

# Architektur von Informationssystemen

Hochschule für angewandte Wissenschaften

Sommersemester 2016

Nils Löwe / [nils@loewe.io](mailto:nils@loewe.io) / @NilsLoewe

Was ist Softwarearchitektur?

Geschichte und Trends

## Sichten auf Architekturen

Qualität und andere nichtfunktionale Anforderungen

Architekturmuster

Dokumentation von Architekturen

Technologien und Frameworks

# Wiederholung

## Sichten auf Architekturen

# Warum überhaupt Sichten?

*"Es ist eine offensichtliche Wahrheit, dass auch eine perfekte Architektur nutzlos bleibt, wenn sie nicht verstanden wird..."*

Felix Bachmann und Len Bass in "Software Architecture  
Documentation in Practice"

# 1.

Eine einzelne Darstellung kann die Vielschichtigkeit und Komplexität einer Architektur nicht ausdrücken.

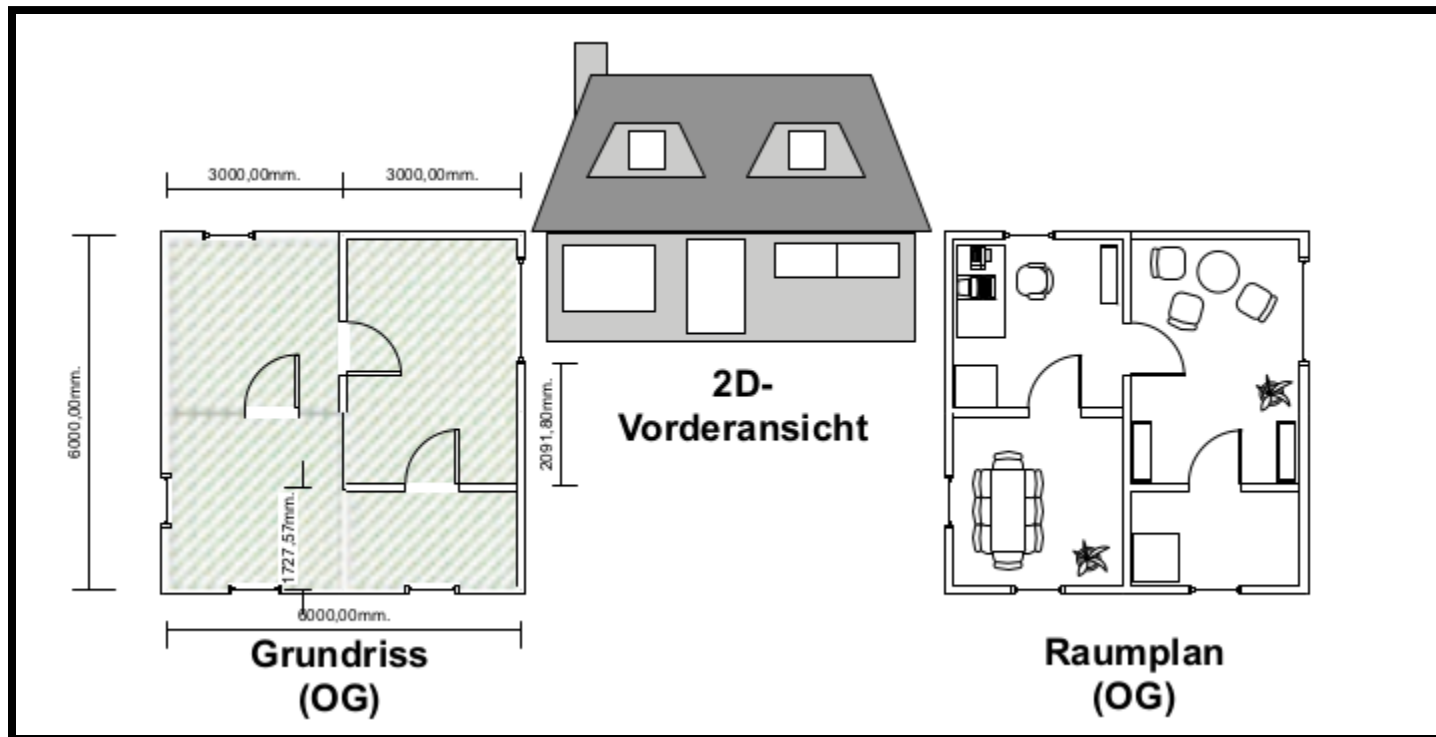
- Genauso wenig, wie man nur mit einem Grundriss ein Haus bauen kann.

## 2.

Sichten ermöglichen die Konzentration auf einzelne Aspekte des Gesamtsystems und reduzieren somit die Komplexität der Darstellung.

# 3.

Die Projektbeteiligten haben ganz unterschiedliche Informationsbedürfnisse.



