Was sind Vorgehensmodelle und wozu sind sie gut?

- immer umfangreichere, komplexere Softwareprodukte -> Vorgehensmodelle gewinnen an Bedeutung, wesentlich für Geschäftserfolg
- **Ziel:** Softwareentwicklung auf etablierte ingenieurmäßige Basis stellen, Verbesserungen in Produktivität, Qualität, Vorhersagbarkeit
- kein optimaler Entwicklungsprozess -> je nach Projekt geeignetes Verfahren wählen

Unterschied zwischen traditionellen und agilen Methoden?

- Agile Methoden besser für Projekte mit raschen 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 + 3 Beispiele)

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

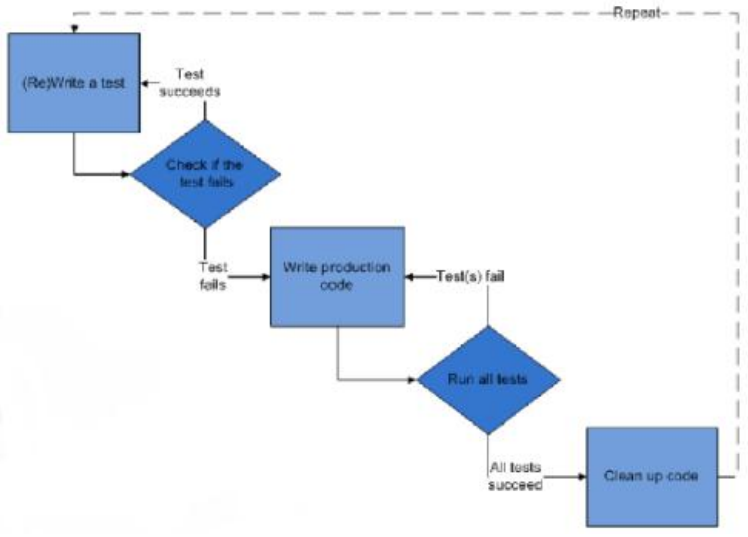
Softwaremetriken in der Qualitätssicherung, warum verwenden? (3 Beispiele objektorientierter Metriken)

- Messung der Qualität von Software
- **Prozessmetriken**: quantitative Daten des Entwicklungsprozesses
- **Produktmetriken:** messen Software bzw. Teile davon, ohne Entwicklungsprozess oder Zustand zu berücksichtigen
- **Umfangsmetrik:** Lines of Code
- **Objektorientierte Metriken:** berücksichtigen auch Beziehungen von Objekten untereinander und deren Struktur
  - **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 -> Ressource-Files
  - Nachrichten nicht stückeln
  - Identische Nachrichten in unterschiedlichem Kontext mehrfach speichern
  - von Beginn an vorsehen

Erklären Sie 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

Was ist „SOA“? Eigenschaften und Paradigmen?

- Service orientierte Architektur
- an Geschäftsprozesse angelehnt -> business driven
- Anwendungen durch mehrere Services gelöst
- Services durch mehrere Anwendungen aufrufbar -> Wiederverwendung, Modularisierung
- Beschreibt Architektur, keine Technologie
- Services auf mehreren Abstraktionsebenen
- Zusammenstellung von Services in Abläufen zur Umsetzung von Geschäftsprozessen (Orchestrierung)
- Paradigmen:
  - **Loose Coupling:** niedrige Kopplung der Services -> Reuse, Flexibilität
  - **Abstraktion:** Reduktion von Neuentwicklungen
  - **Standardisierung:** Zusammenarbeit von Services
  - **Composable:** Services zu Komponenten zusammenfassbar

Was ist BPM?

- Geschäftsprozessmodellierung (Business Process Modeling)
- Geschäftsprozesse oder Ausschnitte abstrahiert, dargestellt (graphisch) und somit modelliert
- Schwerpunkt liegt auf Darstellen des Ablaufs
- Teil des Geschäftsprozessmanagements

Zusammenhang SOA und BPM?

