Fragenausarbeitung SEPM Grechenig 2014

3 Vorteile/Nachteile Teamwork:
+ Gruppendynamik, Teamgeist
+ erhöhte Kreativität durch Wechselwirkung im Team
+ erhöhte Produktivität durch gegenseitiges Anspornen
- Gruppenkommunikation ist zeitaufwändig
- Gruppendiskussionen laufen oft in unproduktive Richtung
- Bei Konflikten kann es zu Motivationsverlust kommen

Was ist ein Stakeholder? 5 Typen nennen
Ein Stakeholder („Anteilhaber“) vertritt gewisse Interessen an einem Projekt. Er hat eine eigene Sicht auf das Projekt (und ist eine Anforderungsquelle).
Typen: Benutzer, Kunde/Autraggeber, Projektleiter, Softwareentwickler, Qualitätssicherung

Stabile und volatile Anforderungen? Auswirkung auf Softwaredesign?
stabil: Anforderung die sich voraussichtlich nicht oder nur kaum ändern werden
volatile: Anforderungen die sich voraussichtlich ändern werden
Auswirkungen:
stabile Anforderungen können mit einfacheren Designs umgesetzt werden. Sie können früher und detaillierter modelliert werden. können früher implementiert werden
volatile erfordern adaptives Softwaredesign

Baseline
Tatsächlich existieren NIE fixe Anforderungen. Trotzdem sind stabile Zustände im Entwicklungsprozess unabdingbar.
Eine Baseline beschreibt den "roten Faden" in den Anforderungen, also Anforderungen, die sehr stabil sind und mit hoher Wahrscheinlichkeit nicht geändert werden. Um die Baseline zu ändern ist ein Change Request notwendig.

Nicht funktionale Anforderungen und **Funktionale Anforderungen**
funktional: spezifizieren WAS das System tun soll
z.b.:
1) Es soll möglich sein sich als Kunde einzuloggen.
2) Es soll möglich sein seine Kundendaten selbst auszudrucken
3) Es soll möglich sein ein Produkt über den Online-Shop zu kaufen

nichtfunktional: spezifizieren WIE das System etwas tun soll (meist hinsichtlich den Qualitätsanforderungen, Leistung, Wartbarkeit, Benutzerfreundlichkeit, Sicherheit)
z.b:
1) Die Speicherung von Passwörtern soll nur verschlüsselt passieren!
2) Anwendung soll zu diesen und jenen Zeit verfügbar sein
3) Eine Produkt-Abfrage soll nicht länger als 2 Sekunden in Anspruch nehmen.

Qualitätskriterien und Probleme beim Schreiben von Anforderungen. Methoden um Vollständigkeit und Qualität der Anforderungen zu testen, 6 Qualitätskritieren für Anforderung nennen

Qualitätskritieren:
- gute Anforderungen sind vollständig (alle Anforderung müssen abgedeckt sein)
- konsistent (Anforderungen enthalten keine Widersprüche oder Konflikte mit anderen Anforderungen)
- Verständlichkeit (ALLE Stakeholder können Anforderung lesen und verstehen)
- atomar (nur eine Anforderungsbeschreibung pro ID)
- Identifizierbar (über ID)
- Priorität

Probleme:

- Verständlichkeit (Softwareentwickler verstehen Sprache/Ausdrucksweisen nicht) ,
- Selbstverständlichkeit (Experten sind so mit Projekt vertraut, dass sie es anderen nicht gut genug darlegen können),
- zu wenig präzise,
- Vermischung von funkt. und nicht - funkt. Anforderungen,
- Mehrdeutigkeit (Beschreibung der Anforderung ist oft mehrdeutig interpretierbar)

UML, Was ist das? Je 5 Verhaltens+Strukturmodelle
UML = Unified Modelling Language
Ist eine Modellierungssprache um die Anforderungen und Komponenten von Softwareprojekten visuell zu modellieren.
untergliedert sich in
Verhaltensmodelle (Usecase, Aktivitätsdiagramm, Zustandsdiagramm, Sequenzdiagramm, Kommunikationsdiagramm) —> KAUZs
Strukturmodelle (Klassendiagramm, Objektdiagramm, Paketdiagramm, Komponentendiagramm, Strukturdiagramm) —> KOKSp

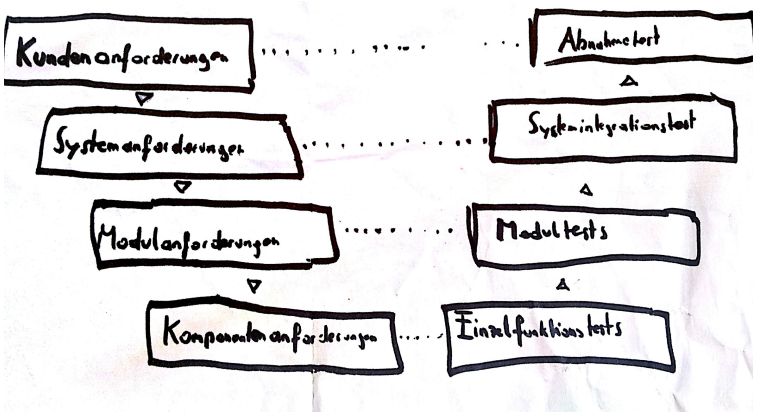
Anforderungsnachvollziehbarkeit (**Traceability**)
Traceability ist die Rückverfolgung einer Information über den gesamten Entwicklungsprozess
Fähigkeit eine Anforderung über ihren gesamten Lifecycle nachverfolgen zu können.

Wie umsetzen:
Woher kommt eine Anforderung und wo wurde sie umgesetzt? (Nachverfolgbarkeit der Quelle)
Welche Artefakte sind von einer Änderung der Anforderung betroffen? (Nachverfolgbarkeit zu abhängigen Artefakten)
Welche Anforderungen sind von einer Änderung der Umsetzung betroffen? (Nachverfolgbarkeit zu anderen Anforderungen)

traditionelle und agile **Vorgehensmodelle**. **V-Modell** erklären, wofür steht das XT im V-XT Modell und Ziele des XT Modells

Vorgehensmodelle: Ein Vorgehensmodell in der Softwareentwicklung dient dazu, die Entwicklung übersichtlicher zu gestalten und die Komplexität beherrschbar zu machen.
Ein Vorgehensmodell ist daher ein Plan, der den Entwicklungsprozess in Phasen aufteilt. Software wird also Schritt für Schritt nach einem Plan hergestellt.

