systems in seine Umgebung (Nachbarsysteme, Stakeholder, Infrastruktur)
- System als Blackbox
- Sehr abstrahiert

## 2. Bausteinsichten

- Aufbau des Systems aus Subsystemen, Komponenten, Teilpaketen, Frameworks, Konfigurationen, ...
- Zusammenwirken der Bausteine (Schnittstellen)
- Top-Down mit Blackboxen und Whiteboxen

## 3. Laufzeitsichten

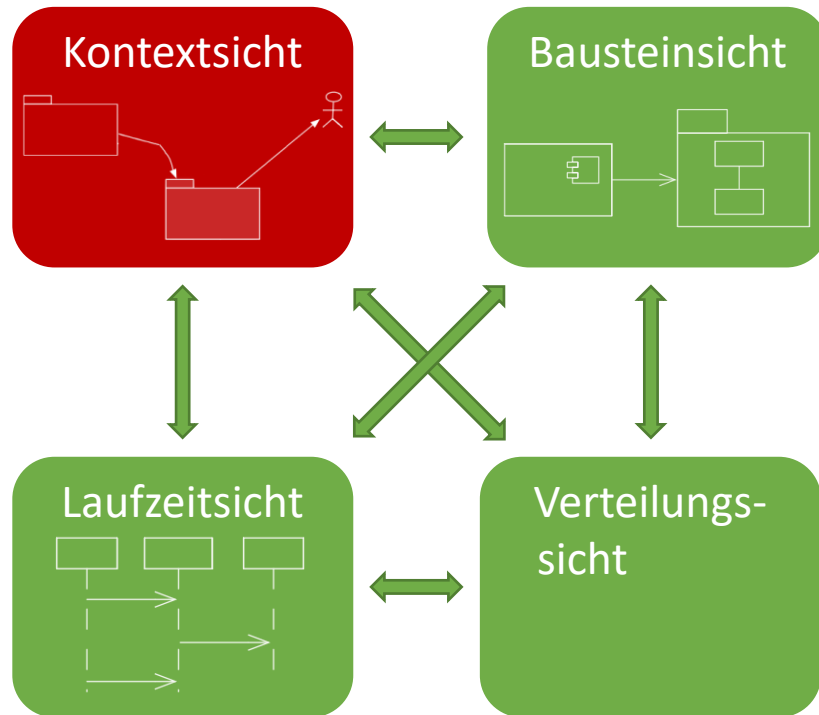
- Dynamische Struktur
- Interaktion von Laufzeitinstanzen

## 4. Verteilungssichten (Infrastruktur-)

- Technische Ablaufumgebung
- Hardwarekomponenten und ihr Zusammenspiel
- Deployment-Einheiten



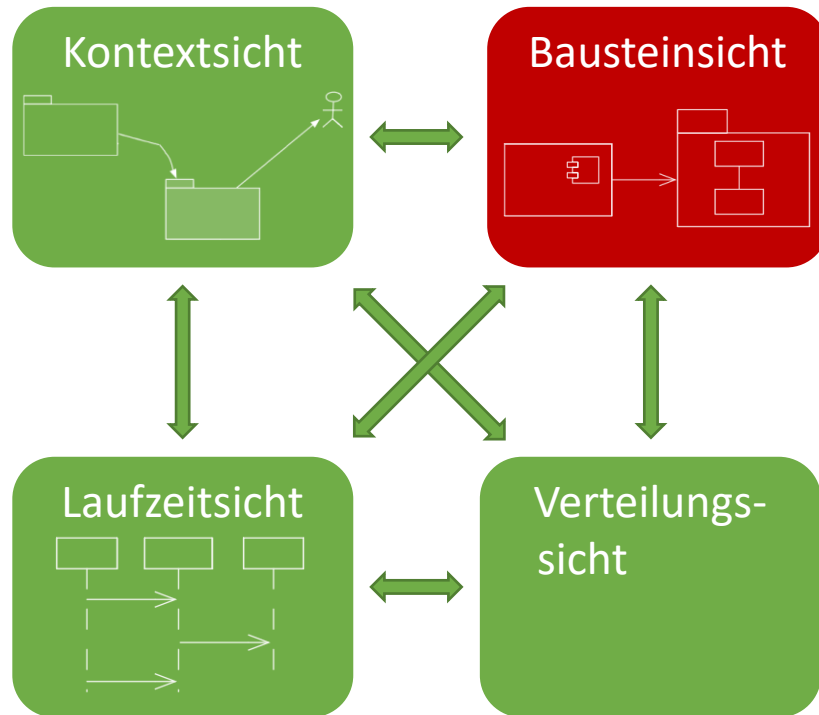
# 4 Sichten nach Starke



## Kontextsicht:

- **Klärt die Frage:**
  - Wie ist System in Umgebung eingebettet?
- **Beschreibt:**
  - System als Black-Box (Außenansicht)
  - Aus Vogelperspektive
  - Schnittstellen zu Nachbarsystemen
  - Interaktion mit wichtigen Stakeholdern
  - Wesentliche Teile der Infrastruktur
- **Zweck:**
  - Abstrakte Beschreibung (Vision) des Systems

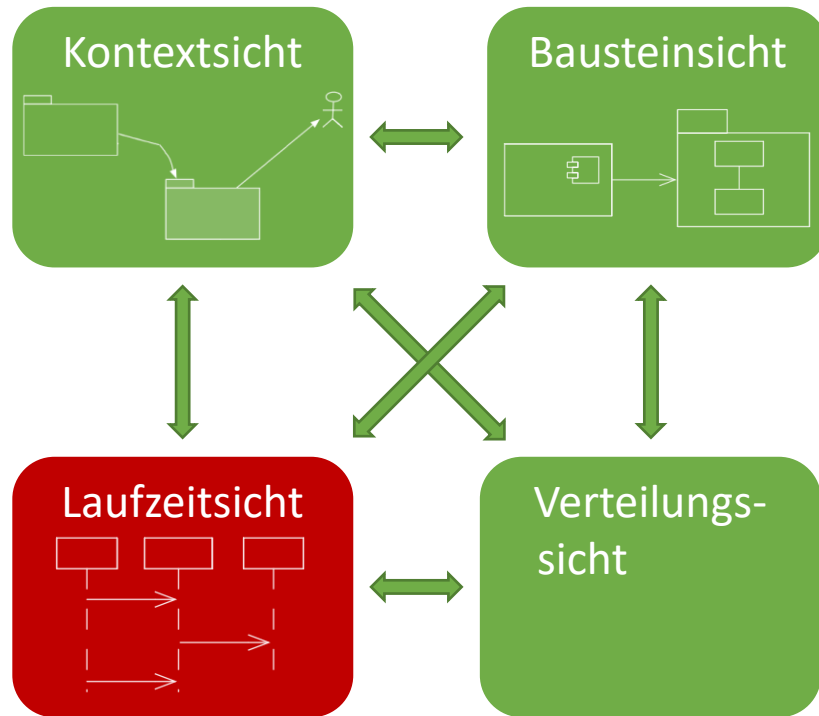
# 4 Sichten nach Starke



## Bausteinsicht:

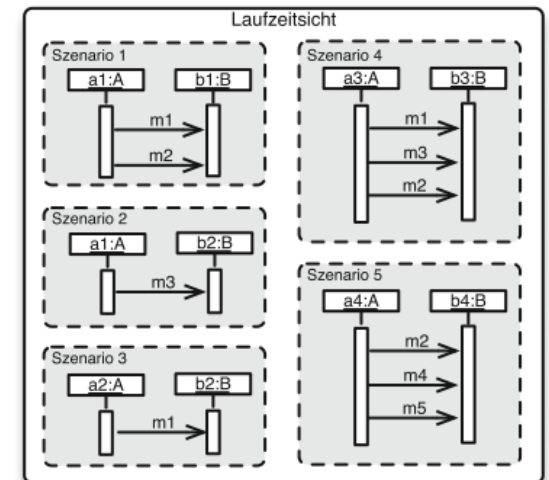
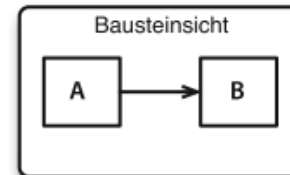
- **Klärt die Frage:**
  - Wie ist das System intern aufgebaut?
- **Beschreibt:**
  - Statische Strukturen des Systems
  - Subsysteme, Module, Pakete, Komponenten, Klassen etc.
- **Zweck:**
  - Unterstützen Projektleiter und Auftraggeber bei Projektüberwachung
  - Dienen Zuteilung von Arbeitspaketen
  - Referenz für Softwareentwickler