Bildquelle: "Effektive Softwarearchitekturen" von Gernot Starke



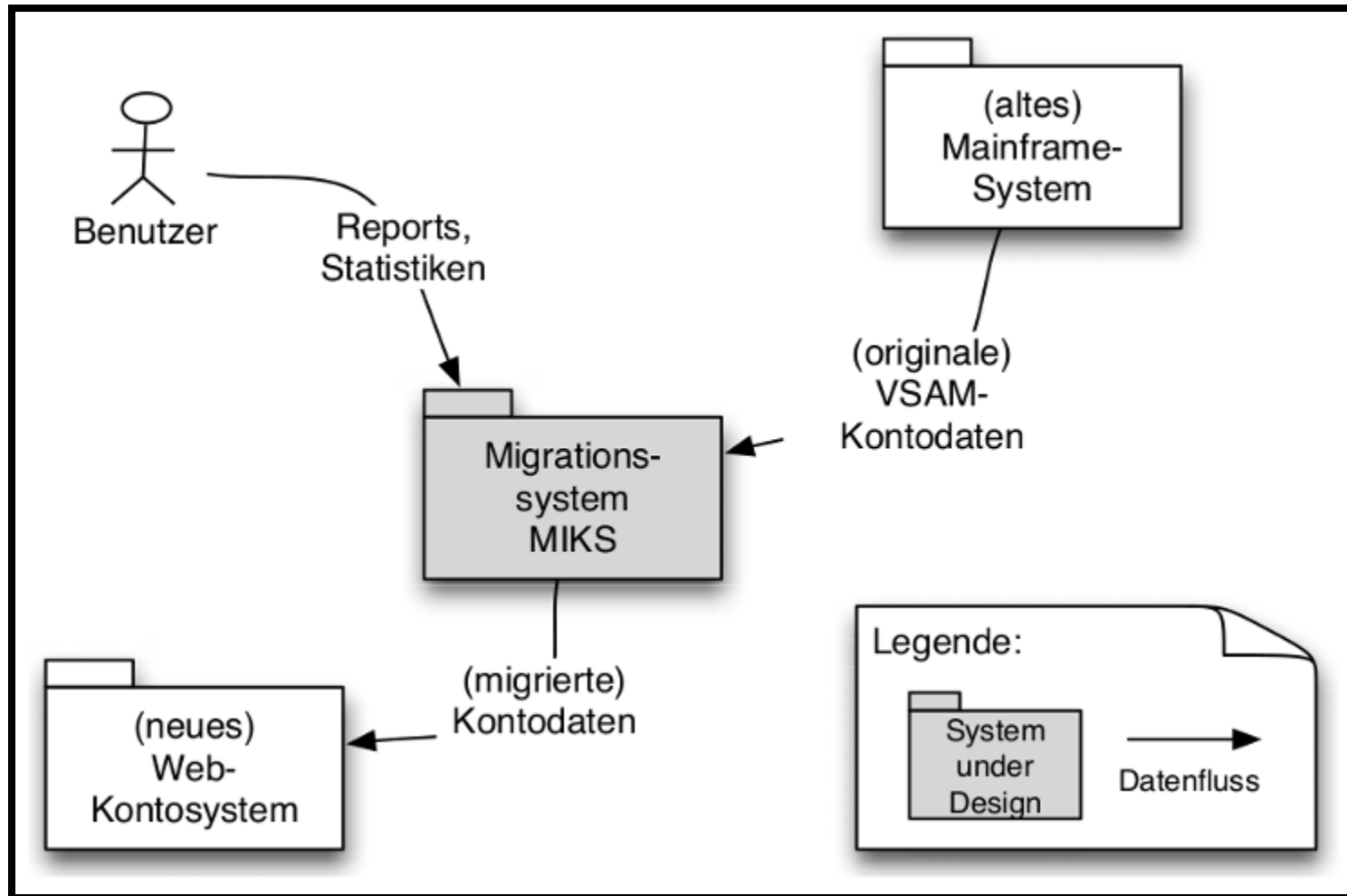
# Kontextsicht

- Wie ist das System in seine Umgebung eingebettet?
- zeigt das System als Blackbox in seinem Kontext aus der Vogelperspektive

## Kontextsicht - Enthaltene Informationen:

- Schnittstellen zu Nachbarsystemen
- Interaktion mit wichtigen Stakeholdern
- wesentliche Teile der umgebenden Infrastruktur

## Kontextsicht - Beispiel



Bildquelle: "Effektive Softwarearchitekturen" von Gernot Starke

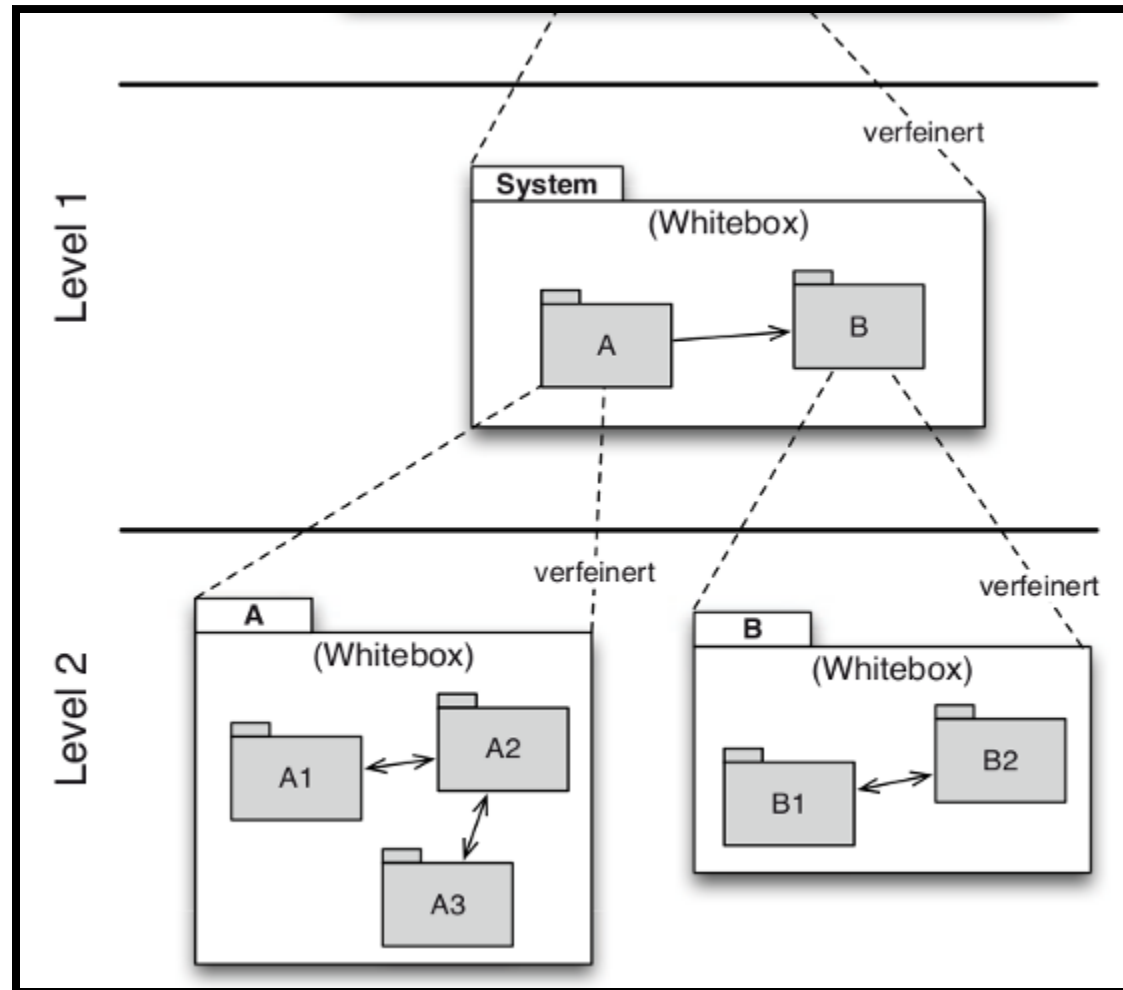
# Bausteinsicht

- Wie ist das System intern aufgebaut?
- unterstützt Auftraggeber und Projektleiter bei der Projektüberwachung
- dient der Zuteilung von Arbeitspaketen
- dient als Referenz für Software-Entwickler

## Bausteinsicht - Enthaltene Informationen:

- statische Strukturen der Bausteine des Systems
- Subsysteme
- Komponenten und deren Schnittstellen