V-Modell:



Am Anfang steht die Analyse der Anforderungen, am Ende die Abnahme des funktionierenden Systems.
Anforderungen steuern gesamten Entwicklungsprozess.

Technische Realisierung wird über mehrere Ebenen schrittweise immer detaillierter erarbeitet.
Das Gesamtsystem wird in immer überschaubarere Komponenten zerlegt.
Ganz unten im V-Modell stehen die Implementation einzelner Komponenten -> Komplexität ist auf ein beherrschbares Niveau reduziert. Vorgaben sind nun klar vorgeben und werden umgesetzt.
Für jede Anforderung muss auf jeder Ebene durch Test gezeigt werden, dass sie erfüllt wurde.
Von unten nach oben wird das geplante System einzeln zusammengebaut und jede Ebene nach dem zugehörigen Testplan geprüft. Der Abnahmetest ist finaler Prüfstein für die Kundenanforderungen.
Also links: Aufteilung der Komplexen Anforderungen in Einzelteile
unten: Implementierung der Komponenten
rechts: Zusammenbau der einzelnen Teile zum komplexen System

V-XT: XT steht für Extreme Tailoring
- das V-Modell ist an die jeweiligen Bedürfnisse anpassbar (=Tailoring).
- Einbindung des Auftraggebers: Vorgehensbausteine für Auftraggeber
- Es gibt nur mehr Vorgehensbausteine, aus denen das konkrete Vorgehensmodell zusammengestellt werden kann (je nach Projekt)
- geht mehr in Richtung agiler und inkrementeller SE

Traditionelle vs. agile Vorgehensmodelle:
Weder agile noch traditionelle Vorgehensmodelle sind für ALLE Projekte sinnvoll. Kommt immer auf Projekt drauf an.
agile SE: Ziel ist Softwareentwicklungsprozess flexibler und schlanker zu machen. Mehr auf zu erreichende Ziele konzentrieren, mehr auf technische und soziale Probleme bei der SE eingehen.
traditionell: oft aufwändiger zu betreiben und viel mehr bürokratische Pflichten

XP? **Extreme Programming**

XP stellt das Lösen der Programmieraufgabe in den Vordergrund der Softwareentwicklung.
Formalisierte Vorgehensweisen wie in traditionellen Vorgehensmodellen haben geringere Bedeutung.
Für Anforderungen die sich häufig ändern. Hauptziel ist das Termingerechte Abliefern von Software.
Teamarbeit und ständige Kommunikation zw. allen Beteiligten (Kunde, User, Entwickler) im Vordergrund.
Einfachheit —> Das einfachste Design wählen, das die Aufgabe löst.
Kunde nimmt aktiv an Herstellung teil.

Was ist RUP?

iteratives und inkrementelles Prozess-Framework von IBM

Rup = Rational Unified Process; beruht auf:
- Software **iterativ** entwickeln
- Anforderungen managen
- Komponentenbasierte Architektur verwenden
- Software visuell modellieren
- Softwarequalität kontinuierlich prüfen
- Änderungen kontrolliert in die Software einbringen

Rup basiert auf der Idee von Tailoring (=anpassen). Der RUP muss für jedes Projekt spezifisch angepasst werden. Dafür gibt es Tools, die das erzeugen eines neuen angepassten Modells anhand bestehender Prozessmodell erlauben.

4 Phasen: Inception (Beginn), Elaboration (Konzept & Design), Construction (Erstellung), Transition(Auslieferung)

Was ist **SCRUM**? Rollen/Elemente

Scrum ist eine Methode zur Softwareentwicklung
Wichtigste Werte:
- Hohe Produktivität
- Hohe Anpassungsfähigkeit
- wenig Risiko und Ungewissheit
- Komfort für Projektmitglieder

Rollen und Elemente:
Product Owner: erstellt gemeinsam mit Kunden Product Backlog (Liste ALLER Anforderungen), Team soll frei von Störungen arbeiten können
Scrum Team: 5-10 Entwickler + Scrum Master (sorgt für gutes Arbeitsklima)
Scrum Sprint: Erstellung von Sprint Backlog (Teilmenge des Product Backlog)
Dauer ca. 2-4 Wochen, jeder Sprint endet mit fertigem ausführbarem Softwareteil
Daily Scrum: Jeder sagt was er bis dato geschafft hat bzw. heute schaffen will
Sprint Review Meeting: Erfahrungen besprechen, Verbesserungsvorschläge diskutieren
Resultat jedes Sprints: fertige Funktionalität (Increment)
Burndown Chart: Diagramm indem aktuelle Fortschritte und Aufwandsabschätzungen vom SCRUM Master eingetragen werden

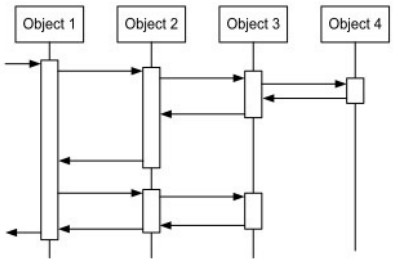
SOA und BPM plus Zusammenhang, Was ist SOAP/WSDL/UDDI

SOA:
Service Orientierte Architektur ist Architekturmuster um Dienste von verteilte IT-Systemen zu strukturieren.
Orientierung an Geschäftsprozesse (Business Process Models = BPM) und deren Abstraktionsebenen: Bsp: „Vergib einen Kredit“, dahinter verbergen sich mehrere Geschäftsprozesse wie etwa „Eröffnen der Geschäftsbeziehung“, „Eröffnen eines oder mehrerer Konten“, „Kreditvertrag...“ und so weiter)
Anwendung werden durch mehrere Services (Orchestrierung) gelöst. Services sollen auch von anderen Anwendungen heraus ausgeführt werden können
Charakteristika: lose Kopplung der Services, Standardisierung, Composability (Zusammenfassung zu Komponenenten)

SOAP = Netzwerkprotokoll, Daten zwischen Systemen austauschen und RPC’s durchführen
WSDL = plattform-, programmiersprachen- und protokollunabhängige Beschreibungssprache für Webservices zum Austausch von Nachrichten auf Basis von XML
UDDI = standardisierter Verzeichnisdienst (enthält Unternehmen, ihre Daten und ihre Services)

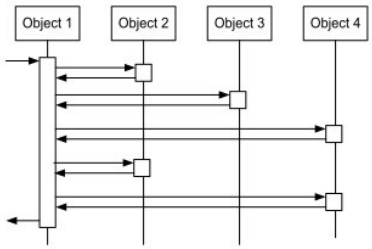
Stair und Fork SW-Architektur erklären + Skizze

Stair bzw. Fork sind zwei SW-Architekturen mit denen man den Kontrollfluss eines Use Case modellieren kann.



