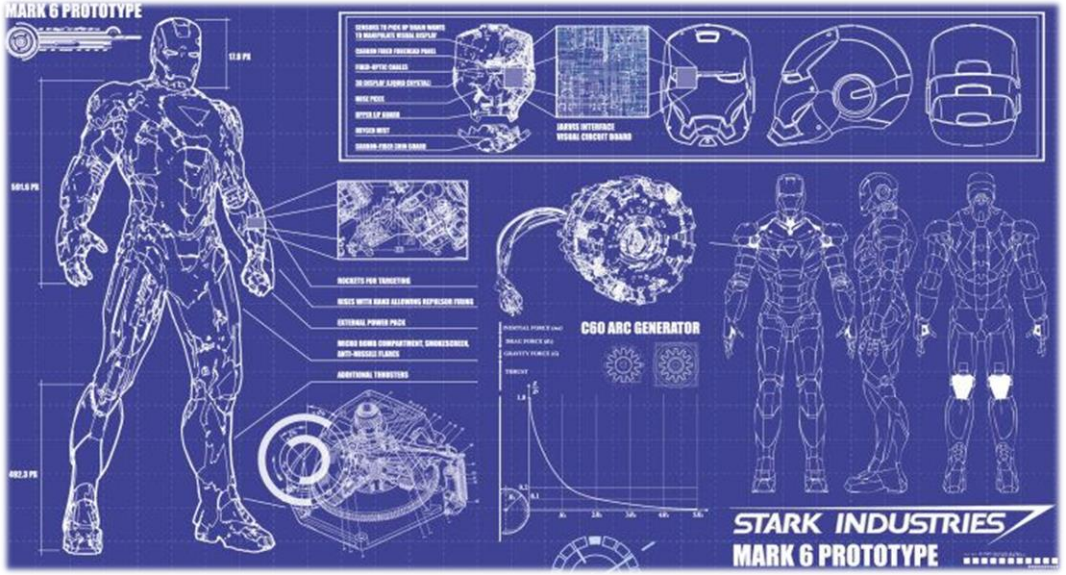
Softwarearchitektur



Jean-Philipp Burnus

Jan Wübbels

Niklas Borgmann

Richard Purk

Agenda

1. **Einleitung**
2. **Einflussfaktoren**
3. **Architektur**
4. **Anwendungen**

Agenda

1. **Einleitung**
2. **Einflussfaktoren**
3. **Architektur**
4. **Anwendungen**

Software

- Software ist eine Kreation von Teilen von Software (Douglas Bell)
  - Siehe z.B. Java

- Software in allen Bereichen des Lebens
  - z.B. Autos, Toaster, Nest Labs

Softwarearchitektur

Was ist Softwarearchitektur?

- Zusammenspiel und Aufbau von Komponenten
- Zerlegung des Gesamtsystems
- Planung von Software
- Entwurf – Softwareblaupause

These

Quick and Dirty

Zeit

Komplex

Softwarearchitektur wird immer bedeutsamer

Koordination

Kosten

Projekt-Teams

Agenda

1. **Einleitung**
2. **Einflussfaktoren**
3. **Architektur**
4. **Anwendungen**

Einflussfaktoren

- Anforderungen
- Größe des Projekts
- Team
- Erfahrung
- Kosten

Einflussfaktoren

- Funktionale Einflussfaktoren
  - Ein Zins soll errechnet werden, …
- Nicht Funktionale Einflussfaktoren
  - Das Programm soll erweiterbar sein, …
- Randbedingungen
  - Oracle ist bereits vorhanden, …