# Bausteinsicht - Beispiel

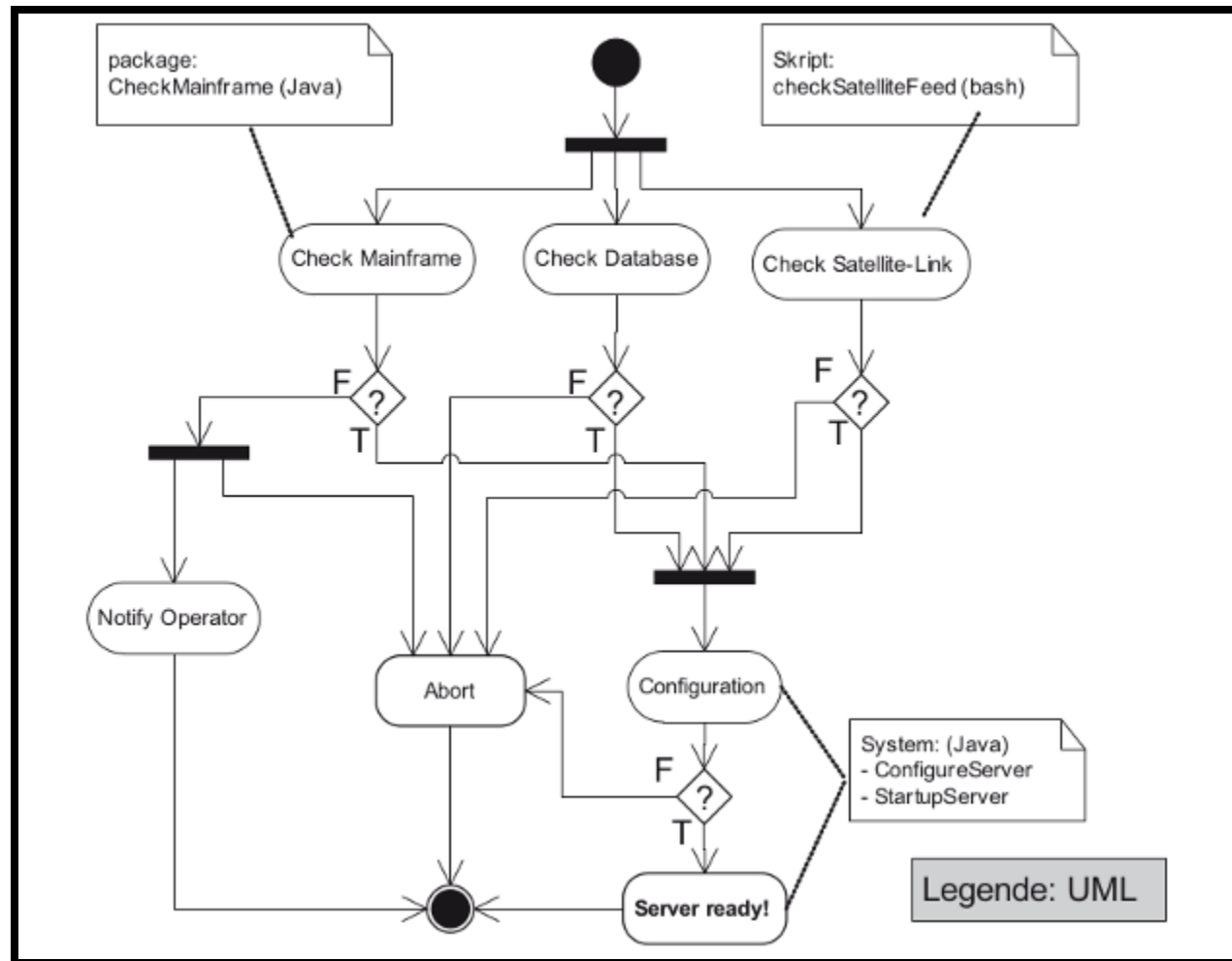


Bildquelle: "Effektive Softwarearchitekturen" von Gernot Starke

# Laufzeitsicht

- Wie läuft das System ab?
- Welche Bausteine des Systems existieren zur Laufzeit?
- Wie wirken die Bausteine zusammen?