Sequence Chart: Stair

- **Stair**
  - Verteilte Kontrolle
  - Schrittweise Ausführung von Funktionen, dadurch wechselt die Kontrolle
  - Verbesserung der Wiederverwendbarkeit der Methoden z.B. durch Vererbung
  - Die spätere Wartung wird erschwert



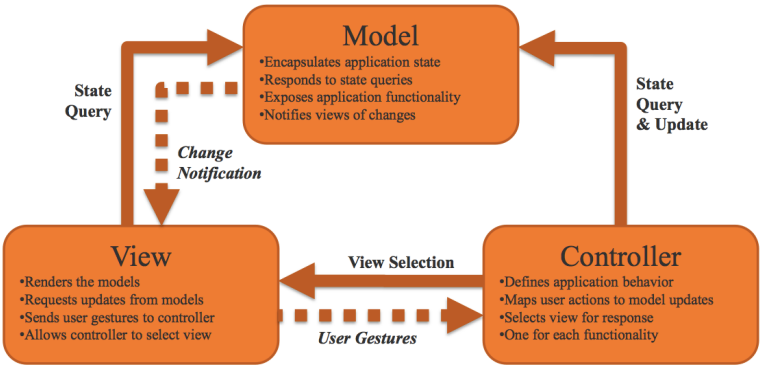
Sequence Chart: Fork

- **Fork**
  - Ein zentrales Objekt kontrolliert den gesamten UseCase
  - Wiederverwendung von Datenobjekten (ohne Business Logik) wird erleichtert
  - Wartbarkeit wird verbessert, da nur ein Objekt geändert werden muss.

Architekturmuster und Designpattern plus 3 Architekturmusterbeispiele, eines davon und erläutern, Für was sind Softwarepatterns gut? MVC, Observer

Patterns sind „Schablonen“ für wiederholt auftretende Strukturen in kommunizierenden Softwarekomponenten. Dienen dazu, ein allgemeines Problem in bestimmten Kontext zu lösen.
2 Arten:
Architekturmuster:
Patterns auf architektonischer Ebene
Für konkrete Softwarearchitekturen. Definieren systemweite strukturelle Eigenschaften einer App.
Beispiele: MVC, 3-Tier/2-Tier, Layered Architechture

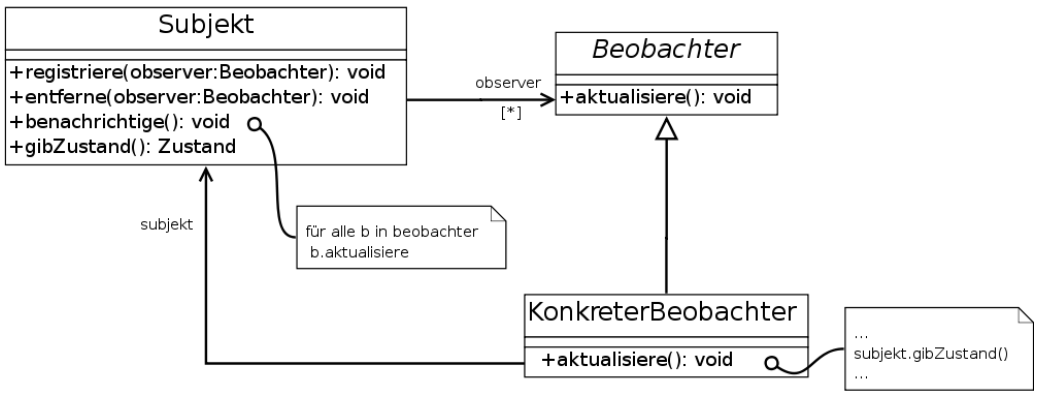
MVC
Problem: Eine Applikation soll auf mehreren Ausgabegeräten, daher auf unterschiedlichen GUI’s laufen. Damit man nicht für jedes Ausgabegerät einzelne Anwendung schreiben muss, kann man mit MVC hohe Wiederverwendbarkeit der Komponenten erzielen und muss nur die View jeweils ändern.



Designmuster:
Muster auf Klassen-Ebene
Creational (Factory)
Structural (Facade, Adapter)
Behavioral (State, Observer)

Observer
Problem: Eine GUI-Komponente stellt Zustand eines Objektes dar. Kennt Schnittstelle des Objekts. Ändert sich der Zustand des Objektes, muss auch GUI-Komponente informiert werden. Objekt soll aber von GUI-Komponente unabhängig bleiben (also ihre Schnittstelle nicht kennen).

Lösung:
Das beobachtete Objekt bietet Mechanismus, um Beobachter (GUI-Komponenten) an- und abzumelden und diese über Änderungen zu informieren. Es kennt alle seine Beobachter nur über die (überschaubare) Schnittstelle Beobachter. Es meldet jede Änderung an jeden angemeldeten Beobachter, braucht also die weitere Struktur dieser Komponenten nicht zu kennen.
Die Beobachter implementieren ihrerseits eine (spezifische) Methode, um auf diese Änderung zu reagieren.



