Prüfung 2011-01-28

Was sind Vorgehensmodelle und wozu sind sie gut?

- immer komplexere, umfangreicher Softwareprodukte → Vorgehensmodelle gewinnen an Bedeutung, sind wesentlich am Geschäftserfolg beteiligt
- Ziel: etablierte, ingeniermäßige Basis für Softwareentwicklung → Verbesserungen in Produktivität, Qualität und Vorhersagbarkeit

Unterschied zwischen traditionellen und agilen Methoden?

- Agile Methoden besser für Projekte mit rascher time-to-market
- **Entscheidungsfaktoren nach Böhm und Turner:**
  - **Teamstruktur:** Qualifikation muss bei agilen Methoden höher sein
  - **Dynamik der Anforderungen:** je häufiger Anforderungen geändert werden, desto einfach muss das möglich sein
  - **Entwicklungskultur:** je nach organisatorischem Umfeld
  - **Teamgröße:** agile Methoden für kleine – mittelgroße Teams
  - **Kritikalität:** Menschenleben vom System abhängig → rigorose Prüfung, Nachvollziehbarkeit auf allen Ebenen

Was sind Buildmanagementsysteme? Erklärung und 3 Beispiele.

- Werkzeug, um immer wiederkehrende Aufgaben zum Bauen einer Applikation automatisiert ablaufen zu lassen
- stellt Abhängigkeiten zwischen Komponenten sicher
- **Beispiele:** Make, Ant, Maven

Was sind Softwaremetriken in der Qualitätssicherung, warum werden sie verwendet? Geben Sie 3 Beispiele objektorientierter Metriken an.

- Messung der Qualität von Software
- **Prozessmetriken:** quantitative Daten des Entwicklungsprozesses
- **Produktmetriken:** messen Software/Teile davon, ohne Entwicklungsprozess, -stand zu berücksichtigen
- Beispiel Umfangsmetrik: Lines of Code
- **Objektorientierte Metriken:**
  - **DIT:** depth of inheritance tree
  - **CBO:** coupling between objects
  - **NOC:** number of childen

Was ist der Unterschied zwischen Internationalisierung und Lokalisierung?

- **Internationalisierung:** Software so gestalten, dass sie möglichst einfach an andere Sprachen und Kultur angepasst werden kann
- **Lokalisierbarkeit:** Wie gut lässt sich ein Produkt übersetzen, ohne in den Source Code eingreifen zu müssen
- **Lokalisierung:** eigentlicher Übersetzungsprozess (Anpassung von Datenformatierungen, Hinzufügen spezifischen Codes): für jede Sprache/Kultur notwendig, vollständig sehr aufwendig
- **Tipps:**
    - Kein Hard-coden von Nachrichten, Beschriftungen → Ressourcen-Files
    - Nachrichten nicht stückeln
    - Identische Nachrichten in unterschiedlichem Kontext mehrfach speichern
    - von Beginn an vorsehen

Erklären Sie das Test Driven Development anhand einer Skizze. Welchen Bezug zu automatisierten Tests zu TDD gibt es?

- Tests zuerst, bestimmen welcher Code geschrieben werden muss
- Code muss getestet werden

Service-orientierte Architektur erklären. Was ist BPM? Zusammenhang SOA und BPM?

- mehrere Abstraktionsebenen:
  - Service Layer
  - Component Layer
  - Class Layer
- unterschiedliche Notationen je Layer:
  - BPMN (Service Layer)
  - Komponentendiagramme (Component Layer)
  - UML (Class Layer)
- **Paradigmen:**
  - **Loose Coupling:** ermöglicht Wiederverwendung, Flexibilität
  - **Abstraktion:** Reduktion von Neuentwicklungen
  - **Standardisierung:** erlaubt Zusammenarbeit von Services
  - **Composible:** Standardisierte Services können zu Komponenten zusammengefasst werden.
- **Business Service:** Schritt in Geschäftsprozess extrahiert und gekapselt, serviceorentierte Architekturen sind an Geschäftsprozesse angelehnt
- **Business Process Modelling** und **SOA** eng miteinander verknüpft: Anwendungen maßgeblich von Geschäftsprozess-Modellierung beeinflusst

Äquivalenzklassenanalyse, Grenzwertanalyse erklären und Unterschiede im Review-Prozess. Warum wichtig? Erklären sie den Ablauf, und die Rollen.

- **Äquivalenzklassenanalyse:**
  - Einteilung möglicher Ein-, Ausgabewerte in Klassen mit gleichem Systemverhalten
  - Werte von Klassen gewählt, bei denen angenommen wird, dass Fehler auftreten, die auch bei anderen Werten der Klasse auftreten können
- **Grenzwertanalyse:**
  - Spezialfall der Äquivalenzklassenanalyse
  - Fehler treten besonders oft an Grenzen der Äquivalenzklassen auf

V-Modell Beschreiben + Skizze

- ausgerichtet auf Durchführungsphase
- Zweck:
  - Vertragsgrundlage bei Auftragsvergabe
  - Arbeitsanleitung für Softwareentwicklung
  - Kommunikationsbasis zwischen beteiligten Parteien
- Submodelle:
  - Projektmanagement
  - Systemerstellung
  - Qualitätssicherung
  - Konfigurationsmanagement
- Entwicklungsphilosophie:
  - inkrementelle Entwicklung
  - zu Beginn Gesamtfunktionalität abgegrenzt, einzelne Funktionen priorisiert
  - Funktionen gemäß Priorität Ausbaustufen zugeordnet, im Rahmen dieser umgesetzt
  - Ergebnis einer Stufe Endanwendern zur Verfügung gestellt
  - Umfang wächst ausgehend von Basisfunktionalität ständig an
- **V-Modell XT = extreme tailoring/extendable:**
  - bessere Anpassung an unterschiedliche Projekte, Organisationen
  - Skalierbarkeit
  - bessere Änderungs-, Erweiterungsmöglichkeit
  - Anpassung an aktuelle Normen, Vorschriften
  - Betrachtung des gesamten Systemlebenszyklus
  - Unterstützung der Einführung, Verbesserung des Modells selbst, Anpassung an organisationsspezifische Gegebenheiten
  - Eigenschaften:
    - ziel- und ergebnisorientiert
    - Produkte im Mittelpunkt
    - Reihenfolge der Produktfertigstellung durch Projektdurchführungsstrategien vorgegeben
    - Auf Basis der Bearbeitung, Fertigstellung von Produkten detaillierte Produktplanung und -steuerung vorgenommen

Stairs und Fork SW-Architektur erklären + Skizze

- **Fork:** zentrales Objekt übernimmt Kontrolle
- **Stair:** Kapselung in Schritten

Prüfung 2010-12-10

Was sind Vorgehensmodelle und wozu sind sie gut?

- immer umfangreichere, komplexere Softwareprodukte → effektive, effiziente Software-Entwicklungs-Prozessmodelle gewinnen an Bedeutung, wesentlich für Geschäftserfolg
- Ziel: Softwareentwicklung auf etablierte ingenieurmäßige Basis stellen, Verbesserungen in Produktivität, Qualität, Vorhersagbarkeit
- kein optimaler Entwicklungsprozess → je nach Projektart geeignetes Verfahren wählen
- 4 Steuergrößen eines Projekts:
    - Kosten
    - Qualität
    - Zeit
    - Umfang

Beschreiben Sie das Spiralmodell von Boehm und zeichnen sie die Phasen ein. Welche Phasen gibt es? Beschreiben sie das Modell.

- berücksichtigt mögliche Projektrisiken der sequentiellen Entwicklung → iterativ aufgebaut
- Gesamtprozess in 4 Phasen gegliedert, alle im Rahmen einer evolutionären Entwicklung mehrmals durchlaufen:
    - Zieldefinition
    - Risikoanalyse
    - Durchführen der Arbeitsschritte
    - Planen der nächsten Phase

Was sind Buildmanagementsysteme? Erklärung und 3 Beispiele.

- Werkzeug, um immer wiederkehrende Aufgaben zum Bauen einer Applikation automatisiert ablaufen zu lassen
- Stellen Abhängigkeiten zwischen Komponenten sicher
- Bsp: Make, Maven, Ant

Was sind Softwaremetriken in der Qualitätssicherung, warum werden sie verwendet? Geben Sie 3 Beispiele objektorientierter Metriken an.

- Ermöglichen Qualität von Software zu interpretieren
- quantitative Daten des Softwareentwicklungsprozesses
- **Produktmetriken:** messen Software, Teile davon ohne Entstehungsprozess, Zustand zu berücksichtigen
- **Umfangsmetrik:** Lines of Code
- **Objektorientierte Metriken:** berücksichtigen auch Beziehungen von Objekten untereinander und deren Struktur
    - **CBO (coupling between objects)**
    - **DIT (depth of inheritance tree)**
    - **NOC (number of children)**
    - **RFC (response for a class)**
    - **WMC (weighted methods for a class)**
    - **LCOM (lack of cohesion** = Zusammenhalt **in methods)**

Was ist der Unterschied zwischen Internationalisierung und Lokalisierung?

- **Internationalisierung:** Software so gestalten, dass einfach an andere Kulturen, Sprachen anpaßbar
- **Lokalisierbarkeit:** Übersetzbarkeit ohne Eingriff in den Source Code
- **Lokalisierung:**
    - eigentlicher Übersetzungsprozess
    - Anpassung von lokalen Datenformatierungen
    - Hinzufügen spezifischen Codes
    - für jede Sprache/Kultur separat
    - vollständig sehr aufwändig
- **Tipps für Lokalisierung:**
    - Kein „hard-coden“ von Nachrichten, Beschriftungen
    - Extrahieren in Ressourcen-Files (Teile, mit denen Benutzer agiert, Kommentierung + sinnvolle Gruppierung)
    - Schriftgrößen nicht einschränken
    - Nachrichten nicht stückeln
    - Identische Nachrichten in unterschiedlichem Kontext mehrfach speichern
    - gleich beim Implementierungsbeginn, später sehr teuer

Erklären Sie das Test Driven Development anhand einer Skizze. Welchen Bezug zu automatisierten Tests zu TDD gibt es?

