University of Applied Sciences
HOCHSCHULE EMDEN·LEER

University of Applied Sciences
HOCHSCHULE EMDEN·LEER

# MODELLIERUNG

NIELS STREEKMANN

SOMMERSEMESTER 2020

VORLESUNG 7

SYSTEMENTWURF

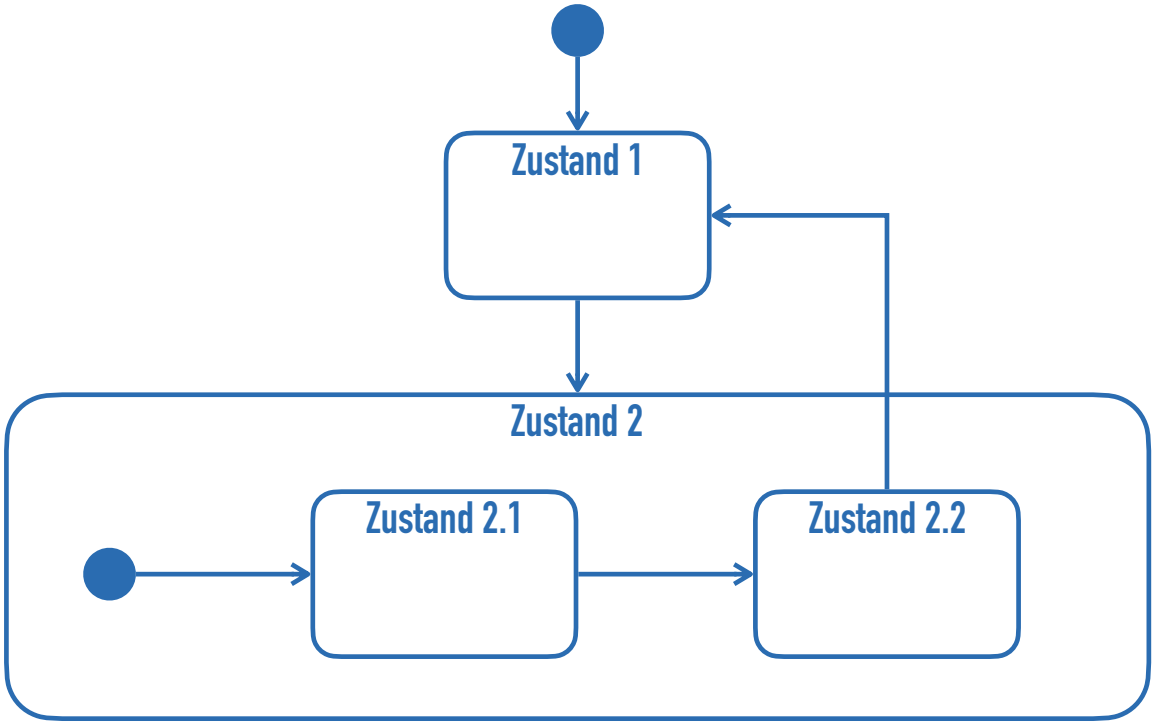
# ZUSTANDSDIAGRAMM – MÖGLICHES VORGEHEN

- Wähle Sequenzdiagramme mit betrachteter Klasse.
- Ordne Ereignisse einer Spalte zu einem Pfad, dessen Kanten mit den Ereignisnamen beschriftet werden.
- Zwischen je 2 Ereignissen liegt ein Zustand.
- Vergib aussagefähige Namen.
- Berücksichtige Sonder- und Fehlerfälle.
- Berücksichtige Ereignisse zu unpassenden Zeitpunkten.

| Buch |
| --- |
| erfassen()<br>ausleihen()<br>zurückgeben()<br>vorbestellen()<br>entfernen() |

# ZUSAMMENGESETZTE ZUSTÄNDE

Ein Zustand kann durch Unterzustände verfeinert werden



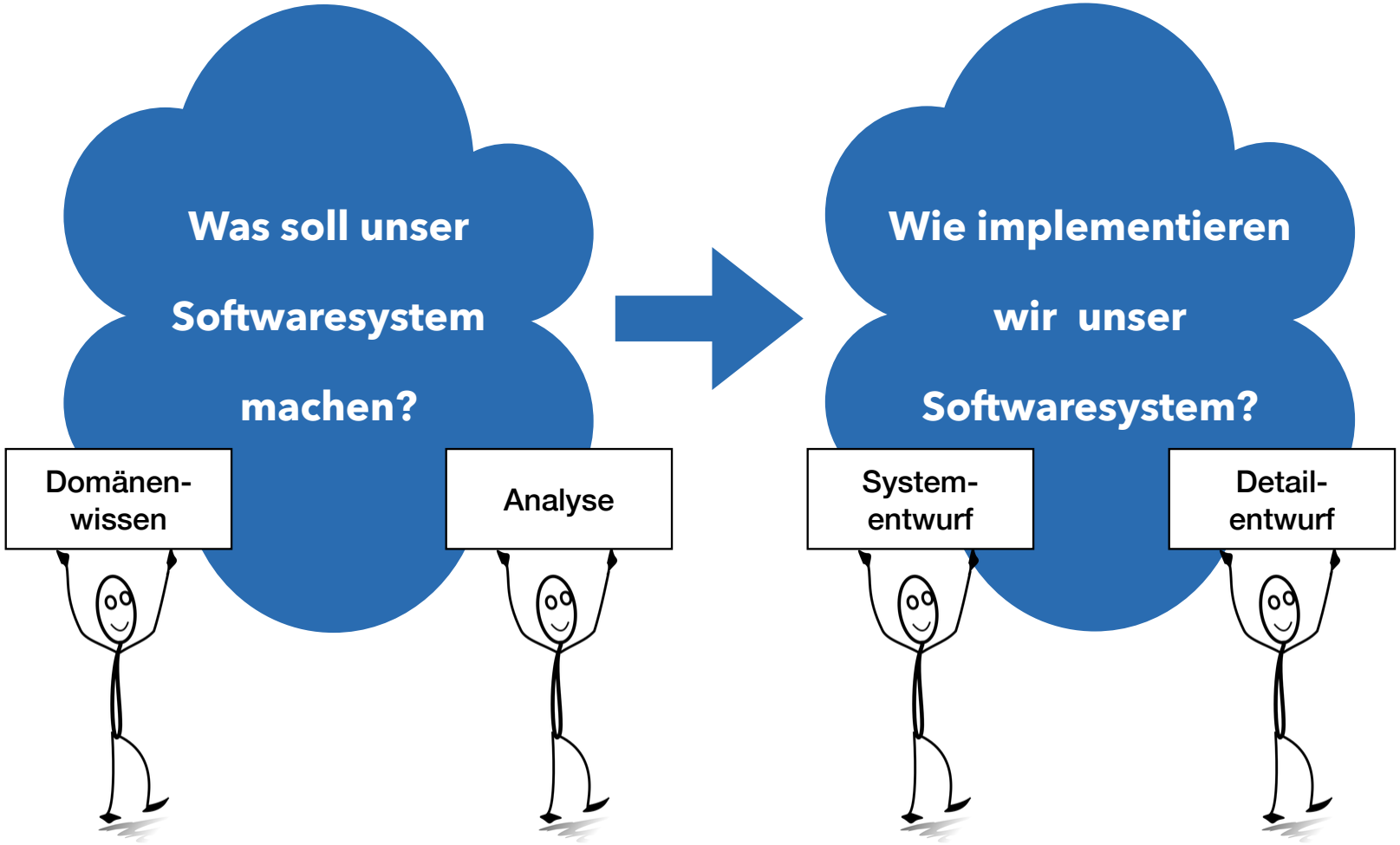
# ZUSTANDSDIAGRAMM – ABGRENZUNG

- Nicht jedes Objekt benötigt ein Zustandsdiagramm. Für Objekte geringer dynamischer Komplexität reicht eine Liste der Eingabeereignisse mit den jeweils erzeugten Ausgabeereignissen.
- Ereignisse zwischen Objekten abgleichen
  - Hat jedes Ereignis einen Sender und einen Empfänger?
  - Handelt es sich bei Start- und Endzuständen um Anfangs- und Endpunkte von Interaktionsfolgen?
  - Haben alle sonstigen Zustände Vorgänger und Nachfolger?
  - Sind Zustandsdiagramme und Sequenzdiagramme konsistent?