Frameworks vs. Bibliotheken, Method Hooks, Hot Spots, Whitebox/Blackbox

Framework:
ist ein vorgegebenes Codegerüst, dass vom Entwickler angepasst und spezialisiert werden kann, um die Anforderungen zu erfüllen. Menge von abstrakten Klassen. Bietet wiederkehrendes Verhalten an. Frameworks rufen eigenen Code selbst auf! (Beispiel: Android Framework)

Bibliotheken:
Bieten wiederkehrende Funktionalität an. Methoden von Bibliotheken werden durch eigenen Code aufgerufen. (Beispiel: GoogleMaps Library)
Method Hook:
sind abstrakte Methoden (Bsp.: onCreate()), die vom Entwickler überschrieben werden um das gewünschte Verhalten zu erzielen.

Hotspots:
sind Teile wo der Programmier, der das Framework benutzt, eigenen Code einfügt um eigene Funktionalität erzielen zu können.

Whitebox und Blackbox Frameworks:
Die Blackbox- und Whitebox-Abstraktion bezieht sich auf die Sichtbarkeit einer Implementierung hinter der Schnittstelle.

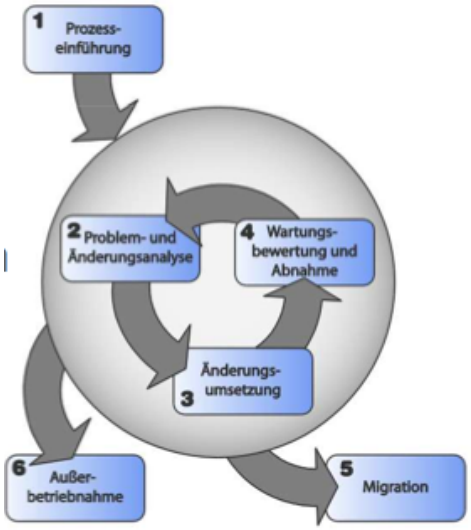
Blackbox: Entwickler kennen keine Details, die sich hinter Schnittstelle befindet.
Whitebox: Implementierung ist vollständig vorhanden und kann vom Entwickler studiert werden ums das Darunterlegende genauer zu verstehen.

Wartungsskriterien/-phasen nach ISO/IEE

Wartungskriterien

korrektive Wartung:	wenn Fehler auftreten, beheben (Bsp.: Bug —> Fehlerkorrektur durch patch, update, workaround)
präventive Wartung:	zur Verhinderung vermutlicher zukünftiger Fehler (z.B. Jahr 2000-Problem)
adaptive Wartung:	wenn sich Hardware/Software/Schnittstellen bzw. generell Umgebung ändert
perfektionierende Wartung:	wenn es darum geht Performance/Wartbarkeit/Usability zu verbessern

Wartungsphasen



1 Prozesseinführung:
Planung des gesamten Wartungsprozesses

2 Problem und Änderungsanalyse:
Analyse des Problems, Änderungsmöglichkeiten werden entwickelt und eine Lösung ausgewählt

3 Änderungsumsetzung
Gewählte Lösung wird umgesetzt

4 Wartungsbewertung und Abnahme
Lösung wird geprüft:
Problem behoben?
JA, Übernahme der Lösung
NEIN, Zyklus startet bei 2

5 Migration
Applikation wird in geänderte Umgebung eingebettet

6 Außerbetriebnahme
Wartungsbetrieb wird eingestellt

Kollektiver Codebesitz. Bedeutung, Vorteile, Nachteile

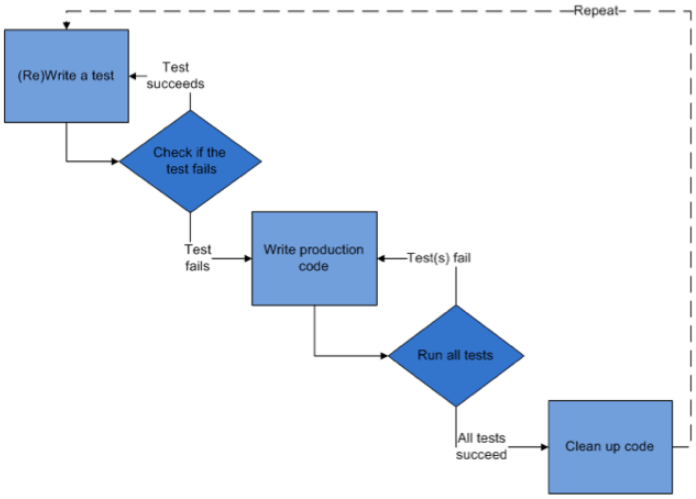
Bedeutung: Jedes Teammitglied verfügt über den gesamten Code, aber nicht jeder ist für den gesamten Code verantwortlich. Dies ist vor allem in agilen Vorgehensweisen üblich.

Vorteile:
+ Jeder kann sein Wissen einbringen
+ Jeder kann Verbesserungen machen
+ Die Auswirkungen des Ausfalls eines Teammitglieds sind geringer

Nachteil:
- die Verantwortlichkeit für schlechten Code ist schlecht nachvollziehbar

Test Driven Development + Skizze, Welchen Bezug zu automatisierten Tests zu TDD gibt es?
TDD ist eine Entwicklungsmethode wo die Programmierer zunächst Tests erstellen und erst nachher Codes zu den Tests. Daher: Der Test gibt vor, welcher Code benötigt wird

Skizze:



automatisiertes Testen:
Oft testen Programmierer ihre Lösungen nur durch „ausprobieren“ oder sehr oberflächliche Tests.
Lösung: automatisierte Funktionstests (Unit Tests)
+ sehr einfach ausführbar (innerhalb IDE)
+ Wichtig zur Vermeidung von ungewollten Nebeneffekten

Refactoring? Welche Methoden? Im Zusammenhang mit TDD erklären?

Refactoring = "Aufräumen des Codes"
nach Refactoring darf sich die Funktion des Programms nicht ändern, Testfälle müssen danach immer noch alle positiv sein!

Refactoring dient zur
- Verbesserung des Quellcodes
- besseren Lesbarkeit des Quellcodes
- besserem Verstehen des Quellcodes

Zusammenhang mit TDD:
Die Vorgehensweise des Refactoring von schlecht strukturiertem Code sollte wie beim TDD erfolgen:
1) Ausreichend viele Tests schreiben, daher hohe Testabdeckung erzielen
2) erst danach Refaktorisierung durchführen
3) Tests durchführen, alle Tests müssen positiv sein

Refactoring-Methoden: Rename Method, Extract Method, Pull Up Method, Split Temporary Variable

Unterschied **Internationalisierung/Lokalisierung**
Internationalisierung: sämtliche Maßnahmen Software so zu gestalten, dass diese möglichst einfach an andere Sprachen + Kulturen angepasst werden kann.