- Tests zuerst, kein ungetesteter Code produktiv
- Tests bestimmen zu schreibenden Code
- **2 goldene Regeln:**
  - Neuen Code nur schreiben, wenn ein automatisierter Test fehlschlägt
  - Duplikate eliminieren
- **Automatisierte Tests:**
  - jeder Entwickler auch Tester
  - Tests oft unstrukturiert, oberflächlich, lückenhaft → automatisierte, feingranulare Funktionstests (Unit Tests)
  - einfach ausführbar (Unterstützung durch IDE, Integration in Build Prozess)
  - Vermeidung ungewollter Nebeneffekte

Service-orientierte Architektur erklären. Was ist BPM? Zusammenhang SOA und BPM?

- **Serviceorientiertes Design:**
  - mehrere Abstraktionsebenen, unterschiedliche Notationen je Layer:
    - **Service Layer: BPMN**
    - **Component Layer: Komponentendiagramm**
    - **Class Layer: UML**
  - Paradigmen der SOA:
    - **Loose Coupling:** der Services ermöglicht hohen Reuse und Flexibilität
    - **Abstraktion:** Reduktion von Neuentwicklungen
    - **Standardisierung:** Zusammenarbeit von Services
    - **Composable:** standardisierte Services zu Komponenten zusammenfassen
- **Service Oriented Architecture (SOA):**
  - **Business Service:** Schritt in einem Geschäftsprozess extrahiert und gekapselt, Business Services können auch Geschäftsziele realisieren, müssen keinen Schritt im Geschäftsprozess darstellen
  - **Business driven:** an Geschäftsprozesse angelehnt
- **Business Process Modelling (BPM):** eng miteinander verknüpft, Anwendungen maßgeblich von Geschäftsprozess-Modellierung beeinflusst
- Zusammenhang BPM und SOA:
  - 2 Prozesse greifen auf gleiches Business Service zu, wird in Form zweier technischer Services gelöst

- **SOA – State-of-the-Art Architektur:**
  - Geschäftsprozessorientiert
  - Anwendungen durch mehrere Services gelöst
  - Services in einer Anwendung sollen auch aus anderen heraus aufgerufen werden können → Reuse, Modularisierung
  - Zusammenstellung von Services in bestimmen Abläufen zur Umsetzung von Geschäftsprozessen
  - Services auf mehreren Abstraktionsebenen (Business Service, Component Service, Technisches Service)
  - beschreibt Architektur, keine Technologe (Webservices, oder auch **CORBA (Common Object Request Broker Architecture):** objektorientierte Middleware)

Blackbox und Whitebox Frameworks. Unterschied? Wie werden diese spezialisiert? Erklären von "Method Hooks" und "Hot Spots".

Frameworks:

- Applikationsskelett, vom Entwickler anpaßbar, um Anforderungen optimal umzusetzen
- wiederverwendbarer Entwurf, oft durch Menge abstrakter Klassen, Zusammenspiel ihrer Instanzen beschrieben
- Verwendung: effizienter **Software Reuse**, reduziert Applikationsentwicklung auf Spezialisierung vorgegebener Strukturen
- **Vorteile:**
  - Reduktion von Entwicklungskosten, kürzere Entwicklungszeiten
  - Wiederverwendung von Quellcode, bewährten architektonischen Strukturen
  - Einhaltung von Standards
- **Nachteile:**
  - Vererben von Fehlern
  - Einarbeitungsaufwand
- **Bibliothek:** Sammlung von Software und Dokumentation

Framework – Bibliothek:

- Bibliothek: wiederverwendbare Funktionalität – Framework: wiederverwendbares Verhalten
- Bibliothek: Methoden durch eigenen Code aufgerufen – Frameworks: eigener Code durch Framework aufgerufen
- Frameworks bestehen aus abstrakten Klassen, bieten Funktionalität in teilweise implementierten Methoden an (können Funktionalität aus Bibliotheken beziehen)

Eigenbau – Framework:

- Deckt Framework Anforderungen ab?
- Wartung, Weiterentwicklung gesichert?
- kommerziell – frei?
- Verfügbarkeit erfahrener Entwickler?
- Eigenentwicklung nur sinnvoll wenn:
  - kein vorhandenes Framework
  - Anpassung zumindest gleich aufwändig
  - spezielle (nichtfunktionale) Anforderungen

Framework-Typen:

- **Persistence Frameworks:** Hibernate
- **Web Application Frameworks:** Ruby on Rails
- **Rich Client Frameworks:** Eclipse RCP
- **Grafik Frameworks:** Direct X

Framework –Eigenschaften:

- **Method Hooks:** abstrakte Methoden, vom Entwickler überschrieben um gewünschtes Verhalten zu erstellen
- **Hot Spots:** Punkte, an denen ein Framework spezialisiert wird
- **Whitebox Frameworks:** innere Struktur muss für Spezialisierung sichtbar sein (erfolgt meist durch Vererbung)
- **Blackbox Frameworks:** enthält stabile Hot Spots, Spezialisierung kompositionsbasiert (Komponenten an Framework übergeben)

Äquivalenzklassenanalyse, Grenzwertanalyse erklären und Unterschiede.

Testabdeckung:

- Ziel der **Verifikation** von Softwaresystemen: möglichst hohe Abdeckung/Coverage
- Testfallselektion, da vollständige taxative Aufzählung aller möglichen Systemzustände unmöglich
- Ziel: Testfälle identifizieren, die mit höchster Wahrscheinlichkeit Fehler finden, größtmögliche Abdeckung erreichen
- Zwei Ansätze:
    - **Strukturelle Abdeckung (White Box-Tests):**
        - Anweisungsabdeckung
        - Bedingungsabdeckung
        - Pfadbabdeckung
    - **Funktionale Abdeckung (Black-Box-Tests):**
        - **Äquivalenzklassenanalyse:**
            - Einteilung möglicher Ein-, Ausgabewerte in Klassen gleichen Systemverhaltens
            - Werte gewählt, bei denen wahrscheinlich Fehler auftreten, die auch bei anderen Werten der Klasse auftreten können
        - **Grenzwertanalyse:**
            - Spezialfall der Äquivalenzklassenanalyse
            - Fehler treten besonders oft an Grenzen der Äquivalenzklassen auf

Review-Prozess. Warum wichtig? Erklären sie den Ablauf, und die Rollen.

- **Review:** formell organisierte Zusammenkunft von Personen zur inhaltlichen/formellen Überprüfung eines Produktteils nach vorgegebenen Prüfkriterien und –listen
- **Vorteile:**
  - schon sehr früh im Entwicklungsprozess möglich, noch bevor ausführbare Programme zum Testen vorliegen → Verkürzung der Latenzzeiten von Fehlern, Kosten der Behebung geringer (kompensieren Review-Aufwand)
- **Rollen:**
  - **Manager:** Projektleiter, hat Auftrag zur Erstellung des Testobjekts gegeben, verantwortlich für Freigabe
  - **Moderator/Review-Leiter:** plant, organisiert, leitet Review, sucht Teilnehmer aus, stellt Unterlagen bereit, organisiert Sitzung
  - **Schreiber:** Protokoll
  - **Autor:** Urheber des Testobjekts oder Repräsentant des Teams
  - **Gutachter/Reviewer:** Begutachten Material in der Vorbereitung, berichten in der Sitzung über Erfahrungen
  - **Leser:** für Aufbereitung zu prüfender Softwareelemente, zeitlich angemessene Sitzungsdurchführung verantwortlich
- **Prozess:**
  - beginnt mit Planungsphase
  - Review durch Initialisierung gestartet
  - Abschluss des Reviews bildet Freigabe
  - Review besteht aus Vorbereitung, Sitzung + Nacharbeit
  - Nacharbeit: Aussagen über Effektivität des Reviews

- **Review-Messung:** Ergebnisse im erwarteten Rahmen?
- **Review-Verfahren:**
  - **Stellungnahme:** Autor stellt Kollegen Kopien zur Verfügung, diese begutachten und retournieren kommentierte Papiere, Autor begutachtet Stellungnahme, korrigiert falls notwendig
  - **Walkthrough:** weniger formell, Diskussion/Interaktion zwischen Vortragendem und Teilnehmern
  - **Technisches Review:** Beurteilung durch qualifiziertes Team hinsichtlich beabsichtigter Verwendung, Identifikation von Abweichungen von der Spezifikation
  - **Inspektion:** formellstes, wirkungsvollstes Review-Verfahren, Zweck: Finden/Identifikation von Anomalien im Source Code, Prüfung auf Erfüllung der Spezifikation und Einhaltung von Richtlinien, Standards
  - **Round Robin Review:** Suche nach positiven Argumenten
  - **Peer Review:** wesentlichster Unterschied zur Inspektion: Prozess nicht definiert, keine speziellen Techniken zur Fehlersuche, Team ad-hoc zusammengestellt/permanent bestimmt Aufgabenstellung und Vorgehensweise

Prüfung 2010-05-17

Erklären sie die Grundcharakteristik, die wesentlichen Elemente und Rollen des SCRUM Prozessmodells.

