Software Engineering

Inhalt

[Leistungsnachweis 2](#_Toc436143325)

[Literatur 2](#_Toc436143326)

[Begriffe und Werkzeuge des Software-Engineering 3](#_Toc436143327)

[Werkzeuge und Ziele 4](#_Toc436143328)

[Lasten- und Pflichtenheft 4](#_Toc436143329)

[Graphische Modellierung 5](#_Toc436143330)

[Lebenszyklus von Softwaresystemen - Entwicklung von Software und Komponenten 5](#_Toc436143331)

[Software-Lifecycle 5](#_Toc436143332)

[Softwareänderungen 5](#_Toc436143333)

[Softwarealterung 6](#_Toc436143334)

[Entwicklungswerkzeuge 7](#_Toc436143335)

[IDE - Integrated Development Environment 7](#_Toc436143336)

[Andere Werkzeuge 7](#_Toc436143337)

[Entwurf 7](#_Toc436143338)

[Komponenten und Schnittstellen 8](#_Toc436143339)

[Vorgehensmodelle - Grundlagen, Basis- und umfassende Modelle 9](#_Toc436143340)

[Dokumentengetriebene Ansätze 9](#_Toc436143341)

[Rollen im Entwicklungsvorgehen 9](#_Toc436143342)

[Phasen in Vorgehensmodellen 9](#_Toc436143343)

[Basismodelle 10](#_Toc436143344)

[Umfassende Modelle 11](#_Toc436143345)

[Zusammenfassung 13](#_Toc436143346)

[Vorgehensmodelle - Agile Modelle 14](#_Toc436143347)

[Schwierigkeiten klassischer Entwicklung 14](#_Toc436143348)

[Grundlagen agiler Vorgehensmodelle - Agile Manifesto (2001) 14](#_Toc436143349)

[Feature Driven Development (2002) 14](#_Toc436143350)

[Extreme Programming (XP) (1999 und 2005) 15](#_Toc436143351)

[Scrum (2001) 17](#_Toc436143352)

[Zusammenfassung 19](#_Toc436143353)

[Rolle und Aufgabe der Software-Architektur 19](#_Toc436143354)

[Software-Architektur 19](#_Toc436143355)

[Aufgaben der Software-Architektur 19](#_Toc436143356)

[Rolle der Software-Architektur 19](#_Toc436143357)

[Rolle des Software-Architekten 19](#_Toc436143358)

[Architektur im Entwicklungsprozess 20](#_Toc436143359)

[Sichten der Architekturbeschreibung 21](#_Toc436143360)

# Leistungsnachweis

* 50% Klausur über 60 Minuten
  + 4-5 Fragen
  + Wie würden Sie hier handeln?
  + Benennen und erläutern..
  + Sie wollen..Wo kann es Schwierigkeiten geben?
* 50% Hausarbeit

# Literatur

* Helmut Balzert: Lehrbuch der Softwaretechnik
* Ian Sommerville: Software engineering
* Bernd Oestereich: Objektorientierte Softwareentwicklung
* Gernot Starke: Effektive Software-Architekturen
* Gernot Starke, Peter Hruschka: Software-Architektur kompakt
* Thorsten Posch, Klaus Birken, Michael Gerdom: Basiswissen Software-Architektur

# Begriffe und Werkzeuge des Software-Engineering

* Probleme der Software-Entwicklung
  + Absprache
  + Technik zu komplex
  + Zeitaufwand der Planung und Entwicklung --> Termintreue
* Besonderheiten von Software
  + Immateriell
  + Black-Box
  + Schnell änderbar
  + Komplexität nicht sichtbar
  + Anforderungen ändern sich schnell und sind nicht leicht zu überblicken
* Software-Krisen
  + Früher war die Hardware prozentual am teuersten; heute Entwicklung und Wartung
  + 70er Jahre
    - Zu teuer, zu spät fertig, Ziel verfehlt --> NATO 1968: Begriff Software-Engineering
  + 80er Jahre
    - Ausweitung des Software-Anteils --> Qualitätsmangel
  + 90er Jahre
    - Es gibt nicht DIE eine Lösung
  + 00er Jahre
    - Modellgetriebene Softwareentwicklung
    - Round-Trip-Engineering
    - DSL (Domain specific Languages)
  + 10er Jahre
    - Mobile devices
    - Kollaboration (bei Entwicklung)
    - Web Services
    - Web APIs
    - Industrie 4.0 - Auch Maschinenbauer müssen programmieren können
* **Software Engineering ist die Anwendung eines systematischen, disziplinierten und quantifizierbaren Ansatzes auf die Entwicklung, den Betrieb und die Wartung von Software, das heißt, die Anwendung der Prinzipien des Ingenieurwesens auf Software. (IEEE 610.12)**
* ****
* **Software-Engineering für**
  + **Produktivität**
  + **Qualität**
  + **Termin**
  + **Kosten**
* **Software-Engineering - Methoden des Ingenieurbaus in die Software-Entwicklung portieren**
  + Genaue Planung
  + Modellierung/Pläne
  + Grenzen des Systems bestimmen
  + Arbeitsschritte und -abfolge planen
  + Vorherige Arbeitsschritte frei testen
  + Einmaligkeit nicht gegeben
* **Qualität messen**
  + Nutzerorientierung
    - Funktionalität
    - Zuverlässigkeit
    - Effizienz
    - Benutzbarkeit
  + Herstellerorientierung
    - Übertragbarkeit (Wiederverwendbarkeit)
    - Änderbarkeit
    - Testbarkeit
    - Transparenz

