DHBW Stuttgart
Duale Hochschule
Baden-Württemberg

Software Engineering: Software-Architekturen

Dr. Eugenie Giesbrecht

Softwarearchitektur

Die grundsätzliche Organisation eines Systems, verkörpert durch dessen Komponenten, deren Beziehung zueinander und zur Umgebung sowie die Prinzipien, die für seinen Entwurf und seine Evolution gelten.

IEEE Standard 1471

- Architektur besitzt Strukturen und Komponenten, die untereinander in Beziehung stehen
- Beschreibt die Bausteine, Merkmale und Beziehungen
- Bauanleitung: Beschreibt Entwurfsentscheidungen, erst durch eine konkrete Implementierung entsteht ein System
- Zwischenschritt von Analyse zur Implementierung: Geforderte Funktionen und Merkmale werden auf Software übertragen

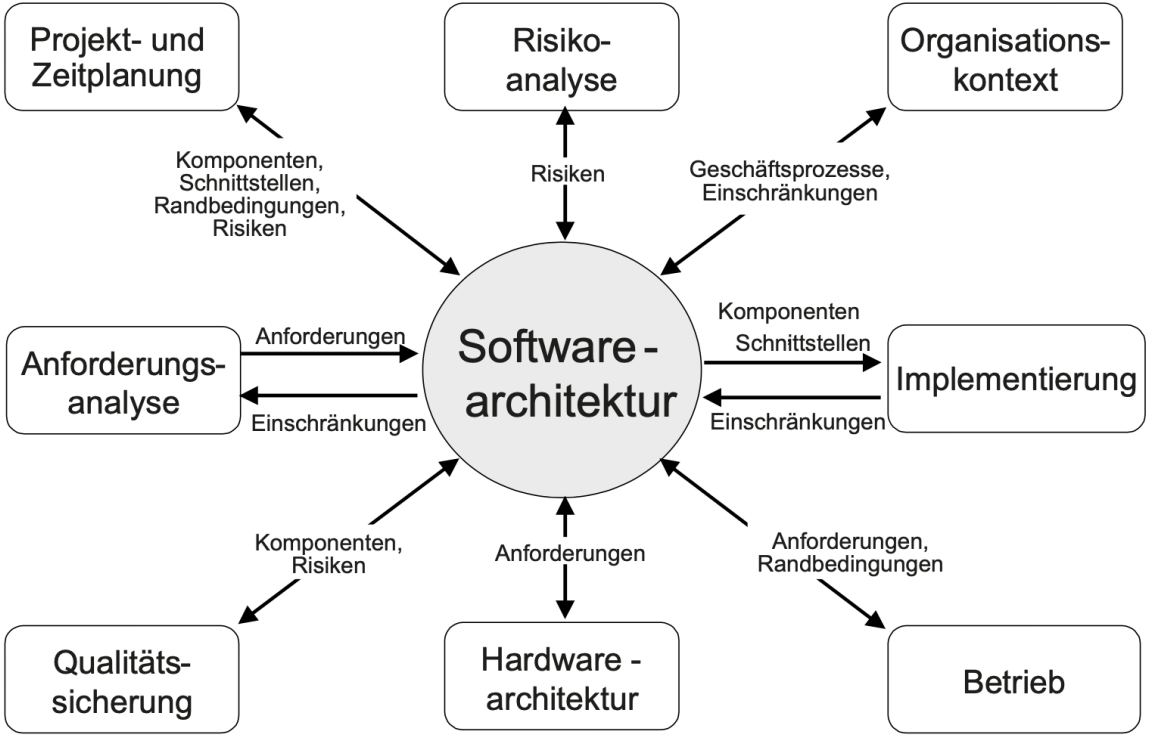
Starke. (2017). Softwarearchitekturen 8.A: Ein praktischer Leitfaden. Carl Hanser Verlag. https://doi.org/10.3139/9783446454200

Was ermöglicht eine Softwarearchitektur?

