SEPM SS13 – Fragenkatalog ausgearbeitet

Vorwort: Ich habe mir die Mühe gemacht, den bestehenden Fragenkatalog von den vorigen Test auszuarbeiten. Ich kann nur jedem empfehlen es min. 1 selbst zu schreiben, da dies ungemein beim Lernen hilft.

Was mir während der Ausarbeitung aufgefallen ist, wäre das es im SS13 neue Themen gab, welche ich hier in keiner Fragenstellung fand und diese zu der Zeit der Ausarbeitung auch Folien mäßig nicht zur Verfügung standen.

Dies wäre als einziger das Thema „Persistenzstrategien“ welches im Folienblock 9 enthalten ist. „Block 09 - Persistenzstrategien.pdf“ Es kam zum Test und umfasste 8 Pkt. Alle anderen theoretischen Fragen waren bei meinem Test hier abgedeckt. Für den theoretischen Teil bis du hiermit gut Vorbereitet.

Quellen:

Folien der LVA SS13 – siehe Tuwel
Wikipedia: www.wikipedia.org
Buch: Best Practice Software Engineering. Eine praxiserprobte Zusammenstellung von komponentenorientierten Konzepten, Methoden und Werkzeugen; Spektrum Akademischer Verlag, 2010, 978-3827424860
Fragenkatalog: Fragenkatalog - Stand- WS 2013.pdf - https://vowi.fsinf.at/wiki/Spezial:Materialien/TU_Wien:Software_Engineering_und_Projektmanagement_VO_(Biffl)

Inhalt

1. Software Wartung (!!!) 5
1.1. Vervollständigen Sie die Tabelle um die Wartungskategorien, die in der Vorlesung besprochen wurden und geben Sie jeweils ein konkretes Beispiel an. 5
1.2. Welche Phasen umfasst der Wartungsprozess? 5
1.3. Nennen und beschreiben Sie zwei Techniken zur Wartung von Software. 6
2. Software Prozesse: Agile Softwareentwicklung (!!!) 6
2.1. Erläutern Sie anhand einer Skizze das Grundkonzept von SCRUM Sprints. 6
2.2. Erklären Sie den Unterschied zwischen dem Produkt-Backlog und Sprint-Backlog. 7
2.3. Was versteht man unter einem Burndown-Chart? 7
3. Traceability (!!!) 7
3.1. Was verstehen Sie unter Traceability im Zusammenhang mit Softwareprojekten? 7
3.2. Beschreiben Sie kurz die unterschiedlichen Arten von Traceability und welche Vorteile sich in einem Softwareprojekt daraus ergeben. 7
3.3. Wie kann Traceability in einem Softwareprojekt umgesetzt werden? 8
4. Qualitätssicherung und Testen (!!!) 8
4.1. Diskutieren Sie die Begriffe Verifikation und Validierung. 8
4.2. Erläutern Sie die folgenden Test-Design-Techniken: Äquivalenzklassenzerlegung und Grenzwertanalyse. 8
4.3. Aus welchen Komponenten besteht eine Testfalldokumentation? Beschreiben Sie die notwendigen Komponenten und geben Sie ein konkretes Beispiel an. Hinweis: Verwenden Sie eine tabellarische Darstellung zur besseren Strukturierung der Testfallbeschreibung. 9
4.4. Skizzieren sie Black Box- und White Box-Testing 10
4.5. Welche Testarten gibt es? 10
4.6. Welche Qualitätskriterien gibt es in der Softwareentwicklung? Beschreibungen Sie diese. 10
5. Test-Driven Development (!!!) 11
5.1. Was versteht man unter Test-Driven Development (TDD)? Beschreiben und skizzieren Sie die grundlegende Idee von TDD. 11
5.2. Welche Vorteile ergeben sich aus der Integration von TDD im Rahmen der Continuous Integration? 11
6. Software Reviews (!!!) 12
6.1. Was sind Software Reviews und wozu werden sie typischerweise eingesetzt? 12
6.2. Welche Arten von Reviews kennen Sie? Erläutern Sie die wesentlichen Aspekte der einzelnen Reviewarten und geben Sie jeweils ein konkretes Beispiel an. 12
6.3. Welche Rollen finden sich typischerweise bei Reviews und welche Aufgaben nehmen die unterschiedlichen Rollen wahr. 13
7. Integration (!!!) 14
7.1. Erläutern Sie den Begriff der Systemintegration. 14

7.2. Skizzieren und erläutern Sie die vorgestellten Integrationsstrategien und welche Vor- bzw. Nachteile damit verbunden sind 14
7.3. Was ist bei der Systemintegration im Hinblick auf die Qualitätssicherung zu beachten? 15
7.4. Welche Integrationsarten existieren bzw. was ist Integration überhaupt? 15
8. Architektur (!!!) 16
8.1. Beschreiben Sie das Data Access Object Pattern (DAO) anhand eines UML-Diagramms und erläutern Sie dessen Funktionsweise. Wann kommt dieses Pattern typischerweise zum Einsatz? 16
8.2. Beschreiben Sie das Factory Pattern. Wann wird es eingesetzt? Geben Sie ein einfaches Beispiel in UML an. 16
8.3. Welche 3 Typen von Pattern gibt es? Nennen sie zu jedem mindestens 2 Beispiele. 17
9. Projektmanagement – Strukturpläne 17
9.1. Was verstehen Sie unter einem Projektstrukturplan? (!!!) 17
9.2. Aus welchen Ebenen besteht ein Projektstrukturplan? (!!!) 17
9.3. Skizzieren und erläutern Sie einen (i) funktions- bzw. aufgabenorientierten Projektstrukturplan und (ii) einen objektorientierten Projektstrukturplan. Worin liegen die wesentlichen Unterschiede und wofür werden sie eingesetzt? 18
10. Projektmanagement 18
10.1. Was ist ein Projekt und durch welche Merkmale ist ein Projekt gekennzeichnet? (!!!) 18
10.2. Was versteht man unter einem Projekterfolg? 19
10.3. Durch welche Rahmenbedingungen wird der Projekterfolg gefördert? Geben Sie für jede Rahmenbedingung zumindest ein konkretes Beispiel an. 19
10.4. Was versteht man unter dem Projektauftrag? (!!!) 19
10.5. Was sind die Aufgaben eines Teamkoordinators? 19
10.6. Was ist Projektmanagement? 19
10.7. Projektmanagement – Erfolgsfaktoren 19
11. Anpassung von Software-Prozessen (!!!) 20
11.1. Erklären Sie die wesentlichen Konzepte des „Process Customizing“ und des „Prozess Tailoring“. Wozu und wann werden sie eingesetzt? 20
12. Requirements (!!!) 20
12.1. Welche Rolle spielen Requirements im SE Life-Cycle? 20
12.2. Wie können sie klassifiziert werden? 20
13. Design-Prinzipien (!!!) 21
13.1. Beschreiben Sie das Design-Prinzip "System Control". 21
13.2. Stair vs. Fork (+ Beispiele) 21
13.3. 4+1 View Model Architecture 21
14. Vorgehensmodelle 23
14.1. V-Modell Prinzip + Vorteile + welche Zweige/Ebenen gibt es (!!!) 23
14.2. V-Modell XT + Tailoring + Einsatzgebiet 23
14.3. (RUP Prinzip + Vorteile + Phasen + Einsatzgebiet) 24

15. Sonstiges 24
15.1. Was ist Continuous Integration? 24
15.2. 4 Kernformen der Arbeitsstruktur 24
15.3. Wie sieht eine moderene 3-Tier-Architektur aus? Beschreiben Sie die folgenden Fälle: a) Thin-Client b) Fat-Client. Beschreiben und skizzieren Sie die beiden Architekturvarianten. 25

1. Software Wartung (!!!)

1.1. Vervollständigen Sie die Tabelle um die Wartungskategorien, die in der Vorlesung besprochen wurden und geben Sie jeweils ein konkretes Beispiel an.