# Laufzeitsicht - Beispiel



Bildquelle: "Effektive Softwarearchitekturen" von Gernot Starke



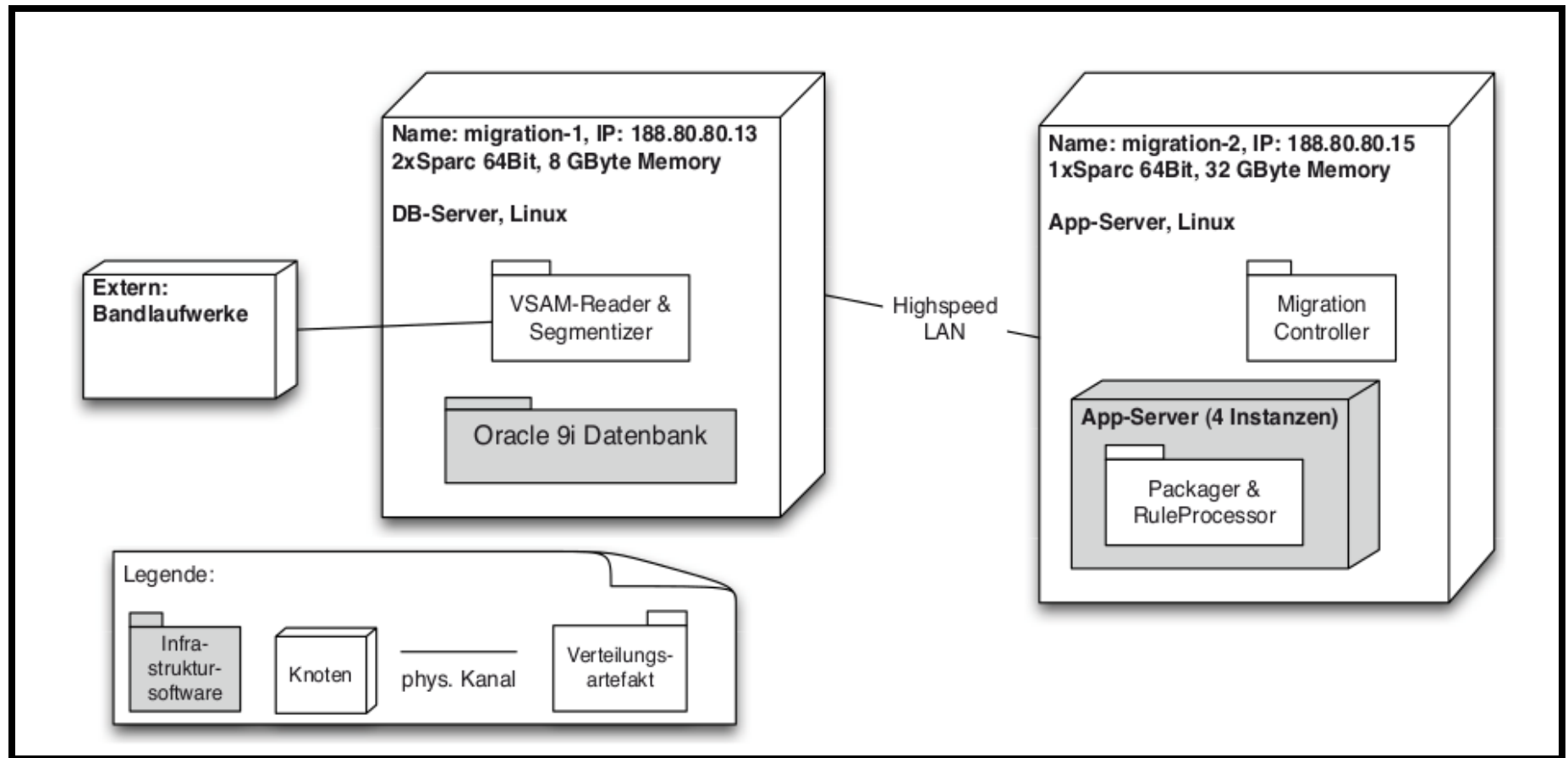
# Verteilungssicht / Infrastruktursicht

- In welcher Umgebung läuft das System ab?
- zeigt das System aus Betreibersicht

## Verteilungssicht - Enthaltene Informationen:

- Hardwarekomponenten: Rechner, Prozessoren
- Netztopologien
- Netzprotokolle
- sonstige Bestandteile der physischen Systemumgebung

## Verteilungssicht - Beispiel



Bildquelle: "Effektive Softwarearchitekturen" von Gernot Starke

## Empfehlung

- Verzichten Sie möglichst auf weitere Sichten
- Jede Sicht kostet Erstellungs- und Wartungsaufwand
- Die grundlegenden Aspekte der Architektur- und Systementwicklung decken die vier Sichten ab

# In welcher Reihenfolge sollten die Sichten entstehen?

Letztlich spielt es kaum eine Rolle, mit welcher Architektursicht Sie beginnen.  
Im Laufe des Entwurfs der Software-Architektur werden Sie an allen Sichten  
nahezu parallel arbeiten oder häufig zwischen den Sichten wechseln.

## Wie viel Aufwand für welche Sicht?

Rechnen Sie damit, dass Sie 60 bis 80% der Zeit, die Sie für den Entwurf der Architektursichten insgesamt benötigen, alleine für die Ausgestaltung der Bausteinsicht aufwenden. Der ausschlaggebende Grund hierfür: Die Bausteinsicht wird oftmals wesentlich detaillierter ausgeführt als die übrigen Sichten.

Dennoch sind die übrigen Sichten für die Software-Architektur und das Gelingen des gesamten Projektes wichtig! Lassen Sie sich von diesem relativ hohen Aufwand für die Bausteinsicht in keinem Fall dazu verleiten, die anderen Sichten zu ignorieren.

*Quelle: Starke/Effektive Softwarearchitekturen*

## Wechselwirkungen dokumentieren

- Bessere Nachvollziehbarkeit von Entscheidungen
- Auswirkungen von Änderungen werden vereinfacht
- Das Verständnis der Architekturbeschreibung wird einfacher, da die Zusammenhänge zwischen den Sichten klarer werden

## Entwurf der Kontextabgrenzung

- Im Idealfall ist die Kontextabgrenzung ein Ergebnis der Anforderungsanalyse
- Zeigen Sie in sämtliche Nachbarsysteme
- Alle ein- und ausgehenden Daten und Ereignisse müssen in der Kontextabgrenzung zu erkennen sein



## Entwurf der Bausteinsicht

- Der Entwurf der Bausteinsicht ist der Kern der Architekturbeschreibung
- Beschreiben Sie exakt, wie das System (strukturell) aufgebaut ist und aus welchen Bausteinen es besteht
- Beginnen Sie mit einer Vogelperspektive der Implementierungsbausteine
- Zerlegen Sie Ihr System dazu in große Architekturelemente, wie Sub- oder Teilsysteme

Erinnerung: Der Aufwand macht 60%-80% der Architekturarbeit aus

## Entwurf der Laufzeitsicht

- Elemente der Laufzeitsichten sind Instanzen der statischen Architekturbausteine, die Sie in den Bausteinsichten dokumentieren
- Ein möglicher Weg zur Laufzeitsicht führt daher über die Bausteinsichten
- Beschreiben Sie die Dynamik der statischen Bausteine, beginnend bei den wichtigsten Use-Cases des Gesamtsystems.
- Einen weiteren Startpunkt kann die Verteilungs-/Infrastruktursicht bilden

## Entwurf der Verteilungssicht

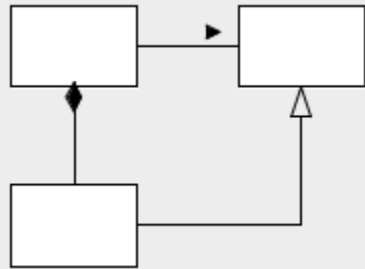
- Die Verteilungssicht sollte eine Landkarte der beteiligten Hardware und der externen Systeme beinhalten
- Genügen die verfügbare Hardware und die Kommunikationskanäle, oder gibt es potenzielle Engpässe?
- Falls Ihre Systeme in verteilten Umgebungen ablaufen, sollten Sie vorhandene Kommunikationsmechanismen, Protokolle und Middleware in die Infrastruktursicht aufnehmen

# UML für Softwarearchitekten

UML für Softwarearchitekten

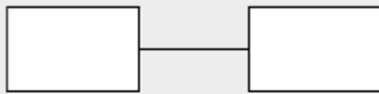
# UML Diagrammtypen

# Strukturdiagramme



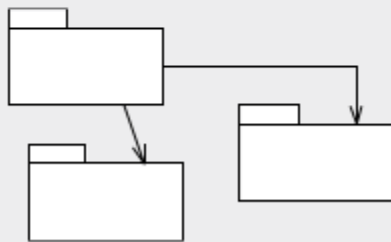
## **Klassendiagramm:**

Zeigt die statische Struktur von Klassen und deren Beziehungen (Assoziationen, Aggregationen sowie Spezialisierungen/Generalisierungen).



## **Objektdiagramm:**

Zeigt eine Struktur von Instanzen sowie deren Verbindungen. Bildet damit einen Schnappschuss des Klassendiagramms.