- Schafft Ordnung und macht Komplexität beherrschbar, indem sie Anforderungen aus der Analyse in Strukturen und Konzepte übersetzt und dokumentiert
- Ermöglicht Überblick über die Komponenten und befähigt Anwender dadurch, die Lösung zu verstehen
- Erweiterbarkeit: Aufgrund der organisatorischen und technischen Natur können sich neue Anforderungen ergeben. Konsistenz in der Architektur stellt sicher, dass Software leicht erweitert werden kann, z.B. werden ähnliche Aufgaben im System durchgängig ähnlich gelöst.
- Stellt Qualität sicher, da die Erstellung von Softwarearchitekturen Anforderungen an Qualitätseigenschaften miteinbezieht
- Ändert sich zu Beginn laufend: Neue Anforderungen während der Erstellung eines Produkts wirken sich auf die Softwarearchitektur aus und erfordern möglicherweise eine Änderung dieser

Starke. (2017). Softwarearchitekturen 8.A: Ein praktischer Leitfaden. Carl Hanser Verlag.
https://doi.org/10.3139/9783446454200

Softwarearchitekturen im Kontext



Starke. (2017). Softwarearchitekturen 8.A: Ein praktischer Leitfaden. Carl Hanser Verlag.
https://doi.org/10.3139/9783446454200

Wie nähert man sich einer SA?

- Entwurfsprinzipien: „Woran soll ich mich halten? “
- Entwurfsmethoden: „Wie gehe ich vor? “
- Architekturmuster: „Welche fertigen Lösungen kann ich nutzen? “

Entwurfsprinzipien (Design Principles)

- Lose Kopplung
- Hohe Kohäsion
- Trennung von Verantwortlichkeiten
- Vermeidung von Redundanz

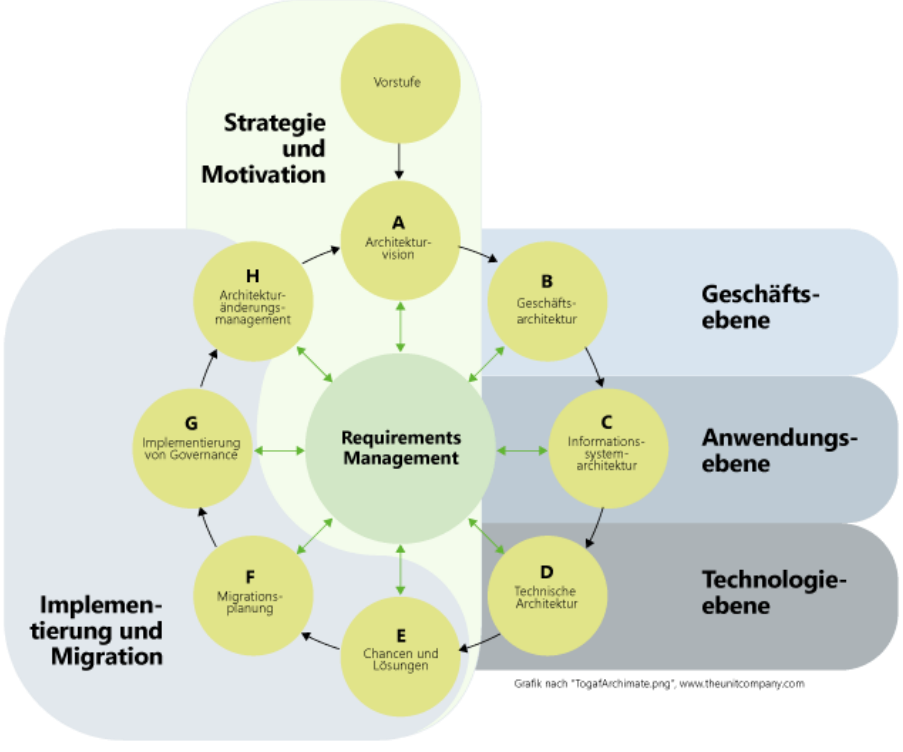
Starke. (2017). Softwarearchitekturen 8.A: Ein praktischer Leitfaden. Carl Hanser Verlag.
https://doi.org/10.3139/9783446454200