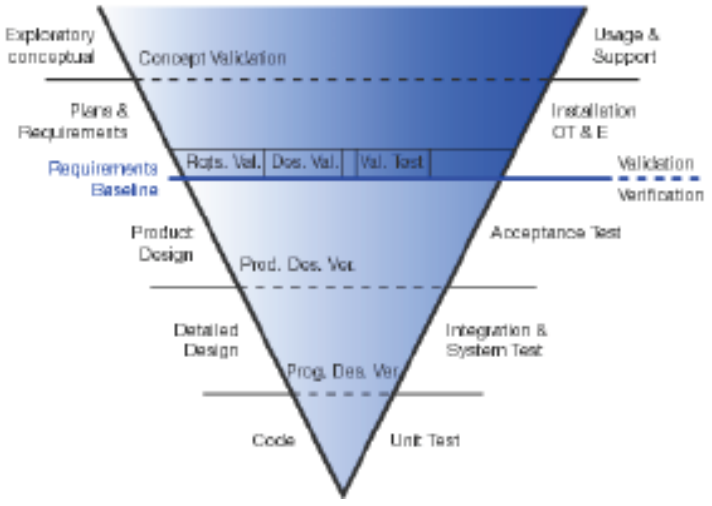
- SOA und BPM eng miteinander verknüpft, Anwendungen maßgeblich von Geschäftsprozessmodellierung beeinflusst
- **Zusammenhang:**
  - Geschäftsprozess A und Geschäftsprozess B greifen auf gleiches Business Service zu
  - Business Service wird auf technischer Ebene durch 2 verschiedene Services gelöst

Äquivalenzklassenanalyse, Grenzwertanalyse erklären und Unterschiede?

- **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

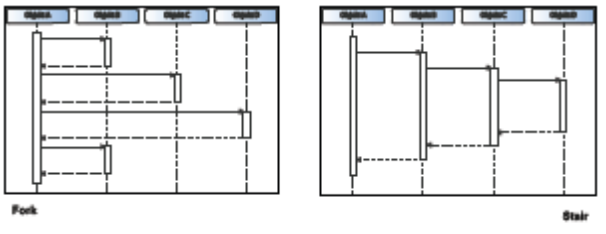
V-Modell Beschreiben + Skizze

- 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? + Skizze

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



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 (Grafik -> links oben)
  - Risikoanalyse (Grafik -> rechts oben)
  - Durchführen der Arbeitsschritte (Grafik -> rechts unten)
  - Planen der nächsten Phase (Grafik -> links unten)
- Anzahl der notwendigen Durchläufe ergibt sich im Projektverlauf, ist von auftretenden Risiken abhängig

Was ist ein Framework? Whitebox, Blackbox Spezialisierung?

- 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
- **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 ist sichtbar, Spezialisierung erfolgt durch Vererbung
- **Blackbox Frameworks:** keine Details, über Struktur hinter Schnittstelle, enthält stabile Hot Spots, Spezialisierung kompositionsbasiert (Komponenten an Framework übergeben)
- **Vorteile:**
  - Reduktion von Entwicklungskosten, kürzere Entwicklungszeiten
  - Wiederverwendung von Quellcode, bewährten architektonischen Strukturen
  - Einhaltung von Standards
- **Nachteile:**
  - Vererben von Fehlern
  - Einarbeitungsaufwand
- **Typen:**
  - **Persistence Frameworks:** Hibernate
  - **Web Application Frameworks:** Ruby on Rails
  - **Rich Client Frameworks:** Eclipse RCP
  - **Grafik Frameworks:** Direct X

Wesentliche Unterschiede zwischen einem Framework und einer Bibliothek?

- **Framework:**
  - wiederverwendbares Verhalten
  - eigener Code wird durch das Framework aufgerufen
- **Bibliothek:**
  - wiederverwendbare Funktionalität
  - Methoden werden durch eigenen Code aufgerufen
- Frameworks bestehen aus abstrakten Klassen, bieten Funktionalität in teilweise implementierten Methoden an, die Funktionalität aus Bibliotheken beziehen können

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:** sehr früh im Entwicklungsprozess möglich, 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

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

- 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 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
- **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.

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

Wozu dient SPICE? 6 Reifegrade!