Einflussfaktoren

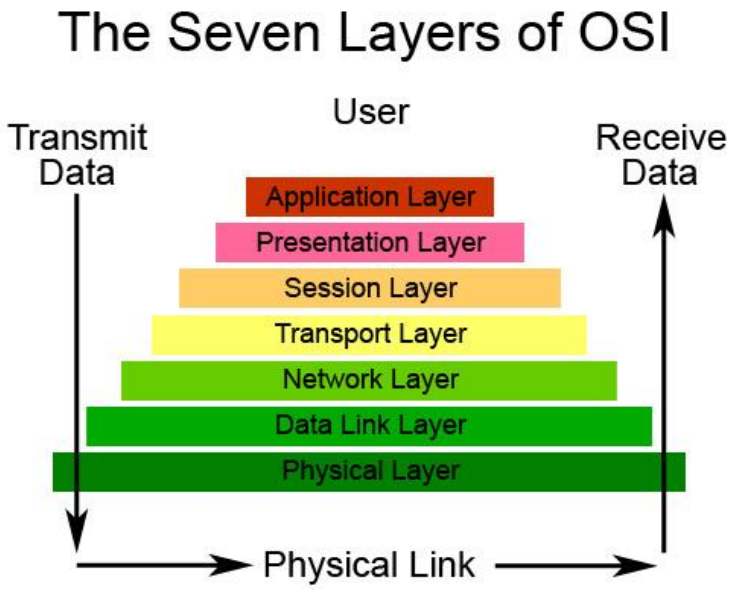
- Qualität von Software – IEC/ISO 9126-Standard
  - Funktionalität
  - Zuverlässigkeit
  - Benutzbarkeit
  - Effizienz
  - Änderbarkeit
  - Übertragbarkeit
- Weitere Qualitätsmerkmale
  - Größe
  - Geschwindigkeit
  - …

Agenda

1. **Einleitung**
2. **Einflussfaktoren**
3. **Architektur**
4. **Anwendungen**

Schichtenmodell

- Strukturierungsprinzip für die Software-Architektur
- Aufteilung des Systems in Schichten
  - Verringerung der Komplexität der Abhängigkeit
  - Hohe Kohäsion
- Unterscheidung in:
  - Zweischichtiger Architektur (z.B. Client/Server)
  - Dreischichtiger Architektur (z.B. MVC)
  - Mehrschichtiger Architektur (z.B. OSI → 7-Schichten)

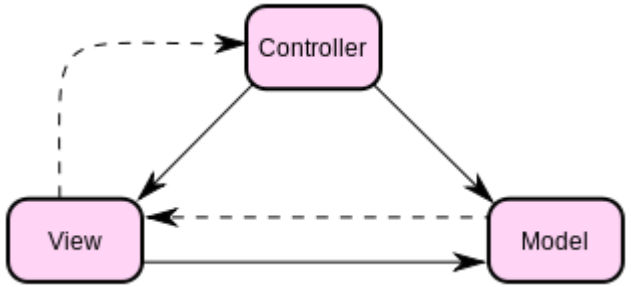


Schichtenmodell

Schichten

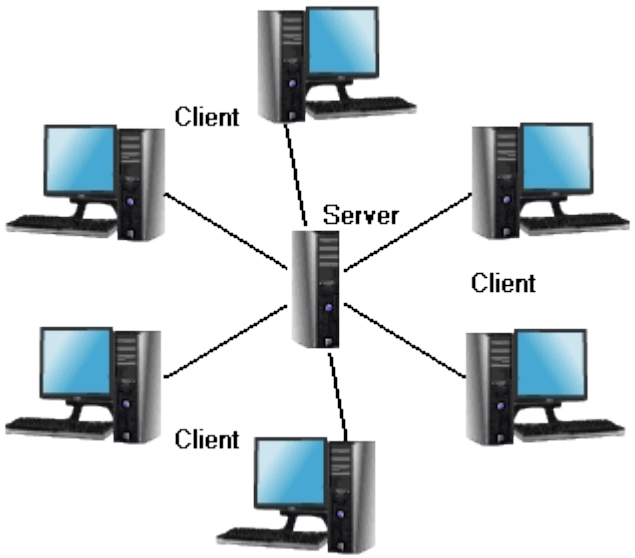
- zweischichtige Architektur
  - nur die höhere hat Zugriff auf die niedrigere Schicht

- dreischichtige Architektur (MVC)
  - Datenhaltung (Model)
  - Präsentation (View)
  - Logik (Controller)



Client / Server

- Aufgabenverteilung im Netzwerk
  - Zentraler Server stellt Dienste zur Verfügung
  - Client nutzt Dienste
- Thin-Client
  - Verlagert Datenhaltung und Verwaltung auf dem Server
  - Nur zuständig für Darstellung
- Fat-Client
  - Nur Datenhaltung auf dem Server
  - Anwendung und Darstellung auf dem Client



Client / Server

Kundendatenbank (Großrechner)

Clients (Kunden)

Peer-to-Peer

- Rechner-Rechner Verbindung (P2P)
- Gleichberechtigung der Peers
  - Inanspruchnahme/Verfügungsstellung von Diensten

- Peers halten eigenen Daten
  - Peer sucht Datei → Wenn Datei gefunden → Peer-to-Peer Übertragung
    - Tauschbörse(Kazaa, Gnutella, …)