## Werkzeuge und Ziele

### Lasten- und Pflichtenheft

* Klassisches dokumentenbasiertes Werkzeug
* Lastenheft: Zusammenstellung aller fachlichen und wirtschaftlichen Anforderungen an Produkt und Auftragnehmer
  + Ausgangssituation und Zielsetzung des Projektes
  + Produkteinsatz
  + Produktübersicht
  + Funktionale Anforderungen (Fachlichkeit), Nichtfunktionale Anforderungen, Rahmenbedingungen (Gesetze)
    - Funktionalität, Benutzbarkeit, Zuverlässigkeit, Effizienz, Änderbarkeit, Übertragbarkeit
  + Risikoakzeptanz
  + Skizze des Lebenszyklus der Systemarchitektur
  + Lieferumfang
  + Abnahmekriterien
* Pflichtenheft: Zusammenstellung der Produktplanung und Leistungsbeschreibung von Seiten des Auftragnehmers
  + Zielbestimmung (Musskriterien, Wunschkriterien, Abgrenzungskriterien)
  + Produkteinsatz (Anwendungsbereiche, Zielgruppen, Betriebsbedingungen)
  + Produktübersicht, Produktfunktionen (Details)
  + Produktdaten (aus Benutzersicht zu speichernde)
  + Produktleistungen (Anforderungen bezüglich Zeit und Genauigkeit)
  + Qualitätsanforderungen
  + Benutzeroberfläche
  + Nichtfunktionale Anforderungen und Rahmenbedingungen
  + Technische Produktumgebung (Hardware: Server + Client getrennt)
  + Organisatorische Rahmenbedingungen
  + Produktschnittstellen
  + Anforderungen an die Entwicklungsumgebung
  + Gliederung in Teilprodukte

### Graphische Modellierung

* Graphiken helfen häufig besser beim Verständnis
  + Modellierung wichtiger Prozesse
* Modellierung mit UML auch für Nicht-Informatiker grob verständlich
* Modellierung von Interaktionen/sequenziellen Abläufen mit Sequenzdiagrammen

# Lebenszyklus von Softwaresystemen - Entwicklung von Software und Komponenten

## Software-Lifecycle

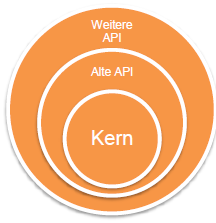
## Softwareänderungen

* **Flexibilität von Software ist erforderlich** --> Architektur auf zu erwartende Änderungen auslegen, Austauschbarkeit durch Komponentenorientierung ermöglichen, inkrementelles Deployment ermöglichen
* Quellen sind: Kundenanforderungen, Plattform, Lizenzen, neue Technologien, Gesetze
* Umgang mit Änderungen (Änderungen sind teuer)
* Änderungen ziehen weitere/neue Änderungen mit sich
* **Änderungsmanagement**
  + Statt geschehen lassen von Änderungen
  + **Ziel: Änderungen an Software geschehen koordiniert und geplant**

|  |  |
| --- | --- |
| **Entwicklung** | **Weiterentwicklung** |
| Neue Funktionen | Anpassung |
| Erstellung | Fehlerbehebung |
|  | Änderungen |

* Frühe Änderungen durch saubere Planung vermeiden

## Softwarealterung

* 4-5 fache geldliche Aufwände für Pflege und Wartung
* **Lebenszyklus: Entwicklung und Etablierung --> Expansion und Diversifizierung --> Alterung und Degeneration**
* Folgen
  + **Unübersichtlichkeit durch Zunahme von Komplexität**
    - Veraltete Methoden/Schnittstellen --> müssen weiter gepflegt werden
    - Alte und neue Technologien
    - Altlasten im Code
  + Größe des Entwickler-Teams --> **Wissen wird stärker fragmentiert** --> Kommunikationsaufwand steigt
  + Zusammensetzung der Teams
    - hohe Personalfluktuation (USA, Indien) bewirkt Verfall des impliziten Wissens
    - typischerweise wenige Köpfe tragen die Kern-Kompetenz
  + **Software wächst nach außen, da neue Schnittstellen entwickelt werden, aber alte erhalten bleiben müssen** --> Funktionalität wird intransparenter, aber aufwändiger zu warten
    - Zwiebelschalencode: 
    - **Rückwärtskompatibilität erfordert Pflege alter Schnittstellen**
    - **Umfang und Komplexität nehmen zu**
* **Vorgehen gegen Überalterung**
  + **Softwareentwicklung mit Entwicklungsmodell**
  + **Restaurierungsarbeiten/Re-Engineering, Re-Factoring**
  + Alle verfügbaren Standards nutzen
  + Kapselung

## Entwicklungswerkzeuge

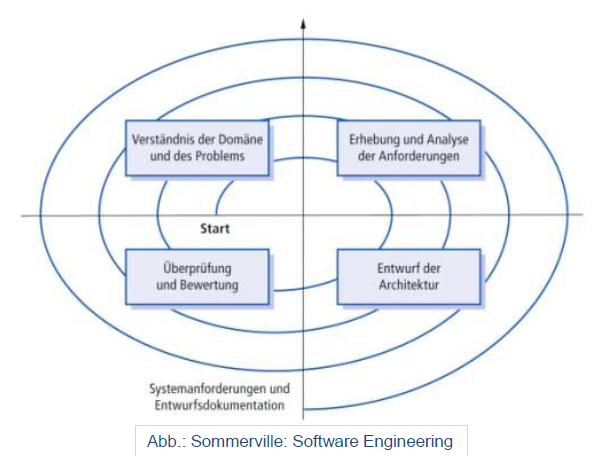
### IDE - Integrated Development Environment