# DIAGRAMME

- Diagramme sollen einen bestimmten, abgegrenzten Sachverhalt darstellen
- Diagramme geben häufig nur eine Übersicht über den Sachverhalt
- Diagramme sollten daher nicht zu viele Elemente enthalten und die klar und übersichtlich anordnen
- Diagramme werden im Rahmen der objektorientierten Analyse zur Kommunikation und zur Dokumentation von Anforderungen verwendet
- Diagramme müssen (je nach Nutzungskontext) ein Gebiet nicht vollständig und in höchstem Detaillierungsgrad abdecken

# PROBLEM- UND LÖSUNGSRAUM



# SYSTEMENTWURF / SOFTWARE-ARCHITEKTUR

Welche Art von System soll erstellt werden?

# SYSTEMENTWURF / SOFTWARE-ARCHITEKTUR

Welche Art von System soll erstellt werden?

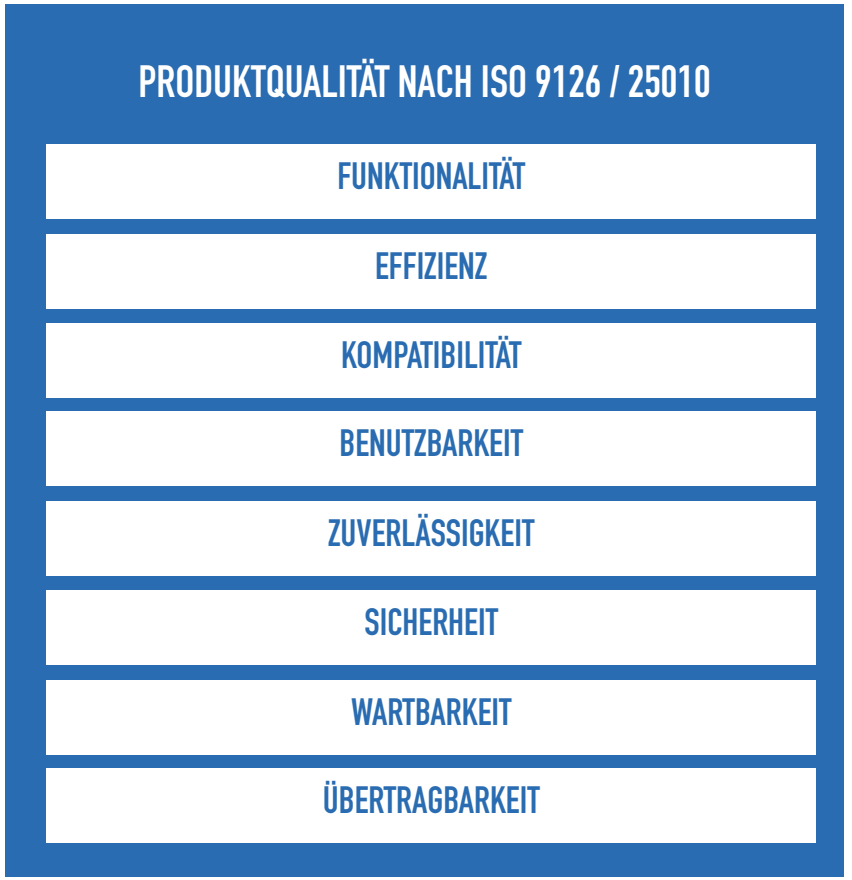
| | | |
|---|---|---|
| DESKTOP-ANWENDUNG | MOBILE APP | WEB-ANWENDUNG |
| KOMMANDOZEILEN-ANWENDUNG | DIENST-ANWENDUNG | EINGEBETTETES SYSTEM |
| BATCH-VERARBEITUNG | VERARBEITUNG KONTINUIERLICHER DATEN | STEUERUNGSSOFTWARE |

# SYSTEMENTWURF / SOFTWARE-ARCHITEKTUR

Welche Qualitätsmerkmale
soll das System zu
welchem Grad erfüllen?

# SYSTEMENTWURF / SOFTWARE-ARCHITEKTUR

Welche Qualitätsmerkmale soll das System zu welchem Grad erfüllen?



# SYSTEMENTWURF / SOFTWARE-ARCHITEKTUR

- Umfang / Aufwand des Systementwurfs abhängig von der Komplexität des Problems
- Fehler im Systementwurf sind später nur mit hohen Kosten zu beheben
- Entwicklung der globalen Problemlösungsstrategie für die Implementierung
  - Entscheidungen zur Konzeption und Vorgehensweise
  - Organisation in Teilsysteme
  - Grundlegende Entscheidungen zu Qualitätsmerkmalen
  - Rahmen für Detailentwurf

# SYSTEMENTWURF / SOFTWARE-ARCHITEKTUR

## WAS IST EIN GUTER ENTWURF?

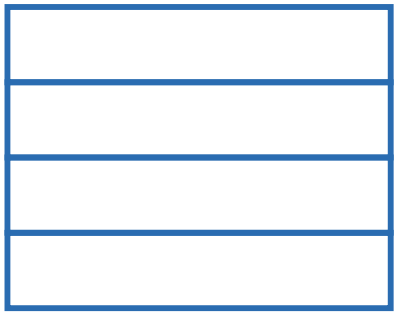
# SYSTEMENTWURF / SOFTWARE-ARCHITEKTUR

## WAS IST EIN GUTER ENTWURF?

- EINE KLARE UND VERSTÄNDLICHE LÖSUNG FÜR DAS PROBLEM
- ERFÜLLT DIE ANFORDERUNGEN DES KUNDEN (FUNKTIONALE ANFORDERUNGEN UND QUALITÄTSMERKMALE)
- ERMÖGLICHT WIRTSCHAFTLICHE ENTWICKLUNG UND BETRIEB DES SYSTEMS
- BERÜCKSICHTIGT GRUNDLEGENDE PRINZIPIEN DER SOFTWARETECHNIK