Entwurfsmethoden

- Top-down-Ansatz
- Bottom-up-Ansatz
- Iterative (TOGAF ADM) oder inkrementelle Entwicklung (SAFe), beides (RUP)
- Domain-Driven Design (DDD)

TOGAF ADM

- The Open Group Architecture Framework, The Open Group Architecture Framework
- Architecture Development Method



Domain Driven Design

Domain Driven Design (Eric Evans *„Domain-Driven Design: Tackling Complexity in the Heart of Software“***)**

- Speziell: Man beginnt den Entwurf mit der Betrachtung der entsprechenden Fachdomäne
- Entwickler arbeiten zuerst mit Fachexperten zusammen, anstatt sich mit der Technik zu befassen: Daraus resultiert ein Domänenmodell und eine domänenspezifische Sprache
- Definition von „Bounded Contexts“: Eng zusammenstehende Themenbereiche, die aus mehreren „Building Blocks“ bestehen
  - Entities
  - Domain Events
  - Services

Starke. (2017). Softwarearchitekturen 8.A: Ein praktischer Leitfaden. Carl Hanser Verlag. https://doi.org/10.3139/9783446454200

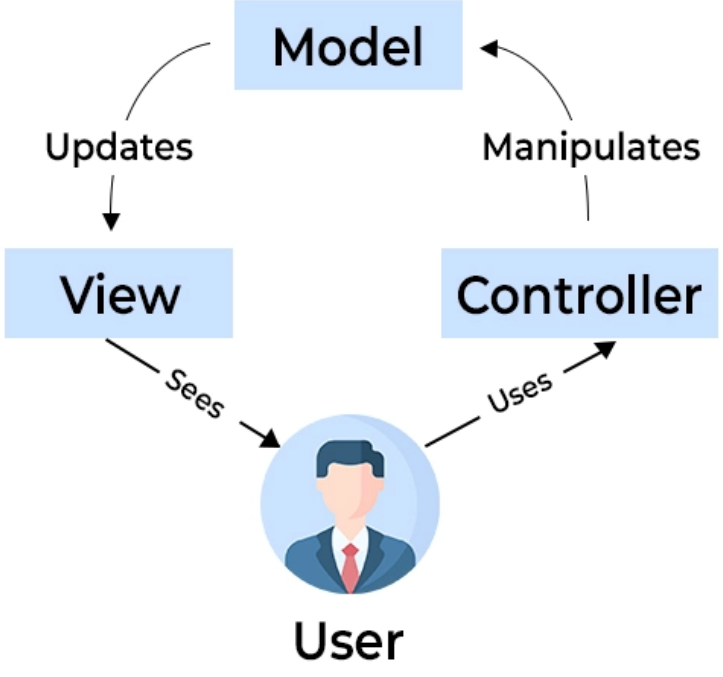
Architekturmuster (Architecture Patterns)

Ein Architekturmuster ist eine Beschreibung bewährter Entwurfspraktiken, die in verschiedenen Umgebungen erprobt und getestet wurden.