* **Rahmen für weitere Entwicklungswerkzeuge (Editor, Compiler, Linker, Debugger)**
* Auswahlkriterien
  + Funktionalitäten, je nach Programmiersprache
    - An Code (autocomplete)
    - Entwicklungszyklus (Debugger, Compiler)
  + Schnittstellen
    - Ticket-System
  + Kosten
  + Plattform- und Sprachunabhängigkeit
  + Erweiterbarkeit

### Andere Werkzeuge

* Versionsverwaltung
* Buildautomatisierung
* Designtools

## Entwurf

* Systemanalyse
  + **Ein System ist eine zweckgerichtete Sammlung von miteinander verbundenen Komponenten, die für ein bestimmtes Ziel zusammenwirken.**
  + Betrachtung der Software in ihrem Umfeld
    - System-Hardware
    - Schnittstellen
    - --> Architektur, Elektrotechnik, Maschinenbau, Oberflächenentwürfe etc. als Thematiken für die Systemanalyse
  + 
* Grober Entwurf der Architektur
* Design der Komponenten

## Komponenten und Schnittstellen

* **Vertrag zwischen einer Komponente, die eine Funktion benötigt, und einer anderen Komponente, die diese Funktion bereitstellt**
* **Komponenten sind Kernbausteine jeder Softwarearchitektur**
  + Kleiner als die Anwendung; größer als eine Klasse
  + Begrenzt offen zur Adaption
  + **Wiederverwendbarkeit, Wartbarkeit, Anpassbarkeit**
  + **Austauschbarkeit innerhalb des Systems durch Modularisierung**
  + Hierarchisch kombinierbar --> innere Struktur ist versteckt
  + **Hat standardisierte Schnittstellen**
  + **Nicht bindende Eigenschaften: sprachunabhängig, plattformunabhängig, Eignung für Einsatz im heterogenen Netzwerk, introspektiv (Infos über Interface offenlegen), anpassbar zur Laufzeit**
* Externe Komponenten: Vertrauensfrage, Integrationsfrage, Leistungsfrage

|  |  |
| --- | --- |
| **Kleine Komponenten** | **Große Komponenten** |
| Wartbarkeit | Leistungsfähigkeit |
| Flexibilität | Performance |

* Adaption von Komponenten
  + Änderungen in der Schnittstelle sollten keine Änderungen in der Komponente zur Folge haben --> Nutzung von Vererbung und Wrapping
* **Testen von Komponenten**
  + Integrationstest der Komponenten in Framework
  + Interaktionstest der Komponenten untereinander
  + Systemtest
  + Ggf. Black-Box-Test
* **Komponentenmodelle**
  + **Standard für Implementierung, Dokumentation und Veröffentlichung von Komponenten**
  + **Bauelemente: Schnittstellen (Schnittstellendefinition), Nutzungsinformationen (Namenskonventionen), Bereitstellung und Veröffentlichung (Dokumentation)**

# Vorgehensmodelle - Grundlagen, Basis- und umfassende Modelle

* **Vorgehensmodell: System von Regeln und festgelegten Abläufen, die das Zusammenarbeiten in einem Entwicklungsprojekt vorgeben**
* Kernkonzepte
  + Stabilisierung der Tätigkeiten durch Iteration
  + Balance zwischen Standardisierung und Freiheit
* Festlegungen für Aktionen und Vorgängen (Projektarbeit, Reihenfolgen, Handlungen)
* Definition von Rahmenbedingungen der Entwicklung (Qualifikationen, Werkzeuge, Verantwortlichkeiten, Eskalationspfade)

## Dokumentengetriebene Ansätze

* **Dokumente sollen den Fortschritt messbar machen**
* **Für jede Projektphase ein Dokument**
* Planung von außen vorgegeben (plangetrieben)
  + Planung sehr aufwendig und schwergewichtig

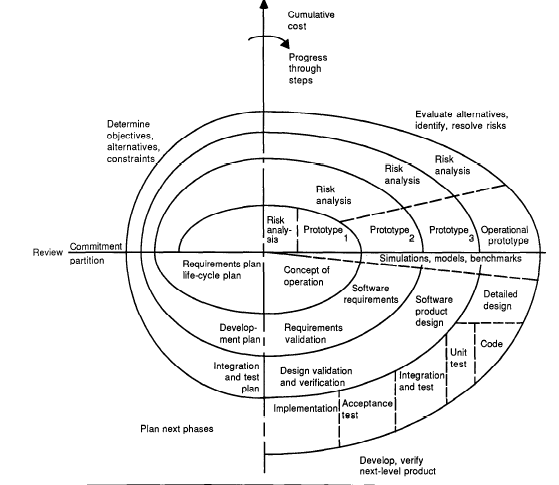
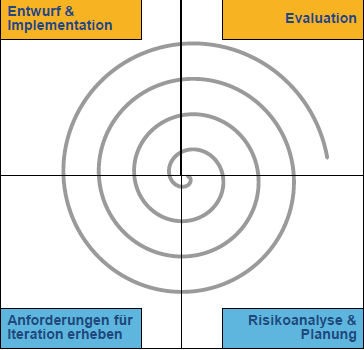
## Rollen im Entwicklungsvorgehen