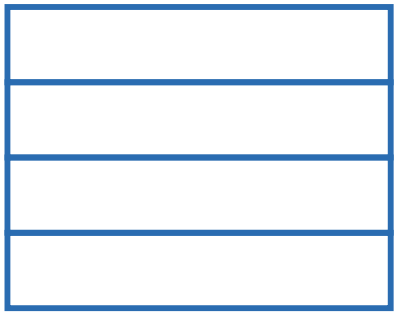
# AUFTEILUNG DES SYSTEMS IN TEILSYSTEME

# AUFTEILUNG DES SYSTEMS IN TEILSYSTEME

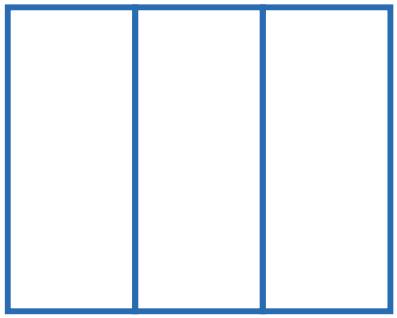


Schichten

# AUFTEILUNG DES SYSTEMS IN TEILSYSTEME

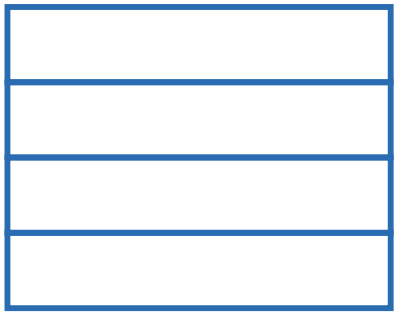


Schichten

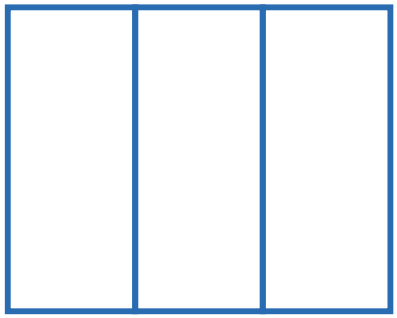


Partitionen

# AUFTEILUNG DES SYSTEMS IN TEILSYSTEME



Schichten

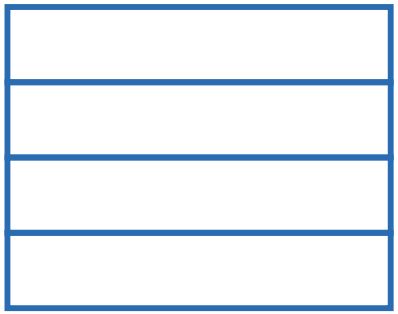


Partitionen

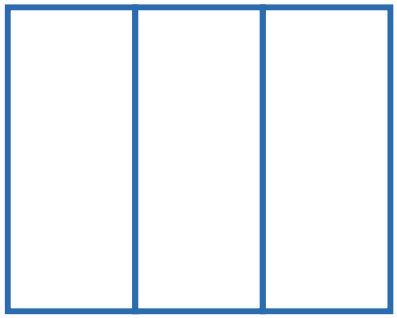
Mischformen

# AUFTEILUNG DES SYSTEMS IN TEILSYSTEME

KEIN UML