# 4 Sichten nach Starke

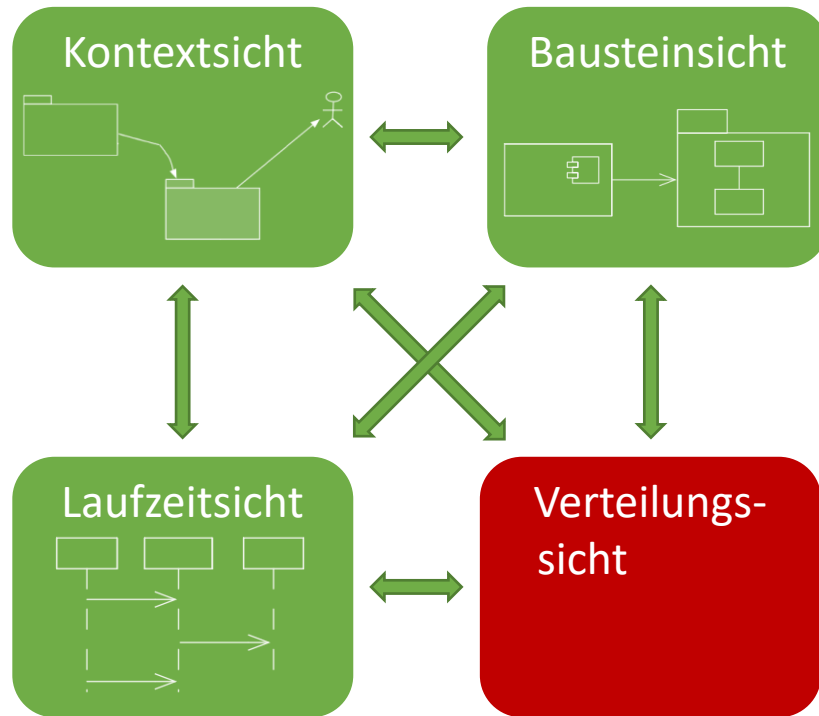


## Laufzeitsicht:

- **Klärt die Frage:**
  - Wie läuft das System ab?
- **Beschreibt:**
  - Welche Bausteine existieren zur Laufzeit
  - Wie interagieren Bausteine miteinander
  - Dynamische Strukturen (im Gegensatz zur statischen Bausteinsicht)



# 4 Sichten nach Starke

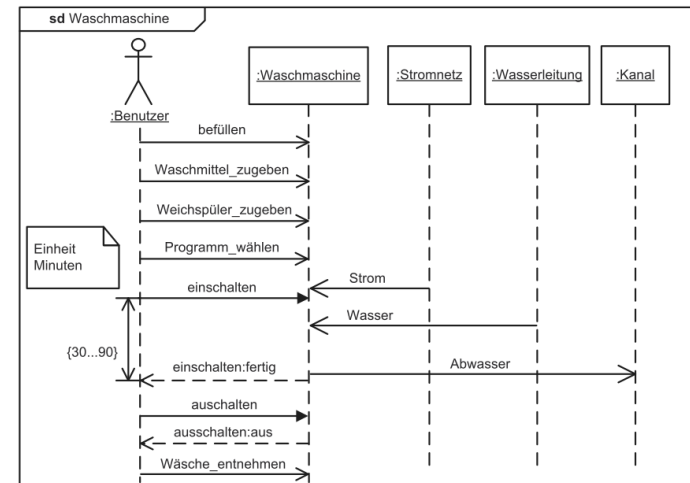
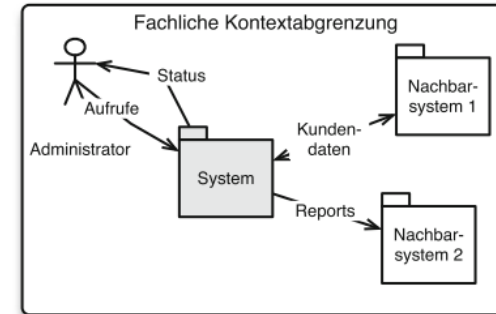


## Verteilungssicht (Infrastruktursicht):

- **Klärt die Frage:**
  - In welcher Umgebung läuft das System ab?
- **Beschreibt:**
  - Hardwarekomponenten
  - Rechner, Prozessoren, Speicher, Netztopologie
  - Sonstige Bestandteile der physischen Systemumgebung

# 4 Sichten nach Starke

- **Kontextsicht**
- Bausteinsicht
- Laufzeitsicht
- Verteilungssicht



Architekturbeschreibung (Gliederungsvorschlag)  
siehe G.R.I.P.S. (Quelle: [www.arc42.com](http://www.arc42.com))

# Kontextsicht

- **Kontextsicht** zeigt **Umfeld des zu implementierenden Systems** sowie dessen **Zusammenhang mit seiner Umwelt**
  - Kontextsicht als konzeptionelle Übersicht (**Vogelperspektive**) auf das System
- Die meisten Stakeholder nutzen sie als Überblick/**Systemlandkarte**
- Sie erleichtert das **Verständnis der übrigen Architektursichten**
- Im Idealfall erhält man die **Kontextabgrenzung** als ein Ergebnis der **Anforderungsanalyse**

# Kontextsicht

- **Kontextsicht zeigt:**

- System als **Black-Box**
- **Schnittstellen zur Außenwelt** inkl. der darüber transportierten Daten/Ressourcen (z.B. Schnittstellen zu Anwendern, Betreibern und Fremdsystemen)
- Die **wichtigsten Use Cases** (Anwendungsfälle) des gesamten Systems
- Die technische Systemumgebung, Prozessoren, Kommunikationskanäle

Damit ist die Kontextsicht eine Abstraktion der übrigen Sichten mit Fokus auf das Umfeld des Systems

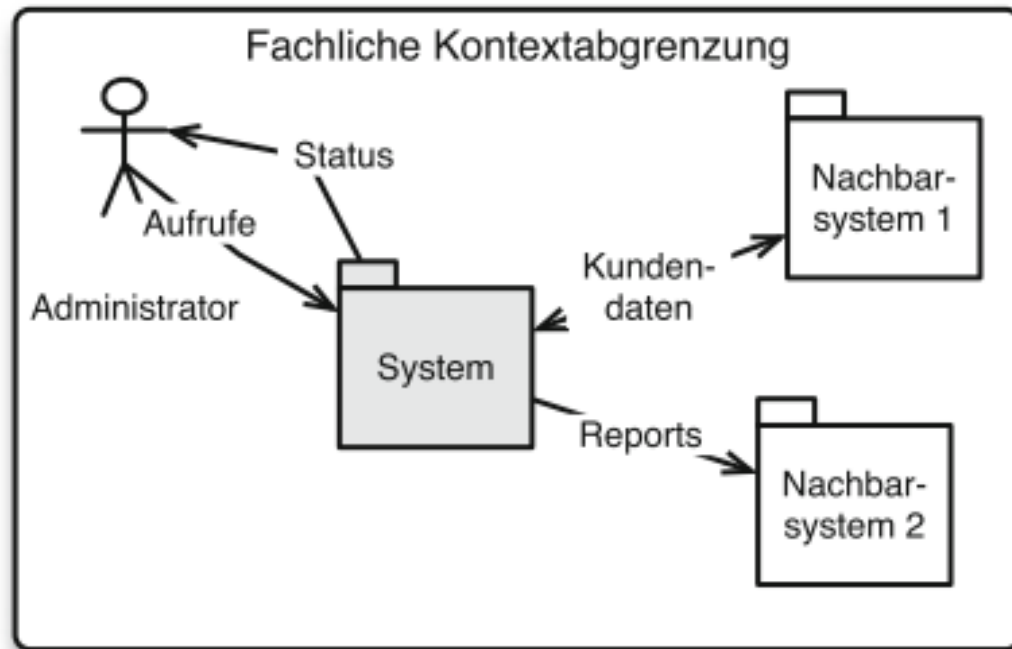
# Kontextsicht: Notation

**Als Abstraktion der übrigen Sichten (Baustein-, Laufzeit- und Verteilungssicht) können deren Notationen verwendet werden:**