Agiles Manifest:

- Individuen und Interaktionen sind wichtiger als Prozesse und Werkzeuge.
- Funktionierende Programme sind wichtiger als ausführliche Dokumentation.
- Die stetige Abstimmung mit dem Kunden ist wichtiger als die ursprüngliche Leistungsbeschreibung in Verträgen
- Der Mut und die Offenheit für Änderungen stehen über dem Befolgen eines festgelegten Plans.

Scrum:

- hyper-produktive Produktentwicklung
- beim Rugby angewandte Strategie (adaptive, schnelle, sich selbst organisierende Vorgehensweise mit wenigen Ruhepausen)
- **Zentrale Werte:**
  - hohe Produktivität und Anpassungsfähigkeit
  - wenig Risiko und Ungewissheit
  - größerer Komfort für die Projektmitglieder
- **Entwicklungsphilosophie:**
  - alle funktionalen, nichtfunktionalen Anforderungen im **Product Backlog** nach Wichtigkeit aufgelistet, **Product Owner** dafür verantwortlich (erhält Inhalte von Stakeholdern, nimmt Priorisierung vor), von allen Beteiligten jederzeit einsehbar
  - eigentliche Entwicklung in kleinen Teams
  - zentrales, wichtigstes Element: **Sprint** (definierte Zeitdauer, ca. 20 Tage), muss mit Fertigstellung neuer Funktionalität enden, **Sprint Backlog**, möglichst schnelles Erreichen des **Sprint Goals**
  - **Daily Scrum Meeting:** vom **Scrum Master** geleitet, immer zur selben Zeit am selben Ort
  - **Sprint Review Meeting:** am Ende des Sprints
- **Elemente:**
  - Sprint
  - Daily Scrum
  - Sprint Backlog
  - **Increment:** Softwareprodukt entsteht in Inkrementen, spätestens im zweiten Sprint lauffähiges Softwaresystem mit Kernfunktionalität
  - **Burndown Chart:** Scrum Master trägt tagesaktuelle Aufwandsschätzungen für aktuellen Sprint ein
- **Rollen:**
  - **Product Owner:** Erstellung, Priorisierung des **Product & Release Backlogs**, Team von Störungen/Unterbrechungen frei halten
  - **Scrum Team:** Umsetzung der vom Sprint Backlog geforderten Funktionalität, Erreichung des Sprint Goal
  - **Scrum Master:** achtet auf Umsetzung der Grundsätze, Regeln von Scrum, schottet das Team von äußeren Einflüssen ab

eXtreme Programming (XP):

- kompakte Methode zur Softwareentwicklung in kleinen bis mittelgroßen Teams, deren Arbeit vagen, sich rasch ändernden Anforderungen unterliegt
- Hauptziel: termingerechtes Abliefern auftragsgemäßer Software
- Merkmale: eigenes Wertesystem, viele Prinzipien, System von sich gegenseitig unterstützenden Techniken
- **Prinzipien:**
  - Kommunikation: zwischen allen Beteiligten
  - Einfachheit
  - Feedback
  - Mut: Offenheit, Transparenz, Änderungen
  - Respekt: vor sich selbst, dem Team, dem Produkt
- **Techniken:**
  - Kunde vor Ort
  - Kurze Release-Zyklen
  - Testen
  - Einfaches Design
  - Refactoring
  - Pair Programming
  - Gemeinsame Verantwortlichkeit
  - 40-Stunden-Woche
  - Programmierstandards
- **Rollen:**
  - Entwickler
  - Kunde
  - XP-Trainer
  - Tracker
  - Tester
  - Berater
  - Big Boss

Meilensteintrendanalyse Bsp. Achsen erklären. Diagramm interpretieren.
Meilensteine:

Meilensteintrendanalyse:

Wozu dient SPICE? 6 Reifegrade!
CMM (Capability Maturity Mode): Ziel: Vergleichbarkeit und Qualität von Softwareprozessen im Rahmen der Vergabe an externe Lieferanten zu verbessern

CMMI (Capability Maturity Model Integration):

- vereinheitlichter, modularer Nachfolger von CMM
- liefert Analyse des Ist-Zustandes eines Unternehmens, kein international anerkanntes Zertifikat
- CMMI-Assessment beurteilt Fähigkeit einer Organisation, Software unter bestimmten Rahmenbedingungen erstellen zu können
- **Prinzipien und Konzepte:**
    - **Quantitative Managementmethoden:** erhöhen Fähigkeit, Qualität zu kontrollieren, Qualität zu steigern
    - **5 Ebenen Capability Maturity Model:** Bewertung der Leistungen, bestimmt zur Erreichung der nächsten Stufe nötigen Aufwände
    - **Generische Prozessgebiete:** Was? – nicht Wie? → Anwendung auf breite Palette von Organisationsumsetzungen
- **5 Reifegradstufen**

SPICE (Software Process Improvement for Capability Determination:

- ähnlich CMMI → Konkurrenz
- zweidimensionales Referenzmodell:
  - **Prozessdimension**
  - **Fähigkeitsdimension**
- **Prinzipien und Ziele:**
  - Zusammenführung verschiedener Assessment-Methoden zu zusammenhängendem Systemmodell
  - Universelle Einsetzbarkeit
  - Hohe Professionalität
  - internationale Akzeptanz, weltweiter Standard
- **Zielsetzungen:**
  - Verstehen des Entwicklungsstandes der eigenen Prozesse → Verbesserungen
  - Eignung der eigenen Prozesse in Bezug auf Anforderungen
  - Eignung der Prozesse anderer Organisationen in Bezug auf Verträge
- **Reifegradstufen:**
  - **0 – unvollständig**
  - **1 – vorhanden:** Process Performance
  - **2 – geleitet:**
    - Performance Management
    - Work Product Management
  - **3 – gesichert:**
    - Process Definition and Tailoring
    - Process Resource Availability
  - **4 – vorhersehbar:**
    - Process Measurement
    - Process Control
  - **5 – optimiert:**
    - Process Change
    - Continuous improvement
- **Erfüllungsgrade:** nicht/teilweise/weitestgehend/vollständig erreicht
- **Prozesskategorien:**
  - Engineering
  - Customer-Supplier
  - Management
  - Support
  - Organisation

Matrix-Projektorganisation erklären.

Projektorganisation:

- **Reine Projektorganisation:**
  - Projektleiter trägt Gesamtverantwortung, volle Weisungsbefugnis
  - Vorteile:
    - eindeutige Verantwortung
    - starke Identifikation der Mitarbeiter mit dem Projekt
    - hohe Motivation
  - Nachteile:
    - schwierige Akquirierung der Mitarbeiter
    - problematische Wiedereingliederung der Mitarbeiter am Ende des Projekts
- **Einfluss-Projektorganisation:**
  - Projektleiter nur Projektkoordinator, kaum Weisungsbefugnis
  - Vorteile:
    - geringe organisatorische Veränderungen
    - hohe personelle Flexibilität
    - Kommunikation mit Fachabteilung bleibt erhalten
  - Nachteile:
    - keine klare Verantwortlichkeit
    - Fehlende Autorität des Projektleiters
    - hohes Konfliktpotential
    - fehlender Teamgeist
- **Matrix-Projektorganisation:**
  - Projektleiter trägt Gesamtverantwortung, keine volle Weisungsbefugnis
  - Vorteile:
    - Projektverantwortung durch Projektleiter
    - Sicherheitsgefühl der Mitarbeiter
    - flexibler Personaleinsatz
  - Nachteile:
    - Mitarbeiter Diener zweier Herren
    - Konfliktpotential Linie/Projekt
    - hoher Koordinationsaufwand

Risikomanagement:

- **Risiko:** unsicheres Ereignis, von dem nicht bekannt ist, ob es eintreten wird und welchen Schaden es verursachen wird
- **Risikofaktor = Risikowahrscheinlichkeit * Schadenshöhe**
- **Risikomanagement:** systematischer Prozess zur Identifizierung, Analyse, Behandlung und Kontrolle der Projektrisiken
- **Risikoidentifikation:**
  - Ziel: vollständige Liste konkret formulierter Risiken
  - Workshops mit Kreativitätstechniken
  - Checklisten, Fragebögen
  - Erfahrungen aus vorangegangenen Projekten
- **Risikoanalyse:**
  - Ziel: qualitative und quantitative Bewertung der Risiken
  - Eintrittswahrscheinlichkeit je Risiko
  - Schadensausmaß je Risiko
  - Risikomatrix zur Priorisierung
- **Risikobehandlung und -kontrolle:**
  - Ziel: Erstellung eines Maßnahmenplans
  - Verhältnismäßigkeit Schadenshöhe/Eintrittswahrscheinlichkeit – Aufwand/Kosten
  - Maßnahmenplan:

Risiko	Wahrscheinlichkeit	Schaden	Klasse	Maßnahme	Verantwortlicher
Prototyp funktioniert nicht	sehr wahrscheinlich	erheblich	hoch	Spezialisten einbinden	Projektleiter
Projekträumlichkeiten zu klein	eher unwahrscheinlich	klein	gering	weitere Projekträume	Auftraggeber

Kopplung und Kohäsion: Zusammenhang und Auswirkung auf Wartung.
Konzepte für wartungsfreundliche Implementierung:

- **Design Patterns:**
  - **Creational Patterns:** Singleton, Prototype
  - **Structural Patterns:** Proxy
  - **Behavioral Patterns:** Observer, Iterator
- **Information Hiding:** Kommunikation von Komponenten über Interfaces
- **Kopplung:** Abhängigkeit **zwischen Klassen**, sollte **möglichst gering** sein
- **Kohäsion:** Abhängigkeit **innerhalb von Klassen**, sollte **möglichst hoch** sein: durch Methodengruppierungen nach Zweck