Schichten

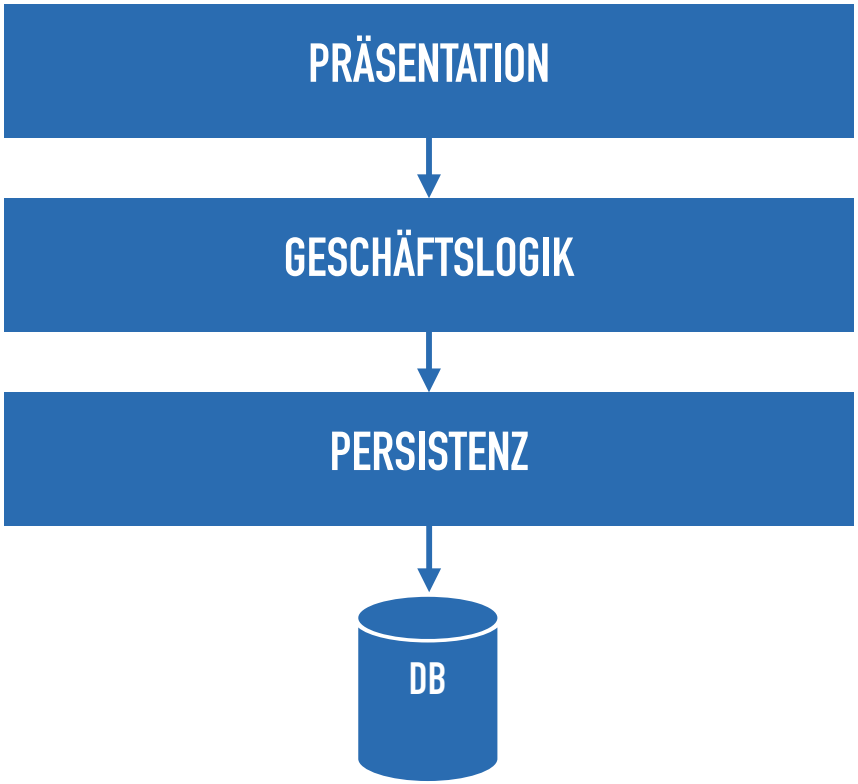


Partitionen

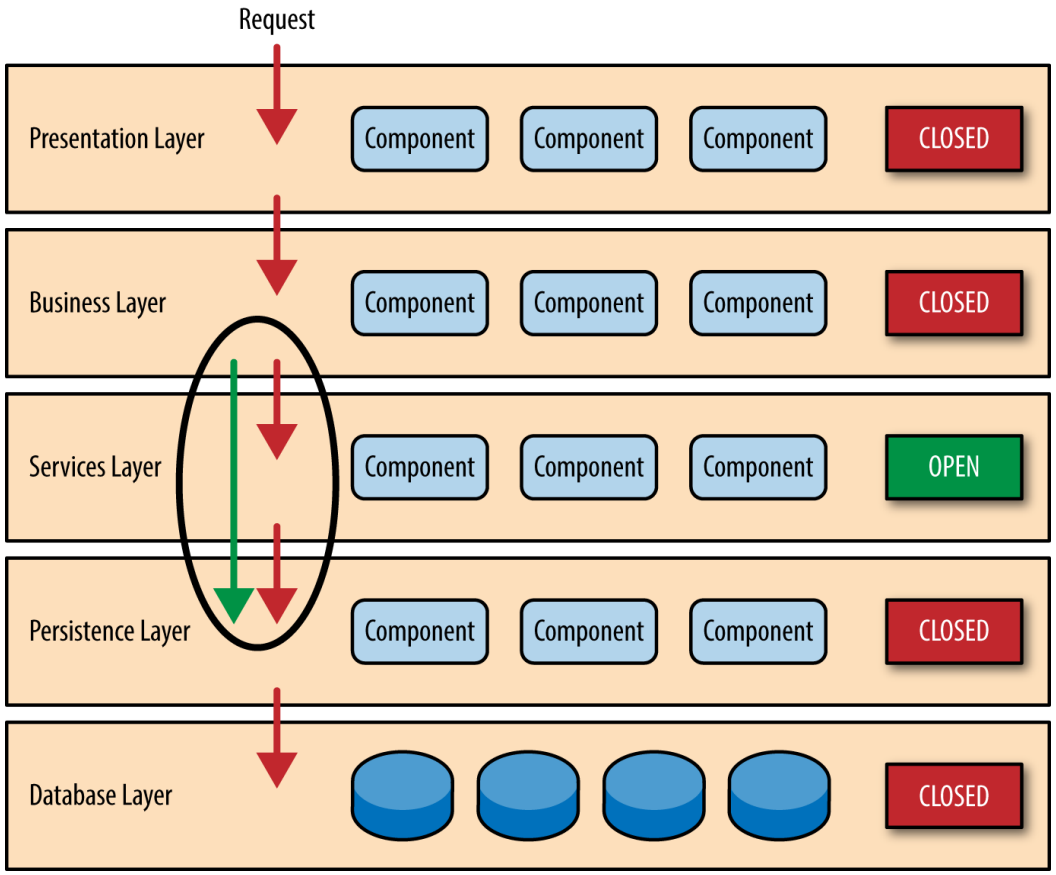
Mischformen

# SCHICHTENARCHITEKTUR

Typische Schichtenarchitektur eines Informationssystems

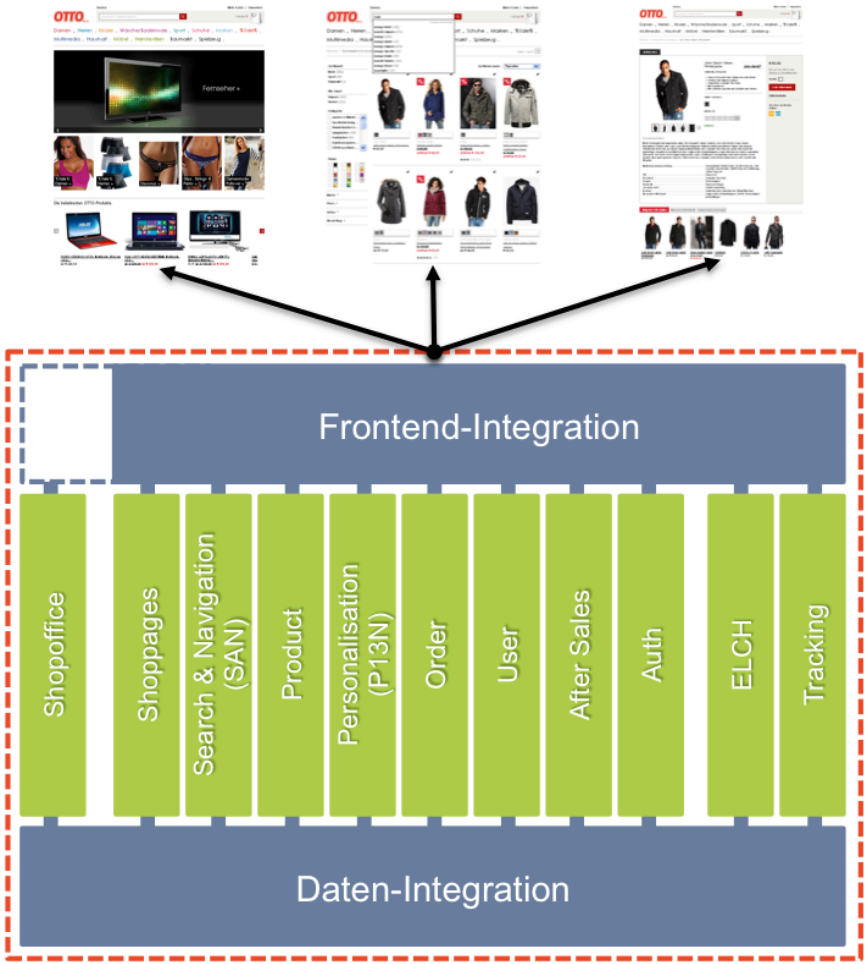


# SCHICHTENARCHITEKTUR



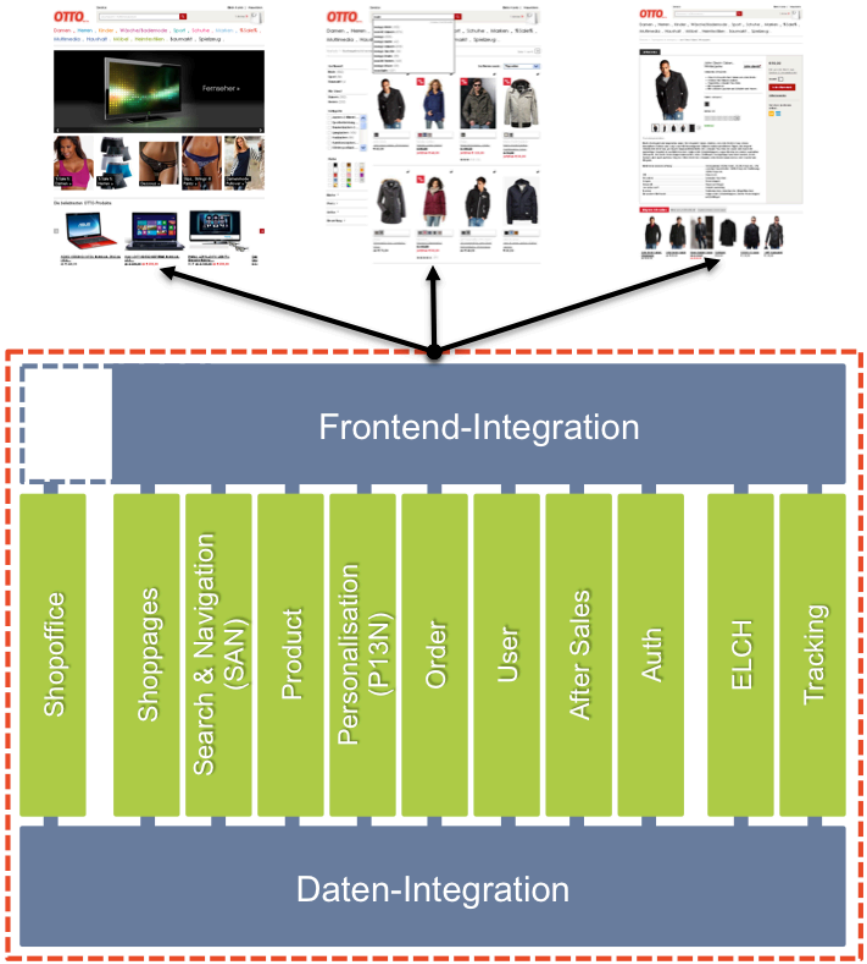

Quelle: Mark Richards - Software Architecture Patterns, O'Reilly, 2015

# VERTIKALE PARTITIONEN – BEISPIEL OTTO




Quelle: Teile und herrsche: Kleine Systeme für grosse Architekturen, OBJEKTspektrum 05/2013