	Correction	Enhancement
Proactive	Preventive	Perfective
Reactive	Corrective	Adaptive

Korrektive Wartung (Corrective)

Beispiel: Bug – Fehlerkorretkur (patches, workarounds, updates)

Adaptive Wartung (Adaptive)

Beispie: Berücksichtigung geänderter externer Anforderungen (Hardware, Softwareänderungen)

Produktpflege und Verbesserung (Perfective)

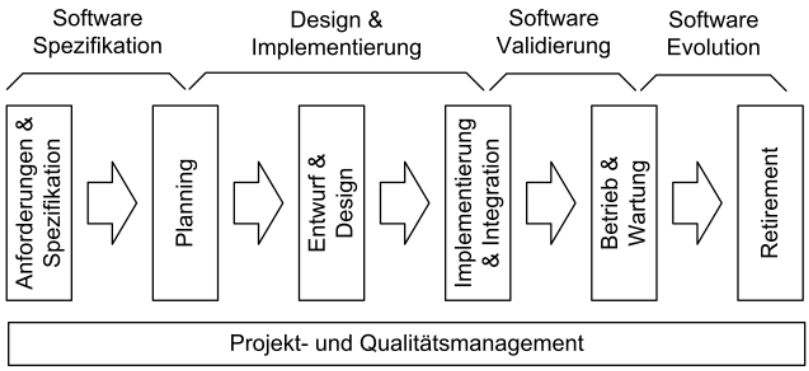
Beispiel: Produktverbesserung (Erweiterung, Verbesserung der Effizienz)

Vorbeugende Wartung (Preventive)

Beispiel: Verbesserung im Hinblick auf zukünftige Wartung (z.B. Ergänzung der Dokumentation)

1.2. Welche Phasen umfasst der Wartungsprozess?

Wartungsphasen beinhalten dieselben Phasen wie der Life-Cycle Prozess; einen speziellen Stellenwert nehmen die Anforderungsänderungen ein.



Anforderungen (Requirements) zeigen die Wünsche des Kunden in Bezug auf das Softwareprodukt (user/custumerview).
-> Anforderungen müssen Testbarsein und getestet werden!

Eine Spezifikation beschreibt das System aus technischer Sicht (engineering view).

Planning: Erstellen des Projektplans bezüglich der Zeit, Dauer und Kosten (project management).

Entwurf & Design: technische Lösung der Systemanforderung (Komponenten, Packages, Datendesign).

Implementierung und Testen: Erzeugen des Softwareproduktes.

Integration und Testen: Zusammenfügen und Tests der einzelnen Komponenten auf Architektur & Systemebene.

Operation and Maintenacne: Fehlerbehebung, Unterstützung, Erweiterung des Softwareproduktes während des laufenden Betriebes.

Retirement: Nach der Einsatzphase, d.h. am Ende des Produktionszyklus, muss das Softwareprodukt kontrolliert aus dem Betrieb genommen werden.

1.3. Nennen und beschreiben Sie zwei Techniken zur Wartung von Software.

Herstellen des Produktverständnisses:

- Das Verständnis „fremder“ Codestücke kann – speziell bei mangeldem Aufzeichnen – sehr Zeitaufwändig sein.
- Key Tools: Code Browsers.
- Produktanforderungen: Klare präzise Dokumentation.

Reengenieering

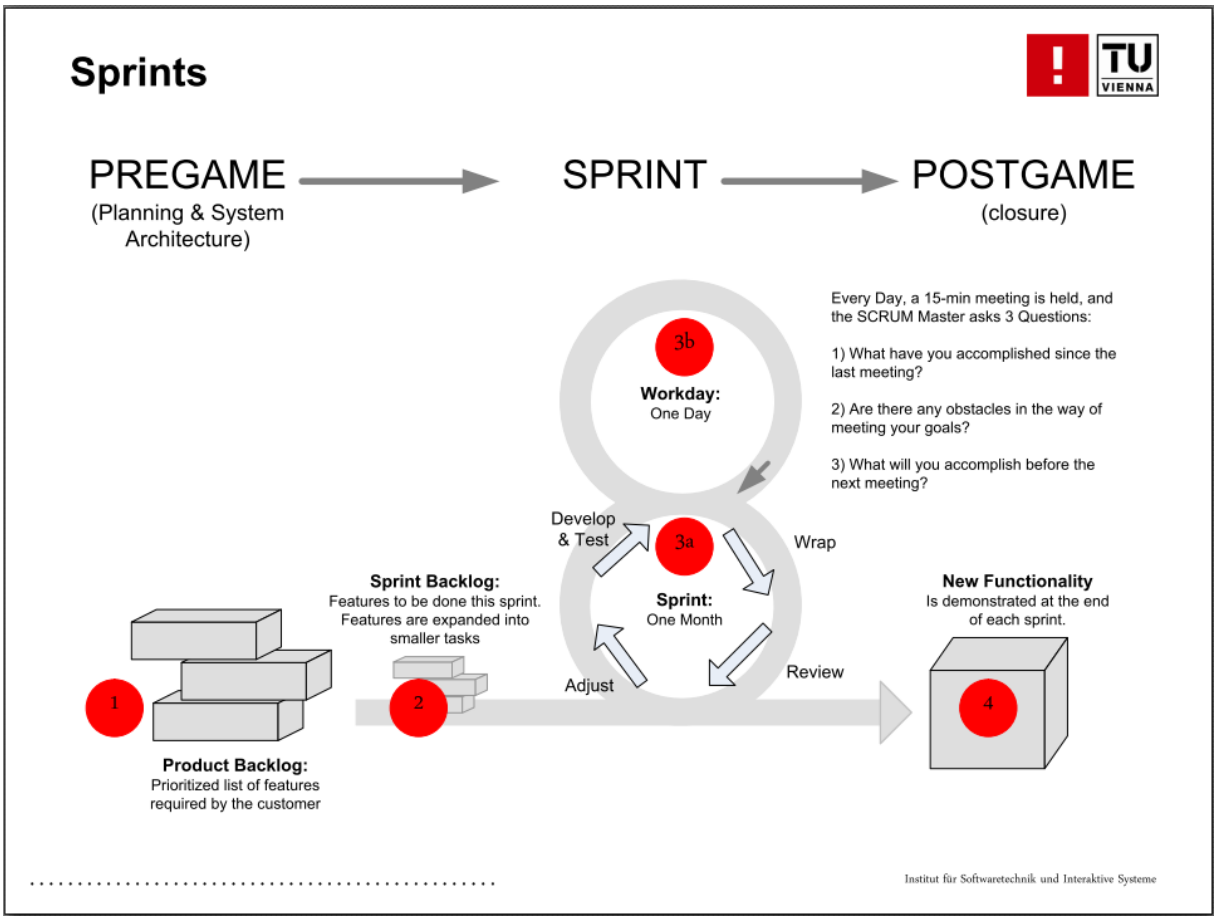
- Überprüfen und Überarbeiten des Software Codes.
- Gravierende und teurere Form der Änderung.

Reverse Engineering

- Analyse der Software im Hinblick auf Komponenten und deren Zusammenhänge.
- Erstellen von Modellen (basierend auf dem Code) auf einem höheren Abstaktionsniveau.
- Z.B. erstellen von UML-Modellen aus C#-Code

2. Software Prozesse: Agile Softwareentwicklung (!!!)

2.1. Erläutern Sie anhand einer Skizze das Grundkonzept von SCRUM Sprints.



Characteristika

- Ein Team erstellt eine Einheit, ein Produkt oder Teilprodukt.
- Klare Arbeitsaufteilung innerhalb des Teams.
- Klar priorisierte Projektergebnisse (Backlog Items)
- Ein gemeinsames Ziel (Erstellung des Produktes)
- Der „Sprint“ ist das zentrale Element
- Das Sprint-Team kann ungestört arbeiten; es sind keine Eingriffe „von aussen“ (z.B. durch den Kunden) zulässig
- Temporal Structure = daily Scrum Meeting + Review + Retrospecitve

Begriffe

- Backlog beinhaltet alle Arbeitspakete, die in “nächster“ Zeit umgesetzt werden müssen (sowohl klar definierte als auch vage Anforderungen): Produkt vs. Sprint Backlog.
- Ein Daily-Scrum ist eine alltägliche Projektbesprechung um etwaige Fragen / Missverständnisse zu klären; Definition des täglichen Arbeitsauftrags.
- Scrum-Team: Funktionsübergreifendes Team, zuständig für den Sprint Backlog.
- Burndown Chart: Visuelle Darstellung des Projektfortschrittes.

2.2. Erklären Sie den Unterschied zwischen dem Produkt-Backlog und Sprint-Backlog.

Produkt-Backlog: Beinhaltet eine Benutzer priorisierte Liste alle Features zum Produkt.

Srpint-Baklog: Beinhaltet alle Feature welche zu einem gewissen Sprint gehöhren. Diese Features werden in kleinere Tasks aufgeteilt.

