Softwareentwicklung in Stichworte

| Phase/Rolle | Tätigkeit | Artefakte

Analyse ---

Analyse (Anwendersicht)	Funktionale Anforderungen	Anforderungsspezifikation Geschäftsprozesse, Anwendungsfälle/User Stories , Benutzungsschnittstelle Glossar, Begriffsmodell
	Nichtfunktionale Anforderungen	
(OO)Analyse (Entwicklersicht)	Transformation der Anforderungsspezifikation	Softwarespezifikation Teilsysteme/Pakete, Domänenklassen und Assoziationen,
Architekturkonzeption	Transformation der Nichtfunktionale Anforderungen in Architekturkonzeption	Festlegung der Struktur-Rahmen des Anwendungssystems, , in welche die Teilsysteme eingebettet werden

Design ---

(OO)Design	Transformation der Softwarespezifikation	Komponentenmodell Pakete -> Komponenten Paketbeziehungen -> Komponenten-Interfaces
(Fein)Design	Entwurf im Hinblick auf die Zielplattform	
Modelling using UML ---	Struktur modelling	UML-Strukturmodelle Klassen-, Objekt-Diagramm
	Funktion modelling	UML-Funktions- und -Verhaltensmodelle Anwendungsfall-, Aktivität-, Interaktion-Diagramm

Agil/Scrum Projekt Management ---

Product Owner	die fachlichen Anforderungen formulieren, pflegen, planen	Product Backlog
	die entwickelte Funktionalität am Ende des Sprints abzunehmen	
Entwickler	Sprint Planning Zerlegung der ausgewählten Anforderungen aus dem Produkt-Backlog in programmierbare Aufgaben (Tasks), die im aktuellen Sprint umgesetzt werden	Sprint Backlog
	Tasks implementieren und Testen schreiben	Code, Teste
	Hindernisse in Impediment Backlog zu tragen	

Scrum Master	die Meetings moderieren Hindernisse, fehlende Ressourcen beseitigen

Continuous-Delivery
-Pipeline
(Automatisierung
des Ausrollen für
jedes "Increment of Software"

DevOps-ler	Auf Git-Server
	Gestaltung "branches" für Projekt (master), Release/Sprint, Tasks verteilung.
	Aufstellen der Regeln für Aktionen bezüglich branches um inkonsistent/konflikt zu vermeiden
DevOps-ler	Auf Build-Server (wie Jenkins)
	Konfiguration für Build, Tests, … Deploy in Staging/Product-Server

Architektur --
- 3(4) Schichtenarchitektur
  - Presentation/ externe Schnittstelle,
  - Application/ Anwendungslogik ,
  - Domain/ Datenobjekte und Geschäftslogik,
  - Infrastructure/ Datenzugriffsschicht
- (Web-) Microservices Architektur

Design Prinzipien –

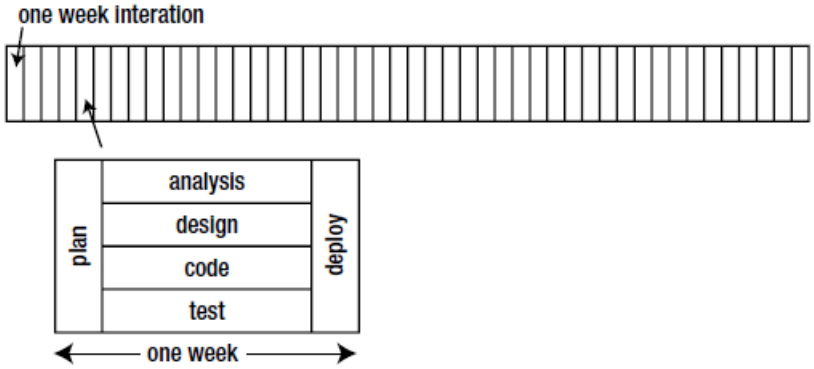
SOLID Prinzipien
Schwache Kopplung und starke Kohäsion
Polymorphismus und dynamisches Binden
Abstrakte Klassen und Interfaces
Dependency Injection

Entwurfsmuster –

MVC (Model View Controller), API-Gateway
Design Pattern
Frameworks

Agiles Projekt Management --

Die Abbildung zeigt Durchführung von Sprints in Projekt nach agilen Prinzipien
Sprint/Iteration ist der Zyklus, nach dem entwickelt wird. In diesem Zeitraum (1-2 Wochen) setzt das Entwicklungsteam die Arbeitspakete des Sprint Backlog in ein potenziell auslieferbares Produkt um.



Agile Prinzipien

- Das Projekt wird in kurze Abschnitte unterteilt, an deren Ende jeweils ein auslieferbares System steht.
- Die Anforderungen werden nur für jeden Abschnitt festgelegt, um beliebige Änderungen während des Projektverlaufs zu ermöglichen.
- Kontinuierliche Kommunikation mit einem Kundenvertreter, idealerweise im Entwicklungsteam.
- Tägliche Kommunikation zwischen den Mitgliedern des Entwicklungsteams.
- Kontinuierliche Unit-Tests und Integration.

CI/CD ---

Die Continuous-Delivery-Pipeline benötigt insbesondere folgende Elemente:

Versions- und Konfigurationsmanagement zur Speicherung aller Artefakte und zur Änderungsverfolgung. Häufig wird getriggert durch das Commit zuerst die Einhaltung von Code-Konventionen überprüft oder erzwungen.

Build-Server, der getriggert durch ein Commit im Versions- und Konfigurationssystem das Bauen und Testen durchführt. Diese
ermöglichen ein automatisiertes Testen und Erstellen von „Nightly“- oder „Release“-Versionen. Diese Versionen können dann automatisiert auf eine Entwicklungs-,Test-, Integrations- und Produktivumgebung deployt werden.

Wesentliche Prinzipien von Continuous Delivery sind:

Test-driven-Development: Der Entwickler erstellt konsequent Tests von den zu testenden Komponenten. Jede Änderung der Software wird getestet.

Build-Server: Neue Releasestände werden an zentraler Stelle nur ein einziges Mal gebaut und bereitgestellt.

Rollforward statt Rollback: Bugs in der Produktion werden nicht gefixt, sondern ein korrigierter Softwarestand in der Continuous-Delivery-Pipeline bereitgestellt.