* **Rollen sind Aufgaben- und Tätigkeitsprofile, die ein einzelner Entwickler einnehmen kann**
* **Eine Person kann mehrere Rollen übernehmen (nacheinander oder gleichzeitig)**
* Wichtige Rollen
  + Software-Projektleiter (Leitung, Verantwortung)
  + Software-Architekt (Strukturierung und Design der Software)
  + Software-Integrator/Konfigurationsmanager (Zusammenstellen der Gesamtsoftware)
  + Software-Entwickler (Detailentwurf einer Komponente)
  + Software-Tester (Testplanung und Durchführung)
  + Tool-Verantwortlicher
  + Qualitätsbeauftragter
  + Sicherheitsmanager

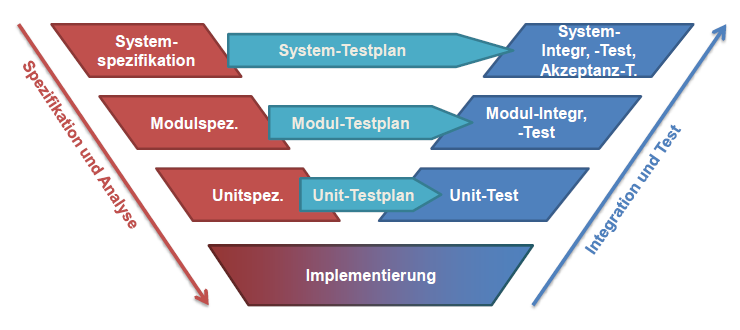
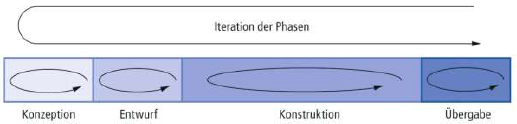
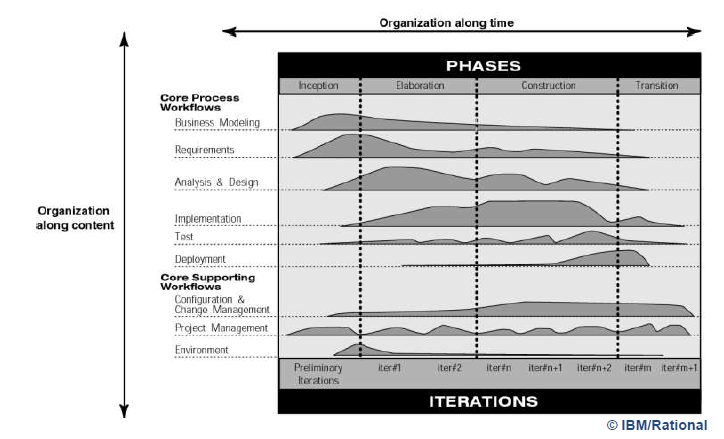
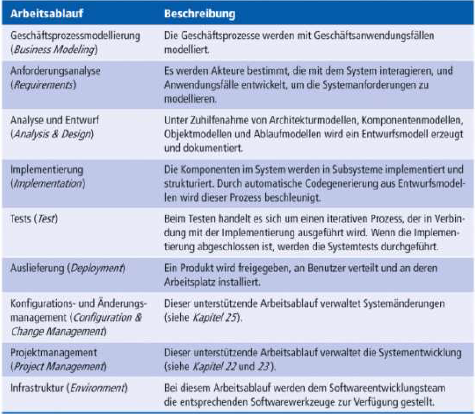
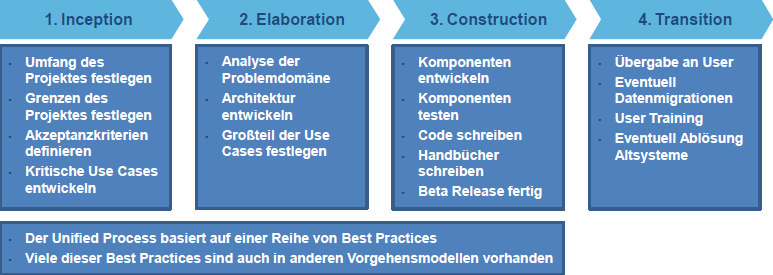
## Phasen in Vorgehensmodellen

* Phasen (überlappen sich)
  + Spezifikation
  + Entwurf und Implementierung
  + Validierung und Test
  + Wartung und Weiterentwicklung
* Vorgehensmodelle abstrahieren, damit
  + Überlappende Aktivität möglich wird (bspw. Definition von Tests parallel zu Spezifikation)
  + Dokumentation standardisiert wird
  + Abläufe automatisiert werden
* Vorgehensmodelle definieren Aktivitätsphasen mit
  + Zuweisungen von Rollen und Aktivitäten
  + Arbeitsprodukten
  + Start- und Endbedingungen (Bewertung: Erfolgs- und Leistungskriterien)