2.3. Was versteht man unter einem Burndown-Chart?

Burndown Chart: Visuelle Darstellung des Projektfortschrittes. Z.B. Gantt-Diagramm im Redmine

3. Traceability (!!!)

3.1. Was verstehen Sie unter Traceability im Zusammenhang mit Softwareprojekten?

„Traceability ist die Nach- oder Rückverfolgbarkeit einer Information durch ihren gesamten Entwicklungszyklus (z.B. bei sicherheitskritischen Anwendungen gefordert).“

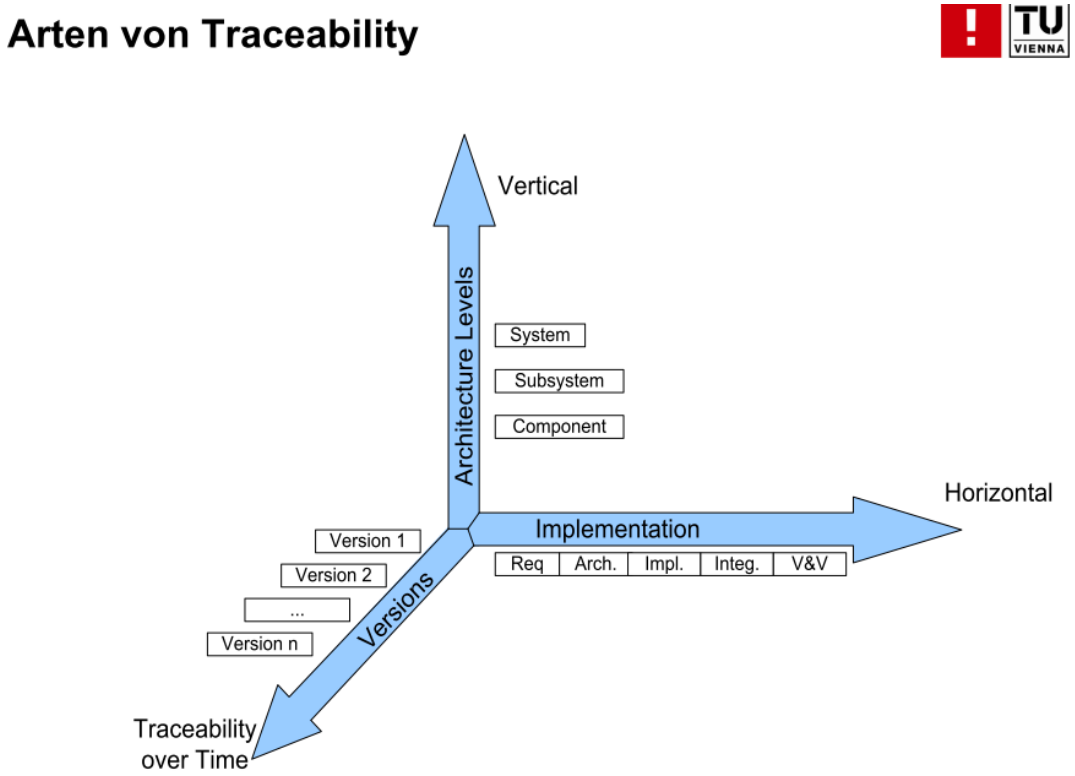
→ Änderungsverfolgung = die Fähigkeit, den Lebenszyklus einer Anforderung vom Ursprung der Anforderung über die verschiedenen Verfeinerungs- und Spezifikationsschritte bis hin zur Berücksichtigung der Anforderung in nachgelagerten Entwicklungsartefakten verfolgen zu können.

3.2. Beschreiben Sie kurz die unterschiedlichen Arten von Traceability und welche Vorteile sich in einem Softwareprojekt daraus ergeben.

Zeitliche Traceability: Zeitliche Nachvollziehbarkeit unterschiedlicher Releases, z.B. durch Konfigurationsmanagement

Vertical Traceability: Beziehungen innerhalb einer Phase und eines Artefaktyps, z.B. System – SubSystem – Komponente

Horizontale Traceability: Beziehungen zwischen unterschiedlichen Entwicklungsartefakten, z.B. Anforderungen – Implementierung – Testfälle.



3.3. Wie kann Traceability in einem Softwareprojekt umgesetzt werden?

Durch typische Fragestellungen:

- Woher kommt eine Anforderung und wo wurde sie umgesetzt?
- Welche Artefakte sind von einer Änderung der Anforderung betroffen?
- Welche Anforderungen sind von einer Änderung der Umsetzung betroffen?

Daraus ergeben sich folgende Vorteile:

- Nachvollziehbarkeit der Information bei Änderung
- (Automatische) Benachrichtigung bei Änderung

Informationen aus dem Headerblock können zur (automatisierten) Realisierung für Requirements Tracing eingesetzt werden (z.B. in IDE´s)

4. Qualitätssicherung und Testen (!!!)

4.1. Diskutieren Sie die Begriffe Verifikation und Validierung.

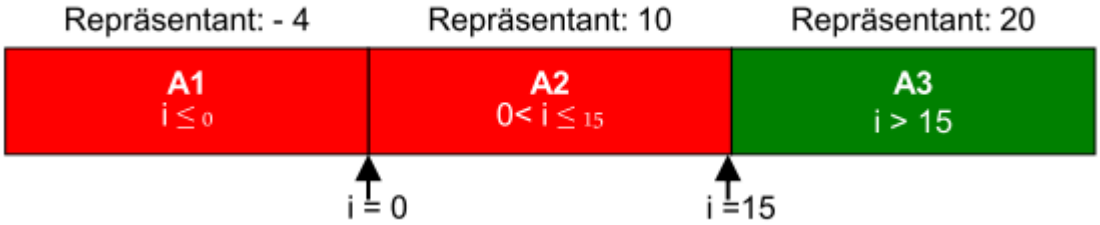
Validierung = „Wurde das richtige Produkt entwickelt?“, z.B. Akzeptanz- und Abnahmetests (Prüfung gegen Anforderungen)
Im Rahmen einer Validierung wird die Frage beantwortet, ob das richtige Produkt entwickelt wurde (also Kunde auch das bekommt, was er bekommen wollte)

Verifikation = „Wurde das Produkt richtig entwickelt?“, z.B. Komponententests (Prüfung gegen Spezifikation)
Unter Verifikation versteht man die Prüfung eines Produktes im Hinblick auf die Spezifikation. „Wurde das Produkt richtig entwickelt“.

4.2. Erläutern Sie die folgenden Test-Design-Techniken: Äquivalenzklassenzerlegung und Grenzwertanalyse.

Äquivalenzklassen

- Gleiches Verhalten aus einer Menge von Eingabedaten mit demselben Ergebnis -> Auswahl eines repräsentativen Wertes für Reduktion der Testfälle.
- Anforderungen: i > 15 -> 3 Äquivalenzklassen



- Testfälle müssen gültige (Normalfall, Sonderfall) UND ungültige Eingabewerte (Fehlerfall) berücksichtigen

Grenzwertanalyse

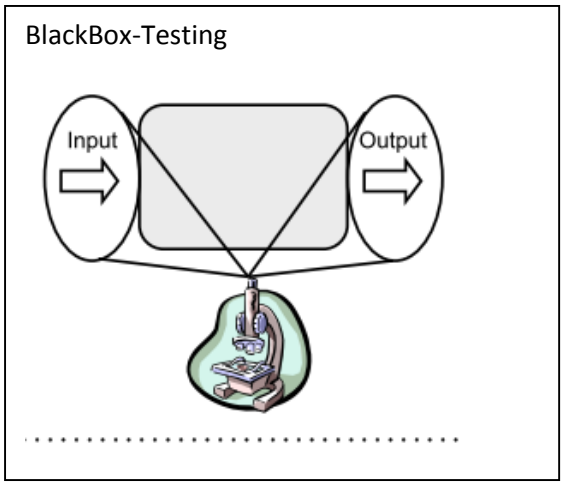
- Spezialfall einer Äquivalenzklasse
- Geeignete Auswahl der Repräsentanten in der Nähe der Datengrenze

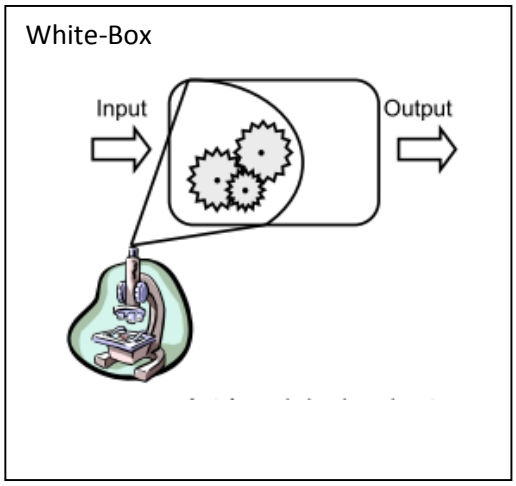
4.3. Aus welchen Komponenten besteht eine Testfalldokumentation? Beschreiben Sie die notwendigen Komponenten und geben Sie ein konkretes Beispiel an. Hinweis: Verwenden Sie eine tabellarische Darstellung zur besseren Strukturierung der Testfallbeschreibung.