# VERTIKALE PARTITIONEN – BEISPIEL OTTO

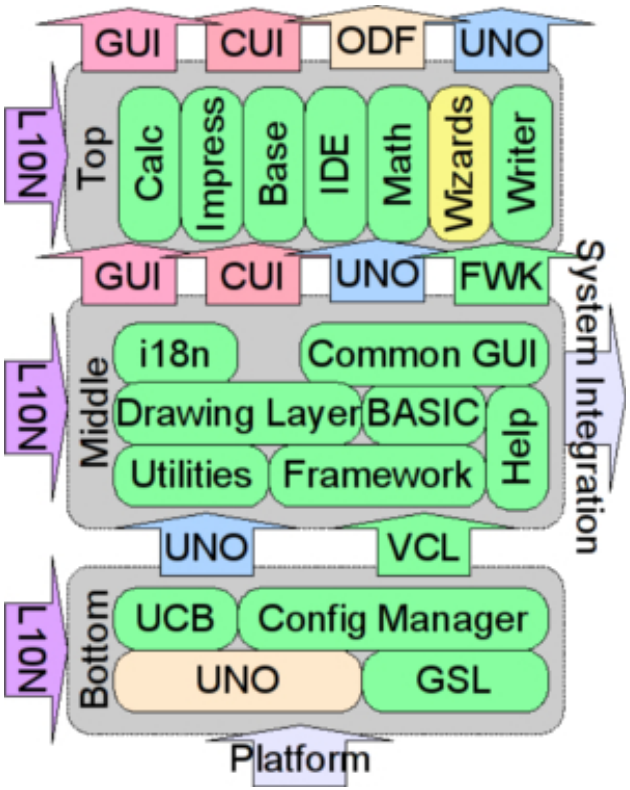



Eigenschaften der Partitionen bei OTTO

- Entwicklung in separaten Teams
- Eigenes Frontend
- Eigene Datenhaltung
- Klar definierte Zuständigkeit für Daten und Features
- Eigene Code-Basis, d. h. kein geteilter Code zwischen den Vertikalen
- Kein geteilter Zustand zwischen den Vertikalen
- Separat deploybare Einheit
  - Kommunikation über definierte REST-Schnittstellen

Quelle: Teile und herrsche: Kleine Systeme für grosse Architekturen, OBJEKTspektrum 05/2013

# MISCHFORMEN – BEISPIEL OPENOFFICE




Quelle: https://wiki.openoffice.org/wiki/Architecture

# ENTWURFSPRINZIPIEN

- ABSTRAKTION
- MODULARISIERUNG
- KAPSELUNG
- WIEDERVERWENDUNG
- NEBENLÄUFIGKEIT

# ENTWURFSPRINZIPIEN – ABSTRAKTION



- Konzentration auf die relevantesten Aspekte des Systems
- Verallgemeinerung durch Vernachlässigung von unwesentlichen Eigenschaften
- Verstehen durch systematisches Vergröbern/Verfeinern
  - Verstehen des Ganzen ohne Details der Teilsysteme kennen zu müssen
  - Verstehen von Teilsystemen, ohne das System in seiner Gesamtheit zu kennen

# ENTWURFSPRINZIPIEN – MODULARISIERUNG

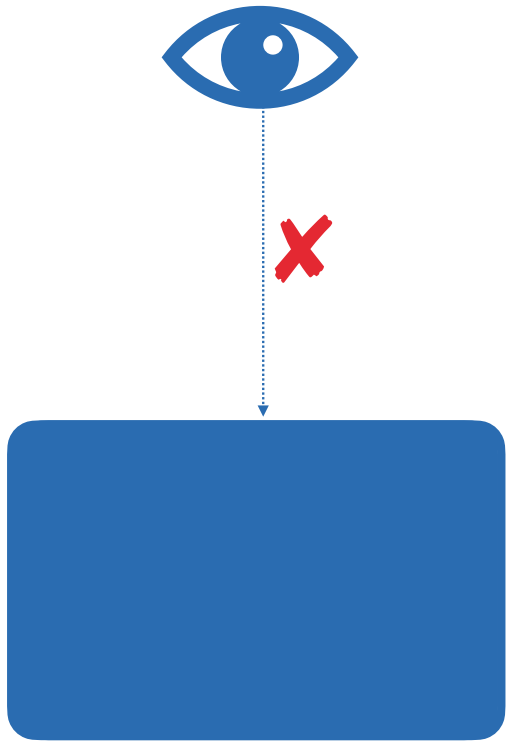


- Definition beherrschbarer, abgegrenzter Einheiten
- Ein Modul
  - ist in sich geschlossen (internes ist von außen nicht sichtbar)
  - hat möglichst wenige, wohldefinierte Interaktionen zu anderen Modulen (schwache Kopplung)
  - ist nur durch Kenntnis seiner Schnittstellen verwendbar
  - hat intern einen starken Zusammenhang (Kohäsion)

# ENTWURFSPRINZIPIEN – MODULARISIERUNG

## Definition beherrschbarer, abgegrenzter Einheiten

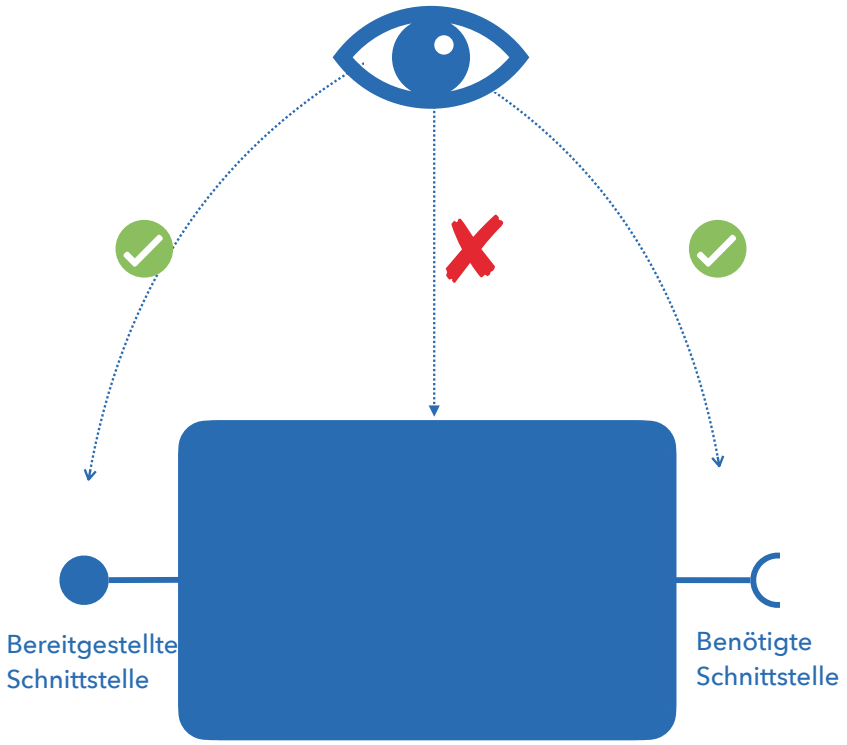
- Ein Modul
  - ist in sich geschlossen (internes ist von außen nicht sichtbar)
  - hat möglichst wenige, wohldefinierte Interaktionen zu anderen Modulen (schwache Kopplung)
  - ist nur durch Kenntnis seiner Schnittstellen verwendbar
  - hat intern einen starken Zusammenhang (Kohäsion)



