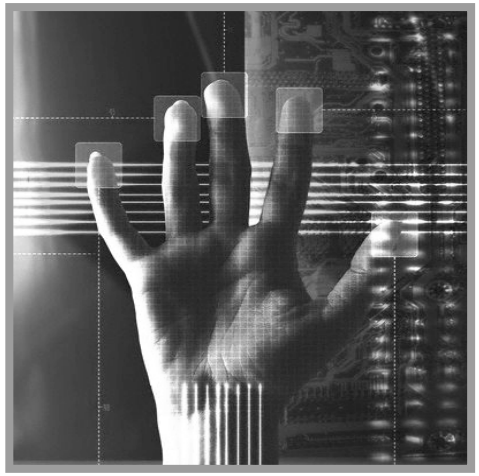
# Software Engineering und Projektmanagement



## Entwurf

**2015W - 29. Oktober 2015**

Andreas Mauczka

Email: andreas.mauczka@inso.tuwien.ac.at
Web: http://www.inso.tuwien.ac.at/lectures/

**INSO - Industrial Software**
Institut für Rechnergestützte Automation | Fakultät für Informatik | Technische Universität Wien

# Motivation

- **Fachliche Sicht trifft auf technische Realisierung**
- **Abbildung der Anforderungen (Funktional und nicht-Funktional)**
  - Ausgangspunkt meistens Pflichtenheft
- **Liefert „Bauplan“ für Gesamt-System**
- **Spezifiziert Struktur und Verhalten der Software und ihrer Komponenten**
- **Entwurf ist ein Prozess zur Lösung von Problemen und dem Planen einer Software Lösung**
  - Definition: wikipedia.org

# Übersicht

- **Erläuterung der unterschiedlichen Aspekte von Architekturen**
  - Welche Sichtweisen von Architekturen gibt es?
- **Transformation der Anforderungen in eine Architektur**
- **Entwicklung der Architektur**
  - Wie hat sich die Softwarearchitektur mit der Zeit verändert?
- **Objektorientierter Entwurf und Service-orientierter Entwurf**
  - Wie modelliere ich entsprechend?
- **Übersicht über Patterns**
  - Was ist die Motivation hinter Patterns, wo könnten Patterns in Projekten eingesetzt werden?

# Stellenwert des Entwurfs und der Architektur

- **Ziele und Aufgabe des Entwurfs**
  - Entwurf eines Systems, das die Anforderungen des Kunden optimal beschreibt
  - Sorgt für Umsetzung der nicht-funktionalen und funktionalen Anforderungen
  - Beschreibung des Systems auf technischer Ebene
  - Ausgangspunkt für Implementierung
  - Festlegung auf Architektur des Systems
    - Auf Bausteinebene
    - Auf Gesamtsystemebene
  - Kommunikationsmittel zwischen den Projekt-Stakeholdern
  - Aufteilung des Systems in funktional getrennte Blöcke
    - Notwendig um Anforderungen an Skalierbarkeit und Wartbarkeit zu erfüllen

# Grundlagen der Softwarearchitekturen

- **Definition Architektur**
  - Eine Architektur ist eine Beschreibung von System-Strukturen (Datenströme, Modulare Strukturen, Prozess Strukturen usw.) [Bass et al. (1998)]
  - Eine Architektur ist das erste Artefakt zur Analyse der Einhaltung von Qualitätseigenschaften [Bass et al. (1998)]
  - Eine Architektur ist  eine Beschreibung der Beziehung von Komponenten und  Verbindungen [Bass et al. (1998)]
  - Eine Architektur ist die fundamentale Organisation eines Systems, verkörpert durch seine Komponenten, deren Beziehungen und deren Umwelt, sowie die Prinzipien hinter dem Design und der Evolution des Systems [IEEE Standard 1471 (2000)]