Peer-to-Peer

Systeme

- Unstrukturierte
  - Zentralisiert P2P: Server zur Verwaltung (Napster)
  - Reine P2P: keine Verbindung zu fremden IP‘s nur zu bekannten
    - Web-of-Trust Netzwerk
  - Hybrid P2P: Dynamische Bestimmung von Servern zur Verwaltung
- Strukturiert
  - Erzeugt eine bekannte Topologie
    - Dies erlaubt eine zielgerichtete „Suche“ nach Inhalten
  - Auch bekannt als „Distributed Hash Table “(DHT)

Agenda

1. Einleitung
2. Einflussfaktoren
3. Architektur
4. Anwendungen

Modularisierung

Modularität im Allgemeinen bedeutet ein Ganzes in viele verschiedene Teile, die auch als Komponenten oder Bausteine bezeichnet werden können, aufzuteilen.

Modularisierung

Kopplung

- Verknüpfung von verschiedenen Komponenten
- Maß für die Stärke dieser Verknüpfungen
- Lose Kopplung
  - Geringer Grad an Abhängigkeiten zwischen Software-Komponenten
  - Änderungen haben nur Lokal Auswirkungen
- Enge Kopplung
  - Hoher Grad an Abhängigkeiten zwischen Software-Komponenten
  - Änderungen bewirken Anpassungen in anderen Komponenten

Kohäsion

- Abbildung einer logischen Aufgabe in einer Komponente
- Starke Kohäsion
  - Eine Komponente ist für genau eine wohldefinierte Aufgabe zuständig
- Schwache Kohäsion
  - Mehrere Komponenten für eine Aufgabe
  - Funktionalitäten können nicht wiederverwendet werden
  - DRY-Prinzip zur Vermeidung

Modularisierung

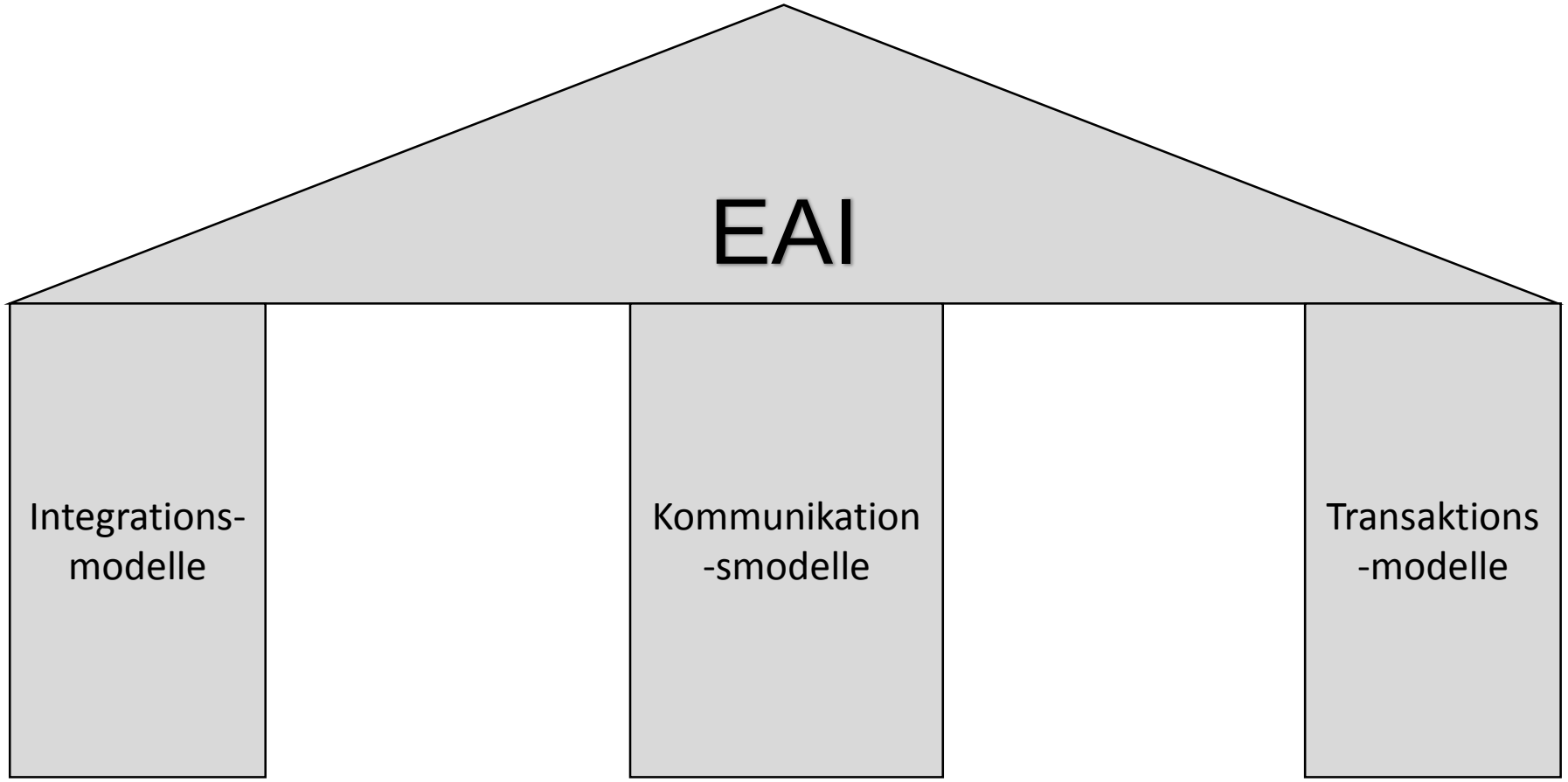
- Vorteile
  - Spezialisierung
  - Kontrollierbarkeit der Komplexität
  - Austauschbarkeit
  - Wiederverwendbarkeit
- Nachteile
  - Aufgabenverteilung
  - Fehler in einer Komponente kann zu Fehlfunktion des Gesamtsystems führen