Lokalisierung:
Darunter versteht man den eigentlichen Übersetzungsprozess. Daher das anpassen von Datenformatierungen (Texte, Zahlen, Datum, Währung, etc.). Ist für jede Sprache/Kultur separate

notwendig

Wozu **Logging**-Framework? Warum nicht System.out.println()?
Logging-Frameworks (Log4J) stellen mehrere Log-Levels zur verfügen, womit man eindeutig das Systemverhalten identifizieren kann, was mit System.out.println nicht möglich ist. Die Log-Levels sind typischerweise:
DEBUG: feingranulare Informationen zur Programmausführung
INFO: wichtige Programmereignisse
WARN: eine unerwartete Situation ist aufgetreten
ERROR: ein Fehler ist aufgetreten, die Fehlerbehandlung wird durchgeführt
FATAL: ein nichtbehandelbarer Fehler ist aufgetreten und das Programm kann nicht stabil fortgesetzt werden

Buildmanagmentsysteme, 3 Beispiele
Buildmanagementsystem: Werkzeug um Aufgaben zum Bauen einer Applikation automatisiert ablaufen zu lassen. Stellt Abhängigkeiten zwischen Komponenten sicher.
Also Aufgaben sind: Code kompilieren, Linken von Bibliotheken
3 Beispiele: Ant, Maven, Gradle

zentrale und dezentrale **Versionierung,** Unterschied zwischen pesimistischer/optimistischer Versionierung

Zentral:
verfolgt den Client-Server-Ansatz. Das Repository liegt auf einem Server, auf den mehrere Clients zugreifen um die Versionierung durchzuführen (z.b.: SVN)

Dezentral:
Es existiert kein zentrales Respository, jeder Client besitzt sein eigenes. Die Änderungen werden zwischen den Repositories ausgetauscht (Update, Merge) (zb.: Git)

pessimistische Versionierung (Lock-Modify-Unlock):
Will ein Benutzer eine Datei ändern, so wird die Datei gesperrt. Nach Abschluss der Änderung wird die Datei wieder freigegeben. Während die Datei gesperrt ist verhindert das System Änderungen durch andere Benutzer.
Vorteil: kein Mergen notwendig, da immer nur einer Datei ändern kann.
Nachteil: man muss auf Freigabe von Datei warten

optimistische Versionierung (Copy-Modify-Merge):
Änderungen einer Datei durch mehrere Benutzer sind möglich. Änderungen werden automatisch oder manuell zusammengefügt (gemergt).
Vorteil: effizientes unabhängiges Arbeiten im Team
Nachteil: manche Dateien lassen sich schwierig mergen

Alle **Teststufen** von agiler Softwareentwicklung (**Testarten** von klein nach groß)

Komponententest:
Testen von Objekten, Klassen, Modulen etc. Die Isolation der Testobjekte erfolgt mit Hilfe von Stubs oder Mocks

Integrationstest:
Prüft die Schnittstellen zwischen den Komponenten

Systemtest:
Testen das System im Ganzen und sollen die funktionalen und nichtfunktionalen Anforderungen abdecken. Sie basieren auf Anforderungen, Anwendungsfällen und Geschäftsprozessen.

Akzeptanztest:
Weist die vereinbarten Leistungen des Systems nach. Wird meist vom Kunden oder Benutzer selbst durchgeführt.

Äquivalenzklassen und **Grenzwertanalyse** erklären.
Äquivalenzklassen:
Modelle im Softwaretesting-Prozess, die Eingaben, die die selben Ausgaben und Reaktionen liefern sollen, in Klassen zusammenfassen.
Grenzwertanalyse:
analysiert konkret die oberen und unteren Grenzen der Klassen, da hier die meisten Probleme auftreten (.: Eingabe ist gültig von 9 <= x < 20, dann werden speziell die Werte 9, 8, 19 und 20 getestet!).

Warum prüft man Komponenten isoliert? Was für **Arten von Isolation** kennen Sie?

Weil es entweder andere Komponenten, die mit der getesteten Komponente agieren müssen, noch nicht gibt oder noch nicht getestet werden. Durch Isolation hat man außerdem die Möglichkeit (auch im Falle, dass andere Komponenten schon alle implementiert sind) Fehler besser zu lokalisieren, da man die abhängigen Komponenten als Fehlerquelle ausschließt.

Arten der Isolation:
- Stubs
- Mocks
- Testtreiber
- Simulatoren

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

Dienen dazu, verschiedene voneinander abhängige Komponenten eines komplexen Systems im Zusammenspiel miteinander zu testen. Es werden also hauptsächlich die Schnittstellen geprüft. Die zu testenden Komponenten sind für sich isoliert bereits fehlerfrei und funktionsfähig.

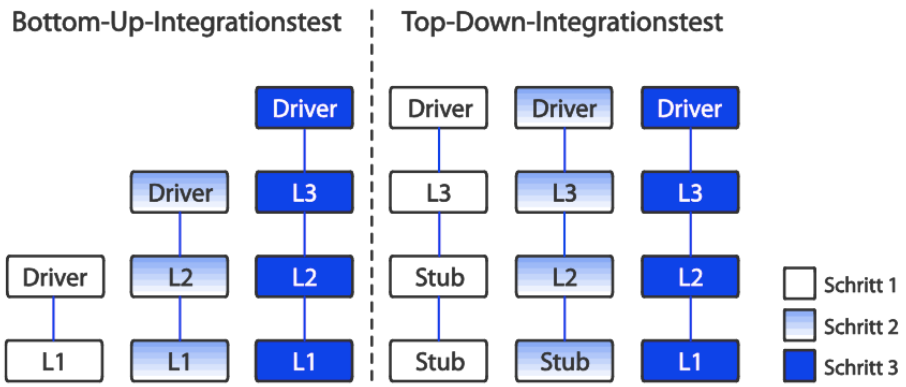
Zwei Ansätze:
<u>Inkrementelle Integrationsstrategie</u>