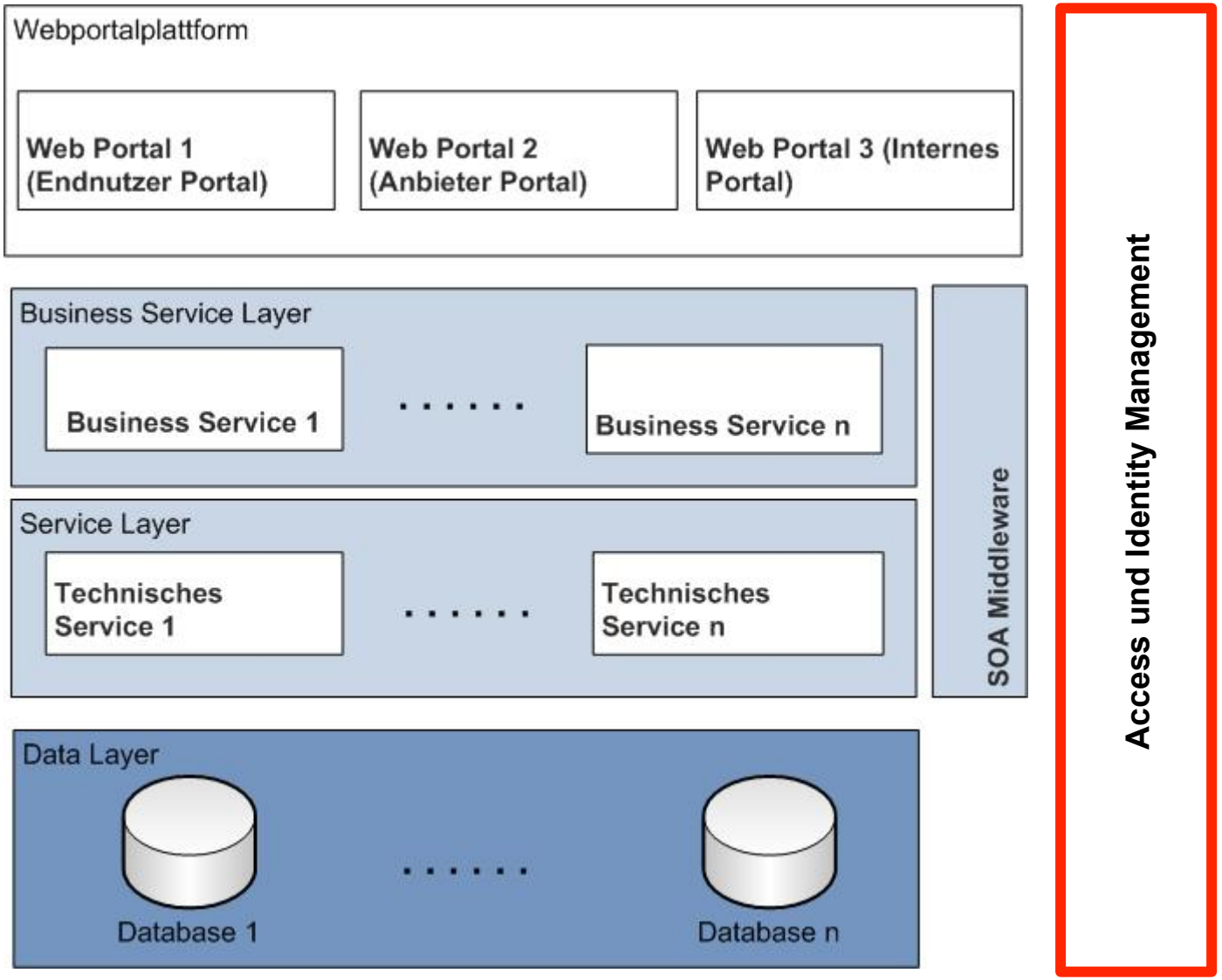
# Grundlagen der Softwarearchitekturen

- **Strategischer Entwurf**
  - Architektur im „Großen“
  - Entwurf auf Systemebene
  - Entscheidungen über Technologie, Gesamtaufbau des Systems, etc.
- **Taktischer Entwurf**
  - Architektur im „Kleinen“
  - Entwurf im klassischen Software Engineering Sinne (auf Bausteinebene, nicht auf Systemebene)
  - Entscheidungen über Aufbau der einzelnen Bausteine (z.B. Entwurf der Klassen, usw.)

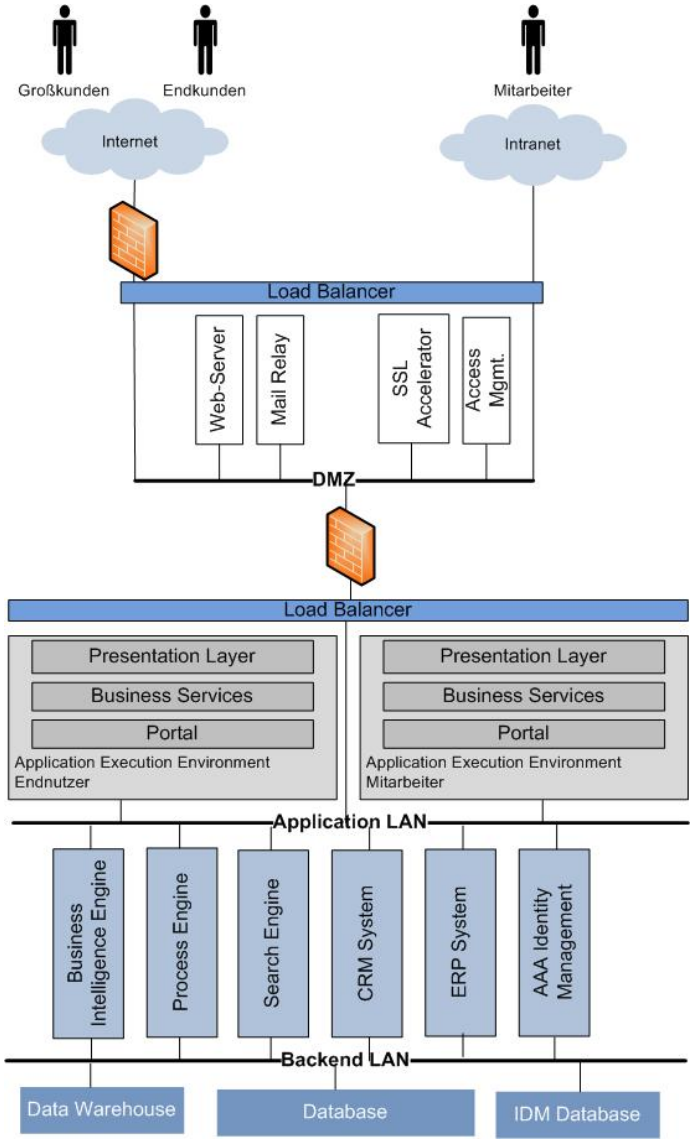
# Sichtweisen auf Architektur

- **Am Beispiel The Open Group Architecture Framework (TOGAF)**
  - Businessarchitektur
  - Informationssystemarchitektur
    - Anwendungsarchitektur
    - Datenarchitektur
  - Technologiearchitektur

# Architektur im Großen und Sichtweisen



# Architektur im „wirklich“ Großen



# Grundlagen der Softwarearchitekturen

- **Von der Anforderung zur Architektur**
  - Architektur ist der Hauptträger der Qualitäten eines Systems – diese sind:
    - Performanz
    - Veränderbarkeit
    - Sicherheit
  - Ziel des Entwurfs: Abbildung eines Systems, das Anforderungen optimal beschreibt
  - Zwei Arten von Anforderungen (funktional und nicht-funktional)
    - Nicht-funktionale Anforderungen (z.B. Zuverlässigkeit, Benutzbarkeit, etc.) sind Faktoren hinter strategischem Entwurf
    - Funktionale Anforderungen (z.B. Verarbeitung von Eingaben, Erstellen von Ausgaben, etc.) sind Faktoren hinter taktischem Entwurf
  - Voraussetzungen um Anforderungen in Architektur umzusetzen
    - Erfahrung des Softwarearchitekten
    - Unterstützung durch Auftraggeber

# „Pflichtenheft“

- **Funktionale Anforderungen**
  - Z.B. in Use Case Form beschrieben
  - Oder als User Story

- **Nicht-Funktionale Anforderungen**
  - In Prosa, in Form von SLAs
  - Wartbarkeit
  - Usability
  - Performance
  - Skalierbarkeit
  - ...