Erklären sie den Unterschied zw. zentraler, dezentraler, lokaler Versionskontrolle + je 2 Bsp.
Versionsmanagement:

- **Repository:** Datenbestand, aus dem Arbeitskopie generiert
- **Checkout:** Übertragung der Daten aus dem Repository
- **Check-in/Commit:** Übertragung der Daten in das Repository
- **Branching:** Anlegen einer Kopie von Objekten im Versionsmanagementsystem
- **Markieren/Tagging:** Tag kennzeichnet konkreten Entwicklungsstand (zB zur Auslieferung bestimmt)
- **Protokollierung/Historisierung:**
  - Welche Änderungen von welchem Benutzer zu welcher Zeit?
  - Wiederherstellung eines beliebigen Versionsstandes aus der Vergangenheit
- **Zentrales Versionsmanagement:**
  - Server + beliebig viele Clients
  - zB Subversion (SVN)
- **Dezentrales Versionsmanagement:**
  - kein zentrales Repository, jeder Entwickler besitzt sein eigenes
  - Änderungen zwischen Repositorys ausgetauscht
  - zB Git
- **Lokales Versionsmanagement:**
  - älteste Form, beschränkt auf Versionierung einzelner Dateien
  - nicht geeignet für Teamprojekte
- Strategien:
  - **Lock-Modify-Unlock-Strategie (Pessimistic Revision Control)**
  - **Copy-Modify-Merge-Strategie (Optimistic Revision Control)**

Stairs und Fork SW-Architektur erklären.

- **Fork:** zentrales Objekt übernimmt Kontrolle
- **Stair:** Kapselung in den Schritten

Was versteht man unter der Spezialisierung von Frameworks? Wie geschieht die bei Whitebox Frameworks und Blackbox Frameworks?

- **Method Hooks:** abstrakte Methoden, vom Entwickler überschrieben, um gewünschtes Verhalten zu erstellen
- **Hot Spots:** Punkte, an denen Framework spezialisiert wird
- **Whitebox Frameworks:** innere Struktur muss sichtbar sein um spezialisieren zu können (meist durch Vererbung)
- **Blackbox Frameworks:** enthält stabile Hot Spots, Spezialisierung kompositionsbasiert, Komponenten an Framework übergeben

5 Beziehungen eines GANTT Diagramms benennen und erklären (Anfang-Ende-Beziehung, etc.)

- älteste, am weitesten verbreitete grafische Methode
- Aufgaben zur Terminplanung dargestellt
- aufgaben- oder personenbezogen

- Pufferzeit eines Vorgangs:
  - **Gesamtpuffer:** Zeitspanne, um die sich ein Vorgang verzögern darf, ohne dass das Projektende verzögert wird
  - **Freie Puffer:** Zeitspanne, um die sich ein Vorgang verzögern darf, ohne dass andere Vorgänge verzögert werden
- **Kritischer Vorgang:** Vorgang ohne Pufferzeit
- **Kritischer Pfad:** enthält nur kritische Vorgänge

Was ist "SOA"? Eigenschaften? Paradigmen?
SOA (Service Oriented Architecture):

- mehrere Abstraktionsebenen, unterschiedliche Notationen je Layer:
  - **Service Layer:** BPMN
  - **Component Layer:** Komponentendiagramme
  - **UML:** Class Layer
- **Paradigmen:**
  - **Loose Coupling:** Reuse, Flexibilität
  - **Abstraktion:** Reduktion von Neuentwicklungen
  - **Standardisierung:** Zusammenarbeit von Services
  - **Composable:** Zusammenfassung standardisierter Services zu Komponenten

Was sind Integrationstests? Unterschiedliche Integrationsformen? Erläutern sie in diesem Zusammenhang die Begriffe Stub and Driver!
Komponententest:

- Prüfung separat testbarer Komponenten (zB Module, Programme, Objekte, Klassen) auf vorhandene Fehler
- Isolation der einzelnen Komponenten vom Rest des Systems mit Hilfe von Platzhaltern (Stubs) bzw. Simulatoren und Testtreiber

Integrationstest:

- Prüft Schnittstellen zwischen Komponenten
- mehrere Integrationsstufen möglich
- mit Größe des Integrationsumfangs wird Isolation von Fehlerwirkungen schwieriger
- Inkrementelle Integrationsstrategien – Bing-Bang-Strategie
- horizontale – vertikale Integration
- **Bottom-Up-Integrationstest – Top-Down-Integrationstest**

Systemtest:

- spezifiziertes Verhalten eines Gesamtsystems oder Produktes
- Abdeckung funktionaler und nichtfunktionaler Anforderungen
- basiert auf:
  - Anforderungsspezfikationen, Anwendungsfällen + sonstigen Beschreibungen
  - Geschäftsprozessen
  - Risikoanalysen
  - Erfahrungen
  - Systemressourcen

Akzeptanztests:

- Nachweis der Erbringung der von Auftraggeber und Auftragnehmer vereinbarten Leistung
- meist von Kunden oder Benutzern durchgeführt
- Arten:
  - Anwender-Abnahmetest
  - Betrieblicher Abnahmetest

  - Vertraglicher Abnahmetest
  - Alpha- und Beta-Test (Feldtest)

Grundsätze des Softwaretests:

- Testen zeigt Anwesenheit von Fehlern
- Vollständiges Testen nicht möglich
- Frühzeitig beginnen
- Häufung von Fehlern
- Wiederholungen haben keine Wirksamkeit
- Testen abhängig vom Umfeld
- Keine Fehler bedeuten nicht zwangsläufig brauchbares System

Prüfung 2010-04-12

Erklären sie die Grundcharakteristik, die wesentlichen Elemente und Rollen des Unified Process.

Unified Process:

- IBM Rational
- **Vorgehensweisen:**
  - iterative Softwareentwicklung
  - Anforderungen managen
  - Komponentenbasierte Architekturen verwenden
  - Software visuell modellieren
  - Softwarequalität kontinuierlich prüfen
  - Änderungen kontrolliert in Software einbringen
- **Tailoring:**
  - selektive Adaption für jedes Projekt/jede Organisation
  - Anpassung an spezifische Gegebenheiten, Kombination mit anderen Modellen
  - **Process Engineer**

Erklären sie die Delphi-Methode.

Aufwandsschätzung:

- Informationen durch Erfahrungen bewertet, in Arbeitsaufwände umgerechnet (Personentage/Personenstunden)
- genaue Wissensbasis erforderlich (Umfang der Arbeitspakete, genaue Ergebnisse, erforderliche Schritte zur Durchführung)
- Besprechung der Arbeitspakete mit den Verantwortlichen
- **Delphi-Methode:**
  - Moderator + Experten
  - jeder Experte gibt zu jedem Arbeitspaket einen Schätzwerk ab
  - Mittelwert, falls alle Schätzwerte in bestimmter Bandbreite (+ 20 %)
  - sonst Argumente austauschen → neue Schätzung
  - zusätzlich 10 – 15 % für Projektmanagement
- **3-Experten-Konzept:**
  - 3 optimistische, 3 realistische, 3 pessimistische Schätzungen
  - Abklärungen, bis Ergebnisse in jeder Kategorie übereinstimmen
  - daraus Schätzwert ableiten

Wozu dient CMM? 5 Reifegrade!

CMM (Capability Maturity Model):

- US-Verteidigungsministerium, 80er-Jahre
- Ziel: Vergleichbarkeit, Qualität von Softwareprozessen im Rahmen der Vergabe an Lieferanten zu verbessern
- 2003 vereinheitlichter, modularer Nachfolger: **CMMI (Capability Maturity Model Integrated)**
- Analyse des Ist-Zustands des Unternehmens (beurteilt Fähigkeit, Software unter bestimmten Rahmenbedingungen erstellen zu können), kein international anerkanntes Zertifikat
- **Prinzipien und Konzepte:**
    - **Quantitative Managementmethoden:** erhöhen Fähigkeiten der Qualitätskontrolle, Produktivitätssteigerung
    - 5-Ebenen-Capability Maturity Model: ermöglicht Bewertung der Leistungen, bestimmt zur Erreichung der nächsten Stufe nötige Aufwände
    - Generische Prozessgebiete: Was? – nicht Wie? → Anwendung auf breite Palette von Organisationsumsetzungen
- **Struktur eines CMMI-Prozessgebietes:**
    - **Spezifische Ziele:**
        - **Typische Arbeitsprodukte**
        - **Teilpraktik**
    - **Generische Ziele:**
        - **Erläuterungen zur generischen Praktik**
        - **Teilpraktik**
- **Reifegradstufen:**
    - **Reifegrad 1 – Initial**
    - **Reifegrad 2 – Wiederholbar:**
        - Softwarequalitätssicherung
        - Projektplanung
        - Anforderungsmanagement
    - **Reifegrad 3 – Definiert:**
        - Peer Reviews
        - Training und Schulung
    - **Reifegrad 4 – Geführt:**
        - Softwarequalitätsmanagement
        - Quantitatives Prozessmanagement
    - **Reifegrad 5 – Optimiert:**
        - Process Change Management
        - Technologietransfer
        - Fehlervermeidung

Was versteht man unter einer "Baseline" in der Analyse? Was passiert üblicherweise bei Änderungen (Changes) nachdem eine Baseline gesetzt wurde?
Anforderungen:

- **Anforderungsvolatilität:**
  - **Stabile Anforderungen**
  - **Volatile Anforderungen:** ändern sich voraussichtlich während Anforderungsanalyse oder Betrieb
  - **Systemdesignauswirkungen:**
    - stabile Anforderungen mit simpleren Design, „hard-wired“
    - Umsetzung volatiler Anforderungen erfordert adaptiveres Systemdesign „soft-wired“
    - stabile Anforderungen früher, detaillierter dokumentieren, modellieren
    - stabile Anforderungen können früher implementiert werden
- **Baseline:**
  - nie echtes **Freeze** der Anforderungen
  - stabile Zustände unabdingbar → **Baseline**: Systemspezifikation, die formal reviewed wurde und auf die man sich geeinigt hat als Basis für die Entwicklung, Änderungen nur formal
  - bei Änderungen: **Change-Impact-Analysis, Re-Costing**
- **Klassifikation von Anforderungen:**
  - **Benutzeranforderungen – Systemanforderungen**
  - **Funktionale Anforderungen:** Funktionen/Services des Systems, Reaktion auf bestimmte Eingaben/Situationen
  - **Nichtfunktionale Anforderungen:** Qualitätseigenschaften, Entwicklungsprozess, einzuhaltende Standards, Normen, gesetzliche Rahmenbedingungen
  - **Domainanforderungen:** besondere Eigenschaften/Einschränkungen der Domäne, oft inhärent, nicht explizit pro Projekt dokumentiert
- **Gute Anforderungen:**
  - komplett, konsistent
  - **Überprüfung:**
    - **Anforderungs-Review:** strukturierte, systematische Überprüfung der Qualität der Anforderungen, Erkennen und Auflösen von Konflikten zwischen Stakeholdern (zB Priorisierung), Beteiligung unterschiedlicher Stakeholder (fachlich, technisch)
    - **Prototyping:** Anforderungen umsetzbar, eindeutig, vollständig?
    - **Abnahmetestfallerstellung:** Testbarkeit erreicht, Prüfung auf Eindeutigkeit und Vollständigkeit ermöglicht
- **Typische Probleme bei Anforderungen:**
  - **Verständlichkeit:** Sprache der Applikationsdomäne
  - **Selbstverständlichkeit:** Domain-Spezialisten sind mit der Domäne zu vertraut
  - **Wenig Präzision:** sonst schwer lesbar
  - **Vermischung:** funktionale und nichtfunktionale Anforderungen
  - **Zusammenführung:** verschiedene Anforderungen gemeinsam beschrieben, nicht genau identifiziert und abgegrenzt
  - **Mehrdeutigkeit:** natürliche Sprache
  - **Überflexibilität:** Beschreibung mehrdeutig interpretierbar, nicht exakt

- **Qualitätsattribute von Anforderungen:**
  - **Vollständigkeit:** explizite Beschreibung, keine impliziten Annahmen der Stakeholder
  - **Eindeutig abgegrenzt**
  - **Verständlich beschrieben:** für alle Stakeholder verständlich
  - **Keine Vermischung**
  - **Identifizierbar:** Anforderungs-ID
  - **Einheitliche Dokumentation:** gleiches Dokument, gleiche Struktur
  - **Notwendigkeit?**
  - **Nachprüfbar:** Abnahmetestfälle
  - **Priorisierung:** Berücksichtigung im Entwicklungsprozess, Auflösung konkurrierender Anforderungen
- **Anwendungsfälle (Use Cases) nach Cockburn:**
  - **Name, Identifier**
  - **Beschreibung**
  - **Beteiligte Akteure:**
    - **primär:** eigentliche Benutzer
    - **sekundär:** überwachen/warten System, unterstützen primäre Akteure bei Zielerreichung
  - **Status:** in Arbeit, bereit zum Review, im Review, abgelehnt, angenommen
  - **Verwendete Anwendungsfälle**
  - **Auslöser:** Grund für Ausführen
  - **Vorbedingungen:** für Ausführung
  - **Invarianten:** dürfen durch Anwendungsfall nicht verändert werden
  - **Nachbedingung/Ereignis:** erwarteter Zustand
  - **Standardablauf:** typisches Szenario (leicht zu verstehen oder häufigster Fall): Schritte numeriert, Aktivitätsdiagramme, Sequenzdiagramme
  - **Alternative Ablaufschritte:** häufig als bedingte Verzweigungen der normalen Ablaufschritte
  - **Hinweise:** besseres Verständnis, eventuelle Nebeneffekte
  - **Änderungsgeschichte**
- **Anwendungsfalldiagramm:**

- **User Stories/Kunde vor Ort:**
  - **Vorteile:**
    - kurz, prägnant, gut wartbar
    - fördern Diskussion/Abstimmung Kunde – Entwickler während gesamter Projektdauer
    - Reduzieren Risiko von Fehlentwicklungen aus Benutzersicht
    - Besonders gut in Kombination mit **Metapher** (gemeinsame Sicht auf das System)
  - **Nachteile:**
    - ohne zugehörige präzisierte Akzeptanz-Testfälle stark interpretierbar, Risiko für erfolgreiche Abnahme
    - setzen starke Involvierung des Kunden/Benutzers während des gesamten Prozesses voraus (oft unrealistisch)
    - personenzentriert
    - können kaum als Vertragsgrundlage dienen
  - **Story Card**
- **Geschäftsprozessmodellierung**
- **Anforderungsdokument:** offizielles Dokument, Vertragsgrundlage
  - MUST, MUST NOT, SHOULD, SHOULD NOT, MAY
  - **Einleitung:** Zweck, Abgrenzung, Definitionen, Referenzen, Überblick
  - **Beschreibung:** Produktperspektive (Systemschnittstellen, Benutzerschnittstellen), Software-Schnittstellen, Produktfunktionen, Nutzermerkmale, Rahmenbedingungen, Voraussetzungen, Abhängkeiten
  - **Spezifische Anforderungen:** externe Schnittstellen, Funktionen, Performance, logische Datenbankanforderungen, Designvorgaben, zu erfüllende Standards, Systemattribute (Zuverlässigkeit, Verfügbarkeit, Sicherheit, Wartung und Pflege, Portierbarkeit)

Erklären sie den Unterschied zw. Optimistic Revision Control und Pessimistic Revision Control.
Versionsmanagement:

- **Zentrales Versionsmanagement:** zentraler Server, beliebig viele Clients, zB Subversion
- **Dezentrales Versionsmanagement:** kein zentrales Repository, Änderungen zwischen Repositorys ausgetauscht, zB Git
- **Lokales Versionsmanagement:** beschränkt auf Versionierung einzelner Dateien, nicht geeignet für Teamprojekte
- **Lock-Modify-Unlock-Strategie (Pessimistic Revision Control):** SourceSafe
- **Copy-Modify-Merge-Strategie (Optimistic Revision Control):** Subversion

Welche Typen der Wartung werden unterschieden? Erklären sie die unterschiedlichen Typen der Wartung. Wieso ist die Unterscheidung wichtig?

Wartung:

- Wozu:
  - Fehlerbehebung
  - Verbesserung des Laufzeitverhaltens
  - Anpassung an geänderte Umgebung
  - keine Weiterentwicklung, keine Änderung des funktionalen Zustands
  - Update der technischen Realisierung
  - Weiterentwicklung parallel möglich
- Auslöser:
  - gefundene Fehler
  - Update einer Komponente, Sicherheitslücken
  - Performance-Probleme
- Aufwand oft höher als Initialentwicklung
- oft wiederholt notwendig
- **Typen:**
  - **Korrektive Wartung:** Ausbesserung von Fehlern, Abweichungen von Spezifikationen
  - **Präventive Wartung:** Verhinderung vermutlicher zukünftiger Fehler
  - **Adaptive Wartung:** Anpassung an geänderte Umgebung, Änderung verwendeter Software/Schnittstellen/Hardware
  - **Perfektionierende Wartung:** Verbesserung der Applikation, keine funktionale Änderung, Benutzerführung, Performance, Wartbarkeit
- **Wartungsmaßnahmen:**
  - **Software Reengineering:**
    - Neuentwicklung bei gleichbleibender Funktionalität
    - Qualitätssteigerung
  - **Reverse Engineering:**
    - Gewinnung von Code und Modellen aus vorhandenen Artefakten (wenn Source Code nicht mehr verfügbar)
    - Re-Dokumentation → trägt zum Codeverständnis bei
    - oft Vorstufe zum Reengineering
  - **Refaktorisierung:** während Inititalentwicklung und Wartung → Erhaltung der Wartbarkeit
  - **Anti Patterns:**
    - Gegenstück zu Software Patterns
    - Sammlung häufiger schlechter Lösungsansätze
    - Grund: mangelnde Erfahrung, Qualifikation
    - bieten Verbesserungsstrategien

Was versteht man unter der Spezialisierung von Frameworks? Wie geschieht die bei Whitebox Frameworks und Blackbox Frameworks?

Frameworks:

- Applikationsskelett, welches vom Entwickler angepasst (spezialisiert) werden kann, um Anforderungen optimal umzusetzen
- wiederverwendbarer Entwurf, oft durch Menge abstrakter Klassen und Zusammenspiel dieser Instanzen beschrieben
- effiziente Art des **Software Reuse**, reduziert Applikationsentwicklung auf **Spezialisierung** vorhandener Strukturen
- **Library:**
  - wiederverwendbare **Funktionalität**
  - **Methoden durch eigenen Code aufgerufen**
- **Framework:**
  - wiederverwendbares **Verhalten**
  - **eigener Code durch Framework aufgerufen**
- Frameworks bestehen aus abstrakten Klassen, bieten Funktionalität in (teilweise) implementierten Methoden, die Funktionalität aus Bibliotheken beziehen können
- **Framework-Typen:**
  - **Persistence Frameworks:** Hibernate
  - **Web Application Frameworks:** Ruby on Rails
  - **Rich Client Frameworks:** Eclipse RCP
  - **Grafik Frameworks:** Direct X
- **Method Hooks (Haken):** abstrakte Methoden, vom Entwickler überschrieben um gewünschtes Verhalten zu erstellen
- **Hot Spots:** Punkte, an denen Framework spezialisiert wird
- **Whitebox Frameworks:**
  - innere Struktur muss sichtbar sein, um spezialisieren zu können
  - Spezialisierung meist durch Vererbung
- **Blackbox Frameworks:**
  - enthält stabile **Hot Spots**
  - Spezialisierung komponentenbasiert, Komponenten an Framework übergeben

Was ist ein "Stakeholder"? 5 Typen des "Stakeholder" nennen!