Enterprise Application Integration

„Konzept zur unternehmensweiten Integration der Geschäftsfunktionen entlang der Wertschöpfungskette, die über verschiedene Applikationen auf unterschiedlichen Plattformen verteilt sind.“

- Stephan Aier, Marten Schönherr

EAI-Architektur



Integrationsmodelle

- Functional Integration Model
  - Integration von Systemen über Funktionen auf der Ebene der Geschäftslogik
- Presentation Integration Model
  - Integration auf Präsentationsebene
- Data Integration Model
  - Integration auf Datenbankebene

Kommunikationsmodelle

- Verantwortlich für Datenaustausch und Kommunikation
- Synchrone Kommunikation
  - Sender schickt Anfrage an Empfänger
  - Sender erwartet ein Ergebnis vom Empfänger
  - Sender ist gesperrt, während Wartezeit
- Asynchrone Kommunikation
  - Nur Transfer von Daten zum Empfänger notwendig
  - Keine Antwort erforderlich

Transaktionsmodelle

- Verhalten bei Fehlerhafter Transaktion
- Einfache Transaktion
  - ACID-Prinzip gilt
  - Daten innerhalb einer Transaktion lesen, ändern und wegschreiben

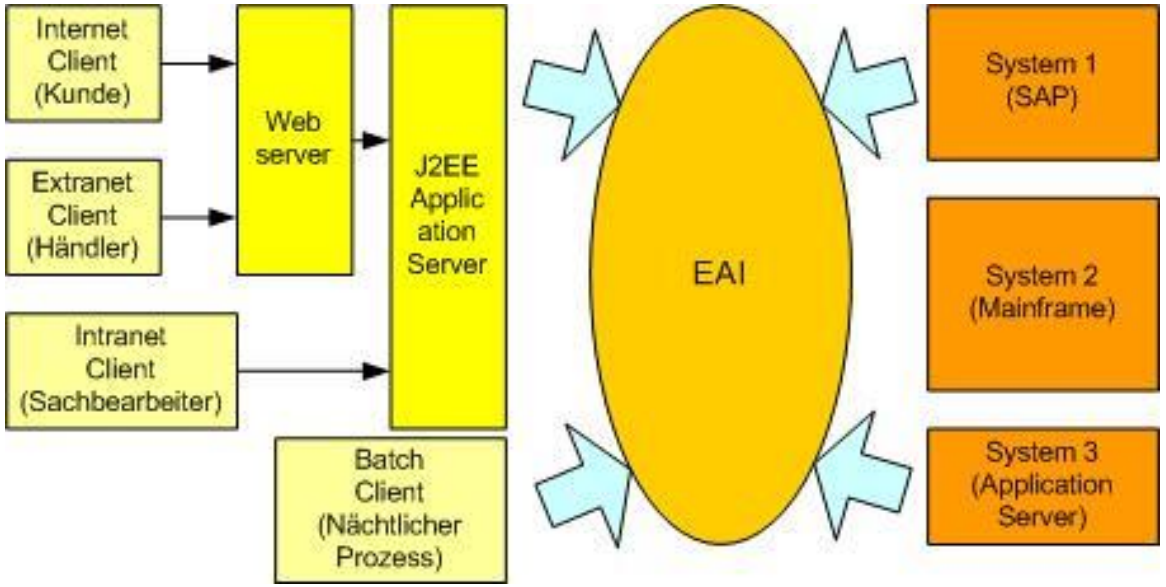
- Verteilte Transaktion
  - ACID-Prinzip gilt
  - Transaktion über mehrere Systeme ausführen
  - Verwaltung von Transaktionsmanager benötigt