## Basismodelle

* **Wasserfall-Methode**
  + **Statisches Modell mit fester Abstufung**
  + Meilensteine nach jeder Stufe
  + Dokumentenzentriert und -getrieben
  + **Rückfall auf vorige Stufen möglich**
  + Absicherungen: Validierung und Verifikation
  + Verständnis der Aufgabe im Voraus ist notwendig
* **Prototypen-Ansätze**
  + **Ein Software-Prototyp ist eine lauffähige Vorabversion des Produktes**
    - **Reduzierter Funktionsumfang**
    - Verminderung von Risiken und falschen Annahmen
    - **Erreichung von Planungssicherheit**
    - Kreative und experimentelle Lösung
    - Hilft beim Kundenverständnis/Zieldefinition
    - **Höherer Aufwand**
    - **Größter Vorteil bei hochindividualisierten Projekten**
  + Varianten
    - **Horizontaler Prototyp** implementiert eine Schicht
      * --> Prüfen: Integration in die betroffene Schicht
    - **Vertikaler Prototyp** implementiert eine Funktion
      * --> Prüfen: Funktionalität/Fachliche Sicht
  + Arten
    - Labormuster
    - Demonstrator
    - Pilotsystem (Alpha-Version)
    - Provisorisches System
* **Spiralmodell**
  + 4 Quadranten/Phasen
  + Insgesamt lineares Vorgehen
  + 
  + 
  + **Risikoanalyse**: Risiken analysieren und **Komponenten mit hohem Risiko zuerst implementieren, damit hinterher der Aufwand nicht umsonst war** --> Big Bang durch inkrementelle Entwicklung vermeiden
  + Wasserfall kann in der Spirale verwendet werden
  + **Risiken als zentrale Treiber des Prozesses** (klare Zieldefinitionen, Prüfung von Rahmenbedingungen etc.)
  + **Hohe Bedeutung der Dokumente**
  + Wenig Flexibilität --> Änderungen werfen selbst Risikomanagement durcheinander
* **Rapid Application Development**
  + Teamorientierter Ansatz
  + Hohe Kundenintegration
  + Unveränderliche Deadline --> Funktionalität begrenzt durch Zeit
    - Must have
    - Should have
    - Could have
    - Won't have
  + 4 Phasen-Modell
    - Joint Requirements Planning
    - Joint Application Design
    - Construction (Implementation)
    - Cutover (Test, Training, Installation)
* Allgemein
  + Vollständige Spezifikation nicht sinnvoll
  + Basismodelle versagen häufig bei großen/neuartigen Projekten
  + Einhaltung von Laufzeiten kaum möglich
  + Innovationen kaum möglich

## Umfassende Modelle

* **Liefern Prozessvorgaben für einen weiten Kreis von Beteiligten des Entwicklungsgeschehen**
* **V-Modell**
  + Integriert eine dedizierte Qualitätssicherung in sequenziellen Ablauf
  + Anpassbar, aber erfordert Festlegung von Handlungen, Zuständigkeiten und Zeitpunkten
  + **Wasserfallmodell + Tests**
  + 
  + **Bietet Sicht für Auftraggeber und Auftragnehmer**
  + Andere Modelle können darin abgebildet werden
  + **Hoher Dokumentations- und Planungsaufwand**
  + Erst für größere Projekte machbar
* **Unified Process**
  + Früher von IBM/Rational: kommerzielles Produkt: Rational Unified Process (RUP)
  + **Use Case-basiert**
  + **Modellierung in UML**
  + **Unterteilung in 9 Workflows und 4 Phasen**
  + Dokumentengetrieben mit agilen Ansätzen
  + 
  + 
  + 
  + 
  + **Paradigmen**
    - **Anforderungen im Voraus haben, ansonsten iterative Planungsprozesse**
    - **Kundenanforderungen im Auge behalten**
    - **Software in Komponenten aufteilen**
    - **Visualisieren der Planung**
    - **Früh und toolgestützt testen**
    - **Änderungsmanagement zur Absicherung**

## Zusammenfassung

* Was ist/beideutet ein Vorgehensmodell?
* Definierte Rollen
* Warum Rollen?
* Hauptphasen - Entwirrung gegen Lebenszyklus
* Definition/Eigenschaften/Besonderheit der Vorgehensmodelle

# Vorgehensmodelle - Agile Modelle

## Schwierigkeiten klassischer Entwicklung

* Deadline verschieben
* Features auf später verschieben
* Flexibilität bei geänderten Anforderungen
* Schlecht getestete Releases
* Vorgehensmodell ist teuer & aufwendig
* Vorstellung des Kunden verfehlt
* Software letztendlich zu teuer
* Reaktion auf spontan auftretende Probleme

## Grundlagen agiler Vorgehensmodelle - Agile Manifesto (2001)

* Individualitäten über Prozesse
* Funktionierende Software über Dokumentation
* Kunde über Vertrag
  + Kunden einbinden
* Flexibilität für Änderungen statt einem Plan folgen