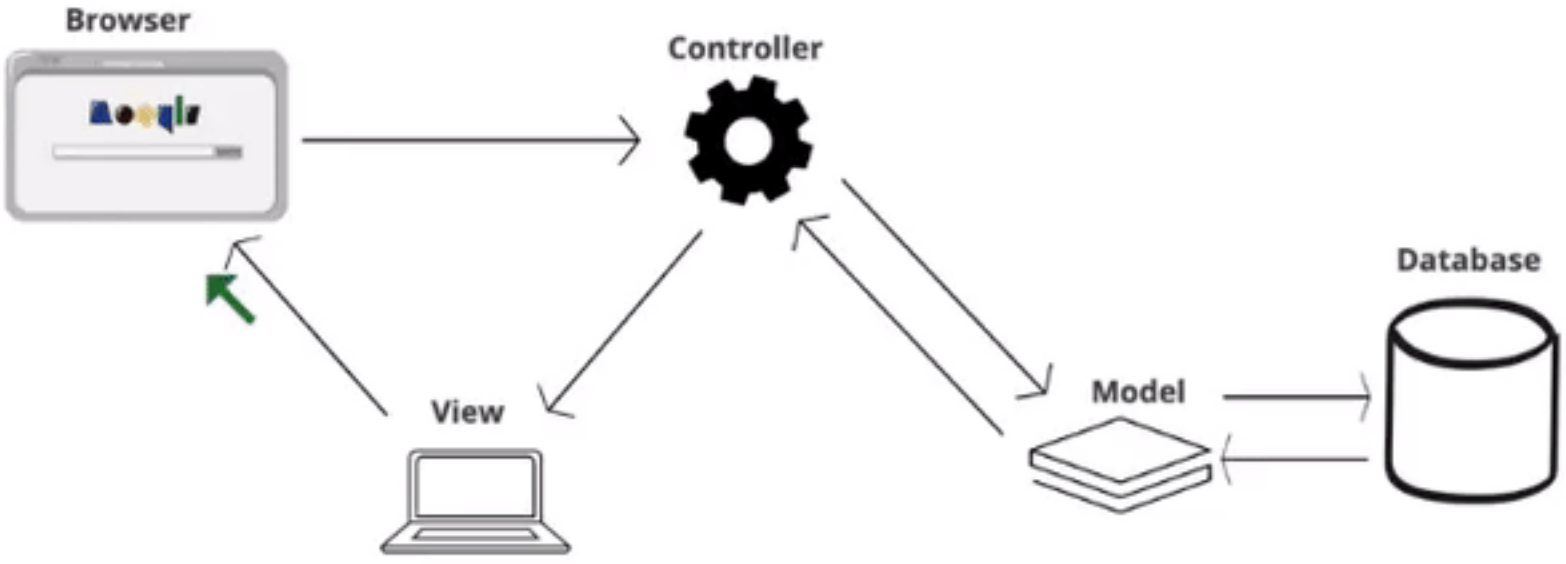
MVC (Model-View-Controller)

- Trennung der Anwendung in drei Hauptkomponenten:
  - Model: Repräsentiert die Daten und Geschäftslogik
  - View: Stellt die Benutzeroberfläche dar
  - Controller: Vermittler zwischen Model und View

- von der Firma Xerox PARC in den 1970er Jahren entwickelt
- bekannt durch Ruby on Rails



