Erklären Sie das Modell des komponentenbasierten Systementwurfs und seine Bedeutung für die Wartung.

Modularisierung: Folie: Design_Architektur, 101ff
Aufgabenteilung: Jedes Modul hat eine genau festgelegte Verantwortung, die es zu erfüllen hat. Bei auftretenden Fehlern kann aufgrund dieser bekannten Verantwortungen die Quelle eines Fehlers eng eingegrenzt werden. Auch Änderungen lassen sich dadurch gezielt an der richtigen Stelle durchführen.

Skalierbarkeit: Durch Aufteilung in Module und die definierte Zusammenarbeit können einem System Module hinzugefügt werden, welche die Funktionalität erweitern. Module können ebenso entfernt werden, ohne die Funktionalität von anderen Modulen zu beeinflussen.

Wartbarkeit: Einzelne Module können gegen neuere Versionen ausgetauscht werden, ohne das Verhalten von anderen Modulen zu beeinflussen.

Welche Typen der Wartung werden unterschieden? Erklären Sie die unterschiedlichen Typen der Wartung. Wieso ist die Unterscheidung so wichtig?(!)

- Präventive Wartung: Jene Maßnahmen, die ergriffen werden, um Fehler zu vermeiden, bevor sie auftreten und somit weiteren Wartungsaufwand vermeiden.

- Korrektive Wartung: Änderungsmaßnahmen, die unmittelbar notwendig werden, wenn ein Fehler im System auftritt.

- Adaptive Wartung: Anpassungen, die durchgeführt werden müssen, wenn sich das Softwareprodukt und die entsprechende Systemlandschaft ändern.

- Perfektionierende Wartung: Änderungen, die das System qualitativ aufwerten sollen.

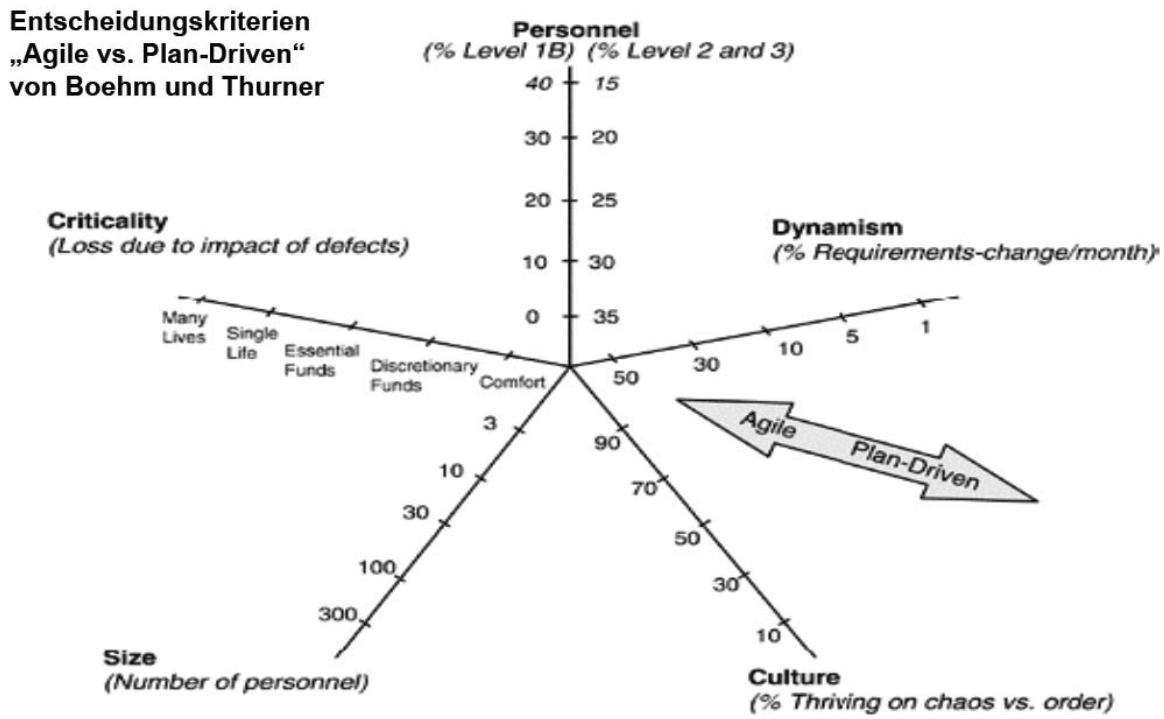
Erklären und erläuern Sie die Entscheidungskriterien für "Agile vs. Plan-Driven" Methoden von Boehm und Thurner.

Agile Sicht: XP
Plan-Driven Sicht: Unified Process
Unified Process:
- Anwendungsfall gesteuert
- Architekturzentriert
- Iterativ und inkrementell (kleine Schritte, geringes Risiko)

Agil (engl. agile)
▪Ergebnisse früh & danach laufend sichtbar (kurze Iterationszeiten)
▪Vage Anforderungen und Änderungen sind kein Problem
▪Produkt, Mensch und Kommunikation stehen im Mittelpunkt
▪Miteinbeziehung des Kunden wird angestrebt
▪Traditionell (engl. oft plan-driven)
▪Späte Ergebnisse
▪Exaktes Verständnis der Anforderungen und somit stabiles Ziel
▪Prozess, Werkzeuge und das Verfolgen eines Plans stehen im Mittelpunkt
▪Direktes Mitwirken des Kunden eher unwahrscheinlich

Boehm und Thurner:
5 Achsen: Size (Number of Personnel), Culture, Dynamism (Requirements - change/month), Personnel (Level 1B, Leven 2 u 3), Criticality (Loss due impact of defects (comfort, discretional funds, essential funds, single life, many lifes...)
Weniger Personal, mehr Kultur, Mehr Dynamik, mehr Level 2,3 Leute, weniger criticality (comfort) -> Agile Methoden.