# ENTWURFSPRINZIPIEN – MODULARISIERUNG

## Definition beherrschbarer, abgegrenzter Einheiten

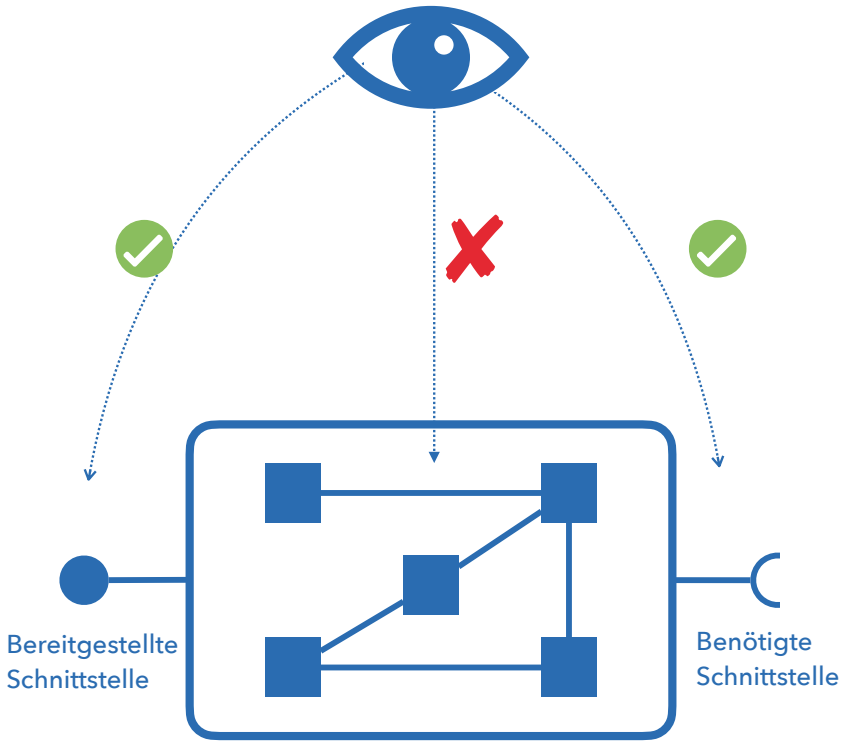
- Ein Modul
  - ist in sich geschlossen (internes ist von außen nicht sichtbar)
  - hat möglichst wenige, wohldefinierte Interaktionen zu anderen Modulen (schwache Kopplung)
  - ist nur durch Kenntnis seiner Schnittstellen verwendbar
  - hat intern einen starken Zusammenhang (Kohäsion)



# ENTWURFSPRINZIPIEN – MODULARISIERUNG

## Definition beherrschbarer, abgegrenzter Einheiten

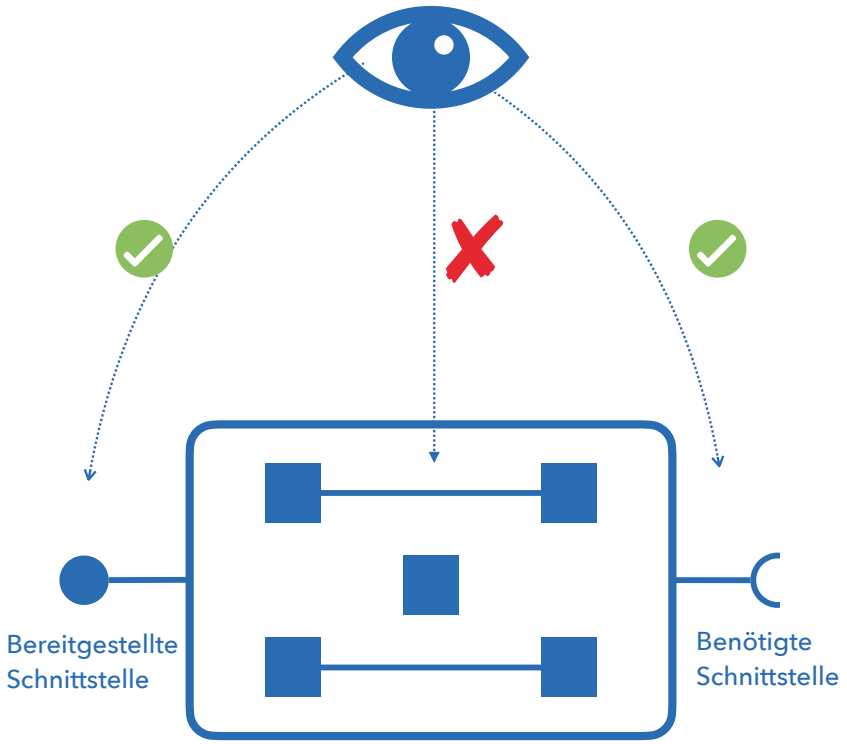
- Ein Modul
  - ist in sich geschlossen (internes ist von außen nicht sichtbar)
  - hat möglichst wenige, wohldefinierte Interaktionen zu anderen Modulen (schwache Kopplung)
  - ist nur durch Kenntnis seiner Schnittstellen verwendbar
  - hat intern einen starken Zusammenhang (Kohäsion)



# ENTWURFSPRINZIPIEN – MODULARISIERUNG

## Definition beherrschbarer, abgegrenzter Einheiten

- Ein Modul
  - ist in sich geschlossen (internes ist von außen nicht sichtbar)
  - hat möglichst wenige, wohldefinierte Interaktionen zu anderen Modulen (schwache Kopplung)
  - ist nur durch Kenntnis seiner Schnittstellen verwendbar
  - hat intern einen starken Zusammenhang (Kohäsion)

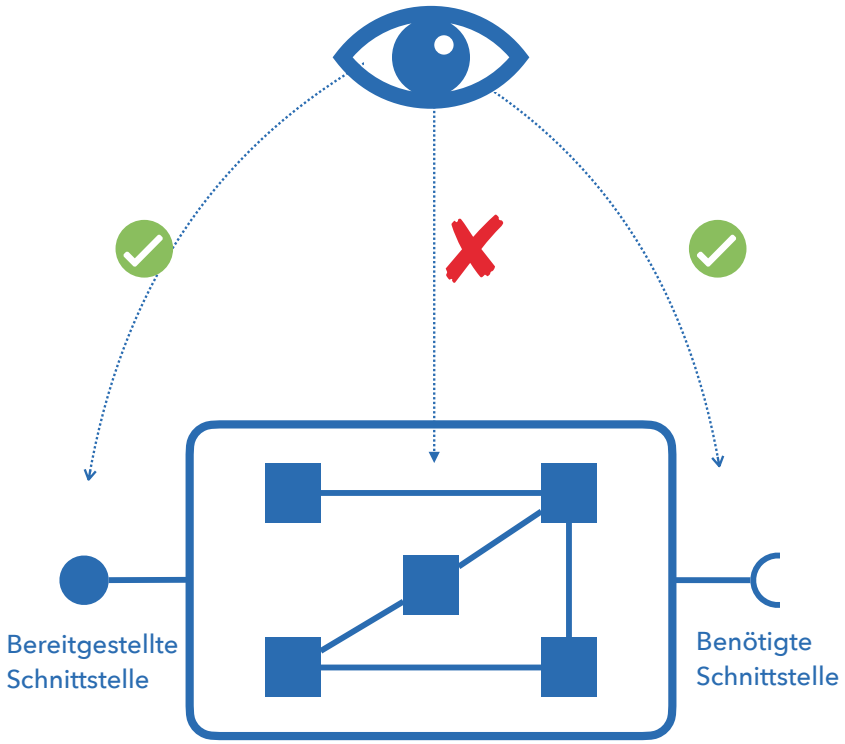


# ENTWURFSPRINZIPIEN – MODULARISIERUNG



## Definition beherrschbarer, abgegrenzter Einheiten

- Ein Modul
  - ist in sich geschlossen (internes ist von außen nicht sichtbar)
  - hat möglichst wenige, wohldefinierte Interaktionen zu anderen Modulen (schwache Kopplung)
  - ist nur durch Kenntnis seiner Schnittstellen verwendbar
  - hat intern einen starken Zusammenhang (Kohäsion)