MVC @ Webapplikationen



https://afteracademy.com/blog/mvc-architecture-in-web-applications/

Aufgabe: Modellierung der TODO-App nach dem MVC-Architekturmuster

1. Definieren Sie für jede der drei MVC-Schichten mindestens die folgenden Elemente:

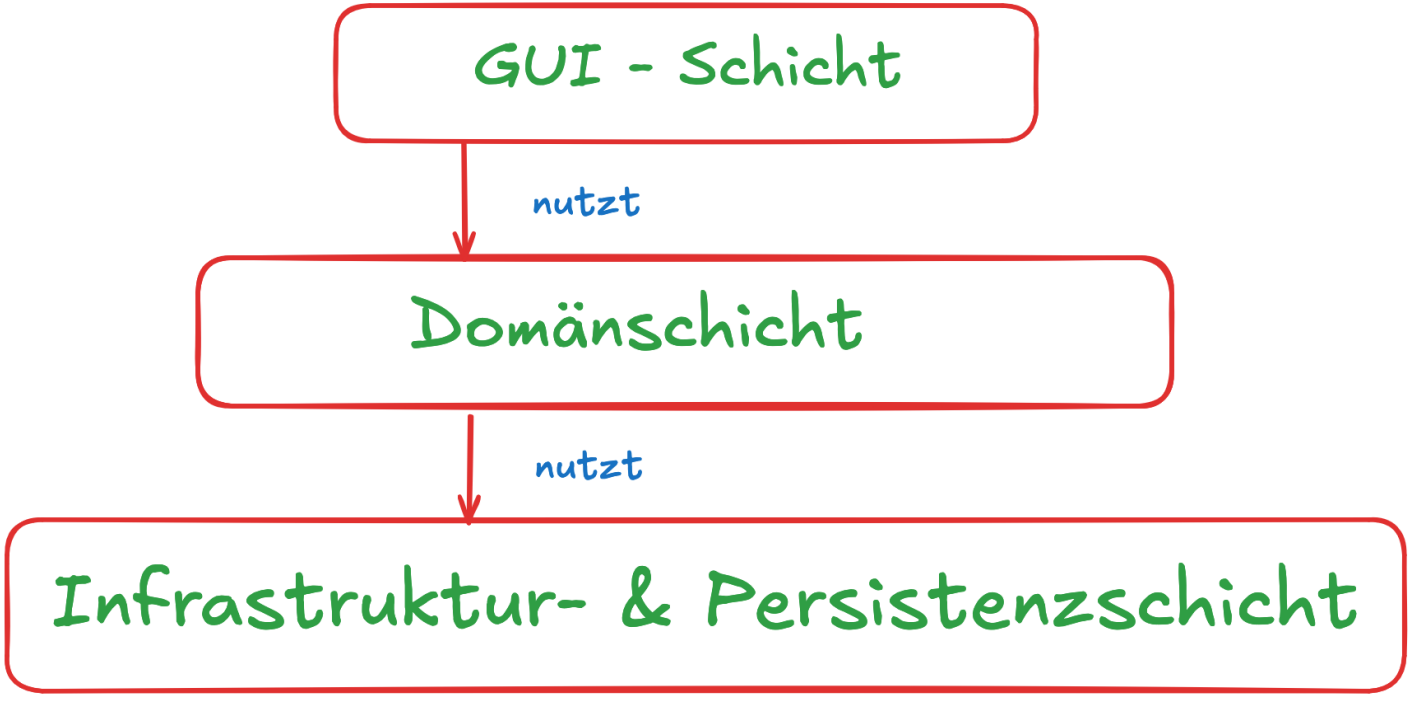
- **Model:**
  - zentrale Domänenobjekte (z. B. „Task“), Datenzugriff, Validierungslogik
- **View:**
  - UI-Elemente
- **Controller:**
  - Steuerungslogik
  - Methoden für das Anlegen, Ändern, Löschen und Anzeigen von Aufgaben (siehe Funktionale Anforderungen)

2. Kurze Beschreibung der Architektur

- Verfassen Sie eine Begründung (max. 10 Sätze), warum MVC für diese Anwendung geeignet ist
- Beschreiben Sie kurz die Verantwortlichkeiten jeder Komponente.

ABGABE: Readme-Datei mit Code (Ende Semester)

Schichtenarchitektur(Layered Architecture)



Schichtenarchitektur: Verwendung

- Cloud-Dienste, wie **AWS Elastic Beanstalk**, **Google App Engine** und **Microsoft Azure App Services,** ermöglichen es, solche Architekturen schnell zu erstellen und zu skalieren
- Schichtenarchitekturen werden auch in Microservices-Anwendungen verwendet
- Auch weit verbreitet in serverlosen Cloud-Umgebungen (**AWS Lambda** oder **Azure Functions**)

Schichtenarchitektur: Betriebssysteme

- Windows:
  - bei der Entwicklung von Desktop-Anwendungen und Webdiensten;
  - Windows **.NET-Framework** (und **.NET Core**)

- Linux:
  - die **LAMP-Architektur** (Linux, Apache, MySQL, PHP)

Schichtenarchitektur: Webanwendungen

- **E-Commerce-Systeme**, **Content-Management-Systeme (CMS)** und **Business Intelligence-Plattformen**

1. **Präsentationsschicht (Frontend):** die UI & UX mit React, Vue.js oder Angular
2. **Anwendungsschicht (Business Logic):** hier kommen serverseitige Technologien wie Node.js, Django oder ASP.NET zum Einsatz
3. **Datenzugriffsschicht (Database):** MySQL, PostgreSQL, MongoDB, etc