- **Darstellung der Systemstruktur:**
  - Klassendiagramme angereichert um Pakete und Komponenten
- **Darstellung von Schnittstellen zur Umwelt:**
  - In Klassendiagrammen über Assoziationen zu Fremdsystemen oder Akteuren
- **Anwendungsfälle oder Abläufe:**
  - Dynamische UML-Diagramme (Sequenz-, Kommunikations-, Aktivitätsdiagramme)
- **Technische Systemumgebung:**
  - Verteilungsdiagramme

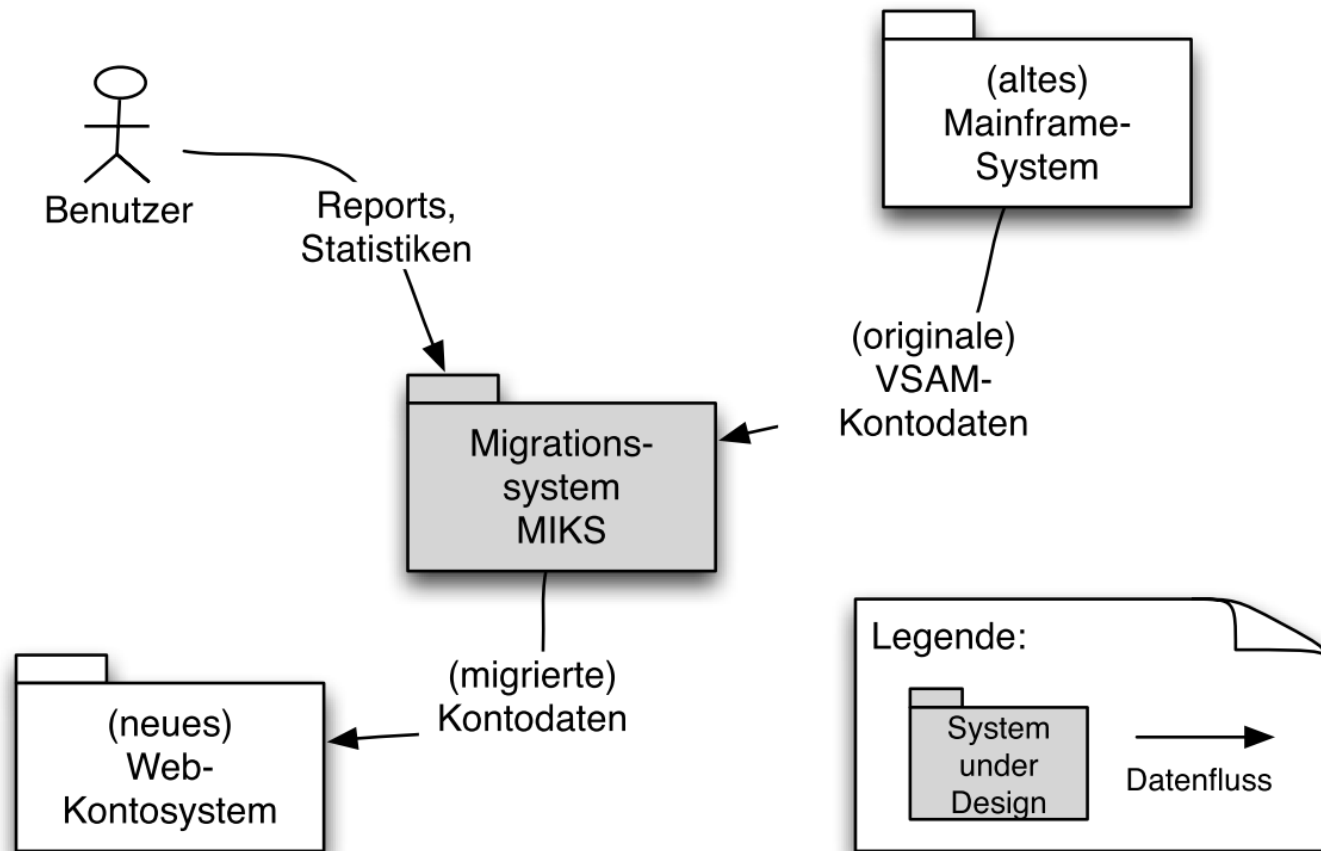


# Kontextsicht: Beispiel



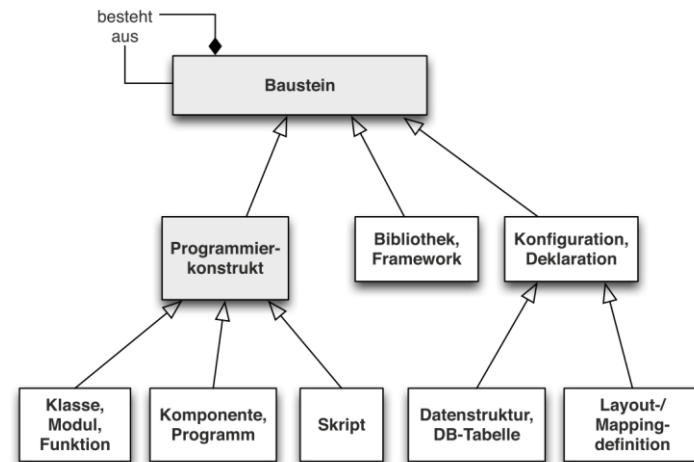
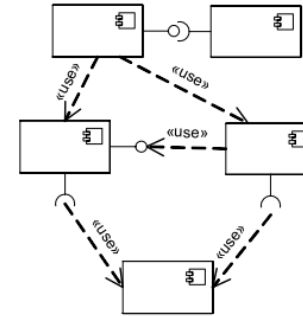
# Kontextsicht: Beispiel

**Beispiel:** Austausch eines Systems durch ein neues



# 4 Sichten nach Starke

- Kontextsicht
- **Bausteinsicht**
- Laufzeitsicht
- Verteilungssicht



Architekturbeschreibung (Gliederungsvorschlag)  
siehe G.R.I.P.S. (Quelle: [www.arc42.com](http://www.arc42.com))

# Bausteinsicht

- Bildet Funktionalität des Systems auf **Software-Bausteine**/Komponenten ab
- Stellt Struktur und **Zusammenhänge zwischen Bausteinen** dar
- Stellt **statische Aspekte** des Systems dar
- Beantwortet folgende Fragen:
  - Wie wird **System in Komponenten, Pakete, Klassen und Subsysteme aufgeteilt?**
  - Welche notwendigen **Abhängigkeiten** zwischen Bausteinen gibt es?