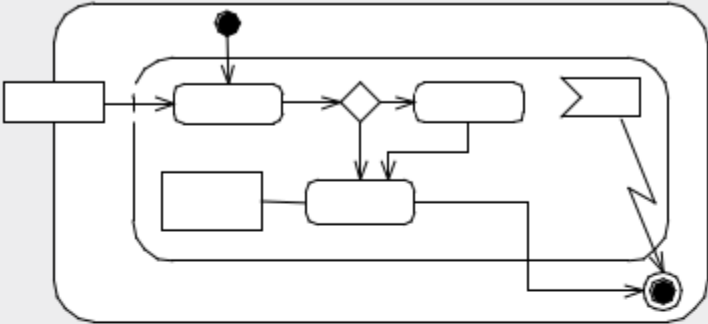
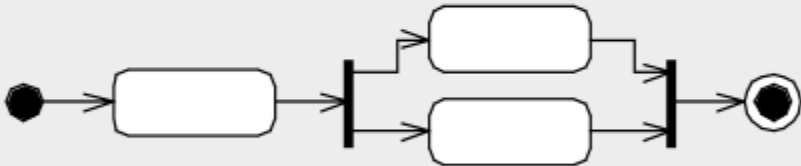
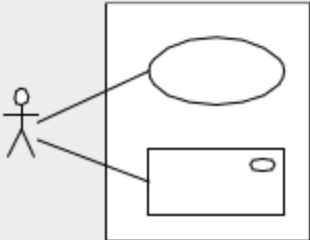


## **Paketdiagramm:**

Gibt einen Überblick über die Zerlegung des Gesamtsystems in Pakete oder Teilsysteme. Enthält logische Zusammenfassungen von Systemteilen.

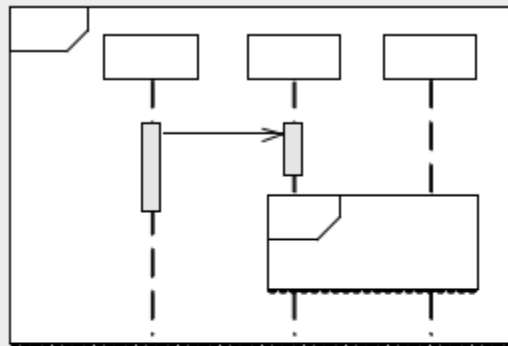
Bildquelle: "Effektive Softwarearchitekturen" von Gernot Starke

# Verhaltensdiagramme

Verhaltensdiagramme	
	<b>Aktivitätsdiagramm:</b> Zeigt mögliche Abläufe innerhalb von Systembestandteilen (etwa: Klassen, Komponenten oder Use-Cases). Kann Algorithmen, Daten- und Kontrollflüsse sehr detailliert beschreiben.
	<b>Zustandsdiagramm</b> ( <i>State Diagram</i> ): Zeigt die möglichen Zustände eines Systembestandteils sowie die erlaubten Zustandsübergänge an und verknüpft Aktivitäten mit diesen Zuständen und Übergängen.
	<b>Anwendungsfalldiagramm</b> ( <i>Use Case Diagram</i> ): Zeigt eine Übersicht über alle Prozesse, mit denen das System auf Wünsche von Akteuren (oder Nachbarsystemen) reagiert.

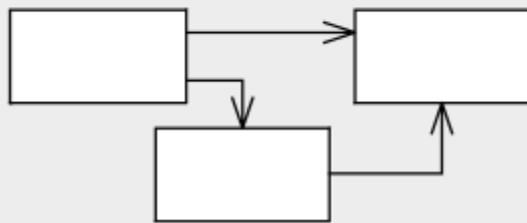
Bildquelle: "Effektive Softwarearchitekturen" von Gernot Starke

# Interaktionsdiagramme



## **Sequenzdiagramm:**

Zeigt den Nachrichtenaustausch zwischen Instanzen von Systembestandteilen für einzelne Szenarien.  
Seit UML 2 schachtelbar.



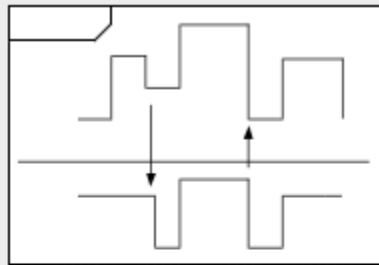
## **Kommunikationsdiagramm:**

In früheren UML Versionen Kollaborationsdiagramm genannt, zeigt die Zusammenarbeit zwischen Instanzen von Systembestandteilen.

Bildquelle: "Effektive Softwarearchitekturen" von Gernot Starke



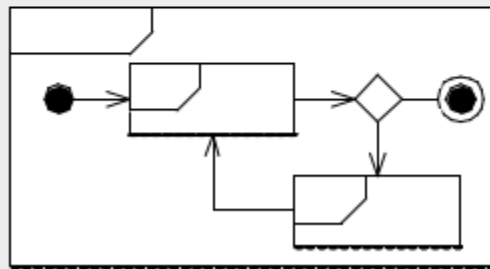
# Interaktionsdiagramme



## **Zeitverhaltensdiagramm**

*(Timing-Diagram):*

Zeigt Zeitabhängigkeiten zwischen Ereignissen beschreibt Zustandsänderungen in Abhängigkeit vom Zeitverlauf.



## **Interaktionsübersichtsdiagramm:**

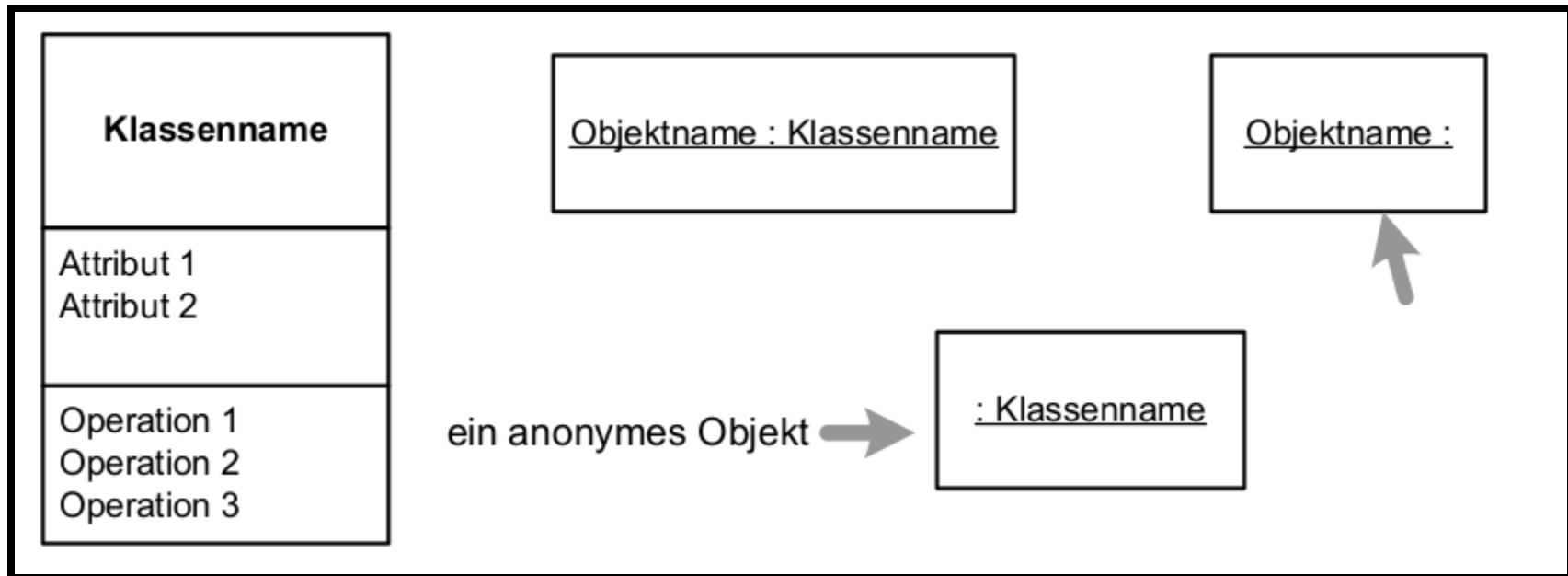
Zeigt das Zusammenspiel verschiedener Interaktionen und besteht in der Regel aus Referenzen auf andere Interaktionsdiagramme. Ein Art Aktivitätsdiagramm auf hoher Abstraktionsebene.