Folien: SE-Prozesse 1, 62ff



Erläutern Sie den Unterschied zwischen Bottom-Up und Top-Down Integration. Wo liegen die jeweiligen Vor- und Nachteile? Welche Variante wird häufiger verwendet und warum? Unter welchen Vorraussetzungen können Sie welche Variante verwenden?

	Bottom-Up Integration	Top-Down Integration
Beschreibung	Hier arbeitet man sich ausgehend von einem niedrigen Level auf die weiter oben liegenden Schichten vor.	Die Komponenten werden von den höchsten Schichten ausgehend nach unten integriert.
Vorteile	Ist deutlich effizienter als die Top-Down Integration.	Eine Demoversion ist sehr früh fertig.
Nachteile	Oben liegende Schichten müssen simuliert werden, der Kunde sieht Ergebnisse ziemlich spät, was Prototypen notwendig macht.	Es müssen Stubs geschrieben werden (Systemkomponenten, die Softwaremodule noch nicht existenter Schichten simulieren). Lowlevelkomponenten, werden sehr spät fertiggestellt. Hohe Kosten für Dokumentation werden erzeugt.

Bottom-Up Integration
Hier arbeitet man sich ausgehend von den hardwarenahen Schichten nach oben vor.
Vorteil ist die frühe Integration der risikobehafteteren Teile (wodurch man später auf ein stabiles Framework aufbauen kann), Nachteil ist dass man Driver (Softwaremodule, oben liegende Schichten simulieren, z.B. GUI-Interaktionen durch ein Script simulieren) schreiben muss. Der Kunde sieht lange keinen Fortschritt, weil die sichtbaren Teile erst spät integriert werden. Es müssen daher zusätzlich Prototypen erstellt werden.

Top-Down Integration
Die Komponenten werden von höher liegenden Schichten der Software-Architektur ausgehend nach unten integriert.
Beispielsweise wird zuerst das GUI mit der Business Logic zusammengefügt, später kommt unten noch der Data-Access-Layer hinzu.
Vorteil ist, dass die höher liegenden Schichten - das sind die für den Benutzer sichtbareren - früh verfügbar sind, was sich für Demo-Zwecke gut eignet. Allerdings sind Stubs zu schreiben (das sind Softwaremodule, die die noch nicht integrierten unteren Schichten simulieren). Außerdem werden hardwarenahe Teile (das sind üblicherweise die fehleranfälligeren) erst spät integriert werden.

Softwarewartung im Unified Process. Welche Rollen gibt es, welche Produkte? Wieso wird zwischen Fehlerbehebung und Erweiterung unterschieden?(!)

- Architekt: Produkt, Auslieferungsplan, Abnahmeprotokoll, Integrations-/Migrationsplan
- Dokumentierer: Anwenderdokumentation
- Wartungsteam: Fehlerbericht, Änderungswunsch

Fehler ist eine Abweichung von den Anforderungen oder einem Dokument welches sich davon ableitet. Wenn Erweiterungen notwendig sind heißt es dass die Anforderungen erfüllt sind aber entweder dort etwas versäumt wurde oder der Kunde es sich einfach nicht gewünscht hat. Dh.: Fehler sind vom Team zu tragen, Erweiterungen können dem Kunden in Rechnung gestellt werden.

Weil es wichtiger ist ein funktionierendes Programm zu haben, als ein 100% an die Bedürftnisse angepasst ist.

Erklären Sie das Konzept des Refactorings inklusive dessen Vor und Nachteile. Nennen und erläutern Sie ein Beispiel für ein Refactoring Muster.

Refaktorisierung: Änderung an der internen Struktur einer Software, um sie leichter verständlich zu machen und einfacher zu verändern, ohne ihr beobachtbares Verhalten zu ändern.
Refaktorisieren: Eine Software umstrukturieren, ohne ihr beobachtbares Verhalten zu ändern, indem man eine Reihe von Refaktorisierungen anwendet.

Vorteile:
- Erhöhung der Wartbarkeit
- Erhöhung der direkten und indirekten Wiederverwendbarkeit
- Erhöhung der Verständlichkeit
- Indirekte Code Inspektion
- Wird oft durch Werkzeuge (IDEs) unetrstützt

Nachteile:
- Nutzen oft nicht direkt messbar
- Veränderung kann unerwünschte Seiteneffekte beinhalten
- Refaktorisierung nicht immer anwendbar
- Verschiedene Refaktorisierungsstrategien anwendbar

zB: Methoden neu definieren: Man hat 2 Methoden, die etwas tun und danach einen Wert ausgeben sollen. Man kann nun eine 3. Methode definieren, die nur Werte ausgibt und dieser Methode einfach den Wert, der ausgegeben soll, übergibt.