# Bausteinsicht: Elemente

- **Klassensymbole:**

- Bezeichnen einzelne Bausteine
- Beispiele:
  - Klassen einer objektorientierten Programmiersprache
  - Aber auch andere ausführbare Software-Einheiten (z.B. Funktionen, Prozeduren, Programme)

- **Komponenten:**

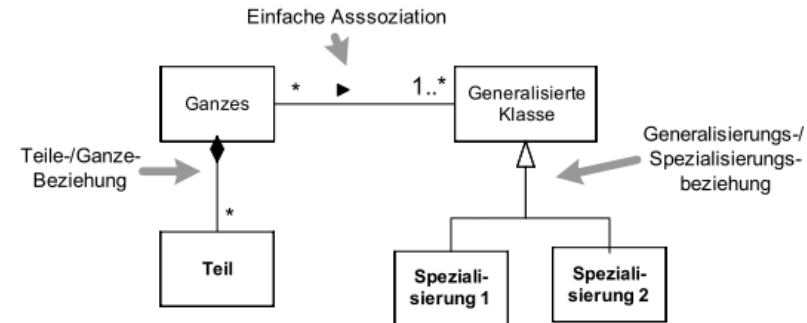
- Bezeichnen einzelne Bausteine
- Bieten im Gegensatz zu Klassensymbolen Möglichkeit, ein- und ausgehende Schnittstellen genau zu spezifizieren

- **Pakete:**

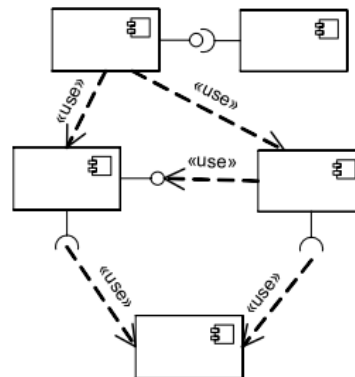
- Beschreiben Gruppen/Mengen/Strukturen von Bausteinen
- Paketsymbole deuten Abstraktion an (die in der Architektur oder im detaillierten Entwurf weiter verfeinert wird)

# Bausteinsicht: Notation

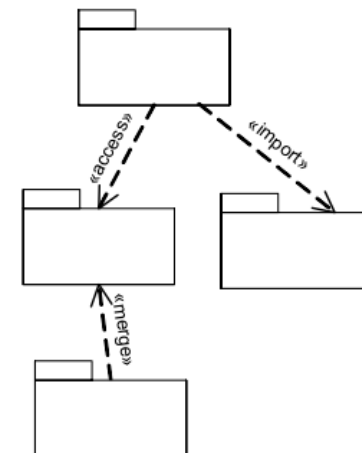
- **Klassendiagramme:**



- **Komponentendiagramme:**



- **Paketdiagramme:**



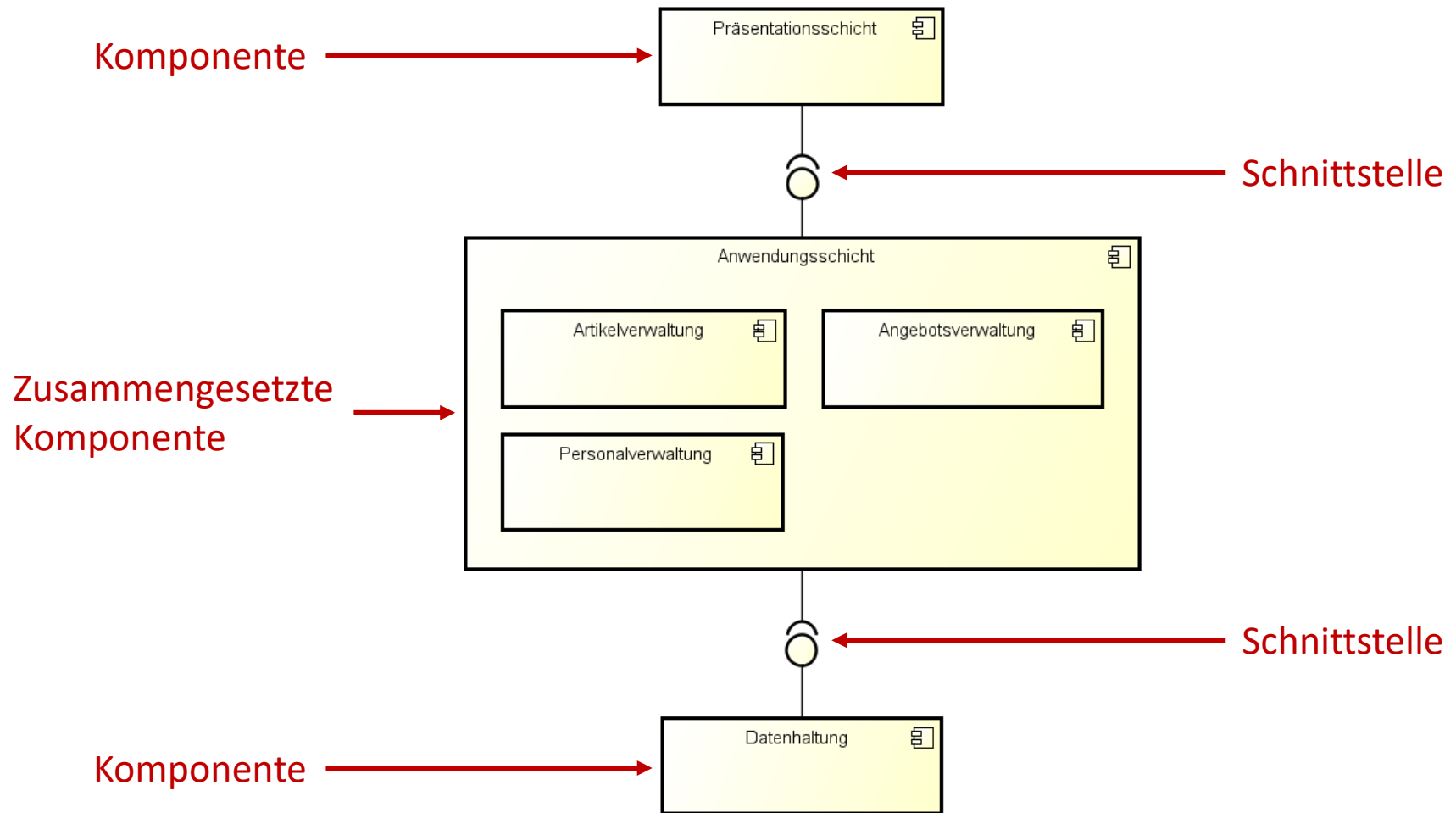
# Bausteinsicht: Adressaten

---

## **Adressaten der Bausteinsicht:**

- Alle an Entwurf, Implementierung und Test von Software beteiligten Projektmitarbeiter
- Qualitätssicherung
- Projektmanagement (zur Definition von Arbeits- und Aktivitätsplänen)

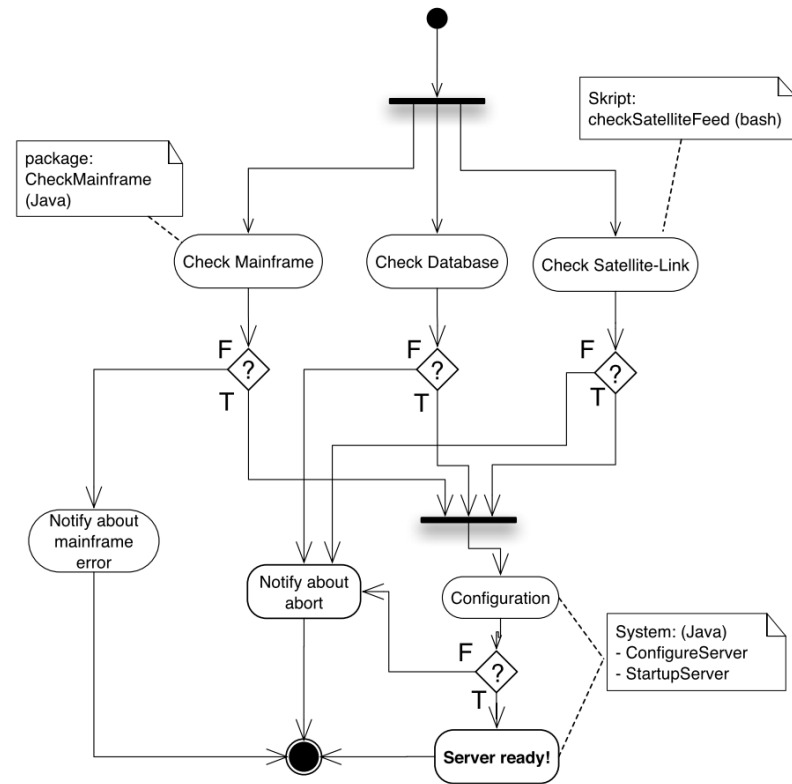
# Bausteinsicht: Beispiel





# 4 Sichten nach Starke

- Kontextsicht
- Bausteinsicht
- **Laufzeitsicht**
- Verteilungssicht



Architekturbeschreibung (Gliederungsvorschlag)  
siehe G.R.I.P.S. (Quelle: [www.arc42.com](http://www.arc42.com))