- „Anteilhalter“, vertritt ein Interesse im Projekt, stellt damit Anforderungsquelle dar
- jede Stakeholdergruppe hat eigene Projektsicht, Konflikte → Anforderungspriorisierung
- Typische Stakeholder:
    - Kunde, Auftraggeber, Sponsor
    - Geschäftsführung, Abteilungsleiter (Auftraggeber- und Auftragnehmerseite)
    - Legacy Owner (Besitzer des Altsystems, der Altdaten)
    - Betreiber, Entwickler von Schnittstellensystemen
    - Benutzer, Benutzergruppen
    - Projektleiter
    - Systemarchitekt, IT-Stratege
    - Softwareentwickler
    - Qualitätssicherung und Test
    - Betrieb, Wartung

Was ist "SOA"? Eigenschaften? Paradigmen?

SOA (Service Oriented Architecture):

- umspannt mehrere Abstraktionsebenen, unterschiedliche Notationen je Layer:
  - **Service Layer:** BPMN
  - **Component Layer:** Komponentendiagramm
  - **Class Layer:** UML
- State-of-the-Art-Architektur
- **Paradigmen:**
  - **Loose Coupling:** hoher Reuse, Flexibilität
  - **Abstraktion:** Reduktion von Neuentwicklungen
  - **Standardisierung:** Zusammenarbeit von Services
  - **Composable:** Zusammenfassung standardisierter Services zu Komponenten
- **business driven:** an Geschäftsprozesse angelehnt → **Business Process Modelling (BPM)** und SOA eng verknüpft
- **Eigenschaften:**
  - Anwendungen durch mehrere Services gelöst
  - Services in einer Anwendung sollen auch aus anderen heraus aufgerufen werden können: Reuse, Modularisierung
  - **Orchestrierung:** Zusammenstellung von Services in bestimmten Abläufen zur Umsetzung von Geschäftsprozessen
  - beschreibt Architektur, keine Technologie (zB Webservices, aber auch CORBA = Common Object Request Broker Architecture)

Was sind Integrationstests? Unterschiedliche Integrationsformen? Erläutern sie in diesem Zusammenhang die Begriffe Stub and Driver!

- Zerlegung in Teststufen ermöglicht frühe Prüfung unterschiedlicher Teile des zu entwickelnden Systems
- Teststufen durch folgende Aspekte charakterisiert:
  - allgemeine Ziele
  - Arbeitsergebnisse → Testbasis
  - eigentliches Testobjekt
  - auftretende Fehlerwirkungen und -zustände, die identifiziert werden sollen
  - Anforderungen an Testrahmen: Werkzeugunterstützung
  - spezifische Ansätze und Verantwortlichkeiten
- **Komponententest:**
  - Prüfung separat testbarer Komponenten (Module, Programme, Objekte, Klassen) auf vorhandene Fehler
  - Isolation einzelner Komponenten vom Rest des Systems mit Hilfe von **Stubs** (Platzhaltern) bzw. Simulatoren und Testtreiber
- **Integrationstest:**
  - prüft Schnittstellen zwischen Komponenten
  - mehrere Integrationsstufen möglich
  - mit Größe des Integrationsumfangs wächst Schwierigkeit der Isolation von Fehlerwirkungen in Komponenten oder Systemen
  - **Inkrementelle Integrationsstrategien vs. Big-Bang-Strategie**
  - können auf Systemarchitektur, funktionalen Aufgaben, Transaktionsverarbeitungssequenzen oder weiteren Aspekten des Systems oder seiner Komponenten basieren
  - **horizontal – vertikal**
- **Systemtest:**
  - spezifiziertes Verhalten eines Gesamtsystems oder Produkts
  - sollen nichtfunktionale und funktionale Anforderungen abdecken
  - basieren auf:
    - Anforderungsspezifikationen
    - Anwendungsfällen, sonstigen Systembeschreibungen
    - Geschäftsprozessen
    - Risikoanalysen
    - Erfahrungen im Produktionsumfeld
    - Systemressourcen
- **Akzeptanztest:**
  - Fokus: Erbringung der von Auftraggeber und -nehmer vereinbarten Leistungen nachzuweisen
  - meist von Kunden/Benutzern eines Systems durchgeführt
  - Arten:
    - Anwender-Abnahmetest
    - Betrieblicher Abnahmetest
    - Vertraglicher Abnahmetest
    - **Alpatest, Betatest** (Feldttest)

Prüfung 2010-01-29

Erklären sie die Grundcharakteristik, die wesentlichen Elemente und Rollen des SCRUM Prozessmodells.

Agiles Manifest:

- Individuen und Interaktionen sind wichtiger als Prozesse und Werkzeuge.
- Funktionierende Programme sind wichtiger als ausführliche Dokumentation.
- Die stetige Abstimmung mit dem Kunden ist wichtiger als die ursprüngliche Leistungsvereinbarung in Verträgen.
- Der Mut und die Offenheit für Änderungen stehen über dem Befolgen eines festgelegten Plans.

Scrum:

- hyper-produktive Softwareentwicklung
- adaptive, schnelle, sich selbst organisierende Vorgehensweise mit wenigen Ruhepausen
- **Zentrale Werte:**
  - hohe Produktivität und Anpassungsfähigkeit
  - wenig Risiko und Ungewissheit
  - größerer Komfort für die Projektmitglieder
- **Entwicklungsphilosophie:**
  - **Product Backlog:** Auflistung aller funktionalen und nichtfunktionalen Anforderungen gereiht nach Wichtigkeit, kann von allen Beteiligten jederzeit eingesehen werden
  - **Product Owner:** verantwortlich für Product Backlog, erhält Inhalte von Stakeholdern, nimmt Priorisierung vor
  - eigentliche Entwicklung in kleinen Scrum Teams
  - **Sprint:** zentrales, wichtigstes Element von Scrum, definierte Dauer, muss mit Fertigstellung neuer Funktionalität enden, möglichst schnelles Erreichen des **Sprint Goal**
  - **Daily Scrum Meeting:** vom **Scrum Master** geleitet, immer zur selben Zeit am selben Ort
  - **Sprint Review Meeting:** am Ende des Sprints
- **Elemente:**
  - **Sprint**
  - **Daily Scrum**
  - **Product Backlog**
  - **Increment:** Softwareprodukt entsteht in Inkrementen, spätestens zweiter Sprint liefert lauffähiges Softwaresystem
  - **Burndown Chart:** tagesaktuelle Aufwandsschätzungen für aktuellen Sprint, vom Scrum Master eingetragen
- **Rollen:**
  - **Product Owner:** Erstellung, Priorisierung des Product & Release Backlogs, hält Team von Unterbrechungen/Störungen frei
  - **Scrum Team:** Mission: Erreichen des **Sprint Goal** (im **Sprint Backlog** geforderte Funktionalität)
  - **Scrum Master:** achtet auf Einhaltung der Grundsätze/Regeln von Scrum, schottet Team vor äußeren Einflüssen ab

Wozu dient das "3-Experten-Konzept"? Erklären Sie dieses.
Aufwandsschätzung:

- **Delphi-Methode:**
  - Moderator + mehrere Experten
  - jeder Experte gibt Schätzwert zu jedem Arbeitspaket ab
  - Mittelwert, falls Schätzwerte in bestimmter Bandbreite, sonst Argumente austauschen und neue Schätzung
  - 10 – 15 % Zuschlag für Projektmanagement
- **3-Experten-Konzept (Mini-Experten-Team):**
  - 3 optimistische, 3 realistische, 3 pessimistische Schätzungen
  - Abklärungen bis Ergebnisse in jeder Kategorie übereinstimmen
  - daraus gemeinsamen Schätzwert ableiten

Wozu dient CMM? 5 Reifegrade!
CMM (Capability Maturity Model)

- 80er-Jahre, US-Verteidigungsministerium
- Ziel: Verbesserung der Vergleichbarkeit und Qualität von Softwareprozessen im Rahmen der Vergabe an externe Lieferanten
- 2003 vereinheitlichter, modularer Nachfolger: **CMMI (Capability Maturity Model Integrated):**
  - liefert Analyse des Ist-Zustandes eines Unternehmens
  - kein international anerkanntes Zertifikat
- **Assessment:** beurteilt Fähigkeit einer Organisation unter bestimmten Rahmenbedingungen erstellen zu können
- **Prinzipien und Konzepte:**
  - **Quantitative Managementmethoden:** erhöhen Fähigkeit, Qualität zu kontrollieren und Produktivität zu steigern
  - **5 Ebenen-Capability Maturity Model:** ermöglicht Bewertung der Leistungen, bestimmt zur Erreichung der nächsten Stufe notwendige Aufwände
  - **Generische Prozessgebiete:** Was? – nicht Wie? → Anwendung auf breite Palette an Organisationsumsetzungen
- **Reifegradstufen:**
  - **1 – Initial**
  - **2 – Wiederholbar:**
    - Softwarequalitätssicherung
    - Projektplanung
    - Anforderungsmanagement
  - **3 – Definiert:**
    - Peer Reviews
    - Training und Schulung
  - **4 – Geführt:**
    - Softwarequalitätsmanagement
  - **5 – Optimiert:**
    - Technologietransfer
    - Fehlervermeidung

Erklären sie den Unterschied zw. Optimistic und Pessimistic Revision Control, Erklären sie diese Strategie anhand einer Skizze.