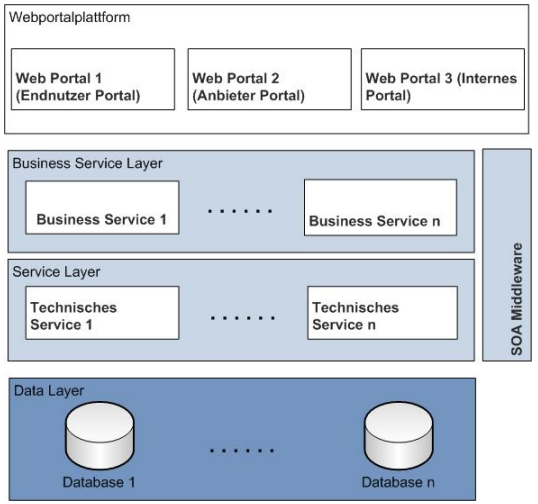
# Umsetzung von Anforderungen, Beispiele

- **Nicht-funktionale Anforderungen**
  - Wartbarkeit
- **Umsetzung: Trennung des Systems in Schichten**

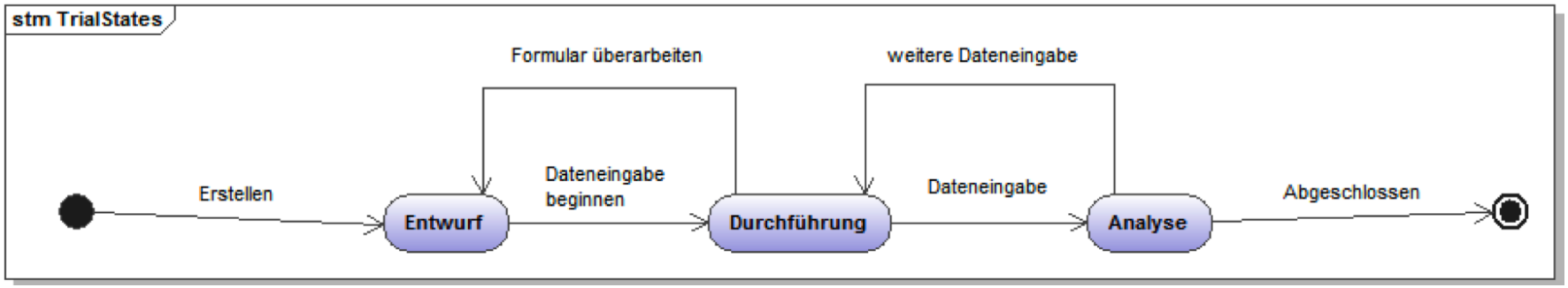
- **Funktionale Anforderung**
  - „Dokumentation durchführen“
  - Eine Dokumentation wird inital erstellt. Nachdem die Dokumentation erstellt wurde, befindet sie sich im Status „Entwurf“. In diesem Status ist die Erstellung und Modifikation von Formularen möglich. Sobald die Formulare fertig entworfen wurden, wird die Dokumentation in den Zustand „Durchführung“ gestellt, bei dem es möglich ist Patienten anzulegen und Daten zu den Formularen einzugeben. Es ist aber jederzeit möglich, den Zustand wieder in den Status „Entwurf“ überzuführen, um nachträgliche Änderungen oder Ergänzungen an den Formularen durchzuführen. Wurde die Dateneingabe abgeschlossen, so wird die Dokumentation in den Zustand „Analyse“ übergeführt. In diesem Zustand ist lediglich die Auswertung (=Export) der Daten möglich. Weder die Formulare noch eingegebene Daten können dann geändert werden.  Auch dieser Zustand ist aber nicht endgültig, d.h. man kann die Dokumentation wieder auf „Durchführung“ setzen, um weitere Daten einzugeben.
- **Umsetzung: Zustandsdiagramm erstellen**

# Beispiele

- **Schichtenarchitektur**




- **Zustandsdiagramm**



# Agile Exkurs

# Architektur im agilen Kontext

- **Emergent Design**
  - Basiert auf Refactoring von erstellten, lauffähigen Komponenten
  - Zuerst Deliverables (Best Practices!) → Dann entsteht Design
  - Beispiele für notwendige Best Practices:
    - Code Conventions
    - Code Reviews
  - High Quality Design ergibt sich aus 4 einfachen Regeln:
  - Test everything; eliminate duplication; express all ideas; minimize entities (YAGNI!!)
- **Rahmenbedingungen trotzdem notwendig**
  - Benachbarte Systeme, existierende Systemlandschaft
  - → strategischer Rahmen muss definiert sein

# Modellieren im agilen Kontext

*„If software development were model building, then the valid measure of the quality of the software or of the development process would be the quality of the models (fidelity, completeness)“* – Alistair Cockburn on Software Modeling

- Probleme von Design Up Front
  - Volatile (oder neue) Anforderungen
  - Nicht jedes Detail kann definiert/bedacht werden
- Barely good enough [1]
- Is Design Dead? [2]

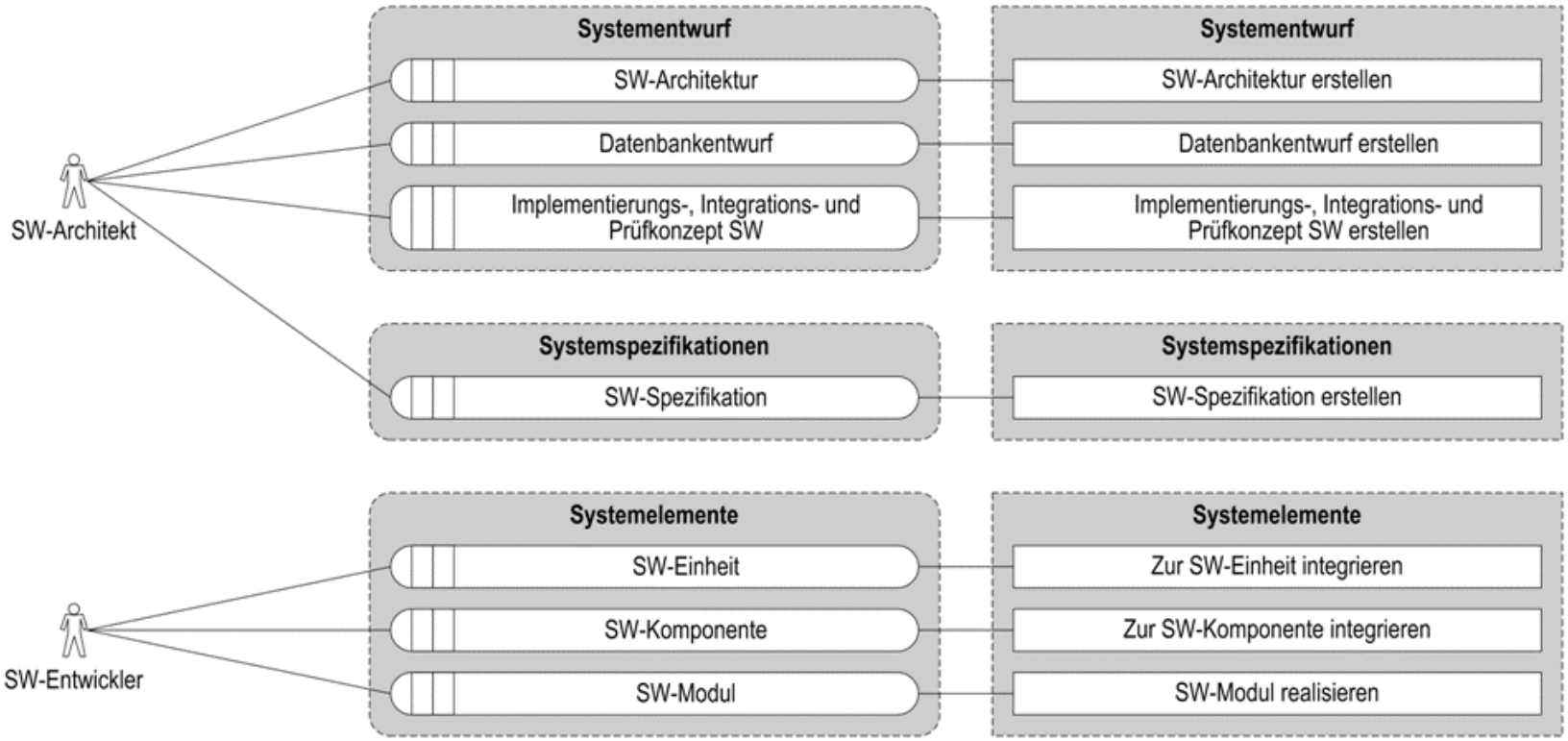
[1] http://www.agilemodeling.com/
[2] http://www.martinfowler.com/articles/designDead.html