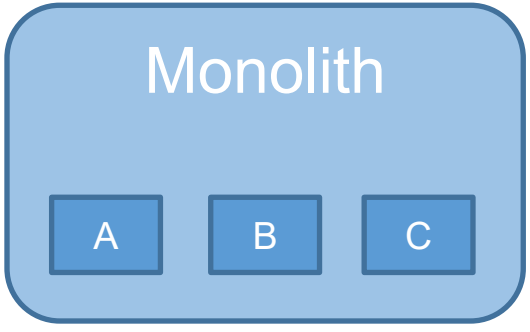
Microservices-Architektur

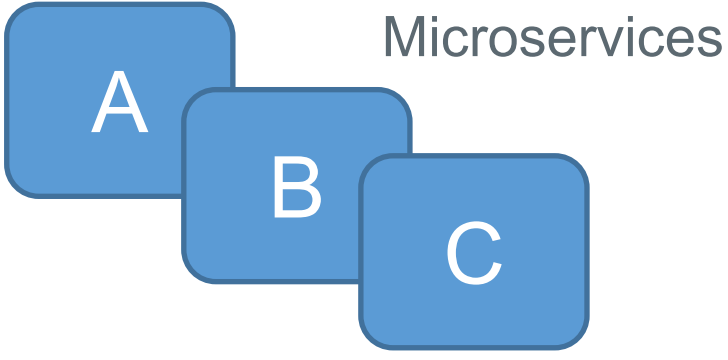
Aufteilung größerer Systeme in kleinere, einzeln entwickel-, deploy- und betreibbare Einheiten

Ziele:

- Schnellere Entwicklung durch kleinere Einheiten, dadurch verbesserte Time-to-Market
- Freie Technologieauswahl
- Eigene Ablaufumgebung pro Service, Kommunikation über Netzwerk

Monolithen vs. Microservices





- Monolithen bündelten alle Funktionen in einer großen Anwendung
- Monolith kann nur als Ganzes von der Entwicklung in den Produktivbetrieb übernommen werden. Das geht nur ein paar Mal im Jahr, weil der gesamte Monolith dafür getestet werden muss. Es sind daher nur wenige Releases im Jahr möglich.
- Neue Funktionalität kann daher nur langsam implementiert werden
  - Fachabteilungen müssen lange auf die Umsetzung ihrer Wünsche warten
  - Entwickler erfahren nur sehr spät von möglichen Fehlern in der Produktion

Eigenschaften von Microservices

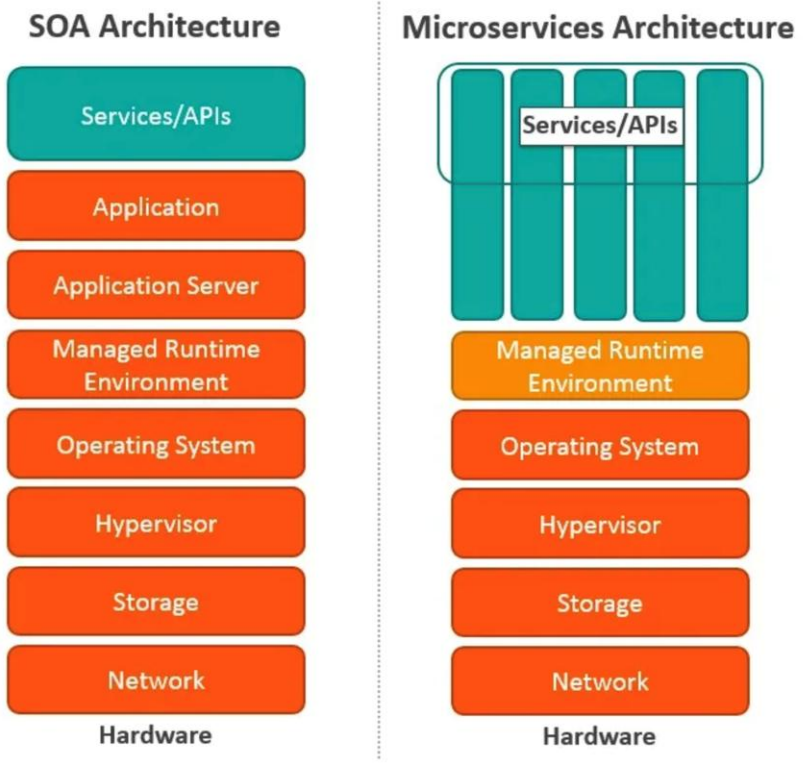
- Eine Anwendung wird modularisiert, indem einzelne Aufgaben/Funktionen/Services als Microservices ausgegliedert werden
- Diese Microservices
  - Interagieren über leichtgewichtige Protokolle miteinander (z.B. HTTP)
  - Haben eine eigene Ablaufumgebung
  - Sind einzeln deploy- und skalierbar
  - Besitzen einfache Kommunikationskanäle
  - Werden getrennt voneinander weiterentwickelt
  - Werden in der Regel automatisiert deployt