Bildquelle: "Effektive Softwarearchitekturen" von Gernot Starke

UML für Softwarearchitekten

# UML Klassen und Objekte

# Klassen und Objekte

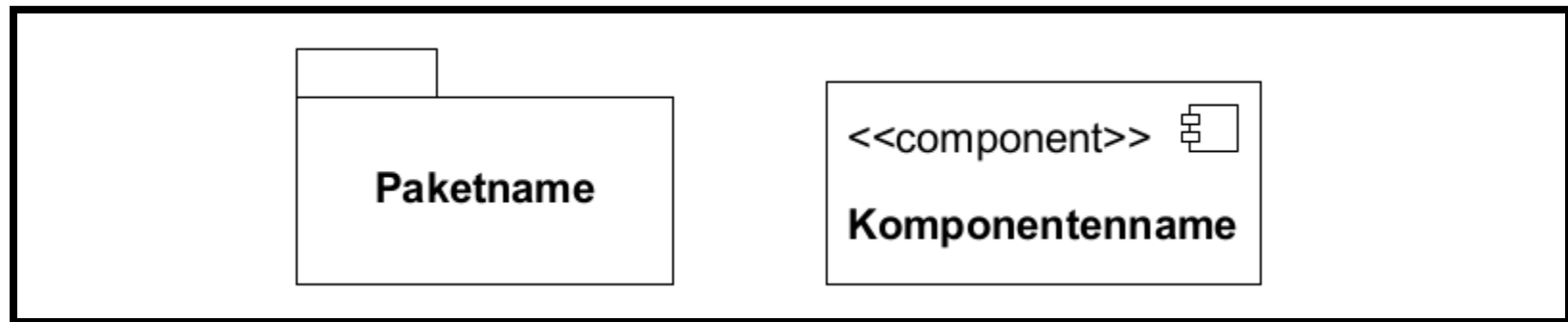


Bildquelle: "Effektive Softwarearchitekturen" von Gernot Starke

UML für Softwarearchitekten

# UML Pakete und Komponenten

## Pakete und Komponenten

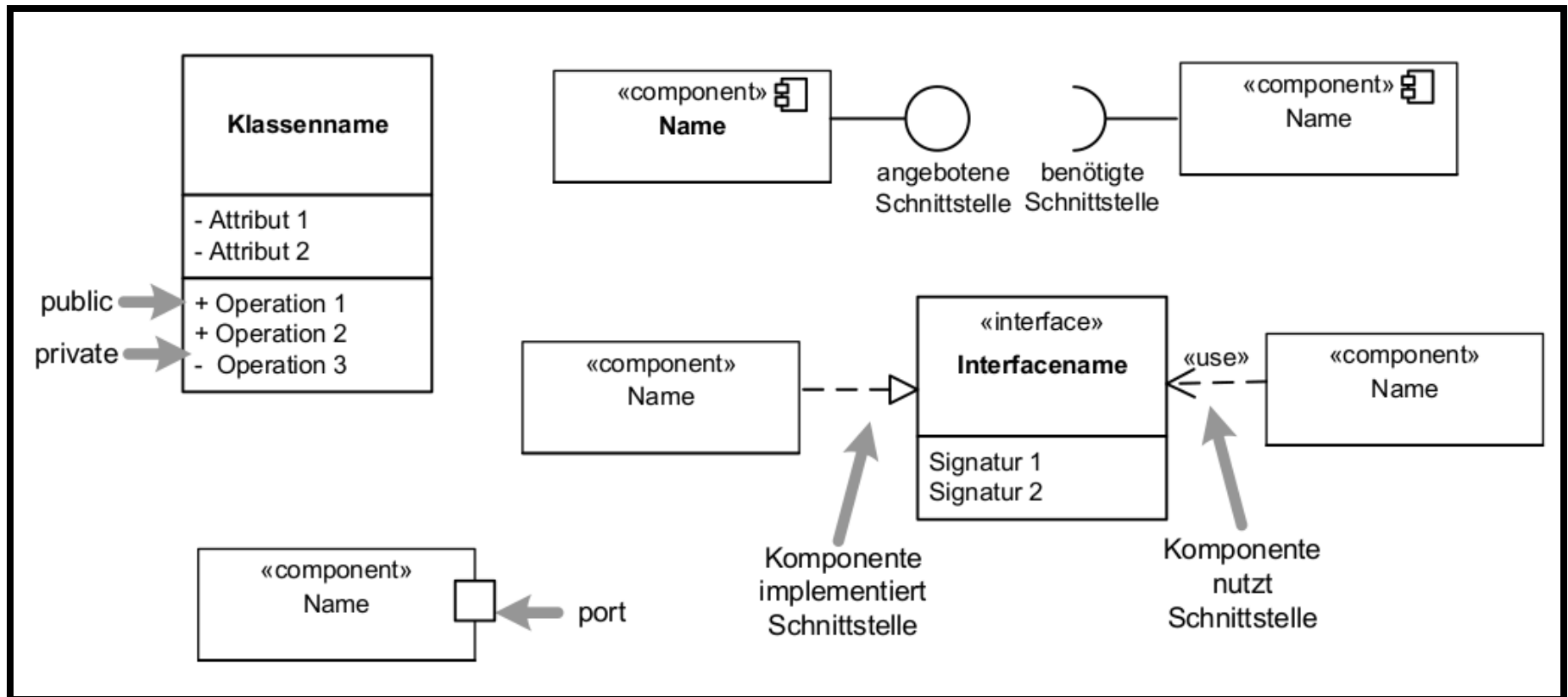


Bildquelle: "Effektive Softwarearchitekturen" von Gernot Starke

UML für Softwarearchitekten

UML Schnittstellen

# Schnittstellen



Bildquelle: "Effektive Softwarearchitekturen" von Gernot Starke

UML für Softwarearchitekten

Welches UML Diagramm für welche Sicht?