# Nicht so schnell..

- **Wie sieht der Entwurf überhaupt aus?**
- **Wann findet er statt?**

# Der Entwurf im V-Modell

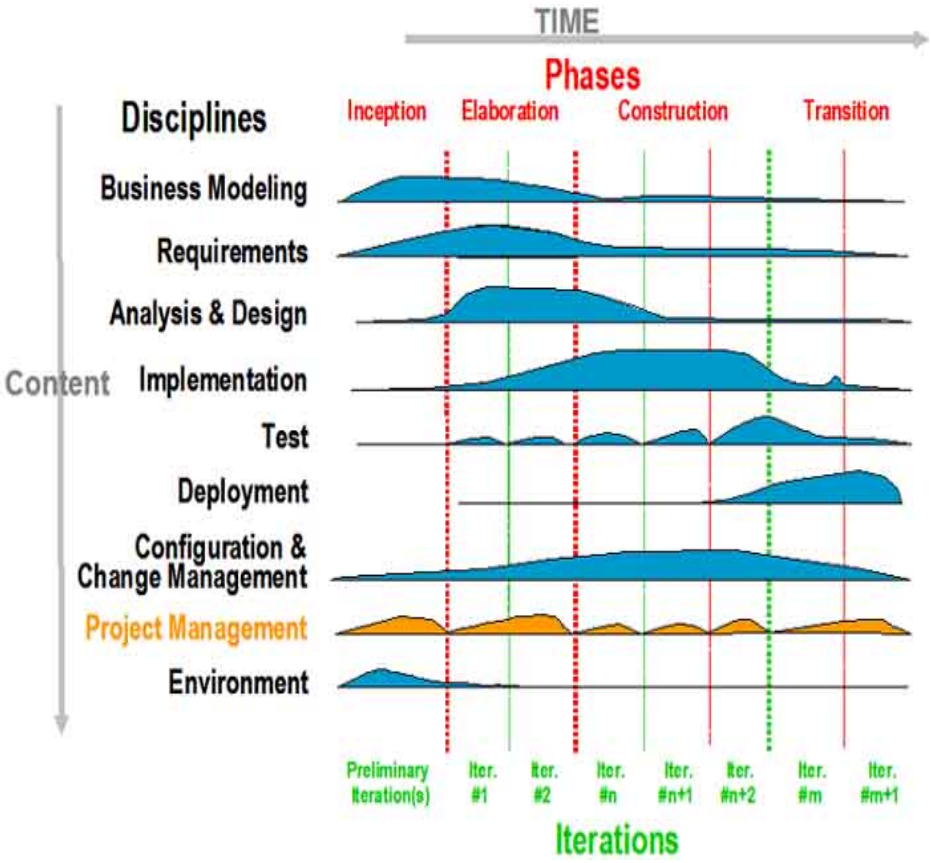
- **Z.B. V-Modell enthält eine Sammlung von Entwurftasks**




- **Quelle:**
  - http://www.cio.bund.de/cln_094/DE/IT-Methoden/V-Modell_XT/v-modell_xt_node.html

# Der Entwurf im RUP

- **Rational Unified Process**
  - Analysis & Design
  - Liefert „Bauplan“ für Software
    - Analysis model
    - Architectural model
    - Design model
  - Iteratives Vorgehensmodell
    - Phasen (Analyse, Entwurf, Implementierung) werden wiederholt durchlaufen!




**Quelle: ibm.com**

# Der Entwurf in einem Vorgehensmodell

- **Unbenanntes Vorgehensmodell**
  - Leitet Pflichtenheft (Software Requirements Specification) in Grob- und Feinspezifikation über
  - Tasks:
    - Erstellen Architekturmodell
    - Erstellen Datenmodell
    - Erstellen Grob- und Feinspezifikation
    - Entwurf der einzelnen Komponenten

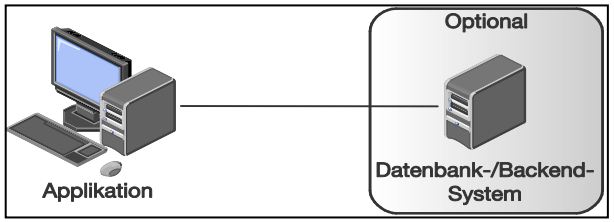
# Entwicklung der Softwarearchitektur

- **Faktoren zur Weiterentwicklung von Softwarearchitektur**
  - Prozedural zu objektorientierten Programmiersprachen
  - Vom Großcomputer zum Heimcomputer
  - Durch objektorientierte Programmiersprachen Trend von monolithischen Applikationen zu modularen Systemen
  - In den 80er, 90er und frühen 2000er Jahren → hauptsächlich *Rich-Client-Applikationen*
  - Durch PHP, Java Server  Pages und ASP.net → Trend zu Webapplikationen
  - Heutzutage Service Oriented Architecture (SOA)
- **Historische Softwarearchitekturen**
  - 2-Tier Architekturen
  - 3-Tier Architekturen
  - Service-orientierte Architekturen