Bottom-Up Integration
Hier arbeitet man sich ausgehend von den hardwarenahen Schichten nach oben vor.
Vorteil: frühe Integration risikobehafteter Teile
Nachteil: man muss Driver simulieren, Kunde sieht lange keinen Sichtbaren Fortschritt

Driver…Softwaremodule, die oben liegende Schichten simulieren, z.B. GUI-Interaktion durch ein Skript simulieren

Top-Down Integration
Die Komponenten werden von höher liegenden Schichten der SoftwareArchitektur ausgehend nach unten integriert. (Bsp.: zuerst GUI - dann Business Logic - dann Data-Access-Layer)
Vorteil: höher liegenden Schichten (GUI) früh sichtbar
Nachteil: man muss Stubs schreiben, hardwarenahe Teile (fehleranfällig!) sehr spät integriert

Stub…Softwaremodule, die noch nicht integrierte Schichten simulieren



Big Bang Strategie:
Alle Komponenten werden zu einem System integriert und dieses wird getestet.
Vorteil: Kein zusätzlicher Zeitaufwand (um fehlende Komponenten zu simulieren)
Nachteil: Fehler schwer lokalisierbar

Kontrollflussorientierte **C0-C4 Tests erklären**. Warum ist 100%-ige Testabdeckung nicht immer möglich/sinnvol?

Anweisungsüberdeckungstest (C0) Jede Anweisung (Knoten) wird mind. 1x ausgeführt

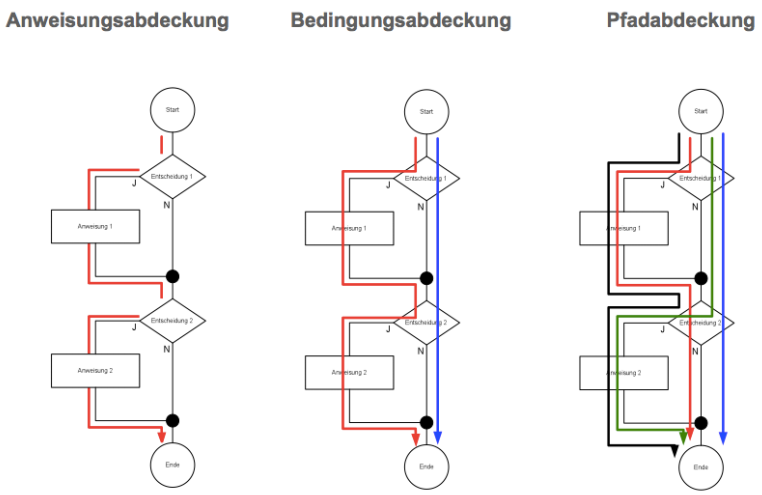
Zweigüberdeckungstest (C1) Alle Zweige (Kanten) werden mind. 1x ausgeführt

Einfacher
Bedingungsüberdeckungstest (C2) jede atomare Entscheidung muss einmal einen wahren und einmal einen falschen Wert annehmen

Mehrfach-
Bedingungsüberdeckungstest (C3) zusätzlich zu den atomaren Teilentscheidungen müssen alle zusammengesetzten Teilentscheidungen und Gesamtentscheidungen einmal mit wahr und einmal mit falsch getestet

Pfad-Überdeckungstest (C4) Alle unterschiedlichen Pfade des Testobjekts zumindest einmal durchlaufen

100% Testabdeckung: Eine vollständige Aufzählung aller möglichichen Systemzustände bei komplexen Systemen ist oft unmöglich und nicht sinnvoll.



Migration - Definition und 3 Arten

Definition:	Ablöse eines laufendes Systems durch ein neues
Eigenschaften:	Darf laufenden Betrieb nicht beeinflussen muss ein geordneter Prozess sein

3 Arten

Softwaremigration:	Austausch bestehender Software durch neue
Datenmigration:	Transfer von Daten von einem System in ein anderes
Hardwaremigration:	Umstellung der verwendeten Hardware

Oft Kombination aus diesen. Beispiel: Softwaremigration bedeutet meist Datenmigration

Continuous Delivery

Software wird kontinuierlich ausgeliefert. Dies geschieht durch automatisierte (Build-) Prozesse.
Bsp.: Facebook liefert tägliche Releases ohne Service-Unterbrechung

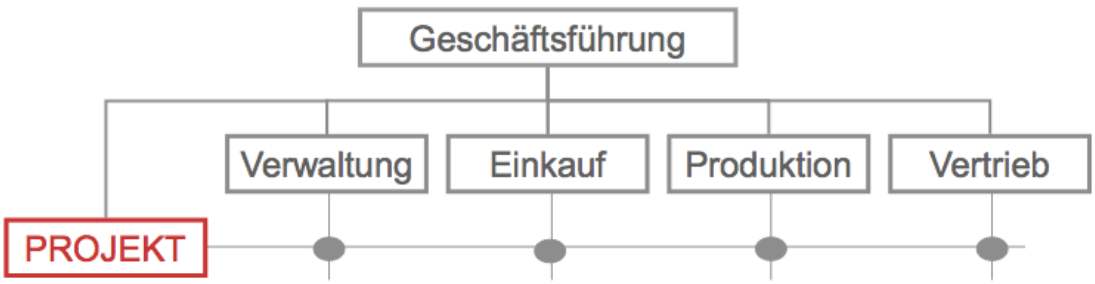
Prinzipien:
- Releaseprozess muss zuverlässig und wiederholbar sein
- Alles ist versioniert (git, svn)
- es existieren automatisiert Build-Tools (maven, gradle, ant)
- oft verwendet man Build-Server (Bsp.: Jenkins) der bei jeder Versionsänderung im Repository automatisiert testet und buildet —> Software ist auslieferungsfähig

Vorteile:
+ hohe Zuverlässigkeit
+ weniger Fehler
+ weniger Stress
+ kürzere Cycle-Time

Matrix-Projektorganisation erklären

Die Matrixorganisation entsteht durch das Überlagern von zwei Weisungslinien. Ein Mitarbeiter hat in dieser Form zwei Vorgesetzte. Er ist nicht nur dem Fachbereichsleiter unterstellt sondern auch dem Leiter des Projektes auf seiner Ebene (siehe Abb.) Dabei sind beide Instanzen gleichwertig