## Feature Driven Development (2002)

* Mehrere Features parallel umsetzen
* Funktionstüchtige Software (zumindest in Teilen) ist schnell erschaffen
* Selbst mit einer implementierten Funktion ist die Software im Grundgerüst schon für viele Funktionalitäten ausgelegt
* Pro Modul ein Entwicklungsverantwortlicher
* **Chef-Architekt**
  + Overall Model
    - Modell vorschlagen (strawman)
    - Modell verfeinern und dokumentieren
* **Chef-Programmierer**
  + Problemdomäne untersuchen, Anforderungen erarbeiten
  + Modell erarbeiten (in Gruppen) und präsentieren
  + Modell verfeinern und dokumentieren
  + Für interne oder externe Bewertungen bereitstellen
* Vorteile
  + Leichte Umsetzung, da es eine Mischform von klassischen und agilen Vorgehensmodellen ist
  + Parallele Entwicklung der Einzelfeatures möglich
* Nachteile
  + Hohe Anforderungen an die Chef-Entwickler vor allem bei großen Systemen
  + Nur für Neuentwicklungen gut geeignet
  + Einzelarbeit relativ starr
* **Grundsätzliche Aktivitäten**
  + Entwicklung Gesamtmodell
  + Erstellung der Feature-Liste
  + Planung jedes Features
  + Entwurf des Features
  + Implementierung des Features

## Extreme Programming (XP) (1999 und 2005)

* Entwickelt von Kent Beck
* Häufig eingesetztes Modell
* Pures agiles Modell
* Nur für kleine Teams sinnvoll --> ein Team (ein Raum)
  + Jeder ist für den gesamten Code verantwortlich
* **Fokus auf Entwickler**
* 
* **Zentrale Werte**
  + **Communication (Kommunikation)**
    - Mangelnde Kommunikation als Hauptfehlerursache
    - Gemeinsame Aufwandseinschätzung
    - Pair Programming
  + **Simplicity (Einfachheit)**
    - Einfache Lösungen bevorzugen
    - Auf zukünftig Benötigtes verzichten
  + **Feedback (Rückmeldung)**
    - Jederzeit Status geben und Rückmeldungen einholen können (Akzeptanztest)
    - Jederzeit Qualität bestimmen können (Modul-/Systemtests)
    - Jederzeit auf Änderungen reagieren können (Machbarkeit- und Aufwandseinschätzung von Change Requests)
  + **Courage (Mut)**
    - Für JETZT entwickeln
    - Korrekturen positiv sehen
    - Hartnäckigkeit
  + **Respect (Rücksichtnahme)**
    - Kollegen nicht behindern
    - Sich selbst nicht überarbeiten
    - Teamgeist und Motivation fördern
* **12 Methoden**
  + **Metapher**
    - Wahl eines gemeinsamen Vokabulars zwischen Programmierer und Kunden
  + **Einfaches Design**
    - Es wird eine bewusst einfache und allgemeine Lösung angestrebt
    - Zukünftige Anforderungen und Spezifikationen werden nicht berücksichtigt
  + **Refactoring**
    - Software wird ständig wartungsfreundlicher gemacht und überarbeitet
  + **Kontinuierliches Testen**
    - Vor den Features werden Testfälle entworfen
    - Unit Test - Test für kleine Gruppe von Klassen (Voraussetzung für Refactoring)
    - Akzeptanztest - Test für Kundenzufriedenheit
  + **Pair Programming**
    - Paarweise Programmierung zur Arbeitsteilung und Fehlerminimierung
  + **Coding Standards**
    - Standards zur Wartbarkeit und Verständlichkeit des Codes
    - Entwurfsmuster, Programmierparadigma, Codestruktur
  + **Planning Game**
    - Planungsphase "Release Planing" mit Entwicklern und Kunden
      * Ermittlung und Anpassungen der Anforderungen
    - Planungsphase "Iteration Planning" nur mit Entwicklern
      * Aufteilung der Aufgaben
  + **Kleine Releases**
    - Kunde erhält nach kurzen Iterationen einen lauffähigen Zwischenstand
    - Entwickler erhält frühzeitig Rückmeldung
  + **Collective Ownership**
    - Keine Verantwortlichkeit eines Einzelnen durch Pair-Programming
  + **Kontinuierliche Integration**
    - Fehlerminimierung durch regelmäßige Integration in kurzen Zeitabständen
    - Kostenminimierung des Integrationsprozesses durch Routinevorgang
  + **Kunde on Site**
    - Kunde immer erreichbar
    - Alternative: eigener Mitarbeiter übernimmt Kundenrolle
  + **40 Std. Woche**
    - Erhöhung der Motivation der Mitarbeiter durch Vermeidung von Überstunden
    - Regelmäßige Überschreitung ist Anzeichen einer schlechten Planung
* Kritische Herausforderungen
  + Kommunikation
  + Disziplin und Homogenität des Teams
  + Stabile Release
* User Stories für Kundenanforderungen
* Varianten
  + Industrial XP
    - Anpassung an industrielle Rahmenbedingungen
  + XP2 (2. Aufl.)
    - Neuer Wert: Respect
    - Prinzipien werden neu gefasst
    - Primärpraktiken als erstes; Folgepraktiken als zweites