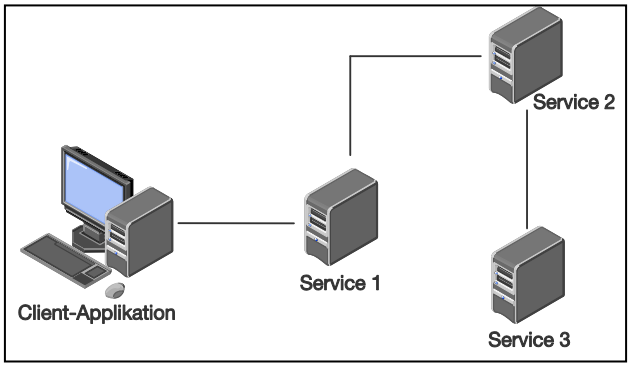
# Entwicklung der Softwarearchitektur

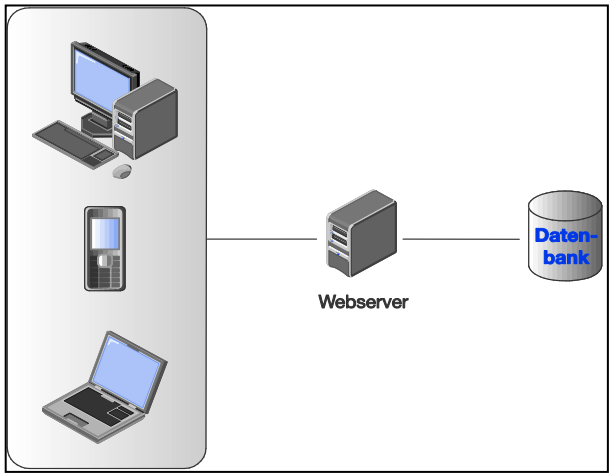
- **Beispiele für Softwarearchitekturen:**

**2-Tier Architektur**




**Service-orientierte Architektur**

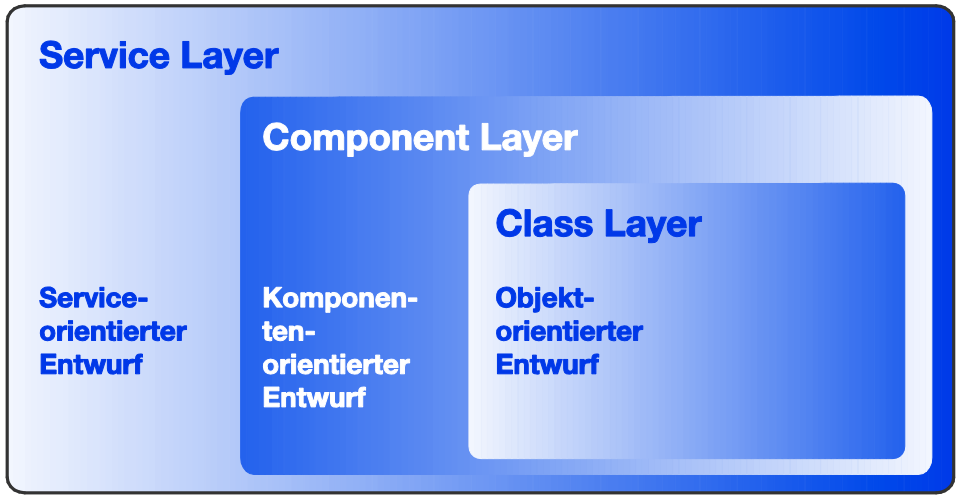



**3-Tier Architektur**

# Entwurfsparadigmen

- **Serviceorientiertes Design**
  - Umspannt mehrere Abstraktionsebenen
    - Service Layer
    - Component Layer
    - Class Layer
  - Unterschiedliche Notationen je Layer
    - Z.B. BPMN (Service Layer), Komponentendiagramme (Component Layer), UML (Class Layer)



# State-of-the-Art Architektur

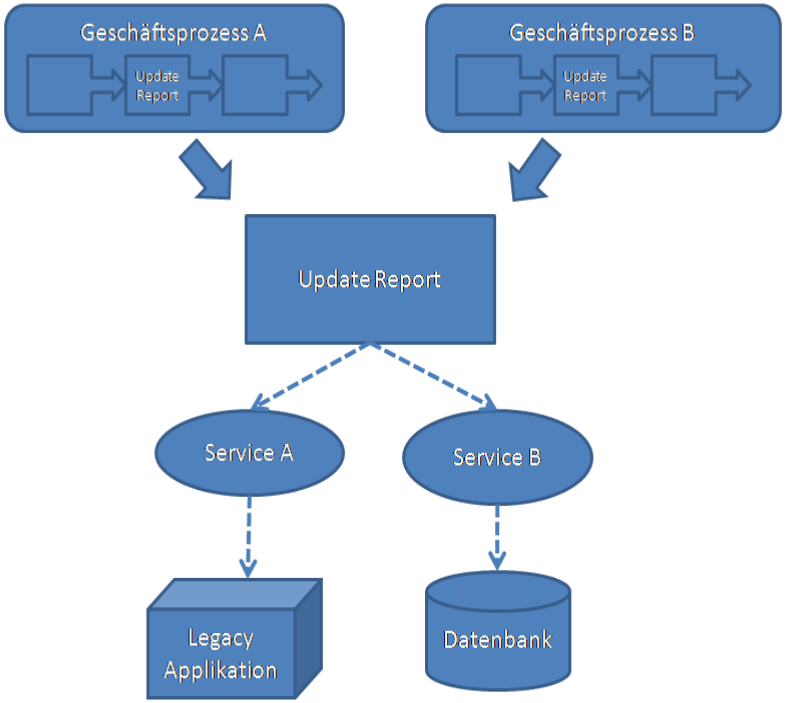
- **Paradigmen der Service-orientierten Architektur**
  - Lose Kopplung
    - Niedrige Kopplung der Services ermöglicht hohen Reuse und Flexibilität
  - Abstraktion
    - Reduktion von Neuentwicklungen – Spannungsfeld Abstraktion vs. Technische Anforderungen (z.B. KÖNNEN manche technische Services einfach nicht abstrakt gelöst werden)
  - Standardisierung
    - Standardisierung erlaubt die Zusammenarbeit von Services
  - Composable
    - Standardisierte Services können zu Komponenten zusammengefasst werden