UML für Softwarearchitekten

## Baustein-Sicht

- Gute Namen wählen!
- Rollennamen anzugeben, Navigationsrichtung vorschreiben und Multiplizitäten festlegen
- Verwenden Sie nur eine Art von Schnittstellendarstellung
- Nutzen Sie Stereotypes für verschiedene Arten von fachlichen und technischen Klassen und Komponenten

UML für Softwarearchitekten

## Baustein-Sicht

- Paketdiagramm
- Komponentendiagramm
- Klassendiagramm
- Aktivitätsdiagramm
- Zustandsdiagramm

UML für Softwarearchitekten

## Verteilungs-Sicht

- Hauptelemente: Knoten und Kanäle zwischen den Knoten
- Knoten sind Standorte, z.B. Cluster, Rechner, Chips, ...
- Kanäle sind die physikalischen Übertragungswege, z.B. Kabeln, Bluetooth, Wireless, ...

UML für Softwarearchitekten

## Verteilungs-Sicht

- Verteilungsdiagramm
- Kontextdiagramm

## UML für Softwarearchitekten

### Laufzeitsicht

- Elemente der Laufzeitsicht sind immer um Instanzen von Bausteinen, die in der Bausteinsicht enthalten sind, also um Objekte zu den Klassen oder um instanziierte Komponenten.

UML für Softwarearchitekten

## Laufzeitsicht

- Objektdiagramm
- Kompositionsstrukturdiagramm
- Sequenzdiagramm
- Laufzeitkontextdiagramm
- Kommunikationsdiagramm
- Interaktionsdiagramm

UML für Softwarearchitekten

## Warum UML?

- UML hat die Kästchen und Striche für uns standardisiert
- Die Bausteine der Architektur lassen sich auf verschiedenen Abstraktionsebenen miteinander in Beziehung setzen
- Die Zusammenarbeit wird effektiver, wenn alle hinter den Kästchen und Strichen das Gleiche verstehen

UML für Softwarearchitekten

Praxisrelevanz?



Fragen?

Was ist Softwarearchitektur?

Geschichte und Trends

Sichten auf Architekturen

Qualität und andere nichtfunktionale Anforderungen

Architekturmuster

Dokumentation von Architekturen

Technologien und Frameworks

Wiederholung

Qualität und andere nichtfunktionale Anforderungen

Was ist Qualität?

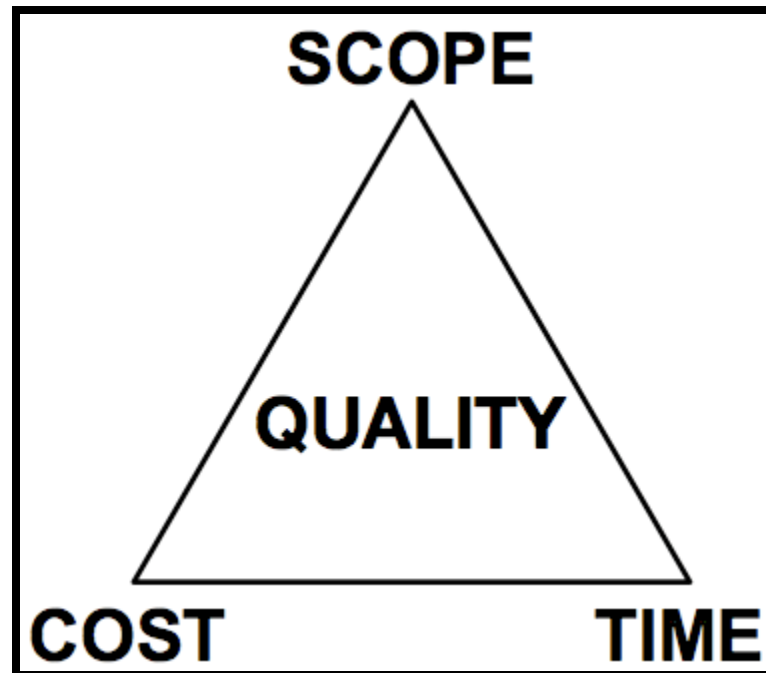
*Was ist Qualität?*

Duden: Qualität=„Beschaffenheit, Güte, Wert“

## *Was ist Qualität?*

Die Qualität stimmt, wenn der Kunde wiederkommt und nicht das Produkt

*Was ist Qualität?*



Quelle: <http://pm-blog.com/>

Qualität ist ein wichtiges Ziel für Software-Architekten



# Probleme von Qualität

- Qualität ist nur indirekt messbar
- Qualität ist relativ (jeweils anders für: Anwender, Projektleiter, Betreiber, ...)
- Die Qualität der Architektur korreliert nicht notwendigerweise mit der Codequalität
- Erfüllung aller funktionalen Anforderungen lässt keinerlei Aussage über die Erreichung der Qualitätsanforderungen zu

Qualitätsmerkmale nach DIN/ISO 9126

Funktionalität

Zuverlässigkeit

Benutzbarkeit

Effizienz

Änderbarkeit

Übertragbarkeit

Die weiteren Details zu Qualität und nichtfunktionalen Anforderungen sind Bestandteil der kommenden Kapitel

Fragen?

Praktikumsaufgabe

## 2. Praktikumsaufgabe

## Praktikumsaufgabe

# REST

Representational State Transfer (REST) bezeichnet ein Programmierparadigma für verteilte Systeme, insbesondere für Webservices. REST ist eine Abstraktion der Struktur und des Verhaltens des World Wide Web.

REST hat das Ziel, einen Architekturstil zu schaffen, der die Anforderungen des modernen Web besser darstellt.

## Praktikumsaufgabe

# REST

- Der Zweck von REST liegt schwerpunktmäßig auf der Maschine-zu-Maschine-Kommunikation.
- REST kodiert keine Methodeninformation in den URI, der URI gibt Ort und Namen der Ressource an
- Eine Ressource kann über verschiedene Medientypen dargestellt werden, auch Repräsentation der Ressource genannt.

Praktikumsaufgabe

## REST Prinzipien

- Client-Server
- Zustandslosigkeit
- Caching
- Einheitliche Schnittstelle
- Mehrschichtige Systeme
- Code on Demand



## Praktikumsaufgabe

# REST Einheitliche Schnittstelle

Dies ist das Hauptunterscheidungsmerkmal von allen weiteren Architekturstilen. Dabei besteht diese aus 4 weiteren Eigenschaften. Ziel ist die Einheitlichkeit der Schnittstelle und somit ihre einfache Nutzung.