# Laufzeitsicht

## Die Laufzeitsicht beschreibt:

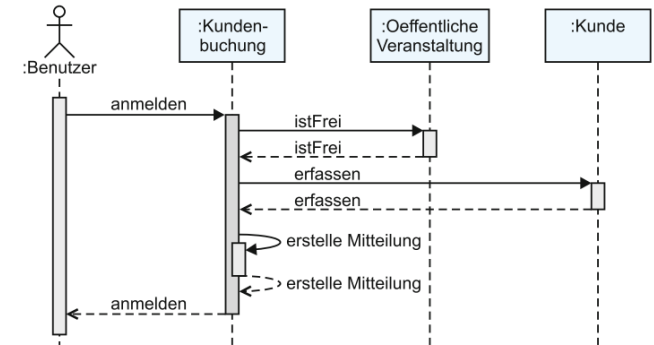
- Welche **Systembestandteile zur Laufzeit** existieren
- Wie diese Systembestandteile **zusammenarbeiten**
- Wie sich Laufzeitkomponenten aus Instanzen von Implementierungsbausteinen **zusammensetzen**

## Hinweis: Suche interessante Laufzeitszenarien, etwa:

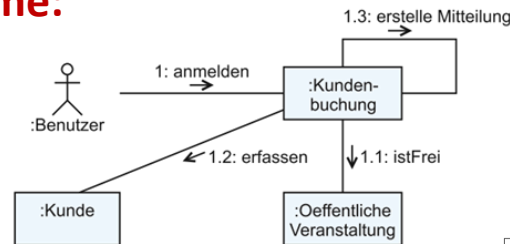
- Welche **Instanzen von Architekturbausteinen** gibt es **zur Laufzeit** und wie werden diese gestartet, überwacht und beendet?
- **Wie startet das System** (z.B.: notwendige Startskripte, Abhängigkeiten von externen Subsystemen, Datenbanken, Kommunikationssystemen etc.)?

# Laufzeitsicht: Notation

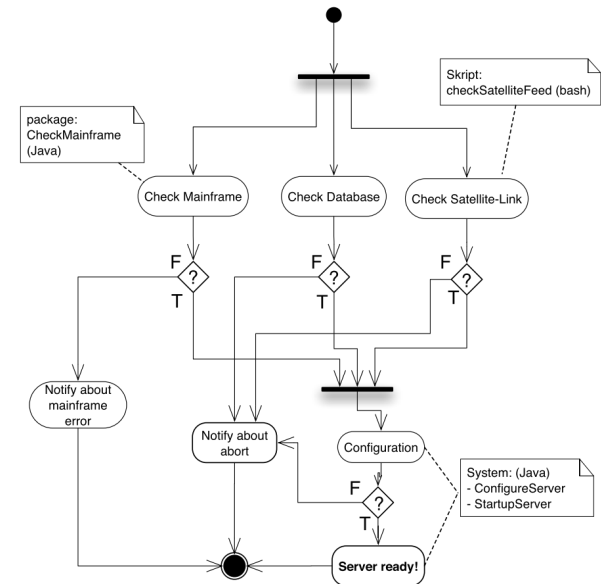
- Sequenzdiagramme:



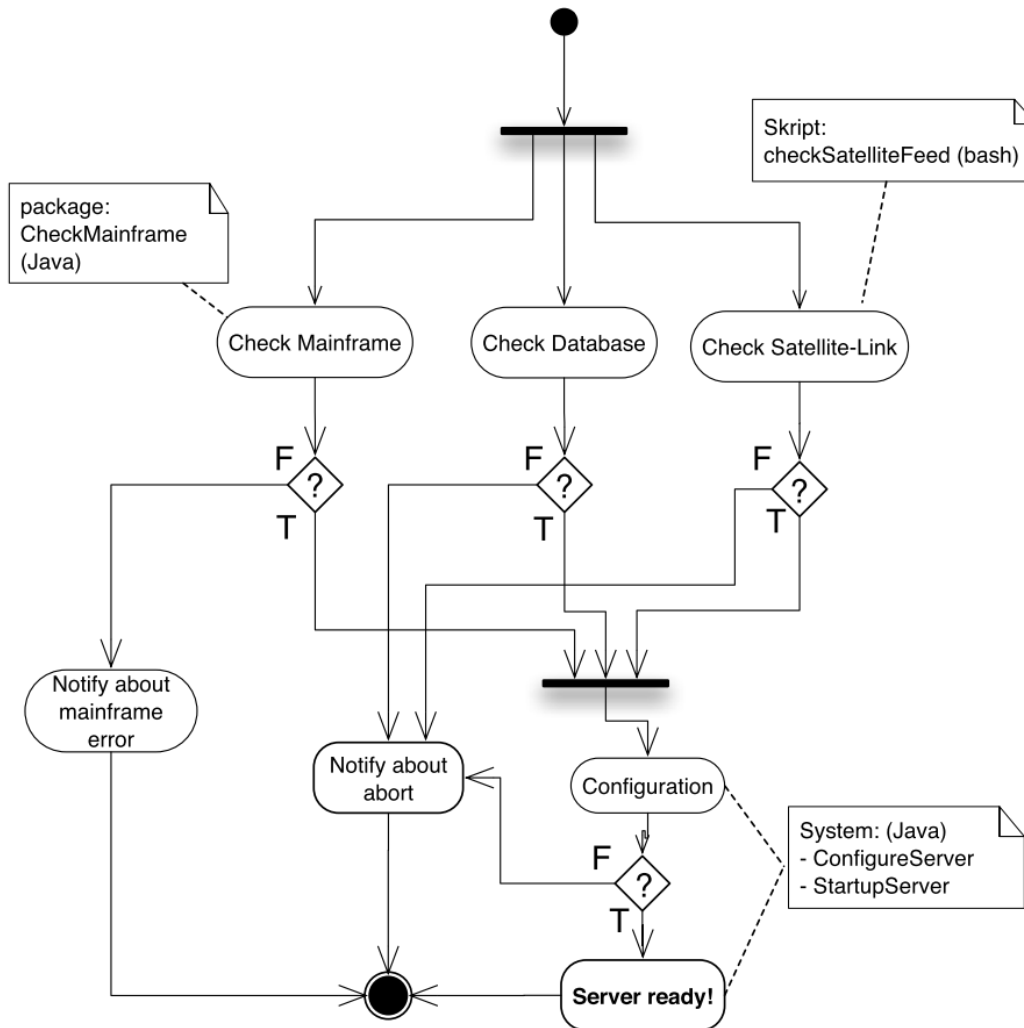
- Kommunikationsdiagramme:



- Aktivitätsdiagramme:

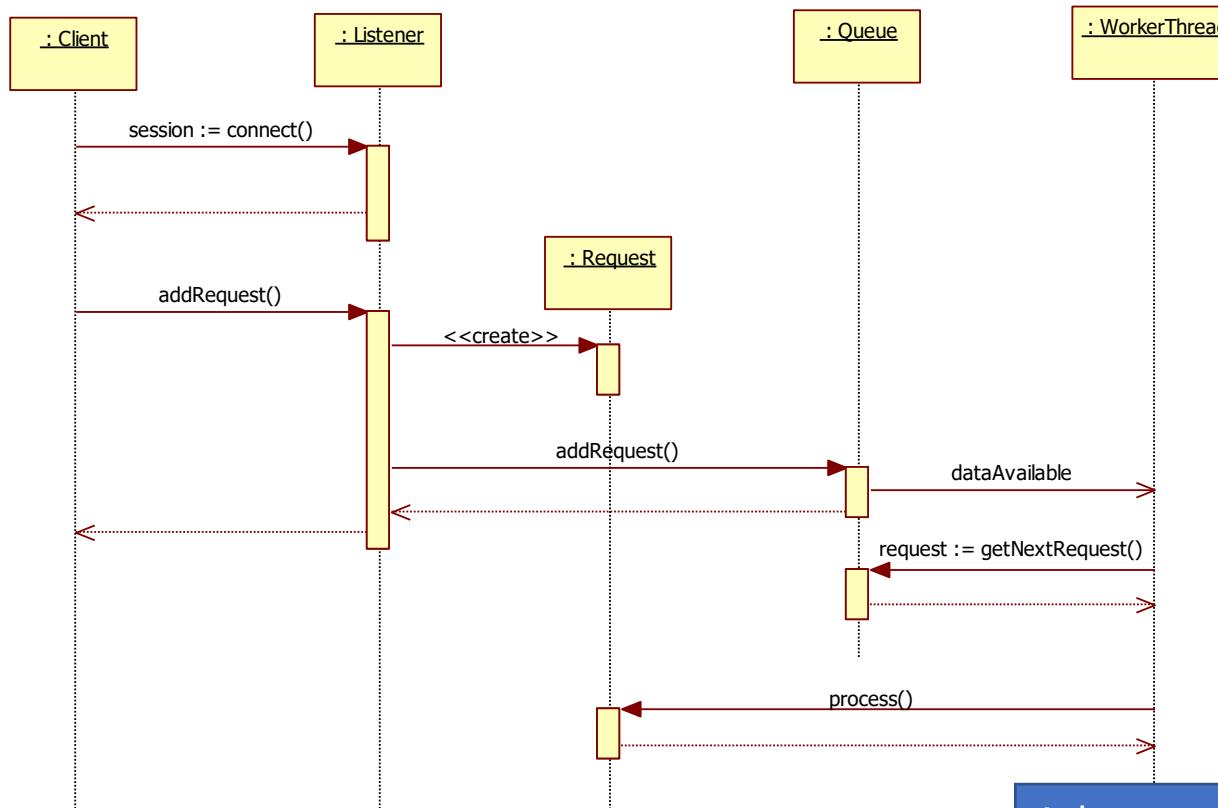


# Laufzeitsicht : Beispiel



Achtung:  
Die Abbildung entspricht nicht  
unseren Konventionen für  
Aktivitätsdiagramme

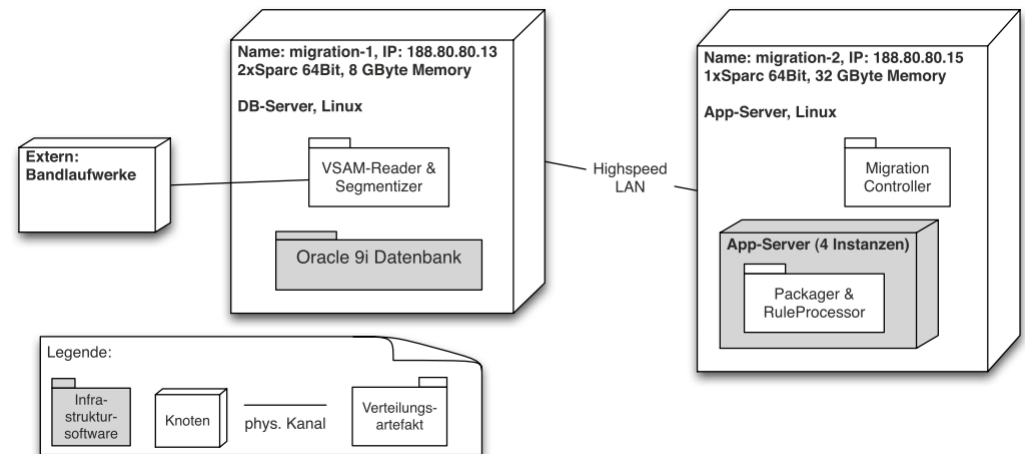
# Laufzeitsicht: Beispiel



Achtung:  
Die Abbildung entspricht nicht  
unseren Konventionen für  
Aktivitätsdiagramme

# 4 Sichten nach Starke

- Kontextsicht
- Bausteinsicht
- Laufzeitsicht
- **Verteilungssicht**



Architekturbeschreibung (Gliederungsvorschlag)  
siehe G.R.I.P.S. (Quelle: [www.arc42.com](http://www.arc42.com))

# Verteilungssicht (Infrastruktursicht)

---

## Die Verteilungssicht beschreibt:

- Welche **Architekturbausteine** laufen **auf** welchen **Hardwarekomponenten**
- Z.B. Computer, Netzwerktopologie, Netzwerkprotokolle und sonstige Bestandteile der physischen Systemumgebung (z.B. Speicher, Router, Firewalls)

Verteilungsschicht besitzt Bedeutung für Betreiber des Systems, Hardware-Architekten, Entwicklungsteam, Management und Projektleitung

# Verteilungssicht: Elemente

- Bestandteile der technische Infrastruktur:

Knoten (z.B. Rechner, Prozessoren, Speicher, Router, Firewalls)

- Laufzeitelemente (Instanzen von Bausteinen), die auf Knoten ablaufen

- Kanäle:

Verbindungen zwischen Knoten (physische Kanäle)

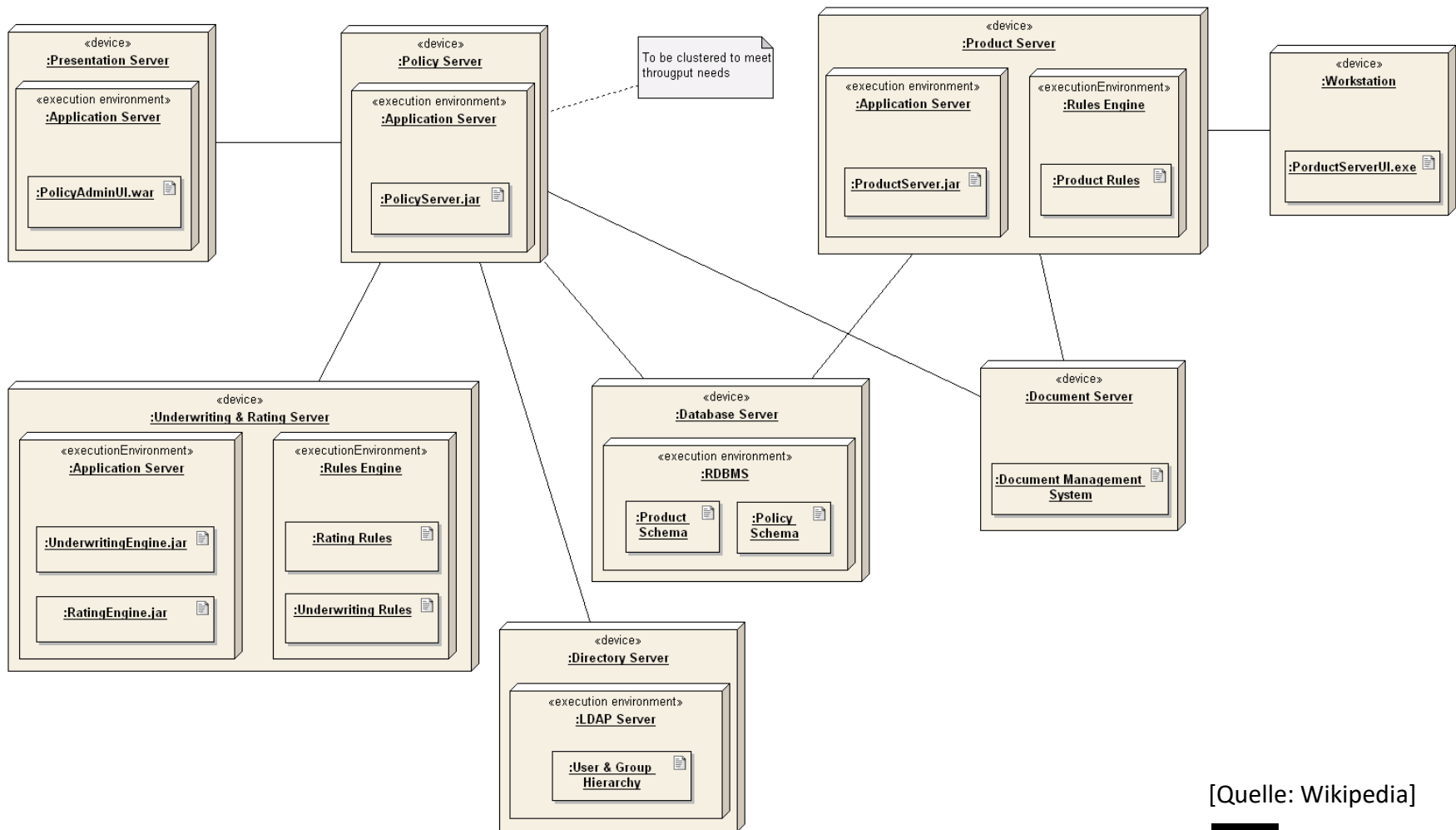
Verbindungen zwischen Laufzeitelementen (logische Kanäle)