Vorteile/Nachteile:
+ Projektverantwortung liegt beim Projektleiter
+ Sicherheitsgefühl bei den Mitarbeitern
+ Flexibler Personaleinsatz
- Mitarbeiter hat 2 Vorgesetzte
- hoher Koordinationsaufwand



Delphi-Methode
Methode zur Aufwandsabschätzung eines Projekts bzw. eines Arbeitspakets. Es gibt einen Moderator & mehrere Experten

- jeder Experte gibt einen Schätzwert für ein Arbeitspaket ab
- falls alle Schätzwerte relativ gut beieinanderliegen (+/- 20%) wird der Mittelwert daraus genommen. falls NICHT: Argumentationen austauschen und neue Schätzwerte abgeben
- Zu jedem Einzelschätzwert werden immer 10%-15% Aufwand für das Projektmanagement gerechnet

3-Experten Konzept
Methode zur Aufwandsabschätzung von Arbeitspaketen (auch Mini-Experten Konzepten).

- Drei Personen schätzen unabhängig voneinander den Aufwand eines Arbeitspakets.
- alle drei Experten jeweils ein optimistisches, ein realistisches und ein pessimistisches Schätzergebnis.
- Diskussion führen, bis Ergebnisse in jeder Kategorie übereinstimmen
- Schätzwert daraus bilden

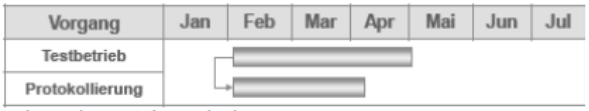
GANNT Diagramm. Kritischer Pfad. 5 Beziehungen eines **GANTT Diagramms** bennen und erklären

GANNT Diagramm: älteste und weitverbreitete grafische Methode zur Terminplanung/ Projektplanung. Stellt zeitlich Abfolge von Aktivitäten in Form eines Balkendiagramms auf einer Zeitachse dar.

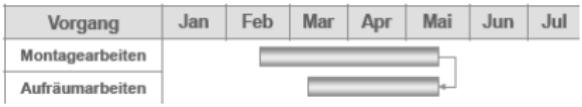
kritischer Pfad
für Aktivitäten wird oft eine Pufferzeit vereinbart (wie lange darf sich Aktivität verzögern)
2 Arten von Puffer:
Gesamtpuffer —> wie lang darf sich Aktivität verzögern ohne Endergebnis zu verzögern.
Freier Puffer —> wie lang darf sich Aktivität verzögern ohne andere Aktivität zu verzögern
kritischer Vorgang —> Aktivität die keinen Puffer erlaubt
kritischer Pfad —> enthält nur kritische Vorgänge

- Ende-Anfangs-Beziehung (EA)
  - Ende Vorgang1 Vorraussetzung für V2
  - Sequentielle Abarbeitung

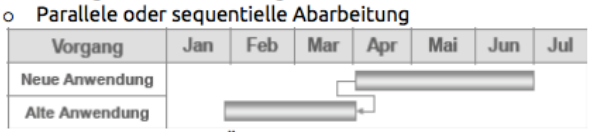
- Anfangs-Anfangs-Beziehung (AA)
  - Anfang V1 ist Vorraussetzung für Anfang V2
  - Parallele Abarbeitung



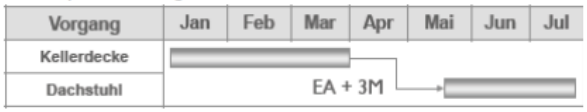
- Ende-Ende Beziehung (EE)
  - Ende V1 Vorraussetzung für Ende V2
  - Parallele Abarbeitung



- Anfangs-Ende Beziehung (AE)
  - Anfang V1 Vorraussetzung für Ende von V2
  - Parallele oder sequentielle Abarbeitung



- Vorgangsbeziehung mit Überlappung oder Verzögerung
  - Zeit/Prozentangaben



Spiralmodell nach Böhm
- ist iterativ aufgebaut
- berücksichtigt Risiken (im Gegensatz zu sequentiellen Modellen, bspw. Wasserfallmodell)
- Prozess in 4 Phasen aufgeteilt die immer wieder durchlaufen werden

1) Definition von Zielen
2) Risikoanalyse
3) Durchführen der Arbeitsschritte
4) Planen der nächsten Phase

- Anzahl der Phasendurchläufe ergibt sich erst während des Projektverlaufs

2 Methoden zu **statischer Qualitätssicherung**

Reviews: können in jeder Phase des Entwicklungsprozesses eingesetzt werden, zeichnen sich daher durch frühe Fehlerfindung aus!

statische Codeanalysen: ermöglichen die Prüfung einzelner Programmteile, bereits lange vor der Integration zu einem lauffähigen System!

Was sind **Softwaremetriken?** 3 Bsps objektorientierter Metriken
Ermöglichen das Messen von Softwarequalität. Im unterschied zu Prozessmetriken (messen Softwareentwicklungsprozess), messen sie die Qualität der Software selbst.

gängige Metriken: Umfangsmetrik (LOC, Lines of Code), McCabe-Metrik (Zyklomatische Zahl)

objektorientierte Metriken sind notwendig um Beziehungen von Objekten und deren Struktur zu berücksichtigen.
CBO - coupling between objects
DIT - depth of inheritance tree
WMC - weighted methods per class

Kopplung/Kohäsion, Zusammenhang**?**
Gehören zu Konzepten von wartungsfreundlicher Implementierung.

Kopplung: Maß der Abhängigkeit zwischen Klassen
Enge Kopplung = viele Klassen müssen interne Details voneinander kennen. Änderungen sind schwieriger. Lose Kopplung erstrebenswert

Kohäsion: Maß der Abhängigkeit innerhalb Klassen
hohe Kähasion: alle Inhalte einer Klasse beziehen sich tatsächlich auf Namen der Klasse
Bsp. schlechte Köhäsion: wenn man Methoden zu „Fahrplan“ in Klassen wie „Zug“ oder „Bahnhof“ findet, sollte man sie in eine Klasse „Fahrplan“ auslagern

Review Prozess, Was sind Rollen? Ablauf?