Name	Beschreibung	Beispiel
Beschreibung	Ist die Komponente im System und eine Funktion (Ablauf aus Anforderungen)	Durch drücken auf die Taste Play/Pause wird das Musikstück abgespielt oder pausiert.
Vorbedingung	Existieren oder eben nicht	Das Programm muss geöffnet und eine Musik muss vorhanden sein.
Eingabewerte	Sind Konkrete Parameterbeschreibungen	
Aktion	Sind Aktivitäten zur Eingabe der Werte.	Drücken auf die Taste Play/Pause.
Erwartete Ergebnisse	Sind Zustände und Ausgangsparameter	Musik wird abgespielt oder gestoppt.
Tatsächliche Ergebnisse	Ergebnisse zeigen Abweichungen zum tatsächlichen Ergebnis.	Die Musik reagiert nicht auf die Aktion.
OK / NOK	Zeigt ob das Testergebnis erfolgreich durchgeführt wurde.	Vergleichen das erwartete Ergebnis zum resultierenden.

No	Type*	Pre-Condition	Test Case Description	Equivalence Classes	Expected Results	Actual Results
1	FF	i=-1	Invalid number, drop error message	AA	Drop Error message, no further action possible.	Error message
2	SF	i=15	Invalid number → drop error message	AB	Drop Error message, no further action possible.	Error message
3	NF	i=20	Valid number → "do something"	AC	Something Done	Error message
..						

4.4. Skizzieren sie Black Box- und White Box-Testing





4.5. Welche Testarten gibt es?

Unit-Tests
Fokus auf Komponenten und Prüfung auf Übereinstimmung zwischen der Umsetzung der Komponente und der technischer Spezifikation.

Integrationstest
Focus auf Interfaces und Verbindung zwischen Komponenten innerhalb des (Sub)Systems.

Systemtest
Fokus auf Übereinstimmung zwischen funktionalen und technischen Bedingungen des Gesamtsystems (nahe an der Zielplatform).

Regressionstest
Testen von geänderten Komponenten. Dadurch sollen Fehler, die durch Änderungen, z.B. auch durch Fehlerkorrektur, entstanden sind, verhindert werden.

Akzeptanztest
Übereinstimmung der festgelegten Anforderungen in der Zielumgebung des Kunden.

Installationstest
Identifizieren von Fehlern während der Installation.

4.6. Welche Qualitätskriterien gibt es in der Softwareentwicklung? Beschreibungen Sie diese.

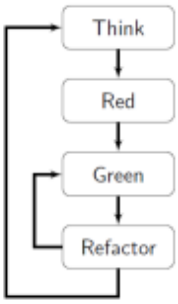
Welche sonstigen Anforderungen müssen umgesetzt werden, die nicht direkt auf die Funktionalität abzielen, sie aber beeinflussen.

- Leistung und Performance (z.B. Optimierung des Informationsflusses)
- Usability und menschliche Faktoren (z.B. Einfachheit der Bedienung, Training)
- Sicherheit (z.B. Zugangskontrolle)
- Wartbarkeit (Trennung von Anwendung und Daten)
- Erweiterbarkeit (z.B. wie Aufwändig ist es ein zusätzliches Modul dem Projekt hinzuzufügen)

5. Test-Driven Development (!!!)

5.1. Was versteht man unter Test-Driven Development (TDD)? Beschreiben und skizzieren Sie die grundlegende Idee von TDD.

Prozessablauf:



1. **Think**: Spezifikation des Tests.
2. **Red**: Implementierung des Tests (Test schlägt fehl!).
3. **Green**: Implementierung der zu testenden Klasse. (Test ist erfolgreich!)
4. **Refactor**: Veränderung einer Implementation → Funktionalität bleibt erhalten.

Zusammenfassung:

- TDD als Basis für zielorientierte Software-Entwicklung (Implementierung und QS).
- Brauchbare Modelle unterstützen das Herstellen effektiver und effizienter Testfälle:
  - Datenmodellierung (Datenbank)
  - Datenfluss und Kontrollfluss (Business Logic)
  - Zustandsübergangsmodelle (z.b. GUI)
- Reviews helfen die Anforderungen und Modelle vorab zu überprüfen, um eine solide Basis für Testfälle herzustellen
- Die Zeit für Unit Tests ist wohl investiert, da die Integrationstests wesentlich vereinfachen (Bottom-Up testing)
- Test Cases sind „living documents“ – unterstützen die Kommunikation innerhalb des Teams.

5.2. Welche Vorteile ergeben sich aus der Integration von TDD im Rahmen der Continuous Integration?

Continious Integration

- Vorteil einer Build-Automatisierung
  - Projekt schnell mit Archetypen aufsetzen
  - Projekt ist portabel
  - Abhängigkeiten leichter darstell- und auflösbar
  - Verbesserung der Projetk-Qualität durch Integration automatisierter Tests
  - Automatisiertes Reporting
- Einsatz im Continous Integraion Kontext
- Best practise Aspekte
  - Ein Sourcode-Repository
  - Automatisierter Build
  - Automatisierte Testabläufe
  - Tägliche Commits der Entwickler
  - Schneller Build
  - Build auf einem neutralen Rechner
    - Daily builds, snapshot
  - Automatisiertes Deployment
  - Transparenz der Entwicklungsprozesses

6. Software Reviews (!!!)

6.1. Was sind Software Reviews und wozu werden sie typischerweise eingesetzt?

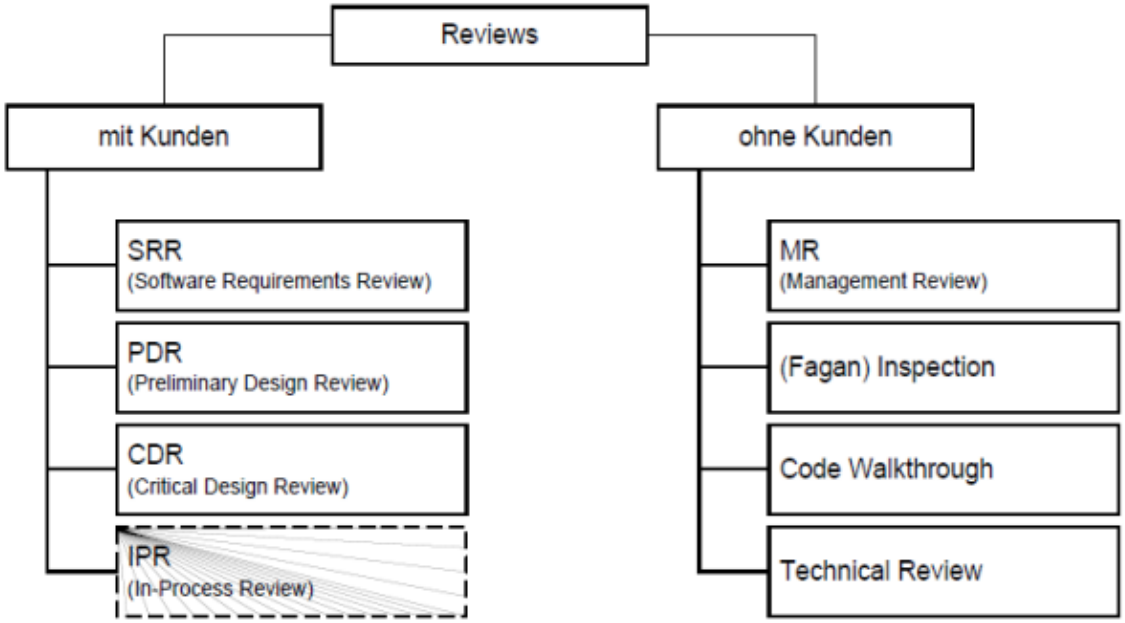
Begriffsdefinition „Review“

„Ein Review ist ein [..] formal geplanter und strukturierter Analyse- und Bewertungsprozess, in dem Projektergebnisse einem Team von Gutachtern präsentiert und von diesen kommentiert oder genehmigt werden.“

- Dienen also vor allem zur qualitativen Beurteilung von Produkten und Prozessen, die quantitativ nur schwer oder gar nicht beurteilt werden können (z.B. Modelle, Dokumente)
- Unterschiedliche Ausprägungen von Reviews zielen auf unterschiedliche Ziele ab.

6.2. Welche Arten von Reviews kennen Sie? Erläutern Sie die wesentlichen Aspekte der einzelnen Reviewarten und geben Sie jeweils ein konkretes Beispiel an.

Arten von Reviews



SRR

- Ein Software Requirements Review (SRR) verfolgt das Ziel, nach der Definition der Anforderungen und vor dem Entwurf ein Review durchzuführen, um alle Unklarheiten sowohl seitens des Kunden als auch seitens des Entwicklerteams auszuräumen. Dementsprechend soll der Kunde an diesem Review teilnehmen. Nach dem SRR müssen die Ziele des Projekts und die Anforderungen allen beteiligten Personen bekannt sein. Im Anschluss daran startet meist die Entwurfs- und Designphase.