(Hinweis: Logische Kanäle müssen immer über physische Kanäle realisiert werden)



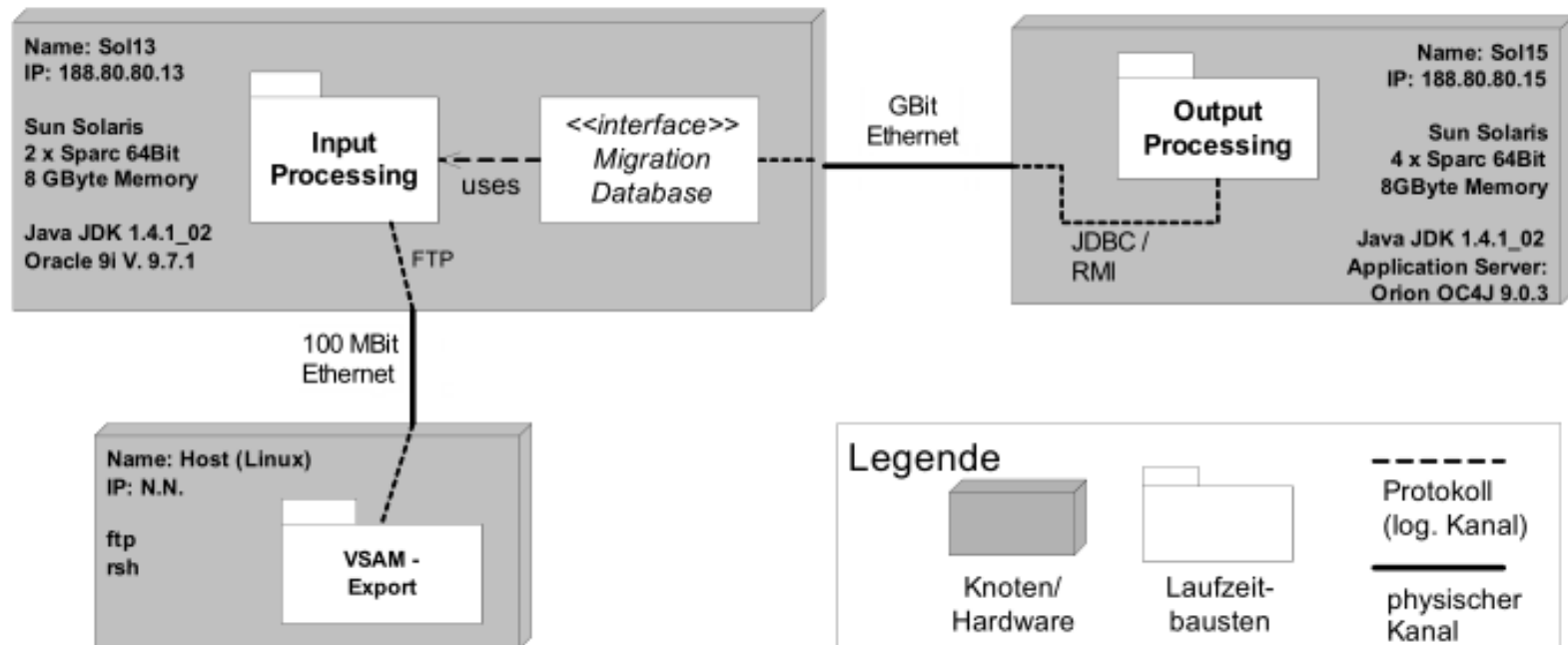
# Verteilungssicht: Notation/Beispiel

## • Deploymentdiagramme (Verteilungsdiagramme)

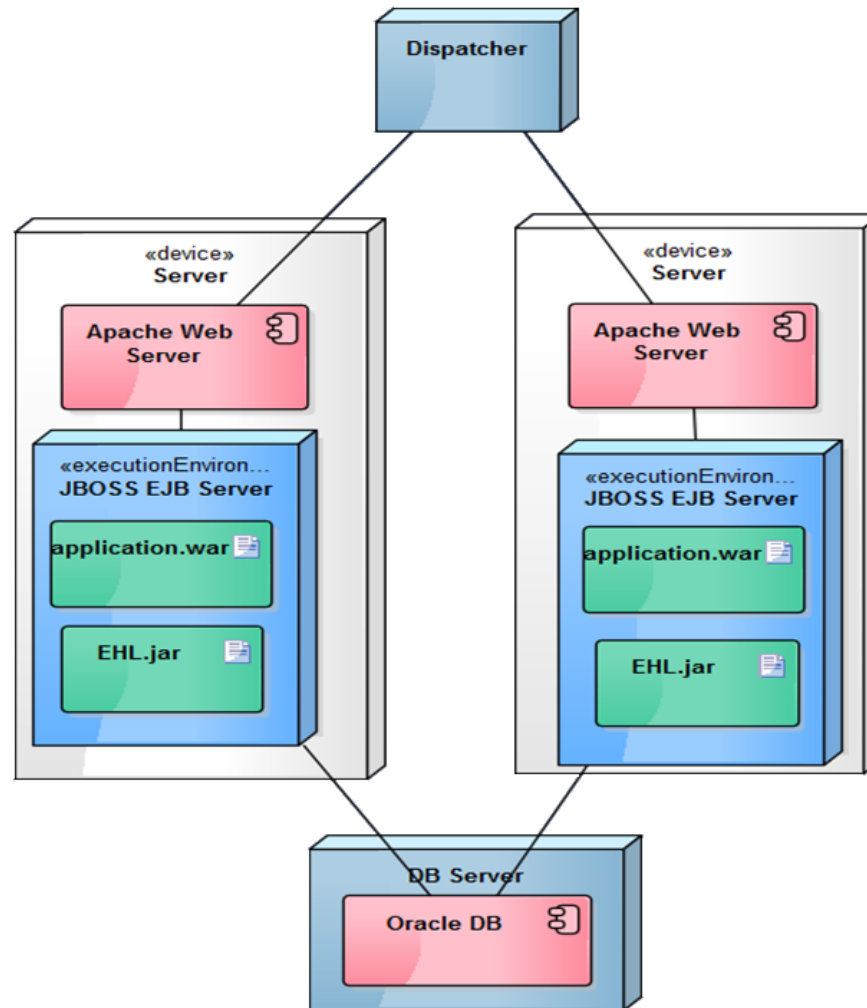


[Quelle: Wikipedia]