Folien: Realisierung Integration 28ff

Scrum beschreiben. Die wichtigsten Elemente und Rollen.

Scrum = das Gedränge

- **Elemente:**

    - Sprint: ca. 30-tägige Periode, nach der ein potentiell auslieferbares System vorhanden sein soll. So schnell wie möglich, keine Vorgaben zu Dokumentation oder Methoden (Modelle und Entwürfe nur für Verständnis da). Abbruch möglich. Das wichtigste ist die gelieferte Funktionalität.

    - Daily Scrum: Tägliches ca. 15-minütiges Meeting, an dem der Scrum-Master jedes Team-Mitglied 3 Fragen zur aktuellen Situation seiner Arbeit fragt.

    - Product Backlog: stellt die gewünschte Funktionalität dar, die vom Product Owner priorisiert wird. Ist von allen Projektbeteiligten jederzeit einsehbar, äußert dynamisch und kann nur vom Product Owner geändert werden.

    - Inkrement: Software-System entsteht in Inkrementen und soll nach spätestens 2 Sprints eine lauffähige Version liefern.

    - Burndown-Chart: Diagramm mit tagesaktuellen Aufwandsschätzungen füraktuellen Sprint. Wird vom Scrum-Master gepflegt und hat als Ziel, möglichst bald Probleme und Hindernisse zu erkennen.

- **Rollen:**

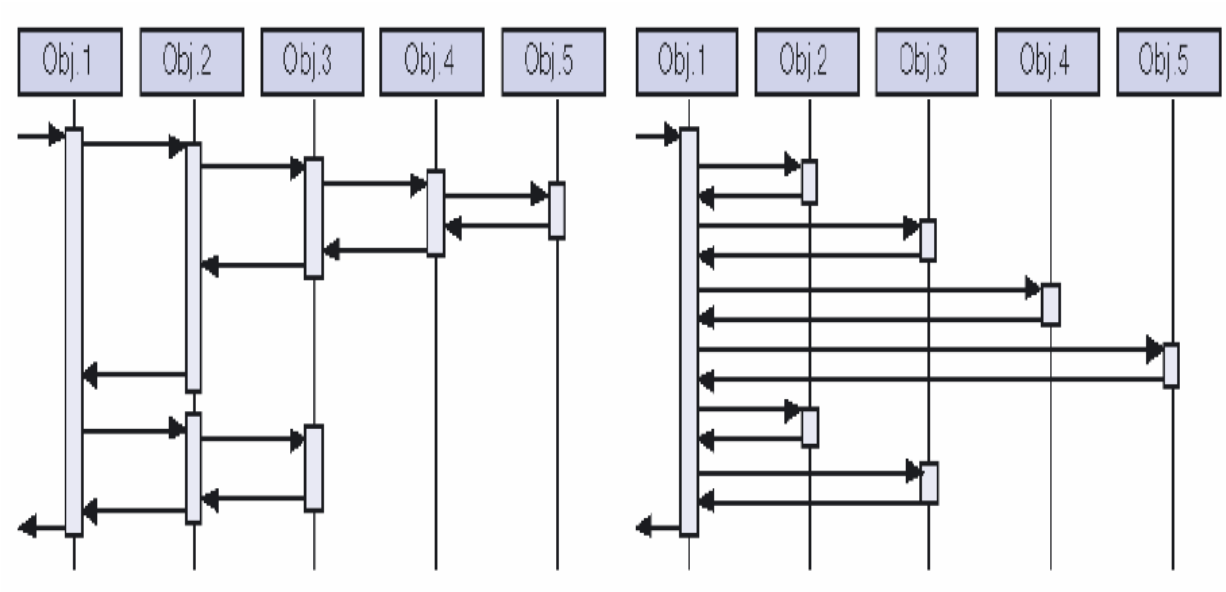
    - Product Owner: Erstellt und priorisiert den Product Backlog. Stellt den ausschließlichen Kommunikationskanal zwischen Team & Außenstehenden dar

    - Scrum Team: Verantwortlich fürdie Erreichung des Sprint Goals und der Umsetzung des Sprint Backlogs

    - Scrum Master: Achtet auf die Umsetzung der Grundsätze und Regeln und schottet das Team vor äußern Einflüssen ab.

Erklären Sie die Unterschiede zwischen Stair - und Fork- Kontrollstil

Kontrollstile beschäftigen sich mit dem Datenfluss zwischen Subsystemen.
Zentralisierte Kontrolle: Ein Subsystem hat die gesamte Verantwortung über die Kontrolle und startet / stoppt andere Subsysteme. (Fork: Ein Objekt kommuniziert immer mit den anderen ohne Zwischenobjekte)
Event-basierte Kontrolle: Jedes Subsystem kann auf Events anderer Subsysteme oder der Systemumgebung reagieren. (Stair: Objekt1 komm. mit Objekt2 komm. mit Objekt 3. Jedes Objekt kümmert sich um sein Folgeobjekt).
Folien: Design und Architektur, Seite 30 ff



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

- Prinzipen der Objektorientierung: Datenkapselung, Objektidentität, Vererbung, Polymorphismus

Prinzipien:
- Anwendungsfall gesteuert (verfolgbarkeit)
- Architekturzentriert (Schnittstellen, Effizient,..)
- Iterativ und inkrementell (kleine Schritte, geringes Risiko)
- Definiert Inhalte in 4 Phasen (Beginn, Ausarbeitung, Konstruktion, Umsetzung)
- Verwendet UML zur Darstellung der Modelle
- Arbeitsschritte: Anforderungen, Analyse, Entwurf, Implementierung, Test

Rollen: (nicht auf folien gefunden)
- Technischer Architekt
- Testkoordinator
- Projektmanager
- Programmierer

Se - prozesse 1, 16ff

Erklären Sie die Begriffe „Funktionale Anforderungen“, „Nichtfunktionale Anforderungen“ und „Domainen Anforderungen“ genau. Welche Typen von nichtfunktionalen Anforderungen können klassifiziert werden?