PDR

- Im Rahmen der Entwurfsphase finden Preliminary Design Reviews (PDR) statt, um den Entwurf der Software-Architektur und die Grobstruktur des Produkts zu überprüfen, bevor die Entwürfe im Detail ausgearbeitet werden. Dadurch wird sichergestellt, dass alle wesentlichen Anforderungen berücksichtigt und umgesetzt werden können.

CDR

- Bei besonders kritischen Komponenten kann ein Critical Design Review (CDR) sinnvoll sein, um den Entwurf und das Design dieser Komponenten vor der Implementierung erneut gründlich zu analysieren. Speziell in phasenorientierten Prozessmodellen mit einer strikt sequenziellen Vorgehensweise ist das in

der Regel der letztmögliche Zeitpunkt, an dem der Kunde das Projekt noch stoppen kann. Bei agilen Prozessen kann das Projekt nach jeder Iteration beendet werden. In der Praxis werden PDR und CDR meist in einem Reviewzyklus zusammengefasst.

IPR

- Sehr große und komplexe Projekte, deren Implementierung sich über einen besonders langen Zeitraum erstreckt, können sogenannte In-Process Reviews (IPR) erforderlich machen, um dem Kunden Fortschritte in der Implementierung, Prototypen oder auch Testfälle zur Begutachtung vorzulegen. Die kontinuierlichen Reviews agiler orgehensweisen entsprechen im Wesentlichen diesem Reviewtyp.

MR

- Management Reviews dienen der formellen Bewertung des Projekts und der Erhebung des aktuellen Projektstatus. Werden Abweichungen festgestellt, müssen geeignete steuernde Maßnahmen eingeleitet werden. Meist werden Management-Reviews im Projektplan zu definierten Zeitpunkten eingeplant und orientieren sich an Phasen und Iterationen, können aber auch bei außerplanmäßigen Ereignissen, beispielsweise bei geänderten Projektzielen, angesetzt werden.

TR

- Innerhalb des Projekts dienen technische Reviews dazu, einen konkreten Teil der Software zu überprüfen und zu bewerten. Weitere Aufgaben von technischen Reviews beinhalten etwa die Überprüfung der Realisierung im Hinblick auf Spezifikationen oder Standards. Auch die Fehlersuche in bestehenden Software-Produkten kann im Fokus eines technischen Reviewssein. Diese Reviews können je nach Bedarf eingeplant werden oder als integraler Bestandteil des Entwicklungsprozesses vorgesehen sein.

IRP

- Beispielsweise ist die Verwendung von Sourcecode-Managementsystemen (SCM) zur Unterstützung der kollaborativen Software-Entwicklung eine gängige Praxis in der modernen Software-Entwicklung. Eine wichtige Frage ist, wie geänderte Codeelemente oder Artefakte in das Repository übernommen werden sollen. Beispielsweise kann ein integrierter Reviewprozess vorgesehen werden, der die Übernahme in das Haupt-Repository erst zulässt, wenn der Sourcecode und die Artefakte zuvor, beispielsweise durch den Hauptentwickler oder Reviewer, überprüft wurden. Dazu wird durch den Entwickler ein Klon des Haupt-Repositories angelegt, der dann die entsprechenden Änderungen oder Erweiterungen vornimmt. Nach Abschluss dieser Änderungen wird dem Hauptentwickler oder Reviewer ein sogenannter „Pull Request“ – eine Anforderung zur Überprüfung der Änderungen – geschickt. Nach dem Review und der Freigabe werden der geänderte Sourcecode bzw. die Artefakte in das Haupt-Repository übernommen.

ICW

- Jetzt stellt sich noch die Frage, wie ein Review konkret durchgeführt werden kann. Eine Möglichkeit zur Beurteilung von Artefakten oder Sourcecode sind beispielsweise Inspection und Code Walkthrough. Diese Methoden Inspection und Code Walkthrough sind spezielle und stark formalisierte Reviewtypen mit der Zielsetzung, Abweichungen und Fehler in einem Produkt zu finden. Aufgrund der systematischen Vorgehensweise können sie auch zur formellen Nachweisführung oder für Schulungszwecke eingesetzt werden. Aufgrund der Bedeutung von Software-Inspektionen wird dieser Themenbereich in Abschnitt 5.4 im Detail vorgestellt.

6.3. Welche Rollen finden sich typischerweise bei Reviews und welche Aufgaben nehmen die unterschiedlichen Rollen wahr.

- Moderator („keeper of the process“): Leiter des Reviews
- Leser („keeper of focus and pace“)
- Gutachter („Reviewer“): Kommentierung des Reviewobjektes.

- Schreiber („preserver of knowledge“): Verfasst Protokoll (Notizen während des Reviews)
- Autor („author“): Klärung offenener Fragen. Keine Kommentier-/Rechtfertigung der Lösung.

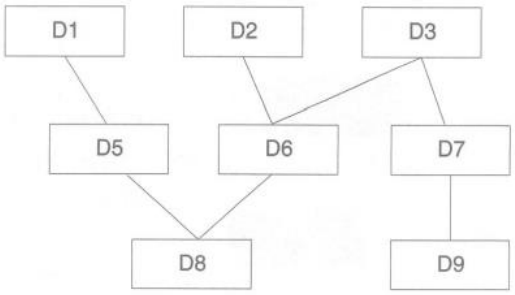
7. Integration (!!!)

7.1. Erläutern Sie den Begriff der Systemintegration.

- Modularisierung erleichtert Lesbarkeit, die Verständlichkeit und Wartbarkeit von Softwarekomponenten.
- (komplexe) Systeme umfassen meist viele unterschiedliche getestete Komponenten (z.B. Unit- oder Kompoententests).
  - ➔ Systemintegration bezeichnet, die Integration unterschiedlicher Komponenten und Komponenten zu größeren (Sub-) Systemen.

7.2. Skizzieren und erläutern Sie die vorgestellten Integrationsstrategien und welche Vor- bzw. Nachteile damit verbunden sind.

Big Bang Integration



Vorgehensweise:

- Alle Komponenten werden gleichzeitig integriert (=“Big Bang)

Vorteile:

- Keine zusätzlichen Testaufwände für Test stubs (um fehlende Funktionalitäten zu simulieren) -> alle Komponenten sind verfügbar.

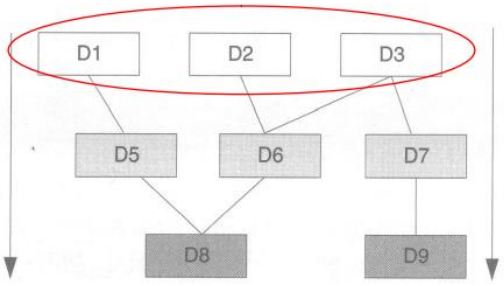
Nachteile:

- Fehler sind sehr schwer zu lokalisieren (z.B. Seiteneffekte)
- Hohes Risiko bei der Integtration

Anwendung:

- Kleine überschaubare Produkte

Top-Down Integration



Vorgehensweise:

- Schrittweise Integration (z.B. D1, D2, D3), ausgehend von den Business Cases.

Vorteile:

- Ausführbares Produkt Framework ist früh verfügbar.
- „Prototypen“ für Demo-Zwecke.
- Framework für Testfälle

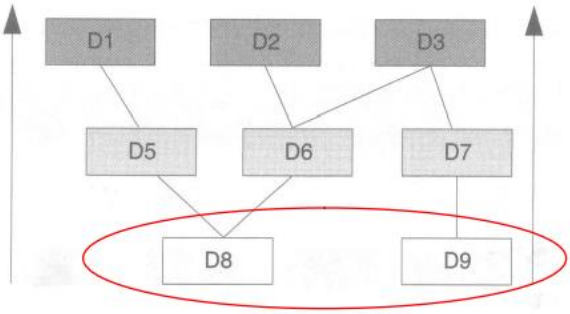
Nachteile:

- Zusätzlicher (hoher) Aufwand für Test stubs (um die fehlende Funktion zu simulieren).
- Integration von Hardware erfolgt spät (zusätzliches Risiko).

Bottoum Up Integration

Vorgehensweise:

- Schrittweise Integration, beginnend bei der Hardware (z.B. D8/D9).