EAI-Architekturkonzepte

- Point-to-Point-Architekturen
  - Direkte Kommunikaton zweier Systeme über Schnittstellen
- Hub-and-Spoke-Architekturen
  - Weiterentwicklung von PtP
  - Kommunikation wird über zentralen Server gesteuert → Message Broker
- Bus-/Pipeline-Architekturen
  - Publish-Subscribe-Prinzip

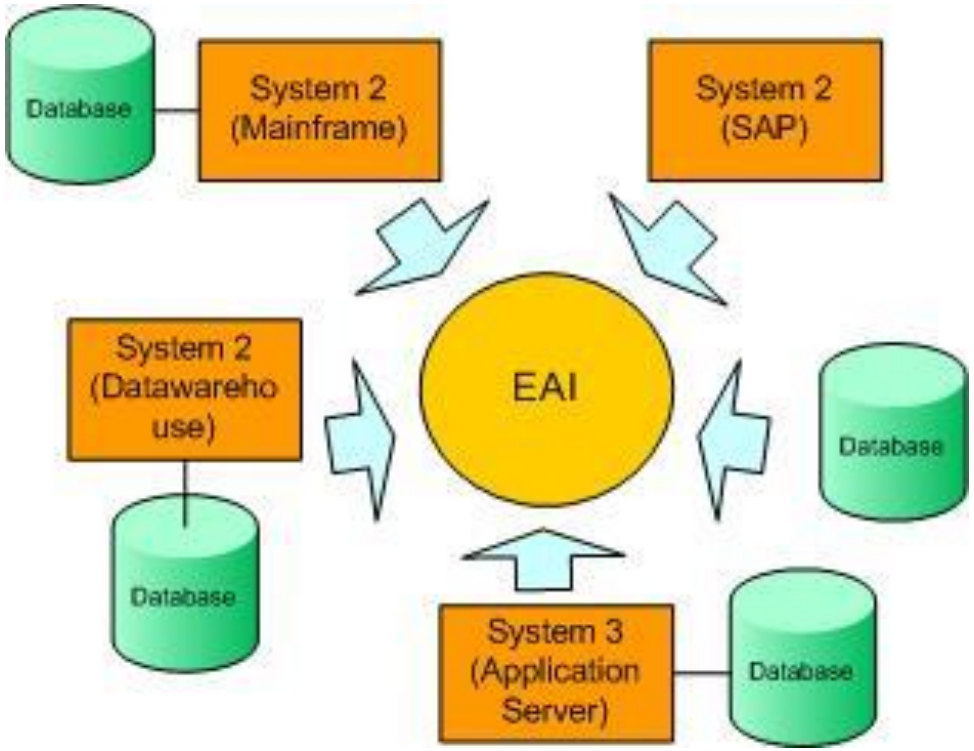
EAI-Anwendung

- Multichannel-Architektur
  - Zugriff auf System über verschiedene Kanäle mit verschiedenen Rechten