Funktionale Anforderungen:

Beschreibung der Funktionen/Services, die das System zur Verfügung stellen soll, wie das System auf auf bestimmte Eingaben reagieren soll, und wie das System in welchen Situationen operieren soll.

Nichtfunktionale Anforderungen:

Anforderungen an die Funktionen/Services, die nicht deren Verhalten, sondern Vorgaben an Qualitätseigenschaften, Entwicklungsprozess, einzuhaltende Standards und Normen, gesetzliche Rahmenbedingungen, etc. beschreiben.

Domainen Anforderungen:

Anforderungen, die von der Domaine des Systems definiert werden, und die besonderen Eigenschaften/Einschränkungen dieses Gebietes darstellen. Domainen-Anforderungen sind oft inhärent, und nicht explizit im Projekt vertreten.

Typen nichtfunktionaler Anforderungen:

Produkt Anforderungen, organisatorische Anforderungen und externe Anforderungen.

Welche Arten des Tailoring im V-Modell XT gibt es? Beschreibung

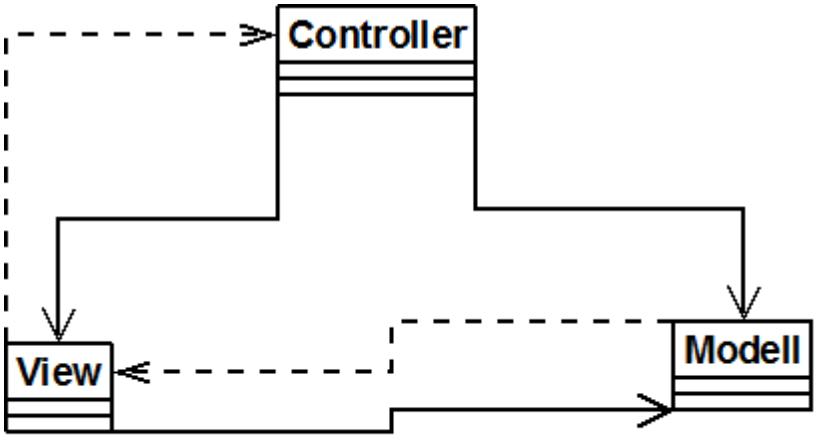
Folie: SE-Prozesse 2, 20ff
Statisches Tailoring:
- ▪Zu Projektbeginn
- ▪Charakterisierung eines Projekts anhand vorgeg. Projektmerkmale (Profil)
- ▪Profil legt verpflichtend bzw. optional zu verwendenden Vorgehensbausteine sowie die möglichen Durchführungsstrategien fest
- ▪Durch Hinzunahme; nicht durch Streichung (nur relevante VB/DS wichtig)

Dynamisches Tailoring
- ▪Während des Projekts
- ▪Durch Hinzunahme/Entfernung von VB (Ausnahmen, Regeln)

Detaillierte Anpassung erst bei Projektplanung
- ▪Produkte (Abhängigkeiten), Aktivitäten

Erweiterungen/Anpassungen (Tool: V-Modell XT-Editor)
- ▪z.B. von Vorgehensbausteinen, Entscheidungspunkten, …

12. MVC Pattern erklären.



Das Model-View-Controller Pattern sieht eine Strukturierung in 3 Einheiten vor: Datenmodell (Modell) , Programmsteuerung (Controller) und Präsentation (View).

Präsentation(View):

Die Präsentationsschicht stellt die benötigten Daten des des Datenmodells dar, wird über Datenänderungen benachrichtigt und reicht Eingaben an die Steuerung weiter, ist aber nicht für die Verarbeitung der Eingaben des Benutzers zuständig.

Datenmodell (Model):

Enthält die Daten, und eventuell die Geschäftslogik. Benachrichtigt die anderen Schichten über Datenaktualisierungen mithilfe des Observer Patterns.

Steuerung (Controller):

Verwaltet eine oder mehrere Präsentationen, und nimmt Benutzeraktionen entgegen. Enthält Mechanismen, um die Benutzerinteraktion der Repräsentation einzurschränken. Präsentation und Steuerung verwenden gemeinsam das Entwurfsmuster Strategie, wobei die Steuerung der Strategie entspricht.

Erklären von MDD.
Was ist der Unterschied zwischen PSM und PIM?

MDD (Model Driven Development): Das System wird in einem abstraktem Modell beschrieben. Dabei wird das abstrakte Modell durch automatische Transformation in Code umgewandelt. Dieser Vorgang kann auch in mehreren Schritten geschehen. Am Schluss der Kette ist die Implementation des Systems.