Cases

Vorteile:

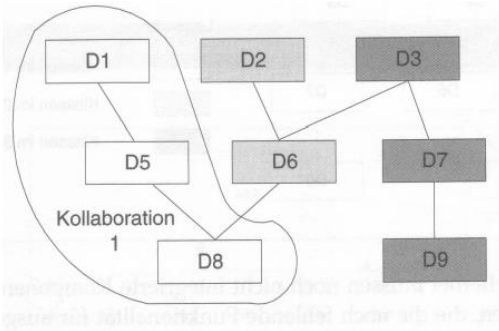
- Stabiles System (basierend auf Hardware Interfaces).
- Schrittweise Integration Richtung Business (Layers).

Nachteile:

- Ausführbares Gesamtsystem ist spät verfügbar.
- Zusätzlicher Aufwand für Prototypen.
- Zusätzlicher Aufwand für Test Drivers, um (low-level) Komponenten zu testen.

Build Integration

Vorgehensweise:



Business

- Schrittweise Integration entsprechend den Cases (über unter-schiedliche Layer hinweg) z.B. D1/D5/D8.
- Phasen-orientierte Integration

Vorteile:

- Frühe Verfügbarkeit von funktionalen Anforderungen (über alle Layer).
- Prototypen und Demo.
- Berücksichtig priorisierter Anforderungen möglich.

Nachteile:

- Wiederverwendung von Komponenten kann schwierig werden.
- Regressions-Tests sind erforderlich.

7.3. Was ist bei der Systemintegration im Hinblick auf die Qualitätssicherung zu beachten?

Eine saubere Automation des Buildprozesses ist eine wesentliche Voraussetzung für die Arbeit im Team und für die Erstellung qualitativ hochwertiger Software. Die Automatisierung hilft auch bei der Erstellung der Dokumentation und verbessert die Qualitätssicherung durch automatisierte Tests und Code-Quality-Checks

7.4. Welche Integrationsarten existieren bzw. was ist Integration überhaupt?

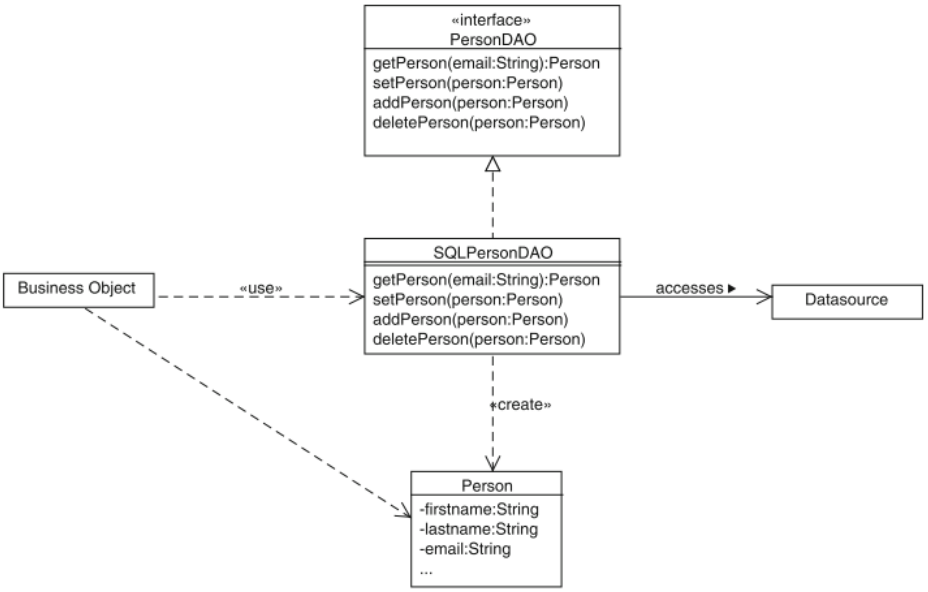
Bezeichnung Integration:

Die Integration in der Informatik, speziell in der Softwaretechnik, dient zur Verknüpfung von verschiedenen Anwendungen. Im Unterschied zur Kopplung handelt es sich hierbei um eine Verringerung und Vermeidung von Schnittstellen. Es lassen sich Funktionsintegration, Datenintegration und Geschäftsprozessintegration unterscheiden.

- Systemintegration
- Funktionsintegration
- Datenintegration
- Geschäftsprozessintegration
- Enterprise Application Integration
- Integrationstest (vielleicht)
- Integrationsmuster (vielleicht)

8. Architektur (!!!)

8.1. Beschreiben Sie das Data Access Object Pattern (DAO) anhand eines UML-Diagramms und erläutern Sie dessen Funktionsweise. Wann kommt dieses Pattern typischerweise zum Einsatz?

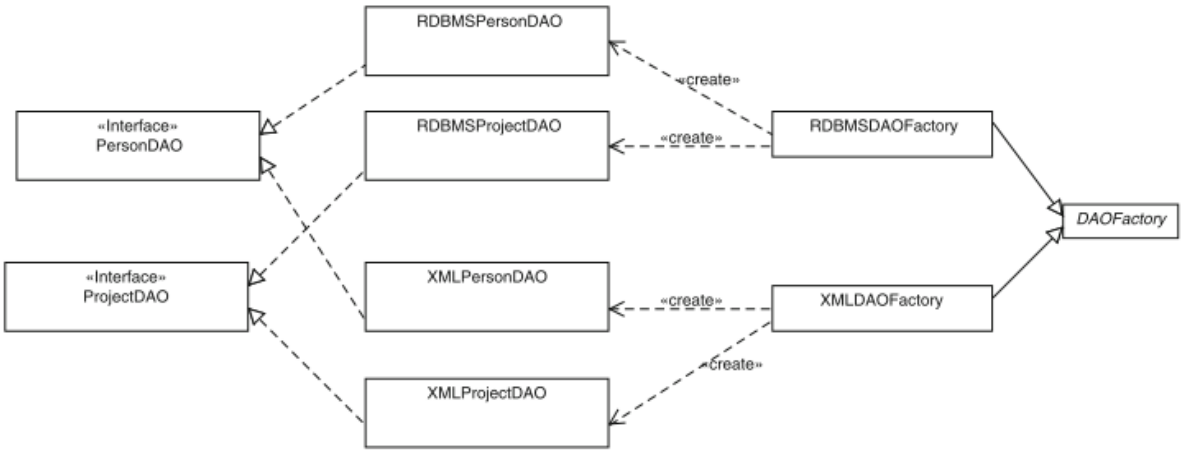


Die grundlegende Idee des DAO-Patterns ist in der Abbildung ersichtlich: Die Order-Klasse enthält ausschließlich Geschäftslogik, also z. B. die Methode zum Berechnen der Rechnung eines Kunden. Wichtig ist dabei, dass sich in Geschäftsmethoden keinerlei direkte Interaktion mit Persistenztechnologien wie relationalen Datenbanken (wie im obigen Beispiel) findet. Um Daten (Domänenobjekte) zu persistieren, definiert man Data Access Object Interfaces, wie in diesem Beispiel TaxDAO und OrderDAO. In diesen Interfaces definiert man möglichst generische „neutrale“ Methoden, die zum Daten speichern, laden oder suchen verwendet werden können. Man erkennt z. B. im OrderDAO-Interface Methoden wie addItem(...) oder getItems(...), mit denen neue Items gespeichert oder geladen werden können, die aber keine bestimmte Persistenztechnologie voraussetzen. Im Beispiel erkennt man auch, dass beide Interfaces in jeweils zwei Vari anten implementiert werden, z. B. DBOrderDAO würde in eine relationale Datenbank persistieren und XMLOrderDAO in ein XML File In der Abbildung wird das gesamte Szenario wie ein Domänenobjekt dargestellt, wobei Person mittels DAO-Pattern persistiert oder aus der Datenbank geladen wird. Man erkennt links wieder ein Geschäftsobjekt, das mit dem Data Access Object SQLPersonDAO interagiert, um das Domänenobjekt Personzu laden oder zu speichern. Das SQLPersonDAO-Objekt ist dabei wieder nur eine mögliche Implementierung des PersonDAO-Interfaces. Diese Implementierung verwendet z. B. eine relationale Datenbank und greift auf diese über eine (JDBC-) Datasource zu.

8.2. Beschreiben Sie das Factory Pattern. Wann wird es eingesetzt? Geben Sie ein einfaches Beispiel in UML an.

- In manchen Anwendungen kommt es vor, dass man Objekte oder Komponenten benötigt, deren Erstellung (Instanziierung) von vielen Rahmenbedingungen abhängig ist, bzw. die selbst wiederum andere Instanzen benötigen, um korrekt zu funktionieren. Man möchte also eine bestimmte Klasse nicht direkt instanziieren, sondern verwendet eine Hilfsklasse/-methode, die einem die gewünschte Instanz zurückgibt. Im Falle komplexerer Instanziierungen (z. B. Data Access Objects, Datenbank-Zugriffsklassen, XML Parser-Klasse usw.) kümmert sich die sogenannte Factory (Fabrik) um die korrekten Initialisierungsschritte. Zudem kann die Verwendung des Factory-Patterns helfen, Objekte voneinander zu entkoppeln. Die aufrufende Klasse muss dann nicht wissen, welche konkrete Instanz sie von der Factory-Methode zurückbekommt, sondern weiß nur, dass diese Instanz ein bestimmtes Interface implementiert oder von einer bestimmten abstrakten Klasse

ableitet.