# BPM und SOA

- **Im Service-orientierten Entwurf wird ein Schritt in einem Geschäftsprozess extrahiert und gekapselt → dabei handelt es sich um ein sog. Business Service**
  - Anmerkung: Business Services können auch Geschäftsziele oder KPI (Key Performance Indicator) realisieren und müssen keinen Schritt im Geschäftsprozess darstellen – z.B. Kostenreduktion einen Report zu generieren kann auch dazu dienen ein Business Service zu definieren
- **Service-orientierte Architekturen sind *business driven*, d.h. sie sind an Geschäftsprozesse angelehnt**
- **Business Process Modelling (BPM) und SOA sind eng miteinander verknüpft – Anwendungen werden maßgeblich von der Geschäftsprozess-Modellierung beeinflusst**

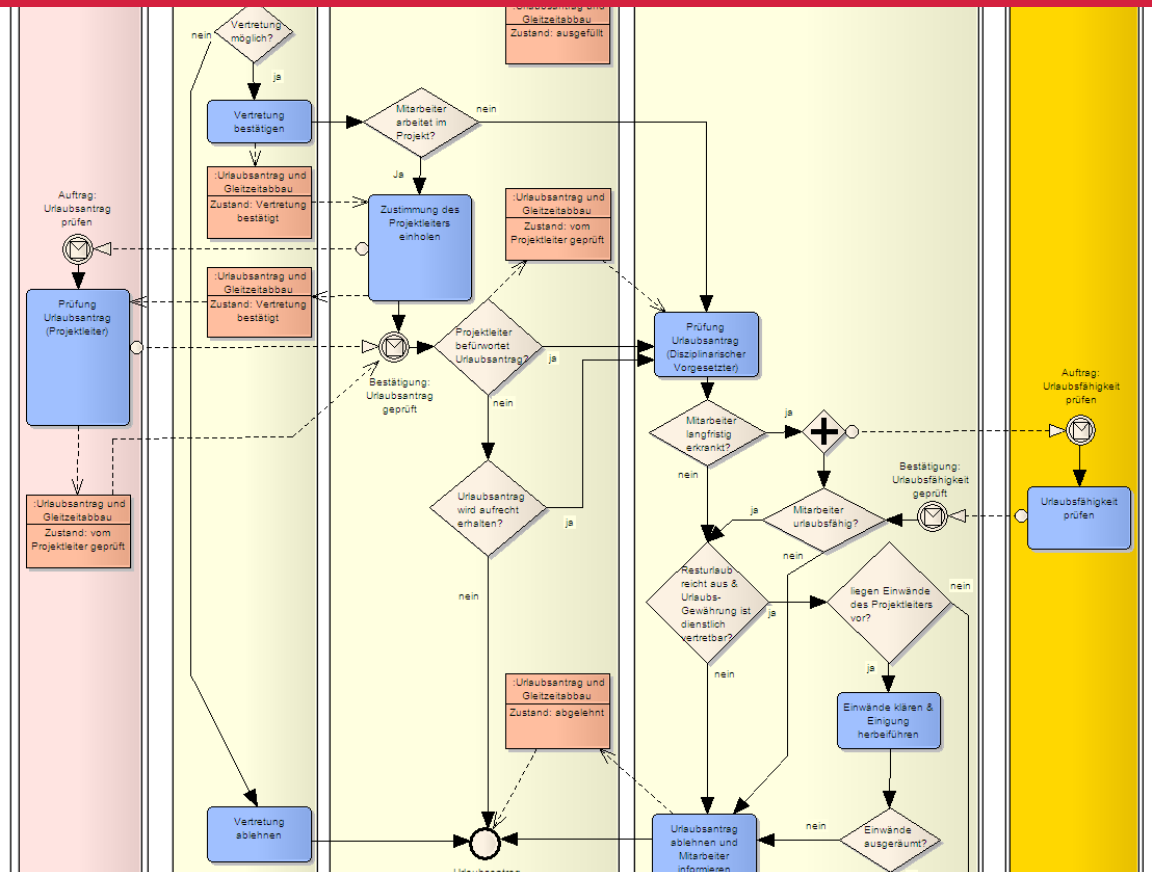
# BPM und SOA

- **Zusammenhang von Business Process Modelling (BPM) und einer SOA**
  - Geschäftsprozess A und B greifen auf das gleiche Business Service „Update Report“ zu
  - „Update Report“ wird in Form von zwei technischen Services gelöst