# ENTWURFSPRINZIPIEN – KAPSELUNG

- Die Schnittstelle eines Moduls beschreibt nur WAS das Modul leistet
  - Entspricht einem Vertrag, der Rechte und Pflichten von Nutzer und Anbieter beschreibt
- WIE das Modul seine Leistung erbringt ist nach außen nicht bekannt
  - Implementierung von Funktionen
  - Repräsentation des internen Zustands –> Attribute
- Ein Modul erbringt seine Leistung vollständig selbst, ggf. unter expliziter Nutzung von Schnittstellen anderer Module

# ENTWURFSPRINZIPIEN – WIEDERVERWENDUNG



- Verwendung bestehender Software / Module statt Neuentwicklung
- Möglichkeiten
  - Bestehende Software, die den Großteil der Anforderungen abdeckt und erweitert bzw. konfiguriert werden kann
  - Nutzung bestehender Teilsysteme bzw. Bibliotheken
  - Nutzung von Frameworks

# ENTWURFSPRINZIPIEN – NEBENLÄUFIGKEIT



- Welche Aufgaben können / müssen gleichzeitig erledigt werden?
- Nebenläufigkeit: Gleichzeitige (parallele oder verzahnte) Ausführung von Anweisungen
- Nebenläufige Aufgaben können durch Prozesse oder Threads ausgeführt werden
- Vorteil: Potentiell höhere Performance durch parallele Ausführung
- Nachteil: Höhere Komplexität

# ENTWURFSPRINZIPIEN – NEBENLÄUFIGKEIT

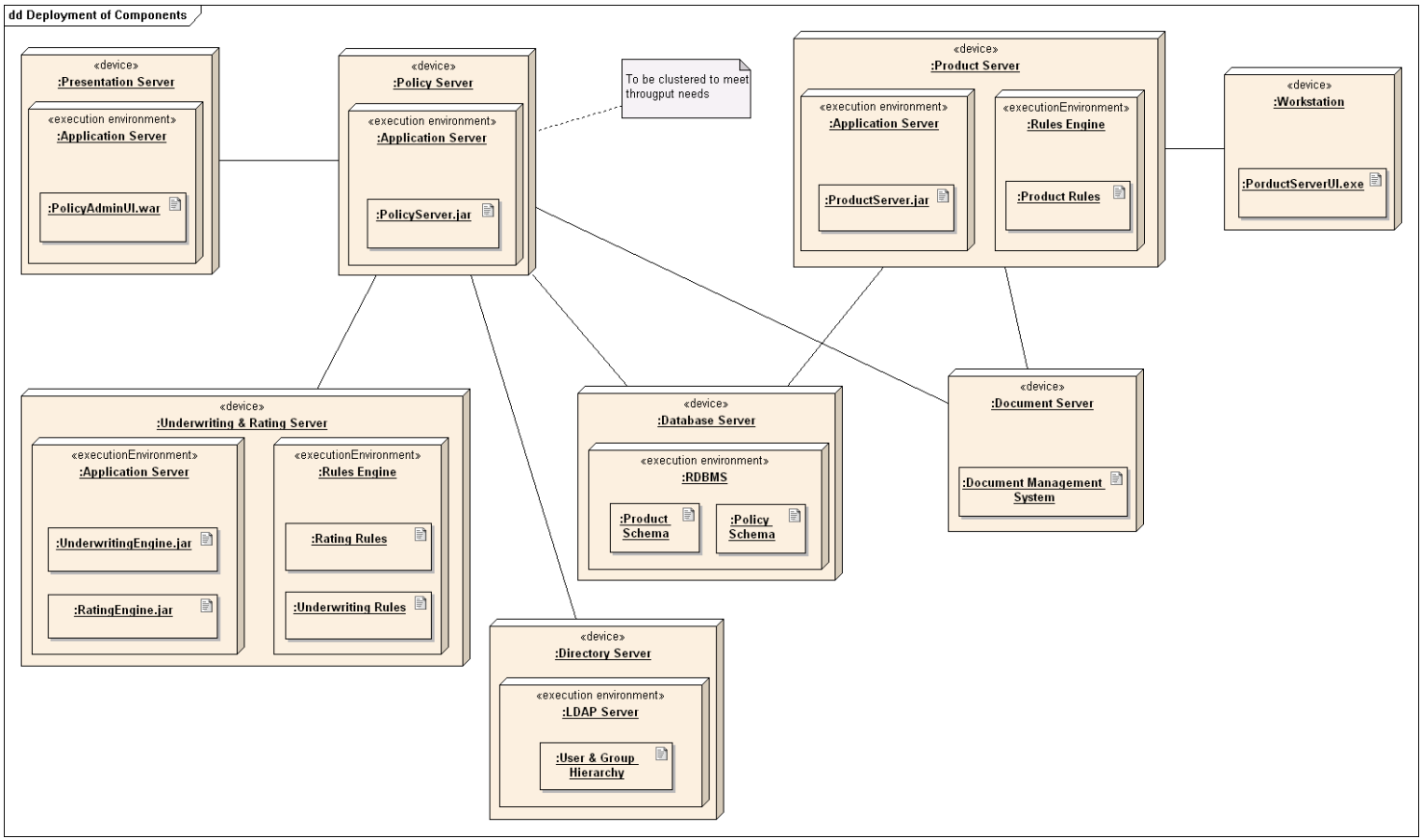


- Entwurfsaufgaben:
  - Welche Aufgaben sollen parallel ausgeführt werden können?
  - Wie werden Aufgaben auf Prozesse verteilt?
  - Wie kommunizieren die Prozesse untereinander?