- PSM (Platform Specific Model): beschreibt, wie das Basismodell auf verschiedenen Plattformen implementiert wird.

- PIM (Platform Independent Model): vollkommen unabhängig von einer plattformspezifischen Syntax.

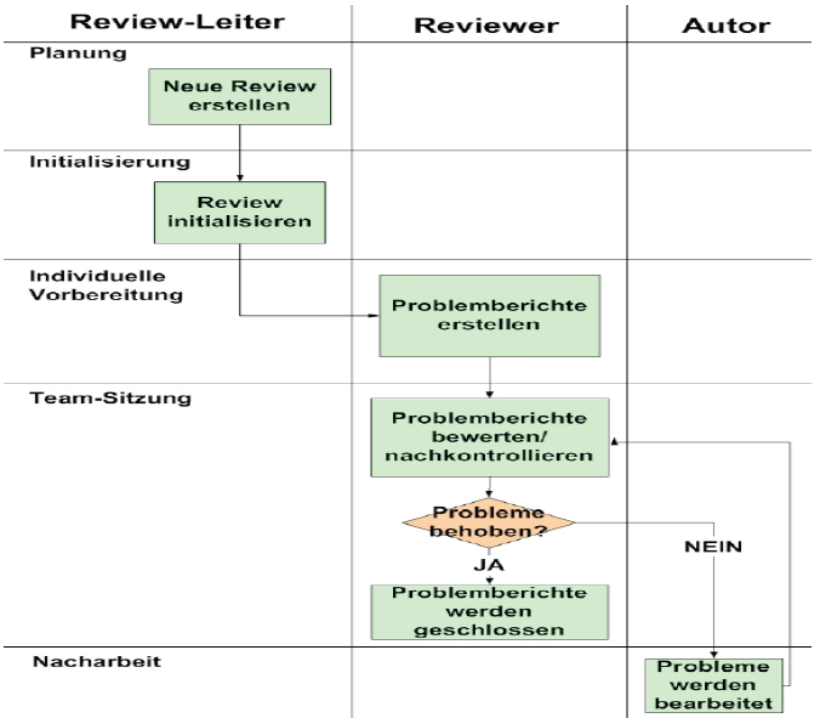
Was sind die wesentlichen Qualitätsanforderungen zur Verifikation von Software-Anforderungen?

se-anforderungen, 33ff

Vollständig
alle Anforderungen des Kunden müssen explizit beschrieben sein, es darf keine impliziten Annahmen des Kunden über das zu entwickelnde System geben.
Eindeutig definiert / abgegrenzt
präzise Definitionen helfen, Missverständnisse zwischen Entwickler und Auftraggeber zu vermeiden.
Verständlich beschrieben
damit sowohl der Auftraggeber als auch der Entwickler mit vertretbarem Aufwand die gesamte Anforderung lesen und verstehen können.
Atomar
es darf nur eine Anforderung pro Abschnitt oder Satz beschrieben sein.
Identifizierbar
jede Anforderung muss eindeutig identifizierbar sein (z. B. über eine Kennung oder Nummer).
Einheitlich dokumentiert
die Anforderungen und ihre Quellen sollten nicht in unterschiedlichen Dokumenten stehen oder unterschiedliche Strukturen haben.
Notwendig
Sind gesetzliche Vorschriften unabdingbar.
Nachprüfbar
die Anforderungen sollten mit Abnahmekriterien verknüpft werden, damit bei der Abnahme geprüft werden kann, ob die Anforderungen erfüllt wurden (Ableitung von Testfällen aus den Abnahmekriterien).
Rück- und vorwärtsverfolgbar
damit einerseits erkennbar ist, ob jede Anforderung vollständig erfüllt wurde und andererseits für jede implementierte Funktionalität erkennbar ist, aus welcher Anforderung sie resultiert, also nicht Überflüssiges entwickelt wird.

Beschreiben des Review-Prozess nach IEEE. Wieso sind Reviews in der Praxis wichtig?

Review-Prozess nach IEEE In SEPM-SE-Qualitaetssicherung_Betrieb_Wartung.pdf , S. 31,
findet sich ein Graph der folgendes beschreibt: Zuerst wird ein Review vom Leiter erstellt und initialisiert. Die Reviewer erstellen anschliessend Problemberichte (in der individuellen Vorbereitung). In einer Team-Sitzung werden die Problemberichte kontrolliert und bewertet. Sind die beschriebenen Probleme behoben, werden die Berichte geschlossen, ansonsten werden die Probleme vom Author bearbeitet (und hoffentlich geloest).
Warum wichtig?
Dazu wird in den Folien nix explizit gesagt. Wohl aber primaer weil es die Qualitaet der einzelnen Entwicklungsstufen sicherstellt und ein "Aufbauen auf Fehler" verhindert.



Wie unterscheidet sich die Anforderungsanalyse im Unified Process (RUP) und im XP?