Herausforderungen von Microservices

- Werden zwar dezentral entwickelt, trotzdem sollte nicht jedes Team völlig individuell entwickeln
  - Einigung auf technische Protokolle für Schnittstellen, einheitliche Kommunikation und Schnittstellendesign
  - Steigerung der Effizient durch “Microservice Template“
- Microservices müssen sich während der Laufzeit gegenseitig finden können
  - Verwendung einer Service-Registry
- Versionierung von Schnittstellen, um bei der Weiterentwicklung die Kompatibilität mit bestehenden Systemen sicherzustellen

Service-Oriented Architecture (SOA)

- Granularität
- Unabhängigkeit und Skalierbarkeit
- Kommunikation
- Zielsetzung



https://medium.com/@SoftwareDevelopmentCommunity/what-is-service-oriented-architecture-fa894d11a7ec

Serverless-Architektur

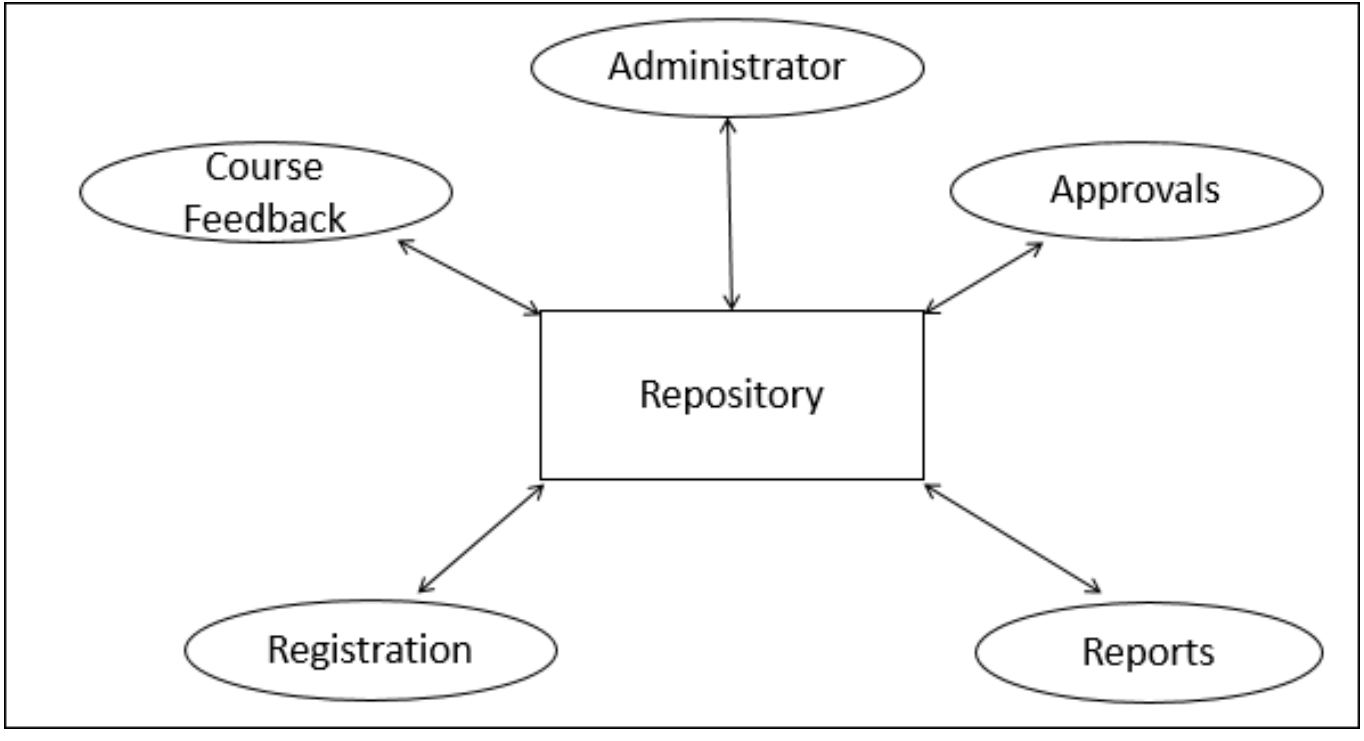
- Entwickler/Operations kümmern sich nicht mehr um die Einrichtung/Betrieb von (virtuellen) Servern. Es muss keine Runtime eingerichtet werden, in der der Code läuft.
- Service Provider stellt Runtime zur Verfügung:
  - Ressourcen werden erst zugeordnet, wenn Code ausgeführt wird
  - Ist die Ausführung beendet, werden Ressourcen wieder entfernt – Ressourcen werden nicht blockiert
  - Apps für Serverless Computing sind daher stateless
- Die Abrechnung erfolgt auf die Millisekunde genau