8.3. Welche 3 Typen von Pattern gibt es? Nennen sie zu jedem mindestens 2 Beispiele.

Name	Beispiel
Grundlegende Muster	Interface, Delegation, Strategy, Immutable, Marker, Annotation
Erzeugungsmuster	Singleton, Factory, Object Pool
Strukturmuster	Fassade, Iterator, Adapter, Proxy, DAO, Generic DAO
Verhaltensmuster	Observer, Decorator, Interceptor

9. Projektmanagement – Strukturpläne

9.1. Was verstehen Sie unter einem Projektstrukturplan? (!!!)

Strukturierung der Arbeit

- Nach zweckmäßigen Kriterien
- In angemessener Detailierung / Tiefe

Ist (die wichtigste) Planungsgrundlage für:

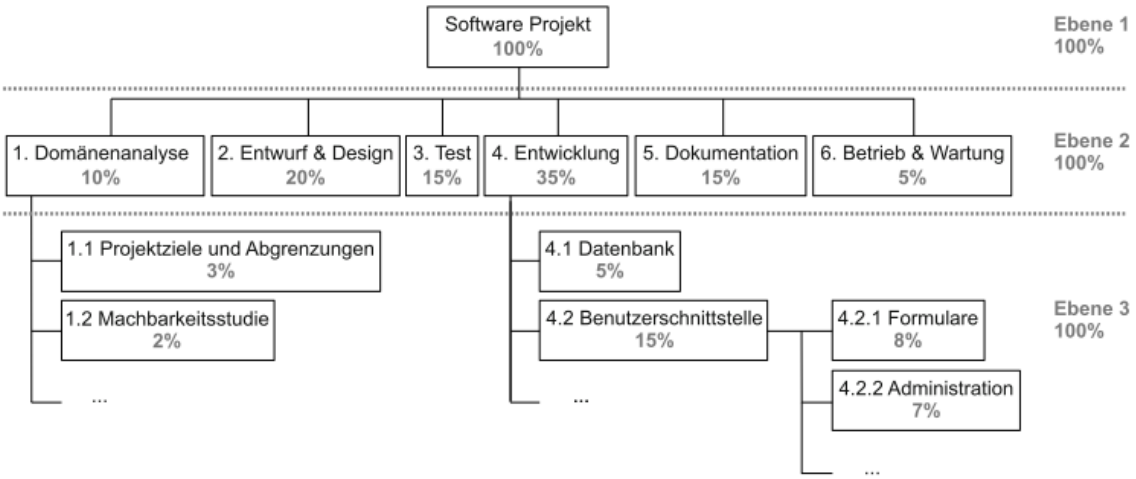
- Aufwandsschätzung
- Ressourcenplanung (Know-How, Skills, ..)
- Personaleinsatzplanung (wer mach was wann?)

Orientiert sich oft- aber nicht immer – an der Projektstruktur

9.2. Aus welchen Ebenen besteht ein Projektstrukturplan? (!!!)

- Projektziel
- Teilziele
- Arbeitspakete
- Meilensteine

9.3. Skizzieren und erläutern Sie einen (i) funktions- bzw. aufgabenorientierten Projektstrukturplan und (ii) einen objektorientierten Projektstrukturplan. Worin liegen die wesentlichen Unterschiede und wofür werden sie eingesetzt?



Im Groben muss jede Art eines Projektstrukturplans immer die 100% / Ebene erreichen!!!

- Funktionsorientierter Projektstrukturplan
  - Es wird hierbei ein Wert auf die Funktionalität des Projektes gelegt. Bessere Übersicht über die verschiedensten Funktionen des Projektes
- Aufgabenorientierter Projektstrukturplan
  - Gibt Übersicht über die Fertigstellung div. Aufgaben (Arbeitspakete) und in welchem Umfang diese fertig sind.
- Objektorienterter Projektstrukturplan
  - Hierbei liegt der Fokus am direkten Objekt. Gibt Übersicht über die Fertigstellung einzelner Objekte z.B. Datenbankanbindung

10. Projektmanagement

10.1. Was ist ein Projekt und durch welche Merkmale ist ein Projekt gekennzeichnet? (!!!)

Projekte sind gekennzeichnet durch:

- Ihre Abgrenzbarkeit bezüglich
  - Aufgabe, Ergebnis
  - Ressourceneinsatz und
  - Zeitrahmen (Beginn - Abschluss)
- Die Komplexität der Aufgabe
- Die Einmaligkeit der Aufgabe

Ein wesentlicher Aspekt: Das Rollenspiel / Verhaltensmuster

Wikipedia Auszug:

Ein Projekt ist ein zielgerichtetes, einmaliges Vorhaben, das aus einem Satz von abgestimmten, gelenkten Tätigkeiten mit Anfangs- und Endtermin besteht und durchgeführt wird, um unter Berücksichtigung von Zwängen bezüglich Zeit, Ressourcen (zum Beispiel Geld bzw. Kosten, Produktions- und Arbeitsbedingungen, Personal) und Qualität ein Ziel zu erreichen.

Merkmale	Messbare Attribute	Beispiele
Größe (PM)	Anzahl der beteiligten Personen	Personen: 50

Dauer und Aufwand (PM)	Anzahl Wochen bzw. Monate	Dauer: 52 Wochen Personen-Jahre: 3 PJ
Verwendete Technologien (SE)	Art, Anzahl und Alter der Technologien	Art: Scripting language, Alter: 4 J, Anzahl: 5
Komplexität (SE)	Anzahl Klassen, Module, Datenbanken; verwendete Technologien, Zeilen Code	Datenbanken: 2 Klassen: 42 Code: 30.000 Zeilen

10.2. Was versteht man unter einem Projekterfolg?

Unter einem Projekterfolg versteht man das Abschließen eines Projektes unter 100%iger Zufriedenstellung und Abdeckung des Projektauftrages mit dem Kunden. -> Der Kunde muss mit dem Projekt zufrieden sein, er muss damit arbeiten können. -> Beide Seiten sind mit der Arbeit zufrieden.

10.3. Durch welche Rahmenbedingungen wird der Projekterfolg gefördert? Geben Sie für jede Rahmenbedingung zumindest ein konkretes Beispiel an.

- Planung
  - Zielvorgaben des Projekts und konkrete Anforderungen an das zu erstellende Produkt, Gemeinsame Ablagen für Dokumente
- Kontrolle
  - Finanzielle, zeitliche und personelle Restriktionen und deren Risiken, Erfüllung der geforderten Aspekte
- Steuerung
  - Auswahl von Artefakten, Einbettung von Teammitgliedern in die Organisation

10.4. Was versteht man unter dem Projektauftrag? (!!!)

Die Projektdefinition ist die Grundlage für eine realistische und realisierbare Projektplanung sowie für eine erfolgreiche Projektdurchführung. Ziel der Projektdefinition ist – aufbauend auf einer freigegebenen Projektidee (Projektvorschlag) – einen Projektauftrag zu erstellen. Diese Festlegungen sind die Basis für die Durchführung des Projekts. Der Projektauftrag beinhaltet unter anderem Ziele und Nicht-Ziele, Rollendefinitionen, Arbeitsstrukturen und Arbeitspakete.

10.5. Was sind die Aufgaben eines Teamkoordinators?

Der Projektleiter oder Teamkoordinator hat die Projektverantwortung und ist für die Projektplanung, das Risikomanagement, die Projektorganisation (insbesondere die Aufgabenverteilung) und Entwicklung von Strategien bzw. Maßnahmen zur Erreichung der Ziele zuständig. Der Projektleiter ist also interner Koordinator des gesamten Entwicklerteams und direkter Ansprechpartner für den Auftraggeber.

10.6. Was ist Projektmanagement?

Management = Getting things done by people

Projektmanagement = Getting the Project done …

Das Projektmanagement (PM) beschäftigt sich mit der Planung, Kontrolle und Steuerung von Projekten und stellt einen organisatorischen Rahmen für eine erfolgreiche Projektabwicklung zur Verfügung.