# Verteilungssicht: Beispiel



# Verteilungssicht: Beispiel

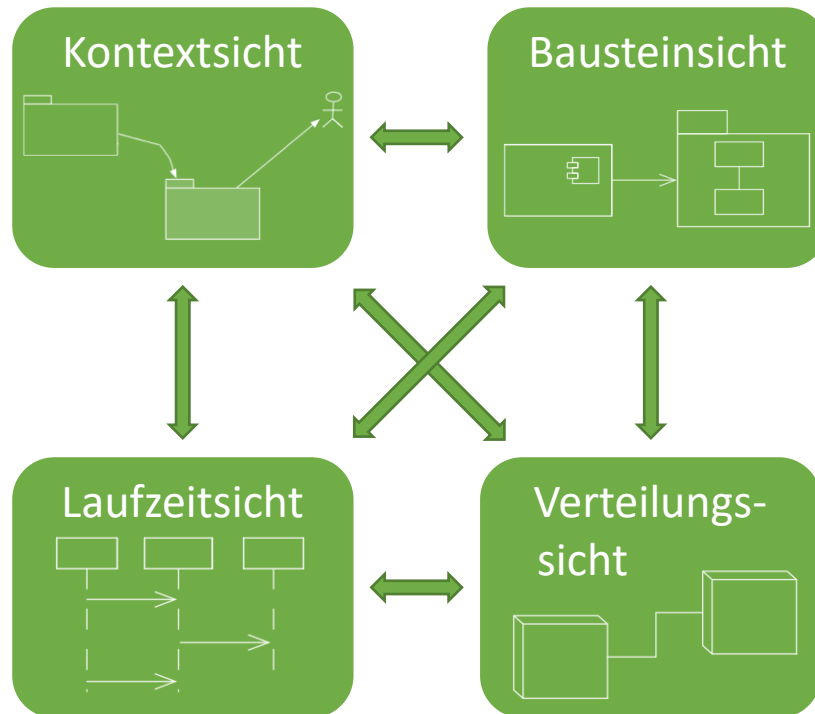


# Weitere Sichten

- Je nach Stakeholder können weitere Sichten gewünscht werden:
  - Szenarien (vgl. “4+1-Sichten” nach Kruchten)
  - Datensicht
  - Sicherheitssicht
  - QS-Sicht
- Vermeide zu viele Sichten (warum?):
  - Alle Sichten müssen gepflegt werden
  - Überschaubarkeit kann leiden
- Nur Sichten, die notwendig sind und zwar in der jeweils benötigten Detailtiefe

# Entwurf von Sichten

- Entwurfsprozess von Sichten ist von deren starken Wechselwirkungen und Abhängigkeiten geprägt  
→ Iteratives Vorgehen



(Abhängigkeiten zwischen Sichten durch Pfeile angedeutet)

# Reihenfolge der Sichten

- **Zunächst:**

- **Kontextsicht**

- **Anschließend:**

- **Bausteinsicht:**

- Wenn Sie bereits ähnliche Systeme entwickelt und genaue Vorstellung von benötigten Implementierungskomponenten haben
    - Wenn Sie ein bereits bestehendes System verändern müssen und damit Teile der Bausteinsicht bereits vorgegeben sind

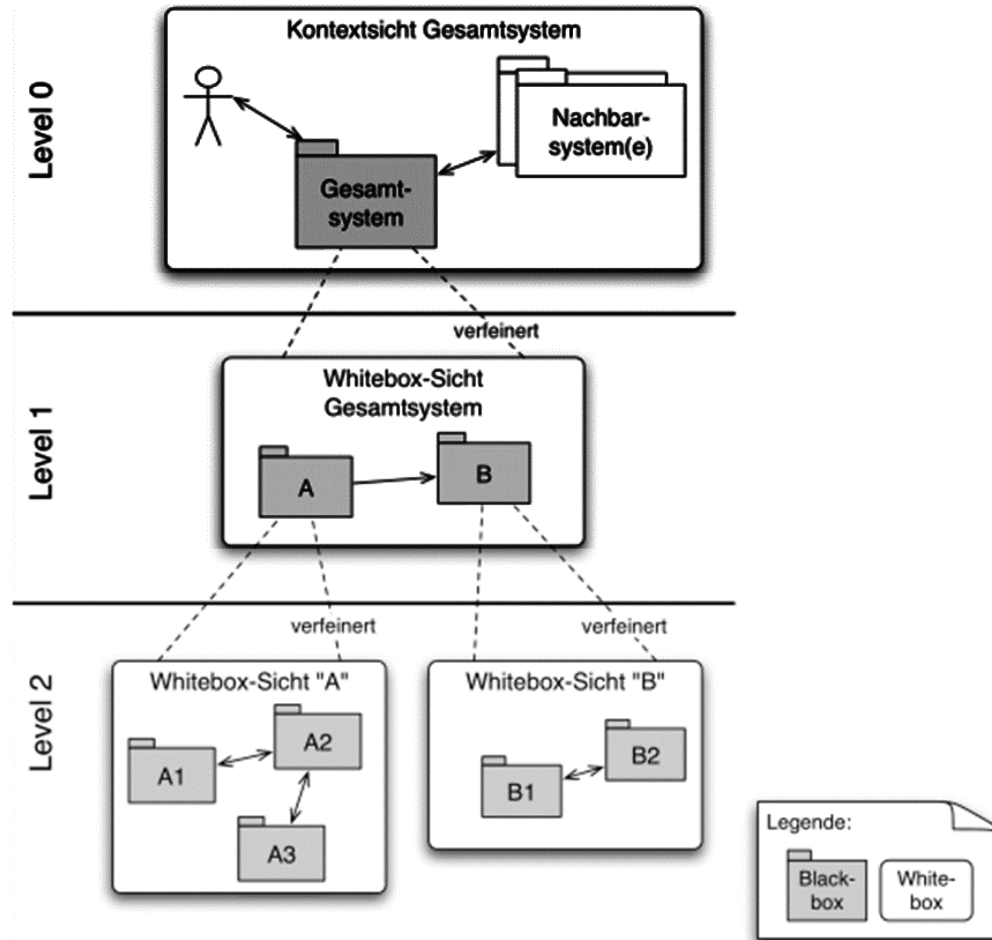
- **Laufzeitsicht:**

- Wenn Sie bereits erste Vorstellungen wesentlicher Architekturbausteine haben und deren Verantwortlichkeiten und Zusammenspiel klären wollen

- **Verteilungssicht:**

- Wenn Sie viele Randbedingungen und Vorgaben durch die technische Infrastruktur, Rechenzentrum oder Administratoren des Systems bekommen

# Iterative Verfeinerung der Bausteinsicht



# Architekturbeschreibung: Gliederungsvorschlag

- [arc42 Template](http://www.arc42.de) (Quelle: [www.arc42.de](http://www.arc42.de))


## 2.3 Konventionen

## 3. Kontextabgrenzung

Die folgenden Unterkapitel zeigen die Einbettung unseres Systems in seine Umgebung.

### 3.1 Fachlicher Kontext

### 3.2 Technischer- oder Verteilungskontext

### 3.3 Externe Schnittstellen

#### 3.3.1 Externe Schnittstelle 1

Identifikation der Schnittstelle

Name / Bezeichnung der Schnittstelle	<Name der Schnittstelle>
Version	
Änderungen gegenüber Vorversion	
Wer hat geändert und warum?	
Verantwortlicher Ansprechpartner / Rolle	

<sup>1</sup> Zwar sind wir an vielen Stellen zu Pragmatismus bereit – hier jedoch bestehen wir auf der vollständigen Auflistung aller (a-I-I-e-r) Nachbarsysteme. Zu viele Projekte sind daran gescheitert, dass sie ihre Nachbarn nicht kannten ☺

### Fachlicher Kontext der Schnittstelle

### Fachliche Abläufe

<Diagramm oder Beschreibung der fachlichen Abläufe>

### Fachliche Bedeutung der Daten

<Beschreibung der fachlichen Bedeutung>

Technischer Kontext

Form der Interaktion

### Anforderungen an die Schnittstelle

### Sicherheitsanforderungen

### Mengengerüste

Laufzeit

Durchsatz / Datenvolumen

Verfügbarkeit

Protokollierung

Archivierung

### Beteiligte Ressourcen

### Syntax: Daten und Formate

Datenformate

Gültigkeits- und Plausibilitätsregeln

Codierung, Zeichensätze

Konfigurationsdaten

### Syntax: Methoden/Funktionen

Prüfdaten