RUP:

- Das Analysemodell soll die Anforderungsanalyse auf Vollständigkeit, Konsistenz und Machbarkeit überprüfen.
- Das System wird nach objektorientierten Gesichtspunkten von innen beschrieben.
- Das System wird in die Klassentypen Schnittstelle, Kontroller und Entitäten unterteilt. Durch die Zusammenarbeit dieser Klassen sollen alle Systemfunktionen für die Benutzer verfügbar gemacht werden.
- Das Analysemodell entspricht einer Aufteilung des Systems nach dem MVC-Paradigma.
- Dynamische Diagramme dienen zur Darstellung von komplexen Abläufen in oder zwischen einzelnen Anforderungen oder innerhalb des Analysemodells.
- Dynamische Diagramme werden nur für besondere Fälle eingesetzt, in denen das Verständnis der dynamischen Funktionsweise des Systems nicht trivial erkennbar ist.
- Die Analyse stellt die Vorbereitung für den Entwurf dar. Während im Entwurf die technische Umsetzung des Systems geplant wird und daher alle technischen Faktoren wie Zielumgebung oder Systemarchitektur eine große Rolle spielen, werden diese in der Analyse außer Acht gelassen.

XP:
Anforderungen werden mit User-Stories beschrieben. Diese werden vom Kunden verfasst und dienen auch als Grundlage für die Aufwandsabschätzung.
Der Kunde muss das ganze Projekt über vor Ort sein.
Die Metapher ist, dass der Kunde und das Entwicklerteam

- eine gemeinsame Sprache im Bezug auf das System sprechen
- eine gemeinsame Vision/Sicht auf das Problem haben
- Generativität: Projektmitarbeiter und Kunde müssen sich gegenseitig umeinander "kümmern".
- Architektur ?

Sind folgende Entwicklungsmodelle traditionell oder agil?

Entwicklungsmodell	Agil	Traditionell
V-Modell 97		x
V-Modell XT	x	x
Personal Software Process (PSP)		x
Team Software Process (TSP)		x
Crystal Clear	x	
Rational Unified Process (RUP)		x
Scrum	x	
eXtreme Programming (XP)	x	
Object-Oriented Software Process (OOSP)		x

Nennen und erklären Sie eine Methode zur Aufwandsabschätzung.

Delphi-Methode: siehe S. 36 in den Folien Projektmanagement

- Moderator & mehrere Experten
- Verfahren:
  - Zu jedem Arbeitspaket gibt jeder Experte einen Schätzwert ab
  - Mittelwert, falls Schätzwerte in einer bestimmten Bandbreite
  - Sonst: Argumente austauschen + neue Schätzung
- Zuschläge zu den Einschätzungen: Plus 10-15% = Aufwand für PM

3 Experten Konzept:

- 3 optimistische, 3 realistische, 3 pessimistische Schätzungen
- Abklärungen durchführen bis Ergebnisse in jeder Kategorie übereinstimmen
- Daraus Schätzwert gemeinsam ableiten

Claim Management erklären. Folien PM2

Bedeutung:
zu Projektbeginn: Vertrag zwischen Auftragnehmer und - geber.
Während des Projekts: Zusatzwünsche des Auftraggebers, Modifikation aufgrund neuer technischer Erkenntnisse, Änderungen aufgrund von Fehlern bei der Projektdefinition.
Frage: Wer trägt die Mehrkosten?

Claim Management: Claim = Forderung eines Vertragspartners infolge von Abweichungen vom Vertrag.
Alle Maßnahmen zur:
- aktiven und frühzeitigen Erkennung von Claim - Situationen
- optimalen Durchsetzung von Claims gegenüber anderen Unternehmen
- Abwehr und Verhütung der an das eigene Unternehmen gerichtete Claims

Claim Vorsorge:
Verhütung eines Anspruches vor seiner Entstehung
Strategisch besonders bedeutend
Günstige Ausgangssituation durch Prävention

Screenshot Folie 11

ProjektMatrix = Matrix Projektorganisation (Folie 21ff, PM1)

Projektleiter trägt Gesamtverantwortung und hat nicht volle Weisungsbefugnis
Vorteile: Projektverantwortung durch Projektleitung, Sicherheitsgefühl der Mitarbeiter, Flexibler Personaleinsatz.
Nachteile: Mitarbeiter ist Diener zweier Herren, Konfliktpotential, Hoher Koordinationsaufwand.

gehört zu: Projektorganisation.
gibt 3 verschiedene:

Reine Projektorganisation

Projektleiter trägt Gesamtverantwortung / Volle Weisungsbefugnis
Vorteil: Verantwortung eindeutig geregelt, Starke Identifikation der Mitarbeiter mit dem Projekt, Hohe Motivation.
Nachteil: Schwierige Akquirierung der Projektmitarbeiter, Wiedereingliederung der Mitarbeiter nach Ende des Projekts problematisch