## Scrum (2001)

* Implementiert agiles Projektmanagement (hat nichts mit Software-Entwicklung zu tun)
* **Framework statt Vorgehensmodell - Rahmenwerk zur Entwicklung und Erhaltung komplexer Produkte**
* Projektsteuerung über
  + Review-Sitzungen
  + Rückkopplungsschleifen
* Rollen
  + **ScrumMaster**
    - Verantwortlich für die Verständnisvermittlung und Einhaltung von Regeln
    - Moderator, der nicht ins Geschehen eingreift --> Optimierung der Zusammenarbeit des Teams
    - Sorgt dafür, dass die Scrum-Meetings stattfinden
    - Verhinderung von Hindernissen (andere Aufgaben) während des Sprints
    - Vertritt das Scrum Team nach außen
  + **Team**
    - Kommunikativ
    - Hohe Eigenverantwortung
  + **Stakeholders**
  + **Managers**
  + **Product Owner**
    - Aufgabe der Wertmaximierung des Produktes
    - Rechenschaftspflichtig für das Product Backlog --> klare, transparente Formulierung und Priorisierung der Einträge
    - Kann von einem Komitee beraten werden
* **Sprints (2 - 4 Wochen)**
  + **Sprint Planning**
    - 1. Entwicklungsteam selektiert Anforderungen aus dem Product Backlog für den nächsten Sprint
    - 2. Entwicklungsteam erstellt einen Systementwurf --> falls zeitliche Über- oder Unterdeckungen festgestellt werden, werden die Ergebnisse der 1. Phase entsprechend angepasst und neu definiert
  + **Daily Scrum (**15 Minuten)
    - Planungssitzung am gleichen Ort zur gleichen Uhrzeit
    - Zentrale Fragen/Themen:
      * Letzte Tätigkeiten seit dem letzten "Daily Scrum"
      * Schwierigkeiten
      * Zukünftige Tätigkeiten bis zum nächsten "Daily Scrum"
    - Verbessert die Kommunikation, Entscheidungsfindung und den gemeinsamen Wissensstand
    - Externe Teilnehmer dürfen zuhören aber müssen schweigen
  + **Sprint Review (4h)**
    - Am Ende eines Sprints, um das Product Backlog zu aktualisieren
    - Scrum Master, Product Owner, Entwicklerteam, Stakeholder nehmen teil
    - Positive und negative Aspekte des Sprints werden ausgearbeitet
    - Marktsituation wird analysiert und mit dieser Kenntnis werden nächste Schritte erarbeitet
    - Zeit- und Budgetplan für das Produkt-Release wird geprüft
  + **Sprint Retrospective (4h)**
    - Findet nach dem Abschluss des Sprint Reviews und vor dem Beginn des nächsten Sprints statt
    - Überprüfung des Teams für die Verbesserung für den nächsten Sprint
    - Selbstreflexion
    - Scrum Master ist für das Meeting verantwortlich und nimmt als Gleichberechtigter teil
    - "Definition of Done" wird angepasst
* Artefakte
  + **Product Backlog**
    - Geordnete Liste, welche als Anforderungsquelle dient
    - Ist niemals vollständig, da es sich mit dem Produkt entwickelt
    - Enthält User Stories, Anforderungen, Aufwandsschätzung (mit Planning Poker)
  + **Sprint Backlog**
    - Aufgabenliste, die aus dem Sprint Planning hervorgeht
    - Enthält Anforderungen des Product Backlogs, welche innerhalb eines Sprints realisiert werden sollen
    - Enthält den Fortschritt der jeweiligen Aufgaben inklusive Restaufwand --> wird täglich aktualisiert und verfeinert
* **Definition of "done"**
  + Definition, wann eine Komponente den Qualitätsansprüchen genügt
  + "Done"-Erarbeitung mit allen Teammitgliedern
  + Enthält Tests für die Komponente zur Integration in das gesamte Produkt
  + Definition of "done" sollte immer strenger erweitert werden --> Qualität wird optimiert
* **Scrum-Werte**
  + **Mut**, um Herausforderungen trotz Risiken anzugehen
  + **Commitment** - Selbstverpflichtung zur Erledigung einer Aufgabe
  + **Offenheit** zur Aufnahme und Teilung von Informationen, für eine verbesserte Einschätzung der Situationen
  + **Fokus** - Einschränkung von Optionen, damit der Kunde zufrieden ist und Risiken gesteuert werden können
  + **Respekt** - Wertschätzung für Beiträge aller Personen
* Vorteile
  + Leichtes Konzept
  + Wenig Training notwendig
  + Optimal für schnell änderndes Umfeld
* Nachteile
  + Kleine, heterogene Teams erforderlich
  + Reines Projektmanagement-Framework (Softwareentwicklung wird nicht modelliert)

## Zusammenfassung

# Rolle und Aufgabe der Software-Architektur

## Software-Architektur

* Brücke zwischen Anforderung und Implementierung
* **Architektur ist die fundamentale Organisation eines Systems, welche sich in dessen Komponenten wiederspiegelt. Darüber hinaus organisiert sie die Verbindungen, die Umgebung, die Leitprinzipien, das Design und die Entwicklung.**
* Beschreibt die Dekomposition eines Systems in Komponenten, Beziehungen, Interaktionen, Qualitätsmerkmalen und Entwurfsrichtlinien
* Hierarchische Beschreibung der einzelnen Module (erst Subsystem, Modul --> dann Klasse und Funktion)
  + Komponenten = Architekturbausteine
  + Schnittstellen = Interaktionsdefinition
  + Verhalten = Funktionalität
  + Sichten = verschiedene Arten von Strukturen (Klassen, Prozesse, Abhängigkeiten, physikalische Struktur etc.)