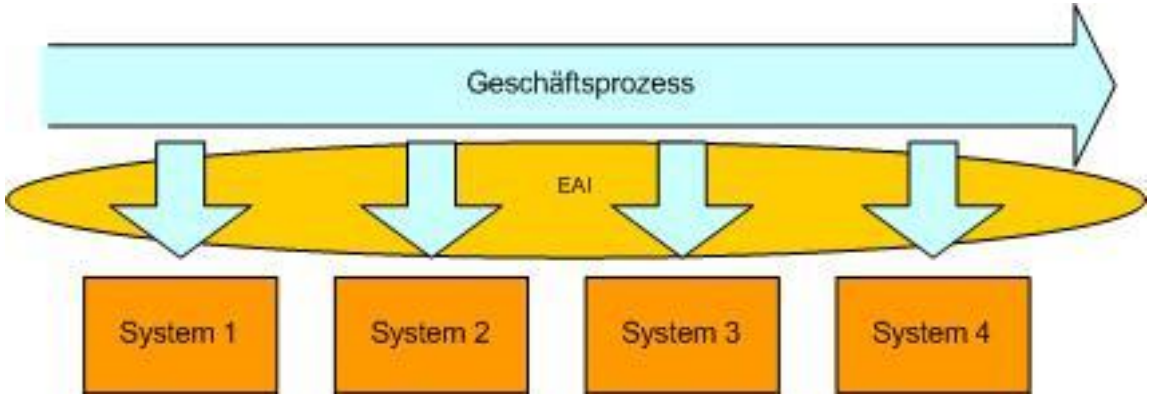
EAI-Anwendung

- Application-to-Application-Integration
  - Zugriff auf verschiedene Applikationen möglich



EAI-Anwendung

- Geschäftsprozessintegration
  - Neueste Architektur
  - Zusammenfassung von benötigten Systemen für einen Geschäftsvorfall



Service-oriented-Architecture

„SOA ist ein Paradigma für die Strukturierung und Nutzung verteilter Funktionalitäten, die von unterschiedlichen Besitzern verantwortet werden.“

- Committee Specification

Komponenten

- Service-Provider
  - Entspricht Dienstleistern
  - Entweder Softwarekomponente oder Unternehmen
  - Allgemein alle Funktionalität einer Ressource
- Service-Registries
  - Angebotene Dienste werden registriert
  - Jeder Provider benötigt eine Registry
  - Weitere Aufgabe ist es, die Services zu kategorisieren
- Service-Requestoren
  - Entspricht Dienstnehmender
  - Fragt die Registries nach bestimmten Diensten

Konnektoren

- Kopplung zwischen Komponenten
- Definiert in vier Basisinteraktionen
  - Register
  - Find
  - Bind
  - Execute
- Finden immer zwischen Komponenten einer SOA statt
- SOA muss immer diese Dienstprimitive anbieten, um weitere Interaktionen bilden zu können

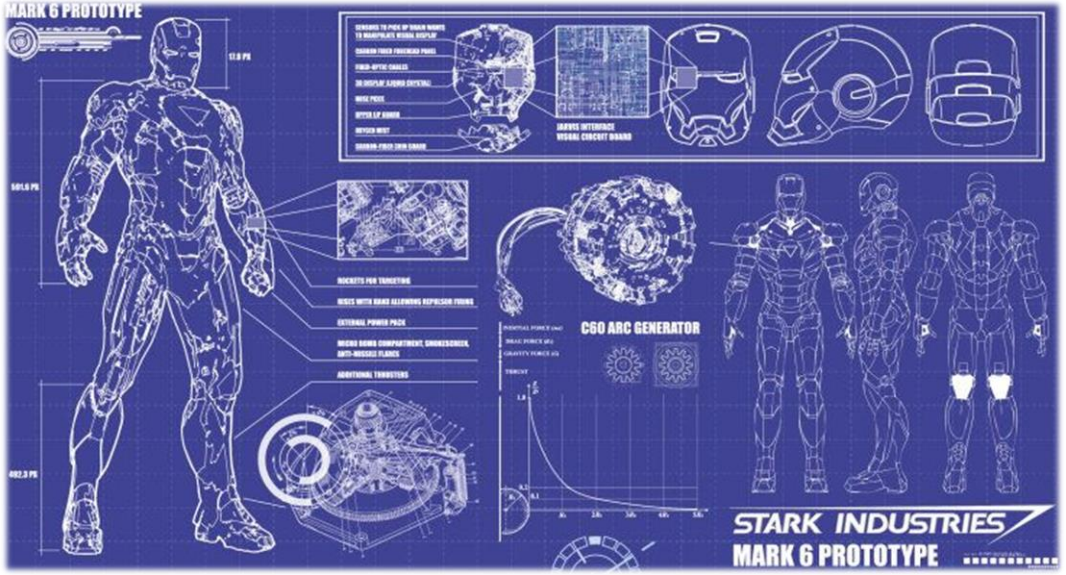
SOA in der Praxis

- Theoretisches Konzept
- Middleware
- Services bereitstellen
- Plattform- und Systemunabhängig
- individuell

Fazit

- Komplexität kontrollieren
- Risikominimierung durch Planung
- Kosten und Zeit optimieren
- Flexibler bei Änderungen

Softwarearchitektur



Jean-Philipp Burnus

Jan Wübbels

Richard Purk

Niklas Borgmann