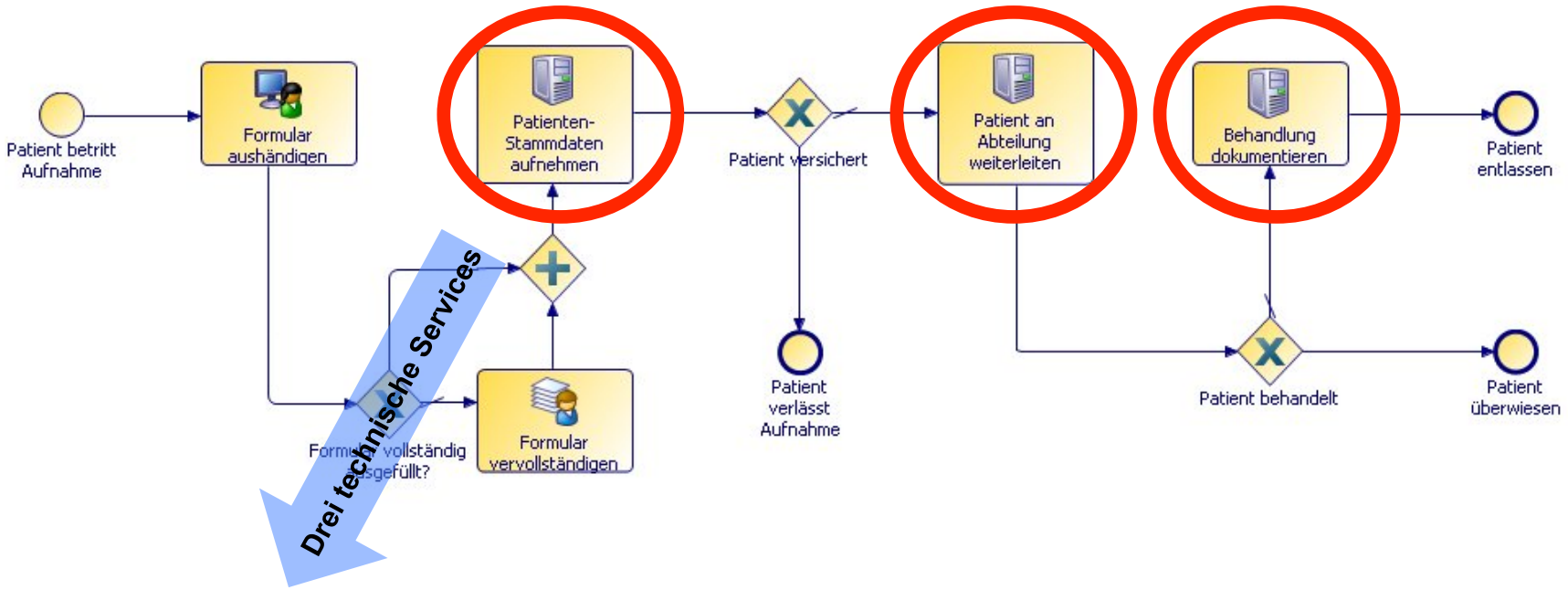
# State-of-the-Art Architektur

- **Service-orientierte Architektur**
  - Geschäftsprozess-orientiert
  - Anwendungen werden durch mehrere Services gelöst
  - Services in einer Anwendung sollen auch aus anderen Anwendungen heraus aufgerufen werden können
    - Reuse
    - Modularisierung
  - Zusammenstellung von Services in bestimmten Abläufen zur Umsetzung von Geschäftsprozessen (Orchestrierung)
  - Services existieren auf mehreren Abstraktionsebenen (Business Service, Component Service, technisches Service, usw.)
  - Beschreibt eine Architektur, keine Technologie
  - Für Interessierte: http://www.activiti.org/ (Free BPM Platform – BPMN 2 Process Engine)




**Beispiel Geschäftsprozess**

# Beispiel zu Services




- **eingabePatientInKIS:** Eingabe der Patientendaten in das Krankenhaus Informationssystem (KIS)
- **abfrageVersichertenStatus:** Abfrage des Versichertenstatus – Aufruf an externes Service
- **anbindungAnZentralesRegister:** Abruf der Behandlungshistorie aus zentralem Register – Verknüpfung mit Patientendaten (Ausgabe: ELAK)

# Entwurfsparadigmen

- **Objektorientiertes Design**
  - Erweiterung der Diagramme und Modelle aus der objektorientierten Analyse
  - Möglichst genaue Abbildung des später zu implementierenden Systems
  - Kein Anspruch auf Vollständigkeit
  - Verwendet folgende Diagramme:
    - Klassendiagramme – Festlegung der Struktur
    - Zustands- und Sequenzdiagramme – Verhalten des Systems
    - Komponentendiagramme – Abbildung der Systemkomponenten und deren Zusammenhang
  - Klassendiagramm im Mittelpunkt
  - Als Notation wird UML verwendet

# Entwurfsparadigmen

- **Vorgehensweise im objektorientierten Entwurf (nach [Booch (1994)])**
  - Klassen und Objekte auf jeder Abstraktionsebene identifizieren
  - Semantiken zwischen Klassen und Objekten festlegen
  - Beziehungen der Klassen und Objekte identifizieren
  - Schnittstellen und Implementationen der Klassen spezifizieren
- **Aufgeführte Schritte teilweise schon in der Phase Analyse durchgeführt**
- **Folgende zwei Punkte sollen erfüllt sein**
  - Festlegen einer Architektur für die Implementierung (logische und physikalische Architektur)
  - Policies festlegen für die Systemkomponenten

# Entwurfsparadigmen

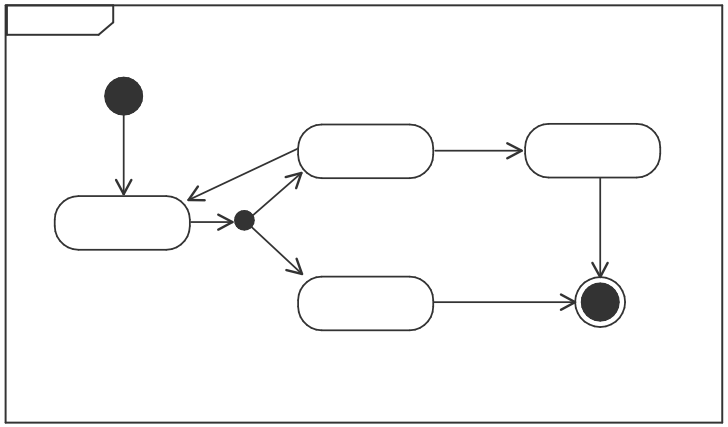
- **Unified Modeling Language (UML) zur Unterstützung des objekt-orientierten Entwurfs**
  - Modellierungssprache der OMG für komplexe und große Systeme
  - Basiert auf objekt-orientierter Entwicklung und Design
  - Sinnvolle Nutzung der UML → gute Dokumentationsbasis
- **Zwei Arten von Modellen**
  - Verhaltensmodelle – beschreiben dynamisches Verhalten, z.B. Aktivitätsdiagramme, Zustandsdiagramme, Sequenzdiagramme
  - Strukturmodelle – beschreiben statische Teile des Systems, z.B. Klassendiagramm, Komponentendiagramm, Verteilungsdiagramm

# Entwurfsparadigmen

- **Beispiel für statisches Diagramm**
  - Klassendiagramm
  - Beschreibt die Struktur eines Systems in Form von Klassen
  - Eine Klasse ist eine Zusammenfassung gleicher Objekte in Bezug auf Eigenschaften und Fähigkeiten

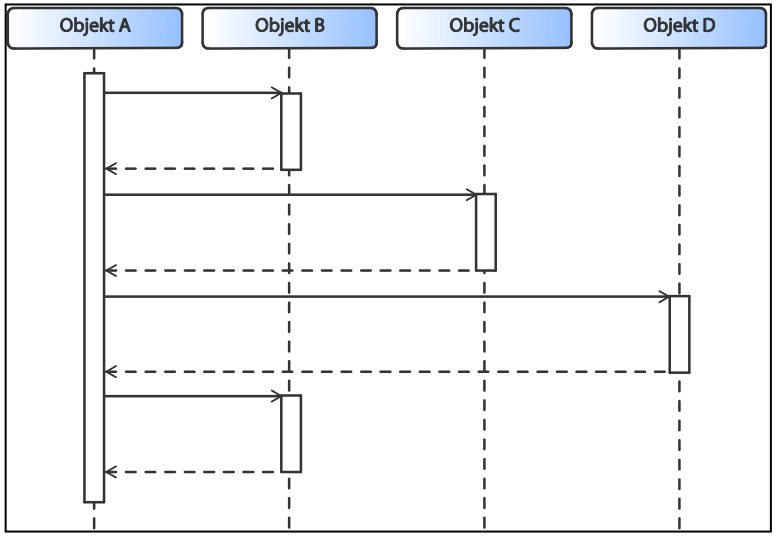
# Entwurfsparadigmen

- **Beispiel für dynamisches Diagramm**
  - Zustandsdiagramm
  - Beschreibt mögliche Systemzustände und bildet Zustandsänderungen sowie auslösende Ereignisse ab
  - Abbildung in Form eines endlichen Zustandsautomaten