- Adressierbarkeit von Ressourcen
- Repräsentationen zur Veränderung von Ressourcen
- Selbstbeschreibende Nachrichten (Verwendung von Standardmethoden wie Http)
- „Hypermedia as the Engine of Application State“ (HATEOAS)

## Praktikumsaufgabe

# HTTP

Das Hypertext Transfer Protocol (HTTP) ist ein zustandsloses Protokoll zur Übertragung von Daten auf der Anwendungsschicht über ein Rechnernetz. Es wird hauptsächlich eingesetzt, um Webseiten (Hypertext-Dokumente) aus dem World Wide Web (WWW) in einen Webbrowser zu laden.

Es ist jedoch nicht prinzipiell darauf beschränkt und auch als allgemeines Dateiübertragungsprotokoll sehr verbreitet.

## Praktikumsaufgabe

# HTTP Verben

- GET fordert die angegebene Ressource vom Server an.
- POST fügt eine neue (Sub-)Ressource unterhalb der angegebenen Ressource ein.
- PUT legt die angegebene Ressource an. Wenn die Ressource bereits existiert, wird sie geändert.
- PATCH ändert einen Teil der angegebenen Ressource.
- DELETE löscht die angegebene Ressource.

Praktikumsaufgabe

# Ein kleines Beispiel

[swagger.io](https://swagger.io)

Was ist Softwarearchitektur?

Geschichte und Trends

Sichten auf Architekturen

Qualität und andere nichtfunktionale Anforderungen

**Architekturmuster**

Dokumentation von Architekturen

Technologien und Frameworks

Was sind Architekturmuster?

A pattern for software architecture describes a particular recurring design problem that arises in specific design contexts, and presents a well-proven generic scheme for its solution. The solution scheme is specified by describing its constituent components, their relationships, and the ways in which they collaborate.

*(1996 / Pattern Oriented Software Architecture)*

Ein Architekturmuster beschreibt eine bewährte Lösung für ein wiederholt auftretendes Entwurfsproblem

*(Effektive Softwarearchitekturen)*



Ein Architekturmuster definiert den Kontext für die Anwendbarkeit der Lösung  
*(Effektive Softwarearchitekturen)*

Warum Architekturmuster?

Erfolg kommt von Weisheit.  
Weisheit kommt von Erfahrung.  
Erfahrung kommt von Fehlern.

Haben Sie jemals einen dummen Fehler zweimal begangen?

– *Willkommen in der realen Welt.*

Haben Sie diesen Fehler hundertmal hintereinander gemacht?

-*Willkommen in der Software-Entwicklung.*

Aus Fehlern kann man hervorragend lernen.

Leider akzeptiert kaum ein Kunde Fehler, nur weil Sie Ihre Erfahrung als Software-Architekt sammeln.

In dieser Situation helfen Heuristiken.

Heuristiken kodifizieren Erfahrungen anderer Architekten und Projekte, auch aus anderen Bereichen der Systemarchitektur.

Heuristiken sind nicht-analytische Abstraktionen von Erfahrung

Es sind Regeln zur Behandlung komplexer Probleme, für die es meist beliebig viele Lösungsalternativen gibt. Heuristiken können helfen, Komplexität zu reduzieren.

Andere Begriffe für Heuristiken sind auch „Regeln“, „Muster“ oder „Prinzipien“.

Es geht immer um Verallgemeinerungen und Abstraktionen von konkreten Situationen.



Heuristiken bieten Orientierung im Sinne von Wegweisern,  
Straßenmarkierungen und Warnschildern.

Sie geben allerdings lediglich Hinweise und garantieren nichts. Es bleibt in Ihrer Verantwortung, die passen- den Heuristiken für eine bestimmte Situation auszuwählen:

Die Kunst der Architektur liegt nicht in der Weisheit der Heuristiken, sondern in der Weisheit, a priori die passenden Heuristiken für das aktuelle Projekt auszuwählen.

# Architektur: Von der Idee zur Struktur

Ein klassischer und systematischer Ansatz der Beherrschung von Komplexität lautet „teile und herrsche“ (divide et impera). Das Problem wird in immer kleinere Teile zerlegt, bis diese Teilprobleme eine überschaubare Größe annehmen.

# Anwendung auf Software-Architekturen:

*klassische Architekturmuster*

Horizontale Zerlegung: „In Scheiben schneiden“

Vertikale Zerlegung: „In Stücke schneiden“

*weitere Architekturmuster*

Alles ist möglich...

# Horizontale Zerlegung

Jede Schicht stellt einige klar definierte Schnittstellen zur Verfügung und nutzt Dienste von darunter liegenden Schichten.

# Vertikale Zerlegung

Jeder Teil übernimmt eine bestimmte fachliche oder technische Funktion.

Prinzipien zur Zerlegung

## Kapselung (information hiding)

- Kapseln von Komplexität in Komponenten.
- Betrachtung der Komponenten als „black box“,
- Definition klarer Schnittstellen
- Ohne Kapselung erschwert eine Zerlegung das Problem, statt es zu vereinfachen (was bekannt ist, wird auch ausgenutzt!)

Prinzipien zur Zerlegung

# Wiederverwendung

- wiederverwendbarkeit verringert den Wartungsaufwand
- Achtung: Nur Dinge wiederverwenden, bei denen es sinnvoll ist



Prinzipien zur Zerlegung

## Iterativer Entwurf

- Überprüfung eines Entwurfs mit Prototypen oder Durchstichen
- Evaluation der Stärken und Schwächen eines Entwurfes
- Explizite Bewertung und Analyse dieser Versuche

Prinzipien zur Zerlegung

## Dokumentation von Entscheidungen

- Warum wurde eine Entscheidung so getroffen?
- Welche Alternativen wurden bewertet?
- Andere Projektbeteiligte werden diese Entscheidungen später kritisieren!

Prinzipien zur Zerlegung

## Unabhängigkeit der Elemente

- Geringe Abhängigkeiten erhöhen die Wartbarkeit und Flexibilität des Systems
- Komponenten sollen keine Annahmen über die Struktur anderer Komponenten machen

Warum sind wir hier?

# Ein Praxisbericht

Why a service oriented architecture is not the holy grail...

Überblick über Architekturmuster

*Arten von Architekturmustern?*

Chaos zu Struktur / Mud-to-structure

Verteilte Systeme

Interaktive Systeme

Adaptive Systeme

Domain-spezifische Architektur

Was ist Softwarearchitektur?

Geschichte und Trends

Sichten auf Architekturen

Qualität und andere nichtfunktionale Anforderungen

**Architekturmuster**

Dokumentation von Architekturen

Technologien und Frameworks



# Fragen?

Unterlagen: [ai2016.nils-loewe.de](http://ai2016.nils-loewe.de)