## Aufgaben der Software-Architektur

* Softwarearchitektur wird an der Erreichung des Ziels gemessen
* **Ziel ist ein gutes Design des Gesamtsystems** (umsetzbar, verständlich, zuverlässig, erweiterbar)
* Klare Zuordnungen der Aufgaben zu jeder ISO/OSI-Schicht
* Bildung klarer/schlanker Schnittstellen --> Kommunikation nur zwischen benachbarten OSI-Schichten
* **Stabilisierung des Systems**
  + Kleine Änderungen der Anforderungen --> kleine Änderungen im System
  + Testbarkeit durch bekannte Wirkungsketten
  + **Wiederverwendbarkeit**
  + **Beherrschbare Komplexität**
  + **Konzeptionelle Integrität (ausgewogen, elegant)**

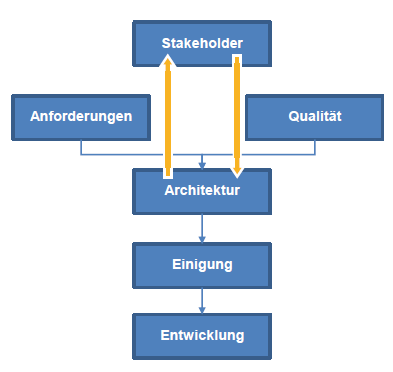
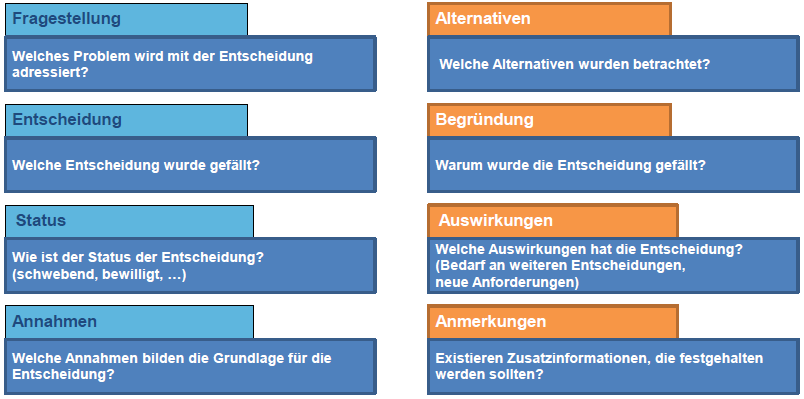
## Rolle der Software-Architektur

* **Rahmen für die Implementierung**
* Abbildung kritischer Use Cases
* Erreichung übergeordneter Projektziele (Ressourcen, Laufzeiten)
* Ansatzpunkte für Änderungs- und Variantenmanagement
* **Entscheidend für den Erfolg eines Systems**

## Rolle des Software-Architekten

* **Position zwischen Projektleitung und Entwicklung**
* Sollte von einer Person bekleidet werden (Sicherung der Intensität der Rolle)
* Aufgaben
  + Kommunikation mit Stakeholdern
  + Designentscheidungen abwägen
  + Designentscheidungen treffen (weitreichende Entscheidung)
  + Systemstruktur festlegen
* Aufgaben nach der Entwicklung der Architektur
  + **Saubere Lösung für Änderungswünsche finden**
  + Koordination der Weiterentwicklung

## Architektur im Entwicklungsprozess

* Einflussfaktoren
  + Rechtliche Rahmenbedingungen
  + Hardware
* Stakeholder sollten Anforderungen nicht nur stellen, sondern auch unter Betrachtung der Architektur anpassen können, damit langfristig die Qualität bestehen bleibt
  + 
* Kerntätigkeiten
  + Auf abstrakter Ebene Orientierung an Vorgehensmodellen
  + Offene Punkte im Backlog verwalten (Baustein in vielen Vorgehensmodellen)
  + Arbeitsweise an individuellen Erfahrungen ausrichten
* Entwicklungsvorgehen
  + Iterativ
  + Inkrementelles (schrittweise) Entwickeln der Lösung
* Hauptschritte
  + Ermitteln von Anforderungen und Risiken
  + Erstellen eines Entwurfs (oft ad hoc)
    - Auf Basis bekannter Muster und spontanen Lösungsideen
  + Überprüfung des Entwurfs, Sicherstellen der konzeptionellen Integrität
    - Anforderungen erfüllt?
    - Absicherung der Risiken
    - Überarbeitung der Schwächen
* Dokumentation von Designentscheidungen
  + Implizit (Erfahrungen, Grundwissen)
  + Explizit und nicht kommentiert --> Gefahr: Entscheidung verblasst mit der Zeit
  + Explizit und extra nicht kommentiert --> taktische Gründe
  + 

## Sichten der Architekturbeschreibung

* **Kontextuelle Sicht**
  + Abgrenzung gegenüber Nachbarsystemen, Geschäftsprozessen und Nutzergruppen
  + Technische Verbindungen zu Nachbarsystemen
* **Physikalische/Verteilungs- Sicht**
  + Beschreibung der Systeme (Speicher, Rechner, Produktions- und Testumgebungen, Datenübertragungskanäle
* **Bausteinsicht (Implementierungssicht)**
  + Softwarebausteine, Komponenten, Beziehungen, Verfeinerungen der Bausteine
* **Laufzeitsicht**
  + Darstellung des dynamischen Zusammenspiels von Bausteinen
  + Betrachtung von Vorgängen (Fehlerbehandlung, Booten)
  + Szenarienbetrachtung (detaillierter als Bausteinsicht)
* **Datensicht**
  + Beschreibung der Daten --> schwer zu ändern
  + Gruppierung der Daten
  + Zusammenhänge der Daten
  + Verifizierung der Daten