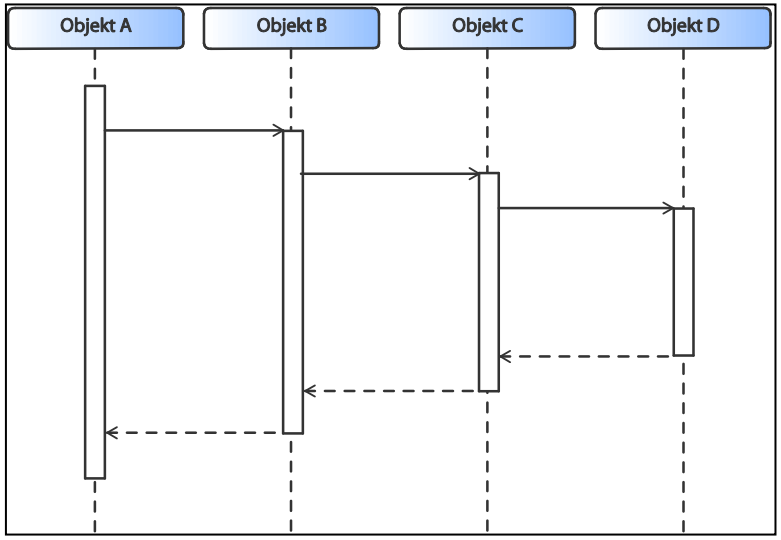


# Entwurfsparadigmen

- **Kontrollfluss am Beispiel Stair vs. Fork**
  - Fork – Zentrales Objekt übernimmt Kontrolle
  - Stair – Kapselung in den Schritten




**Fork**




**Stair**

# Architekturmuster

- **Designpatterns**
  - Beschreiben wiederholt auftretende Strukturen von miteinander kommunizierenden Komponenten
  - Dienen dazu ein allgemein definiertes Problem innerhalb eines bestimmten Kontexts zu lösen
  - Patterns auf Abstraktionsebene unter Architekturpatterns
- **Ein Beispiel zu Designpatterns in Einheit Implementierung**

# Patterns allgemein

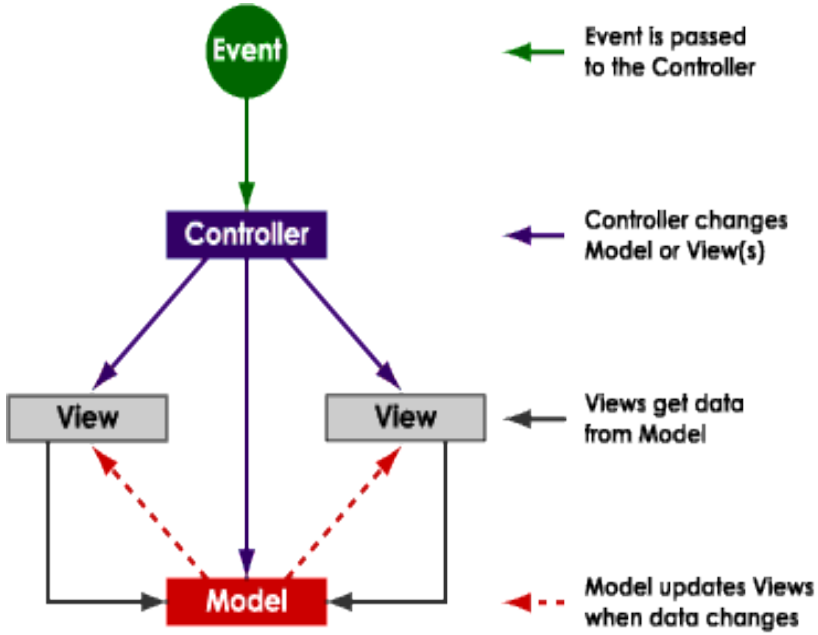
- **Patterns stellen allgemeine Lösungsstrategien für wiederkehrende Probleme der Softwareentwicklung dar**
  - Abstrakt
  - Weitestgehend technologie-unabhängig
  - Wesentlicher Input in Entwurfsentscheidungen
- **Zwei Arten von Patterns nach Abstraktionsebene**
  - Architekturpatterns
    - Muster auf architektonischer Ebene (Makroarchitektur-Ebene)
  - Designpatterns
    - Muster auf Klassen-Ebene (Mikroarchitektur)
- **Pattern Language**
  - Sammlung von Entwurfsmustern
  - Schaffung einer einheitlichen Sprache für Probleme und Lösungen

# Architekturmuster

- **Architektur-Patterns**
  - Schablonen für konkrete Softwarearchitekturen
  - Spezifizieren systemweite, strukturelle Eigenschaften einer Anwendung
  - Großer Einfluss auf die Architektur der eigenen Subsysteme
  - Fundamentale Designentscheidung
  - Literaturhinweis: *Pattern-oriented Software Architecture – Buschmann et al. (1996)*
- **Bekannte Architekturmuster**
  - Model-View-Controller
  - Pipes and Filters
  - Layered Architecture

# Architekturmuster

- **Model View Controller**
  - Szenario: Interaktive Anwendungen mit einer flexiblen Schnittstelle zwischen Mensch und Computer
  - Problem: GUI wird oft geändert
  - Lösung: Aufteilung in Verarbeitung, Output und Input
- **Model kapselt Daten und Funktion**
- **View stellt Informationen dar**
- **Controller empfängt Events**



# Zusammenfassung

- **Historische Entwicklung des Entwurfs von Softwaresystemen**
  - 2-Tier → 3-Tier → Service-orientierte Architektur
- **Im Entwurf werden Entscheidungen zur Mikro- und Makroarchitektur getroffen (taktisch vs. strategischer Entwurf)**
- **Mehrere Sichten und Abstraktionsebenen einer Architektur, je nach betrachtender Rolle**
- **Anforderungen aus der Analysephase müssen in einen Entwurf transformiert werden**
- **Architekturpatterns sollen den Entwurf vereinfachen und bekannte Probleme effizient lösen**