Versionsmanagementsysteme:

- Computersysteme, die Änderungen an Dateien/Verzeichnissen über die Zeit hinweg archivieren und in einem/mehreren Entwicklungszweigen die gemeinsame Bearbeitung durch mehrere Personen bereitstellen
- **Repository:** Datenbestand, aus dem Arbeitskopie generiert wird
- **Checkout:** Übertragung/Abholung der Daten aus dem Repository
- **Check-in/Commit:** Übertragung der Daten in das Repository
- Wichtige Funktionen:
    - **Verzweigung/Branching:** Anlegen einer Kopie von Objekten in Versionsmanagementsystemen
    - **Markieren/Tagging:** Tag bezeichnet konkreten Entwicklungsstand (zB zur Auslieferung bestimmt)
    - **Protokollierung/Historisierung:**
        - Welche Änderungen von welchem Benutzer zu welcher Zeit?
        - Wiederherstellung eines beliebigen Versionsstandes
- **Zentrales Versionsmanagement:**
    - zentraler Server, beliebig viele Clients
    - zB Subversion
- **Dezentrales Versionsmanagement:**
    - kein zentrales Repository, jeder Entwickler besitzt sein eigenes, Daten dazwischen ausgetauscht
    - zB Git
- **Lokales Versionsmanagement:** älteste Form, beschränkt auf Versionierung einzelner Dateien, nicht geeignet für Teamprojekte
- **Lock-Modify-Unlock-Strategie (Pessimistic Revision Control):** Datei während Bearbeitung gesperrt
- **Copy-Modify-Merge-Strategie (Optimistic Revision Control):** Konflikte, Zusammenfügen

Erklären Sie die wesentlichen Unterschiede zwischen einem Framework und einer Bibliothek!
Framework:

- Applikationsskelett, welches vom Entwickler angepasst/spezialisiert werden kann, um Anforderungen optimal umzusetzen
- wiederverwendbarer Entwurf, oft durch Menge abstrakter Klassen und Zusammenspiel ihrer Instanzen beschrieben
- effiziente Art des **Software Reuse**, reduziert Applikationsentwicklung auf **Spezialisierung** vorgegebener Strukturen
- Vorteile:
    - Reduktion von Entwicklungskosten, Entwicklungszeiten
    - Wiederverwendung von Quellcode, bewährten architektonischen Strukturen
    - Einhaltung von Standards
- Nachteile:
    - Vererben von Fehlern
    - Einarbeitungsaufwand
- **Bibliothek:**
    - Sammlung von Software und dazugehöriger Dokumentation
    - **wiederverwendbare Funktionalität**
    - **durch eigenen Code aufgerufen**
- **Framework:**
    - **wiederverwendbares Verhalten**
    - **eigener Code durch Framework aufgerufen**
    - bestehen aus abstrakten Klassen, bieten Funktionalität in teilweise implementierten Methoden an (diese können Funktionalität aus Bibliotheken beziehen)
- **Vorhandenes Framework:**
    - Anforderungen
    - Wartung, Weiterentwicklung gesichert? Verfügbarkeit erfahrener Entwickler?
    - kommerziell – frei?
- **Eigenentwicklung:**
    - kein geeignetes Framework/Anpassung mindestens gleich aufwändig
    - spezielle (nichtfunktionale) Anforderungen
- **Frameworktypen:**
    - **Persistence Frameworks:** zB Hibernate
    - **Web Application Frameworks:** zB Ruby on Rails
    - **Rich Client Frameworks:** zB Eclipse
    - **Grafik Frameworks:** zB Direct X
    - **Metaframeworks**
- **Eigenschaften:**
    - **Method Hooks:** abstrakte Methoden, können vom Entwickler überschrieben werden, um gewünschtes Verhalten zu erstellen
    - **Hot Spots:** Punkte, an denen Framework spezialisiert wird
    - **Whitebox Frameworks:** innere Struktur muss für Spezialisierung sichtbar sein, erfolgt meist durch Vererbung
    - **Blackbox Frameworks:** enthält stabile Hot Sports, Spezialisierung kompositionsbasiert (Komponenten an Framework übergeben)

Was versteht man unter "Stakeholder"? Nennen sie fünf typische Stakeholder in einem Softwareprojekt.

- **Stakeholder:** „Anteilhalter“, vertritt Interesse im Projekt, stellt Anforderungsquelle dar
- jede Stakeholdergruppe hat eigene Sicht auf das Projekt, Konflikte → **Anforderungspriorisierung**
- Typische Stakeholder:
  - Kunde/Auftraggeber/Sponsor
  - Geschäftsführung/Abteilungsleiter (Auftraggeber + Auftragnehmer)
  - Legacy Owner (Besitzer des Altsystems/der Altdaten)
  - Betreiber/Entwickler von Schnittstellensystemen
  - Benutzer/Benutzervertreter
  - Projektleiter
  - Systemarchitekt/IT-Stratege
  - Softwareentwickler
  - Qualitätssicherung und Test
  - Betrieb, Wartung

Wozu dienen Architekturmuster? Beschreiben Sie das Model View Controller Architekturmuster, und nennen sie zwei weitere Architekturmuster!

Patterns:

- allgemeine Lösungsstrategien für wiederkehrende Probleme der Softwareentwicklung
- abstrakt, weitgehend technologieunabhängig, wesentlicher Input in Entwurfsentscheidungen
- **Pattern Language:** Sammlung von Entwurfsmustern
- **Architekturpatterns:**
  - Makroarchitektur (architektonische Ebene)
  - spezifizieren systemweite, strukturelle Eigenschaften einer Anwendung
  - großer Einfluss auf Architektur der Subsysteme, fundamentale Designentscheidung
  - **MVC (Model-View-Controller):**
    - Interaktive Anwendung, GUI oft geändert → Aufteilung in Verarbeitung, Input, Output
    - **Model:** kapselt Daten und Funktionen
    - **View:** stellt Informationen dar
    - **Controller:** empfängt Events
  - **Layered Architecture:**
    - funktionell getrennte Ebenen: Datenbankzugriff, Geschäftslogik, UI
    - Kommunikation nur mit darunter liegender Ebene
    - keine Abhängigkeiten auf darüber liegenden Ebene
    - Kommunikation mittels Interfaces, Exceptions nach Ebene gekapselt
- **Entwurfsmuster (Design Patterns):**
  - Mikroarchitektur (Klassenebene)
  - gute, bewährte Lösung für bestimmtes Problem
  - **Creational Patterns:** Abstract Factory, Singleton, Prototype
  - **Structural Patterns:** Proxy, Decorator
  - **Behavioral Patterns:** Iterator, Observer

Was versteht man unter "SOA"? Welche Eigenschaften zeichnen SOA aus? Welche Paradigmen werden dabei verfolgt?

SOA (Service Oriented Architecture):

- Serviceorientiertes Design umspannt mehrere Abstraktionsebenen, unterschiedliche Notationen je Layer:
  - **Service Layer:** BPMN
  - **Component Layer:** Komponentendiagramme
  - **Class Layer:** UML
- **State-of-the-Art-Architektur**
- **Paradigmen:**
  - **Loose Coupling:** der Services → hoher Reuse, Flexibilität
  - **Abstraktion:** Reduktion von Neuentwicklungen
  - **Standardisierung:** Zusammenarbeit von Services
  - **Composable:** Zusammenfassung standardisierter Services
- **Business Service:** Schritt in Geschäftsprozess extrahiert und gekapselt, Anwendungen durch mehrere Services gelöst, Zusammensetzung von Services in bestimmten Abläufen zur Umsetzung von Geschäftsprozessen (**Orchestrierung**)
- **Business driven:** an Geschäftsprozesse angelehnt
- **BPM (Business Process Modelling):** eng mit SOA verknüpft, Anwendungen maßgeblich von Geschäftsprozess-Modellierung beeinflusst
- Services auf mehreren Abstraktionsebenen: **Business Service**, **Component Service**, **technisches Service**
- beschreibt Architektur, keine Technologie (zB Webservices oder **CORBA**)

Was sind Integrationstests? Erklären sie unterschiedliche Integrationsformen!
Softwaretests:

- **Ziele:**
  - Testfälle identifizieren, mit denen die höchste Wahrscheinlichkeit gegeben ist, festzustellen, ob das Softwaresystem korrekt funktioniert
  - guter Test: möglichst hohe funktionale oder nichtfunktionale Abdeckung
  - Fehler sollen identifiziert werden
  - wenn Fehler gefunden → Test erfolgreich
- **Grundsätze:**
  - Testen zeigt Anwesenheit von Fehlern
  - Vollständiges Testen nicht möglich
  - frühzeitig testen
  - Häufung von Fehlern
  - Wiederholungen haben keine Wirksamkeit
  - abhängig vom Umfeld
  - kein Fehler bedeutet nicht automatische brauchbares System
- **Test- und Integrationsstufen:**
  - Zerlegung der gestellten Aufgabe in beherrschbare Teile bei der Herstellung von Softwaresystemen