# VERTEILUNG DES SYSTEMS

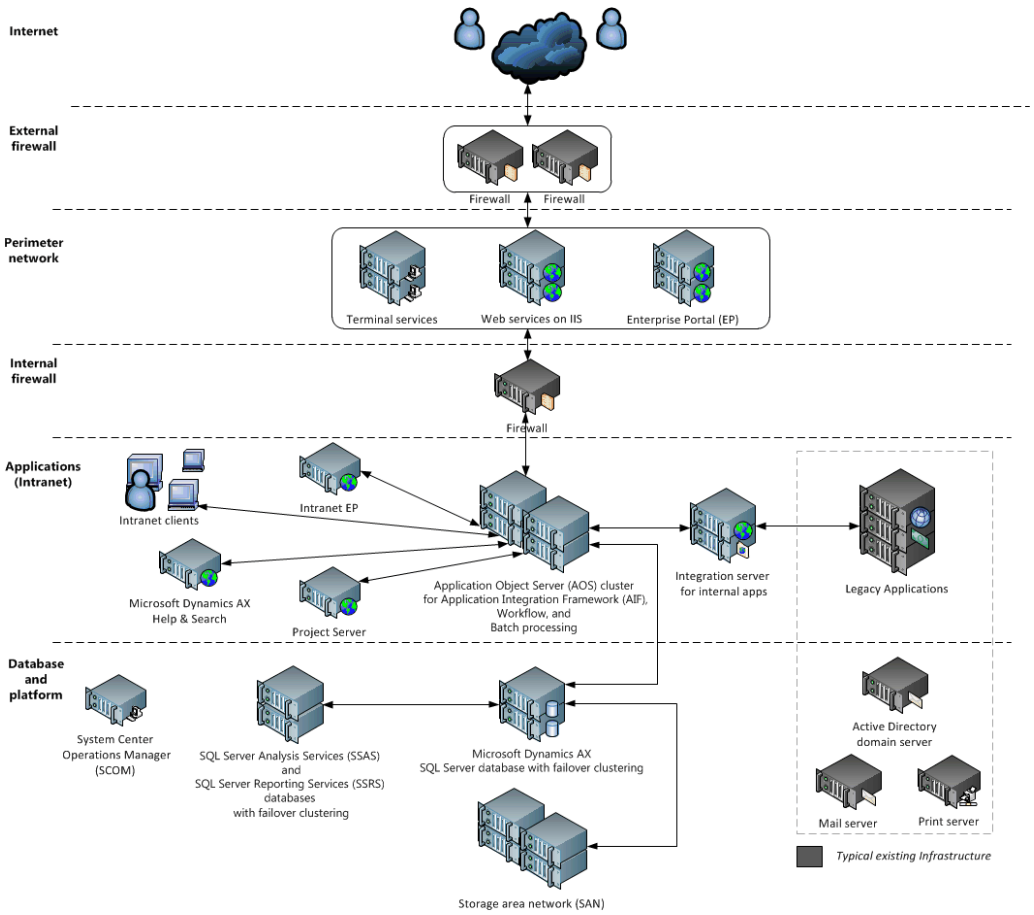
- Bei großen verteilten muss festgelegt werden wie Teilsysteme auf unterschiedliche Hardware verteilt werden sollen / dürfen
- Weitere Fragestellung: Welche Teilsysteme sollen redundant ausgelegt werden, um Zuverlässigkeit bzw. Performance zu erhöhen?

# VERTEILUNG DES SYSTEMS




Quelle: https://en.wikipedia.org/wiki/Deployment_diagram

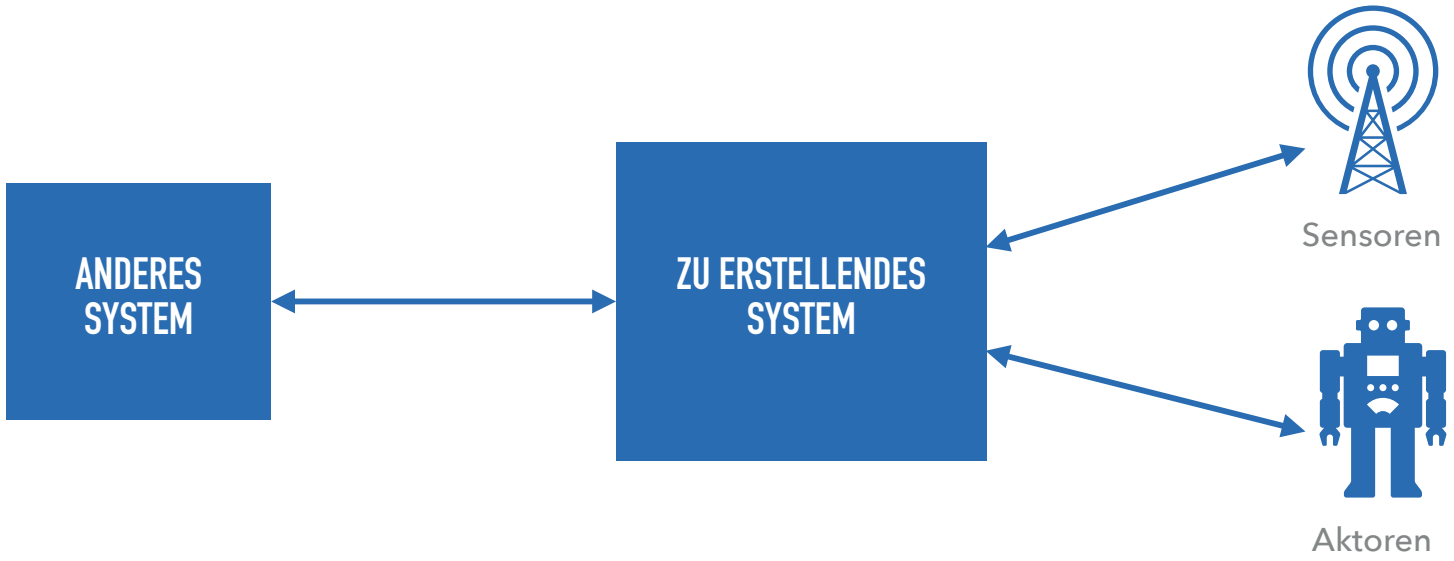
# VERTEILUNG DES SYSTEMS




Quelle: https://docs.microsoft.com/fi-fi/dynamicsax-2012/appuser-itpro/large-scale-deployment

# SCHNITTSTELLEN ZU EXTERNEN SYSTEMEN

- Wie sehen Schnittstellen zu externen Systemen aus?
- Welche Teilsysteme übernehmen die Kommunikation mit externen Systemen?



# BEHANDLUNG VON GRENZBEDINGUNGEN

- Initialisierung, Terminierung, Absturz
- Behandlung von Dingen, die eigentlich nicht auftreten dürften (Exception Handling)
- Sammlung von Information über Fehlersituationen (Logging, Monitoring)

# ERFÜLLBARKEIT DER GEFORDERTEN QUALITÄTSMERKMALE

- Welche Anforderungen sind die wichtigsten? Mögliche Prioritäten z.B.:
  - Speichereffizienz vor Schnelligkeit
  - Benutzerfreundlichkeit vor Schnelligkeit
  - Wartbarkeit hat höchste Priorität
  - Portabilität und Verständlichkeit vor schneller Entwicklung
- Systemweite Prioritäten sollen vermeiden, daß in einzelnen Teilsystemen entgegengesetzte Ziele verfolgt werden

# VORGEHEN BEI ENTSCHEIDUNGEN

- Betrachtung des gesamten Lösungsraums
- Bei Betrachtung mehrerer Lösungsvarianten
  - Wie kritisch ist die Lösung für das Gesamtsystem?
  - Wieviel kann ich in die Evaluation der Lösung investieren?
- Bei kritischen Entscheidungen Varianten möglichst gut evaluieren (z.B. durch geeignete Modelle oder Prototypen)
- Entscheidungen dokumentieren (inkl. der nicht gewählten Alternativen)
- Bei Kostenbetrachtungen sämtliche Kosten über den gesamten Lebenszyklus der Software betrachten

# MODELLE DES SYSTEMENTWURFS

## ÜBERSICHT

- INFORMELLE DIAGRAMME, KEIN UML

## TEILSYSTEME UND IHRE BEZIEHUNGEN

- UML-KOMPONENTENDIAGRAMME
- ARCHITECTURE DESCRIPTION LANGUAGES (ADL)

## SCHNITTSTELLEN UND IHRE NUTZUNG

- KLASSENDIAGRAMME
- SEQUENZDIAGRAMME / ZUSTANDSDIAGRAMME

## VERTEILUNG

- UML-DEPLOYMENT-DIAGRAMME
- TOPOLOGIE-DIAGRAMME