- 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

Projektorganisation(mögliche Fragen -> Reine-/Einfluss/-Matrix-Projektorganisation erklären + Unterschiede):

- **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:

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

Unterschied zw. zentraler, dezentraler, lokaler Versionskontrolle + je 2 Bsp.

- **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
  - z.B.: Subversion (SVN)
- **Dezentrales Versionsmanagement:**
  - kein zentrales Repository, jeder Entwickler besitzt sein eigenes
  - Änderungen zwischen Repositorys ausgetauscht
  - z.B.: Git
- **Lokales Versionsmanagement:**
  - älteste Form, beschränkt auf Versionierung einzelner Dateien
  - nicht geeignet für Teamprojekte
- **Strategien:**
  - Lock-Modify-Unlock-Strategie (Pessimistic Revision Control) -> Datei während Bearbeitung gesperrt
  - Copy-Modify-Merge-Strategie (Optimistic Revision Control) -> Konflikte und Zusammenfügen

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 sind Komponenten-/System-/Akzeptanz-/Integrationstests? Verschiedene Integrationsformen? Erläutern sie in diesem Zusammenhang die Begriffe Stub and Driver!

- **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
- **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:
    - Anforderungsspezifikationen, 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)

Erklären sie die Grundcharakteristik, die wesentlichen Elemente und Rollen des 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?

- 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 gibt es bei Änderungen:**
  - Change Impact Analysis
  - Re-Costing

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

- 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 Initialentwicklung und Wartung -> Erhaltung der Wartbarkeit
  - **Anti Patterns:**
    - Gegenstück zu Software Patterns
    - Sammlung häufiger schlechter Lösungsansätze
    - Grund: mangelnde Erfahrung, Qualifikation
    - bieten Verbesserungsstrategien

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

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

- 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

Vorgehen bei „SQL Injection“ Angriffen? – Wie kann man sich davor schützen?

- Manipulation des SQL-Statements durch Angreifer
- Einfachen Hochkomma: „te‘st“ -> Fehlermeldung aus Datenbank, kann für weiteren Angriff analysiert werden
- Auslesen / Änderungen in der Datenbank möglich …
- Gegenmaßnamen:
  - Jede Eingabe muss validiert werden (nicht ausreichend nur Formular zu validieren)
  - Filterung spezieller Zeichen -> einfaches Hochkomma
  - Blacklist Filterung -> Prüfung der Eingabe auf verbotene Zeichen
  - Whitelist Filterung -> Prüfung auf erlaubte Zeichen (besser)
  - Precompiled Statements verwenden -> werden auf Server mit Platzhalter kompiliert, im Nachhinein nicht mehr änderbar
  - Minimierung des Schadens durch Begrenzung der Rechte

Wasserfallmodell nach Royce 1970

- Am weitesten verbreitet, viele Ausprägungen und Varianten
- In Grundform vollständig sequentiell
- Erweiterungsstufe lässt Rückschritt auf vorherige Phase zu
- Arbeitsschritt erst dann abgeschlossen, wenn alle vorgesehenen Produkte fertig
- Berücksichtigung wesentlicher Dokumente und Review-Schritte im Modell
- Wird in modernen SW-Entwicklung als Risiko betrachtet -> führt zu Entwicklung iterativer Modelle

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 nummeriert, 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 (z.B.: evtl. Nebeneffekte)
- **Änderungsgeschichte:** Versionierung mit Autor, Datum, …

User Stories/Kunde vor Ort:

- **Vorteile:**
  - kurz, prägnant -> gut wart bar
  - 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

Nicht funktionale Anforderungen. / Klassifikation von Anforderungen:

- **Funktionale Anforderungen:** Funktionen/Services, die System zur Verfügung stellen soll, Reaktion auf bestimme Eingaben/Situationen
- **Nichtfunktionale Anforderungen:** Anforderung an die Funktion/das Service, beschreiben nicht das Verhalten, sondern Qualitätseigenschaften, Entwicklungsprozess, einzuhaltende Standards, Normen, gesetzliche Rahmenbedingungen
- **Domänenanforderungen:** funktional oder nichtfunktional Anforderungen, die von der Domäne definiert werden, 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)