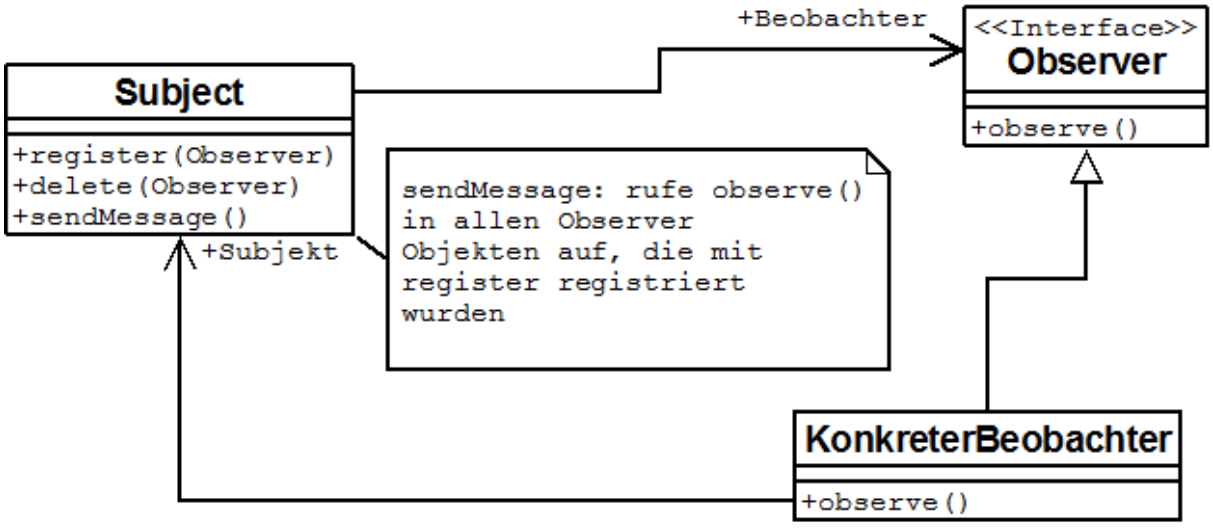
Einfluss Projektorganisation

Projektleiter ist nur Projektkoordinator, kaum Weisungsbefugnis
Vorteil: Geringe organisatorische Veränderung, Hohe personelle Flexibilität, Kommunikation mit Fachabteilung bleibt erhalten.
Nachteile: Keine klare Verantwortlichkeit, Fehlende Autorität des Projektleiter, Hohes Konfliktpotential, Fehlender Teamgeist.

Matrix Projektorganisation

siehe oben

Observerpattern erklären



Ein Subjekt wird von mehreren Beobachtern beobachtet. Alle Beobachter implementieren eine Schnittstelle (Observer), über welche das Subjekt die Beobachter über Zustandsänderungen in Subjekt benachrichtigen kann. Die Schnittstelle von Subjekt bietet Mechanismen an, über die sich die Beobachter für das Beobachten des Subjekts an und abmelden können.

Beispiel: In Java ist ein JButton ein Subjekt, und die Listener die Beobachter des JButtons.

Was versteht man unter "Round Trip Engineering" beim MDD (Model Driven Development)?

Es kann nicht nur am Modell etwas geändert werden, sondern auch am Code. Diese Änderungen werden dann automatisch ins Modell transformiert. Man kann allerdings nicht die Abstraktionsebene wechseln, nur die Darstellung.

Was ist der Unterschied zwischen Verifikation und Validierung von Anforderungen?

Validierung: Überprüfen, ob die Anforderungen wirklich abbilden, was der Benutzer / Auftraggeber benötigt.
Anforderungs-Validierungsfehler sind die teuersten im SE Prozess: Ausbessern eines Validierungsfehlers nach Auslieferung kostet ca. 100x mehr als eine Fehlerausbesserung in der Entwicklung.

SE-Anforderungen, 33ff

Was versteht man unter objektorientierter Kopplung und objektorientierter Kohäsion?

- Kopplung: Ist der Grad der Abhängigkeit zwischen den einzelnen Klassen. Geringe Kopplung erhöht die Wartbarkeit des Systems. Es wird unterschieden zwischen
    - Interaktionskopplung: Gibt Maß an Kopplung auf Grund von Nachrichten zwischen Objekten an.
    - Vererbungskopplung: Beschreibt das Maß der Abhängigkeit zwischen erbender und vererbender Klasse.
- Kohäsion: Ist der Grad der Abhängigkeit innerhalb einer Klasse. Hohe Kohäsion begünstigt geringe Kopplung und Verständlichkeit. Man unterscheidet
    - Klassenkohäsion
    - Servicekohäsion
    - Generalisierung/Spezialisierung-Kohäsion

Was ist ein thin client / fat client. Beispiele und Anwendungen.

3 Schichten:
Benutzerschnittstelle
Business Logic (Ganze logik, die implementiert wird)
Datenspeicherung (zb datenbank)

thin client: Server kümmert sich um business logic, client hat nur benutzerschittstelle (Bankomat). Wird verwendet, wenn alte Systeme auf einen Server ausgelagert werden.
Nachteil: Belastung für Netzwerk und Server

fat client: business logic und Benutzerschnittstelle beim client implementiert. Server verwaltet nur daten. Beispiel: C/S Systeme, bei denen Leistungsdaten des Clients vorher schon bekannt sind. Komplexer als thin client. neue versionen der applikationen müssen auf allen clients installiert werden.

Qualitätsanforderungen bei Extreme Programming.

Test Driven Development:
1. Write test
2. Write code
3. Run automated tests
4. Refactor
5. Repeat.

Test Driven Development:
Zuerst Test, dann wird code dazu geschrieben.

Was ist ein hybrides SE Prozessmodell?
Nennen Sie Beispiele und erklären Sie die aktuelle Bedeutung dieser Modelle.

Kombination von verschiedenen Aspekten aus Modellen um eine neue Charakteristik für ein Prozessmodell zu schaffen.

Beispiel: V-Modell mit agiler Entwicklung

Modelle werden als Model Frameworks beschrieben. Anpassung und Implementierung mit starker Ausrichtung auf unterschiedliche Einsatzzwecke. Anpassung erfolgt über entsprechend mitgelieferte Werkzeuge.