Begriff: Zusammenkunft von Personen zur Überprüfung eines Produktteils (Dokument, Programmteil) nach vorgegebenen Prüfkriterien.
Alle Entwicklungsergebnisse können mit Reviews geprüft werden.

Der Prozess:
- Planungsphase
- Initialisierungsvorgang
- Vorbereitung
- Sitzung
- Nacharbeit

- (Analyse)

Rollen:

Manager	Projekleiter, gibt Auftrag zur Erstellung eines Testobjekts, verantwortlich für dessen Freigabe
Moderator	plant, organisiert, leitet Review
Schreiber	notiert alle Erkenntnisse des Teams in einem Protokoll
Autor	Repräsentiert das Team, welches das Testobjekt erstellt hat
Gutachter	Begutachten Testobjekt in der Vorbereitung, berichten ihre Erfahrungen im Review
Leser:	bereitet Softwareelemente des Testobjekts vor, ist für zeitliche Durchführung des Prozesses zuständig

Reviews wichtig weil:
- sehr früh durchführbar (noch bevor ausführbare Programme vorliegen)
- Latenzzeit eines Fehlers kann verkürzt werden
- Kosten eines Fehlers sind geringer

Audit vs. Review

Audit: Einhaltung vorgegebener Vorgehensweisen, Standards, als auch deren Sinnhaftigkeit wird geprüft
Review: Zusammenkunft von Personen zur Überprüfung eines Produktteils nach Prüfkriteren

Was ist CMM? 5 Reifegrade
Modell zur
- Beurteilung der Qualität von Softwareprozessen einer Firma
- Verbesserung der Softwareprozesse

Ein **CMM-Assessment** beurteilt die **Fähigkeit einer Firma**, Software unter bestimmten Rahmenbedingungen erstellen zu können.

1) Initial	Keine Anforderungen. Diesen Reifegrad hat jede Organisation automatisch.
2) Managed	Projekte werden geführt. Ein ähnliches Projekt kann erfolgreich wiederholt werden.
3) Defined.	Projekte nach Standardprozess durchgeführt. Es gibt kontinuierliche Prozessverbesserung.
4) Quantitatively Managed	Statistische Prozesskontrolle wird durchgeführt.
5) Optimizing	Arbeitsweisen werden mittels statistischer Prozesskontrolle verbessert.

Wozu dient **SPICE**? 6 Reifegrade!

Software Process Improvement and Capabiltiy
Ist dem CMM-Modell sehr ähnlich (80-90% Überdeckung)

Reifegrade:
0) unvollständig
1) vorhanden
2) geleitet
3) gesichert
4) vorhersehbar
5) optimiert

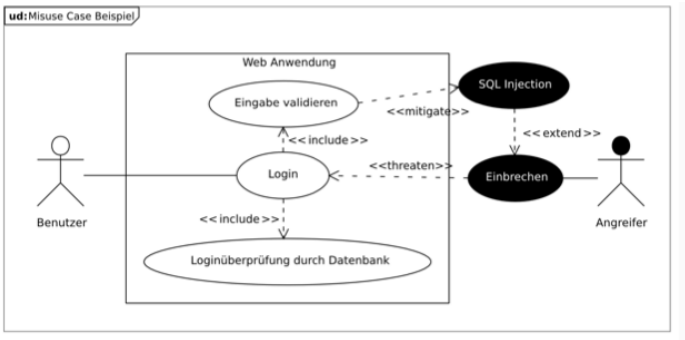
Bewertung jedes Attributs erfolgt anhand vierstufiger Skala:
nicht erfüllt: 0%-15%
teilweise erfüllt: 15% - 50%
weitgehend erfüllt: 50% - 85%

vollständig erfüllt: 85% - 100%

Misuse-Case + Beispiel

- Erweiterung zu normalem Use Case
- Security-Aspekte sollen schon in der Analyse- und Entwurfsphase berücksichtigt werden
- stellen einen Use Case für mögliche Angreifer dar
- Sicherheitsrisiken in bestimmten Use-Cases aufzeigen

Beispiel:
Use Case "Eingabe Validieren" —> Misuse Case "SQL Injection"
Use Case „Login“ —> Misuse Case „Einbrechen"



SQL Injection & Gegenmaßnahmen

SQL Injeciton ist ein Angriff bei dem nicht sichere SQL-Statements in Webanwendungen manipuliert werden.
Ein Angreifer versucht über die Anwendung eigene Datenbankbefehle einzuschleusen.
Sein Ziel: Daten ausspähen, verändern, Kontrolle über Server erhalten oder einfach größtmöglichen Schaden anrichten.

Injection Möglichkeiten:
- Durch Metazeichen wie „Hochkamma“ wird Fehler in Datenbank anzeigt. Dadurch ist Schwachstelle erkannt. Zusätzlich werden manchmal Informationen zum Aufbau der DB angezeigt
- Verwenden von Tautologien wie ‚1‘ = ‚1‘ in WHERE Klausel
  Beispiel: SELECT * FROM USER WHERE password=‚ OR ‚1‘ = ‚1‘
  Gibt alle User zurück
- Auslesen von Daten aus anderen Tabellen mittels einbringen von UNION Klauseln

Gegenmaßnahmen:
- Filterung spezieller Datenbankzeichen
  Blacklist-Filtering: Filterung von Zeichen wie „Hochkommata“
  Whitelist-Filtering: Prüfung der Eingaben auf gültige Zeichen
- Prepared/Precompiled Statements verwenden
  Statement wird mit Platzhaltern kompiliert, danach eingefügt
- Rechte einschränken in Datenbank, Principle of „Least Priviledge“

Software Reengineering und **Reverse Engineering**

Software Reengineering
Neuentwicklung von Software bei gleichbleibender Funktionalität
Motivation ist bessere Qualität des Produkts (Code)

Reverse Engineering
Gewinnung von Code aus vorhandenen Projekten
Notwendig wenn Source-Code nicht verfügbar

Meilensteintrendanalyse Bsp. Achsen erklären. Diagramm interpretieren.
Meilensteintrendanalyse dient der Projektplanüberwachung

x-Achse: Berichtszeitpunkte
y-Achse: Meilensteintermine