- Zerlegung in Teststufen: ermöglicht frühe Prüfung unterschiedlicher Teile des zu entwickelnden Systems
- **Komponententest:**
  - Prüfung separat testbarer Komponenten (Module, Programme, Objekte, Klassen) auf vorhanden Fehler
  - Isolation einzelner Komponenten vom Rest des Systems mit Hilfe von **Stubs** (Platzhaltern) bzw. Simulatoren und Testtreiber
- **Integrationstest:**
  - Schnittstellen zwischen Komponenten
  - mehrere Integrationsstufen, betreffen Testobjekte unterschiedlichster Größe
  - mit Größe des Integrationsumfangs wächst Schwierigkeit der Isolation von Fehlerwirkungen in Komponenten oder Systemen
  - **horizontale – vertikale Integration**
  - **Bottom-Up-Integrationstests – Top-Down-Integrationstests**

  - Fokus auf spezifiziertem Verhalten eines Gesamtsystems oder Produktes
  - sollten funktionale und nichtfunktionale Anforderungen an das System abdecken
  - basieren auf:
    - Anforderungsspezifikationen
    - Anwendungsfällen, sonstigen Beschreibungen eines Systems
    - Geschäftsprozessen
    - Risikoanalysen
    - Erfahrungen im Produktionsumfeld
    - Systemressourcen
  - **Akzeptanztest:**
    - Nachweis der Erbringung der von Auftraggeber und Auftragnehmer vereinbaren Leistungen
    - Arten:
      - Anwender-Abnahmetest
      - Betrieblicher Abnahmetest
      - Vertraglicher Abnahmetest
      - Alphatest, Betatest (Feldtest)
- **Typische Quellen für Testfälle:**
  - definierte Anforderungen
  - versteckte Anforderungen
  - häufige Fehler/Erfahrung
  - bekannte Schwachstellen im Testobjekt
  - Struktur des Testobjekts
- **Testabdeckung:**
  - Ziel der **Verifikation** von Softwaresystemen: möglichst hohe **Coverage**
  - **Testfallselektion:** da vollständige taxative Aufzählung aller mögliche Systemzustände unmöglich
  - **Ziel:** Identifikation von Testfällen, die mit höchster Wahrscheinlichkeit Fehler finden, größtmögliche Abdeckung erreichen
  - **Strukturelle Abdeckung (White-Box-Tests)**

- **Anweisungsabdeckung**
- **Bedingungsabdeckung**
- **Pfadabdeckung**

- **Funktionale Abdeckung (Black-Box-Tests)**
  - **Äquivalenzklassenanalyse:**
    - Einteilung möglicher Ein- und Ausgabewerte in Klassen mit gleichem Systemverhalten
    - Werte von Klassen gewählt, bei denen angenommen wird, dass Fehler auftreten, die auch bei anderen Werten der Klasse auftreten können
  - **Grenzwertanalyse:**
    - Spezialfall der Äquivalenzklassenanalyse
    - Fehler treten besonders oft an den Grenzen der Äquivalenzklassen auf
  - **Klassifikationsbaummethode:**
    - systematische Unterstätzung von Black-Box-Tests
    - Testfälle anhand möglicher Eingabebereiche oder Systemzustände in **Klassifikationen** eingeteilt
    - **Klassifikationen** in disjunkte Teilmengen zerlegt
    - Testfälle durch Kombination verschiedener Klassen erstellt, von jeder Klassifikation nur eine Klasse

Was versteht man unter einer "Baseline" in der Analyse? Was passiert üblicherweise bei Änderungen (Changes) nachdem eine Baseline gesetzt wurde?

- niemals echter **Freeze** der Anforderungen, aber stabile Zustände in der Analyse unabdingbar
- formale Spezifikation, reviewed, vereinbart zwischen Auftraggeber und Auftragnehmer, Basis für weitere Entwicklung, durch formale Mechanismen änderbar
- **Nach dem Setzen einer Baseline bei Änderungen:**
  - **Change Impact Analysis**
  - **Re-Costing**

Welche Typen der Wartung werden unterschieden? Erklären sie die unterschiedlichen Typen der Wartung und erläutern sie warum diese Klassifizierung erforderlich ist.

Wartung:

- Fehlerbehebung
- Verbesserung des Laufzeitverhaltens
- Update der technischen Realisierung, Anpassung an geänderte Umgebung
- keine Weiterentwicklung, keine Änderung des funktionalen Zustandes
- Weiterentwicklung parallel möglich
- **Mögliche Auslöser:**
  - gefundene Fehler, Sicherheitslücken
  - Update einer Komponente
  - Performance-Probleme
- **Wartungstypen:**
  - **Korrektive Wartung:** Ausbesserung von Fehlern, Abweichungen von der Spezifikation
  - **Präventive Wartung:** verhindert vermutliche zukünftige Fehler
  - **Adaptive Wartung:** Anpassung an geänderte Umgebung (verwendet Software, Schnittstellen, Hardware)
  - **Perfektionierende Wartung:** Verbesserung der Applikation (Benutzerführung, Performance, Wartbarkeit), keine funktionale Änderung
- **Wartungsmaßnahmen:**
  - **Software Reengineering:**
    - Neuentwicklung bei gleichbleibender Funktionalität
    - Motivation: Qualitätssteigerung
  - **Reverse Engineering:**
    - Gewinnung von Code und Modellen aus vorhandenen Artefakten
    - wenn Source-Code nicht mehr verfügbar
    - Re-Dokumentation
    - oft Vorstufe zum Reengineering
  - **Refaktorisierung:**
    - während Initialphase und Wartung
    - Ziel: Erhaltung der Wartbarkeit
  - **Anti Patterns:**
    - Gegenstück zu Software Patterns
    - Sammlung häufiger schlechter Lösungsansätze (mangelnde Erfahrung, Qualifikation)
    - dienen schneller Erkennung, bieten Verbesserungsstrategien

Praxis

Klassendiagramm

Welches Vorgehensmodell?

- **Agile Methoden:**
  - rasche **time-to-market**
  - höhere Qualifikation erforderlich
  - **dynamische Anforderungen**
  - **Teamgröße:** kleine – mittelgroße Teams
- **Traditionelle Vorgehensmodelle:**
  - **Kritikalität:** Menschenleben → höhere Maßstäbe an Prüfung, Nachvollziehbarkeit
- **Weitere Entscheidungsfaktoren:**
  - **Entwicklungskultur**

Anwendungsfallbeschreibung Cockburn.

- Name + Identifier
- Beschreibung
- **Beteiligte Akteure:**
  - **Primäre Akteure:** eigentliche Benutzer
  - **Sekundäre Akteure:** überwachen, warten System, unterstützen primäre Akteure bei Zielerreichung
- **Status:** Fortschritt der Arbeit am Anwendungsfall
  - in Arbeit
  - bereit zum Review
  - im Review
  - abgelehnt
  - abgenommen
- **Verwendet Anwendungsfälle:** Querverweise mit Name, Identifikation
- **Auslöser:** Gründe für das Ausführen
- **Vorbedingungen**
- **Invarianten:** Bedingungen, die durch den Anwendungsfall nicht verändert werden dürfen
- **Nachbedingung/Ereignis:** zu erwartender Zustand nach erfolgreichem Durchlauf des Anwendungsfalls
- **Standardablauf:**
  - typisches leicht verständliches Szenario oder häufigster Fall
  - am Ende steht Zielerreichung des Primärakteurs
  - Schritte numeriert, Darstellung als Ablaufpläne, Aktivitätsdiagramm oder Sequenzdiagramm
- **Alternative Ablaufschritte:** Szenarien, die neben Standardablauf noch auftreten können (unabhängig von Zielerreichung), häufig als bedingte Verzweigungen dargestellt
- **Hinweise:** besseres Verständnis des Anwendungsfalls (zB evtl. Nebeneffekte)
- **Änderungsgeschichte:** Versionierung mit Autor, Datum, …

User Stories/Kunde vor Ort:

- Vorteile:
  - kurz, prägnant → gut wartbar
  - fördern Diskussion, Abstimmung zwischen Kunde und Entwickler
  - reduzieren Risiko von Fehlentwicklungen aus Benutzersicht
- Nachteile:
  - ohne zugehörige präzisierte Akzeptanz-Testfälle sehr stark interpretierbar
  - setzen starke Involvierung des Benutzers/Kunden während des gesamten Prozesses voraus
  - personenzentriert
  - können kaum als Vertragsgrundlage dienen
- **Storycard:**
  - Aufgabenstellung
  - Aufwandsschätzung
  - Referenzen
  - Nr., Datum

Risikoanalyse.

- Nr.
- Bezeichnung
- Beschreibung
- Eintrittswahrscheinlichkeit
- Schadensschwere
- **Risiko:** Eintrittswahrscheinlichkeit * Schadensschwere
- **Risikomanagement:**
  - Risikonr.
  - Maßnahme
  - Verantwortlicher

Nicht funktionale Anforderungen.
Klassifikation von Anforderungen:

- **Funktionale Anforderungen:** Funktionen/Services, die System zur Verfügung stellen soll, Reaktion auf bestimme Eingabe/Situationen
- **Nichtfunktionale Anforderungen:** nicht Verhalten, sondern Qualitätseigenschaften, Entwicklungsprozess, einzuhaltende Standards/Normen/gesetzliche Rahmenbedingungen
- **Domänenanforderungen:** funktional oder nichtfunktional, von der Domäne definiert, besondere Eigenschaften/Einschränkungen dieses Gebiets, oft inhärent, nicht explizit pro Projekt definiert

Pflichtenheft:

- **Funktionale Anforderungen:** Use Cases oder User Storys
- **Nicht-funktionale Anforderungen:** Wartbarkeit, Usability, Performance, Skalierbarkeit
  - in Prosa
  - Nr., Anforderung, Kategorie (Zuverlässigkeit, Benutzbarkeit, Effizienz, Änderbarkeit, Übertragbarkeit)