Video: https://www.youtube.com/watch?v=W_VV2Fx32_Y

Serverless: Beispiele

- Netflix & Mapbox → AWS Lambda
- Alexa Skills → AWS Lambda
- Chatbots und virtuelle Assistenten → das Backend der Chatbots
- Datei- und Datenumwandlungen

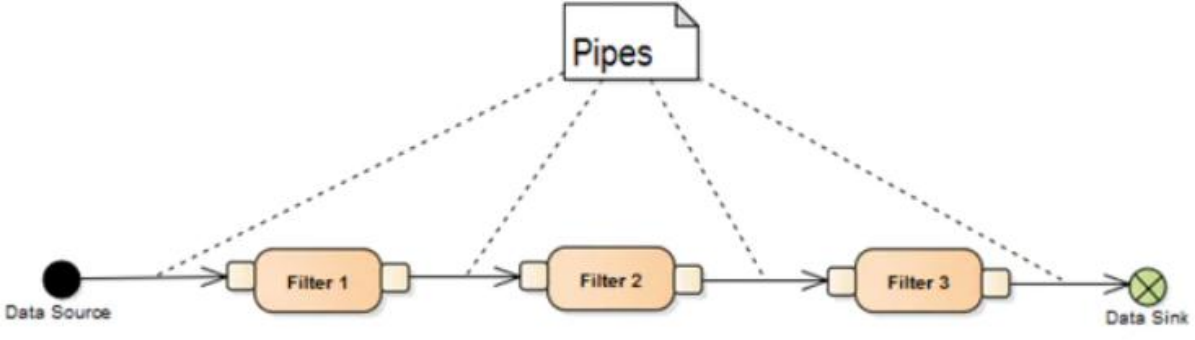
Repository-Architektur



Pipes-und-Filter-Architektur

- Ein Softwarearchitekturmodell, das auf der Verarbeitung von Datenströmen basiert
- Besonders nützlich für Anwendungen, die eine sequenzielle Datenverarbeitung erfordern

Struktur



Aufgabe: Implementieren Sie die Komponenten der MVC-Architektur für die TODO-App

- Implementieren Sie für die TODO-App jeweils eine grundlegende Version der drei zuvor erstellten MVC-Schichten
- Ergänzen Sie die README-Datei um eine kurze Beschreibung, welche Klassen welche Rollen innerhalb des MVC-Architekturmodells übernehmen

What‘s next?

Design Patterns