Softwarewartung im Unified Process. Welche Rollen gibt es, welche Produkte? Wieso wird zwischen Fehlerbehebung und Erweiterung entschieden?

Rollen: Architekt, Dokumentierer, Wartungsteam
Produkte: Auslieferungsplan, Abnahmeprotokoll, Integrations- Migrationsplan, Anwenderdokumentation, Fehlerbericht, Änderungswunsch

Fehlerbehebung ist wichtiger als Erweiterung, da ein funktionierendes Programm wichtiger ist als eines das viele Funktionen hat die nicht gut funktionieren.

Rational Unified Process (RUP) Erläutern Sie die grundlegenden Konzepte des Rational Unified Process (RUP). Aus welchen Phasen besteht der RUP? Welches sind die wesentlichen Vorteilevon RUP? Wann wird es typischerweise eingesetzt?

Phasen:

Inception phase

Hier werden eine Projektbeschreibung, die Erfolgsfaktoren des Projektes sowie eine Kosten- und Risikoabschätzung erstellt. Weiters wird ein grundsätzliches Use Case- Modell und ein Basis-Projektplan festgelegt. Auch Qualitätsstandards für dieses Projekt werden hier definiert. Grob zusammengefasst: hier geht es um die Ziele des Projektes.

Elaboration phase

Diese Phase entspricht in etwa der klassischen Design-Phase: Hier wird die grundsätzliche Software-Architektur festgelegt und die Problem-Domain-Analyse durchgeführt (was soll konkret umgesetzt werden? was existiert bereits?). Grob zusammengefasst: hier geht es um die Architektur.

Construction phase

Dies ist die eigentliche Entwicklung und entspricht der klassischen Implementierungs-Phase. Es entsteht eine erste Version der Software, die auch nach außen releasedwerden kann.

Transition phase

Hier findet die Einschulung der End-User, das Beta-Testing und die abschließende Qualitätskontrolle statt.

Vorteile:

Es gibt Tool-Unterstützung

Nachteile:

RUP ist komplizierter als viele andere Modelle,
Dokumentationslastig
Proprietär

Einsatzgebiet:

Wegen dem relativ komplexen Aufbau wird RUP vor allem bei großen Projekten eingesetzt.

Software Implementierung und Integration Was versteht man unter der Integrationsphase im Softwareentwicklungsprozess? Welche Integrationsstrategien kennen Sie? Wie unterscheiden sie sich? Welche Vorteile / Nachteile haben diese unterschiedlichen Strategien?

Integration meint das Zusammenfügen einzeln entwickelter Komponenten zu einem Teil oder Gesamtsystem. Es gibt dafür verschiedene Strategien mit verschiedenen Vor- und Nachteilen.

Big-Bang-Integration
Damit ist die gleichzeitige Zusammenfügung aller Komponenten gemeint. Sie lässt sich nur bei sehr kleinen Projekten anwenden. Vorteil ist, dass keine Stubs oder Driver (siehe unten) geschrieben werden müssen und dass keine Regressionstests notwendig sind. Nachteil ist aber, dass auftretende Fehler kaum lokalisierbar sind.

Top-Down-Integration
Die Komponenten werden von höher liegenden Schichten der Software-Architektur ausgehend nach unten integriert.
Beispielsweise wird zuerst das GUI mit der Business Logic zusammengefügt, später kommt unten noch der Data-Access-Layer hinzu. Vorteil ist, dass die höher liegenden Schichten - das sind die f¨ur den Benutzer sichtbareren - früh verfügbar sind, was sich für Demo-Zwecke gut eignet. Allerdings sind Stubs zu schreiben (das sind Softwaremodule, die die noch nicht integrierten unteren Schichten simulieren).
Außerdem werden hardwarenahe Teile (das sind üblicherweise die fehleranfälligeren) erst spät integriert werden.

Bottom-Up-Integration
Hier arbeitet man sich ausgehend von den hardwarenahen Schichten nach oben vor. Vorteil ist die frühe Integration der risikobehafteteren Teile (wodurch man später auf ein stabiles Framework aufbauen kann), Nachteil ist dass man Driver (Softwaremodule, oben liegende Schichten simulieren, z.B. GUI-Interaktionen durch ein Script simulieren) schreiben muss. Der Kunde sieht lange keinen Fortschritt, weil die sichtbaren Teile erst spät integriert werden. Es müssen daher zusätzlich Prototypen erstellt werden.

Build-Integration
Darunter versteht man die gleichzeitige Integration mehrere Komponenten auf verschiedenen Ebene der Architektur, die gemeinsam einen Use Case umsetzen.
Beispielsweise könnte GUI, Business Logic und Data-Access-Layer, die für den Abruf von Kontoauszügen benötigt werden, gemeinsam integriert werden. Andere Geschäftsfälle sind dann aber noch nicht abgedeckt. Vorteil ist, dass schnell ein Use Case vollständig durchgeführt werden kann (gut für Präsentationen). Allerdings bauen oft verschiedene Use Cases auf gleichen Modulen auf, wodurch eventuell später ein Fehler in einem Modul erkannt wird, das vorher in Verbindung mit einem anderen Use Case verwendet wurde und dort funktioniert hat. Es sind in diesem Fall Änderungen und Regressionstests notwendig.