10.7. Projektmanagement – Erfolgsfaktoren

- Einbringung und Berücksichtigung der Anwender
- Adäquates Projektmanagement: nicht zu viel aber auch nicht zu wenig
- Anforderungen müssen eindeutig beschrieben, realisierbar und auch überprüfbar sein
- Flexibler, realistischer Projektplan, der mögl. Verzögerungen berücksichtigt
- Realistische Kostenabschätzung und Buget, inkl. Risikoanalyse
- Angemessene Ziele
- Schlüsselteammitglieder haben genügend Projekterfahrung

- Gute Teamarbeit, funktionierende Kommunikation im Team

11. Anpassung von Software-Prozessen (!!!)

11.1. Erklären Sie die wesentlichen Konzepte des „Process Customizing“ und des „Prozess Tailoring“. Wozu und wann werden sie eingesetzt?

Prozesstailoring (bezogen auf der jeweilige Projekt)

- Anpassbarkeit eines generischen Entwicklungsprozesses an spezifische Projektgegebenheiten durch Prozesstailoring.
- Ersetzen einzelner Prozess-Schritte (oder Vorgehensbausteine) durch passende alternative Lösungen.
- Wiederverwendung von Best-Practices (Methoden / Tools).
- Individuelle Anpassung des Projektplans.

Achtung:

- Tailoring erfordert erfahrene Projektleiter, da ein fundiertes Verständnis des Modellaufbaus erforderlich ist.
- Berücksichtigung von definierten Tailoring-Kriterien und Produktabhängigkeiten.

Customization (Standardisierung)

Verallgemeinerte Form des „Tailorings“.

- Anpassung des Vorgehensmodells an unternehmensspezifische Gegebenheiten.
- Effizienzsteigerung von Tailoring für ähnliche Aufgaben durch Customization:
  - Unternehmensstandards: Anpassung an das Unternehmen.
  - Projektstandards: Anpassung an ähnliche Projekte (z.B. Webapplikationen).
- Diese “angepassten Prozesse” dienen als Grundlage für projektspezifisches Tailoring.

Achtung:

- Die Kompatibilität zum zugrunde liegenden Prozess muss sichergestellt werden!

12. Requirements (!!!)

12.1. Welche Rolle spielen Requirements im SE Life-Cycle?

Requierements (Anforderungen) zeigen die Wünsche des Kunden in Bezug auf das Softwareprodukt (user/customer view).

- Anforderungen müssen testbar sein und getestet werden.

12.2. Wie können sie klassifiziert werden?

Funktionale Anforderungen

- Was bzw. welche Funktion soll umgesetzt werden?
- Wie soll sich das System in definierten Situationen verhalten?
- Datenformate

Nicht-funktionale Anforderungen (z.B. Qualitätskriterien)

- Welche sonstigen Anforderungen müssen umgesetzt werden, die nicht direkt auf die Funktionalität abzielen, sie aber beeinflussen! Z.B.
  - Leistung und Performance (z.B. Optimierung des Informationsflusses)
  - Usability und menschliche Faktoren (z.B. Einfachheit der Bedienung, Training)
  - Sicherheit (z.B. Zugangskontrolle)

- Wartbarkeit (z.B. Tennung von Anwendung und Daten)
- Erweiterbarkeit

Designbedinungen

- Worauf soll beim technischen Entwurf geachtet werden, z.B. Schnittstellen, Plattformen und Entwicklungsumgebung, Verteilte Entwicklung.

Prozessbedingungen

- Rahmenbedingungen im Softwareprojekt, z.B. Ressourcen / Dokumentation

13. Design-Prinzipien (!!!)

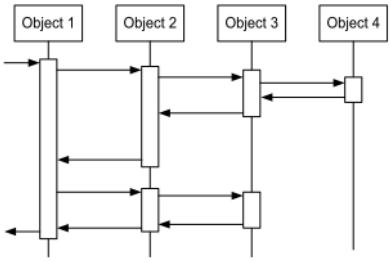
13.1. Beschreiben Sie das Design-Prinzip "System Control".

System Control (Wer übernimmt die Kontrolle im System)

Es gibt 2 Arten um das Design Prinzip „System Control“ auszuführen:

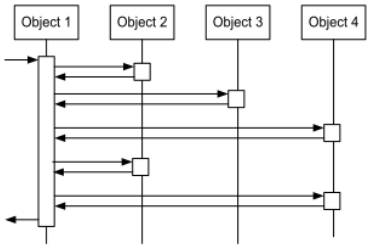
- Stair
- Fork

13.2. Stair vs. Fork (+ Beispiele)



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

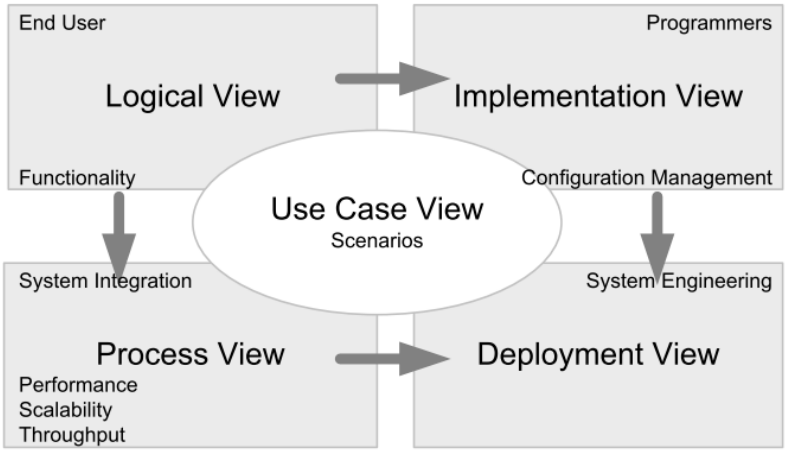


Sequence Chart: Fork

- **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 musss.

13.3. 4+1 View Model Architecture

- Teilaspekte der Architektur und des Designs und die spezifischen Eigenschaften eines Systems über verschiedene Sichten (views) beschrieben.
- Stammt aus der UML-Familie.



Logical View

- Funktionale Anforderungen.
- Fokus auf den End-User.
- Beispiele: Design Packages, Subsysteme, Klassen.
- UML 2: Klassendiagramme, State-Machines, Package Diagrams usw.

Implementation View

- Beschreibt statische Softwarekomponenten.
- Fokus auf Implementierung.
- Beispiele: Configuration Management, Source Code
- UML 2: Component Diagram der vorhandenen Softwareteile.

Process View

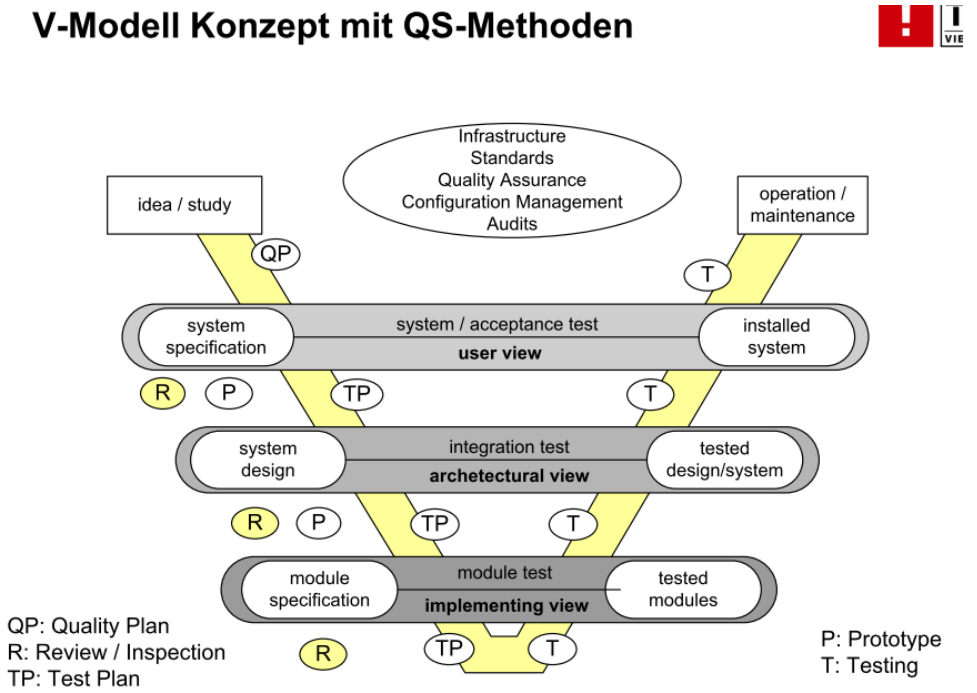
- Nicht-funktionale Anforderungen.
- Fokus auf Systemintegration.
- Beispiele: Laufzeitbedingungen, wie Concurrency, Lastverteilung, Fehlertoleranz.
- UML 2: Sequence, Activity Diagrams, Communication Diagrams.

Deployment View

- Ausführbare Applikationen (zur Laufzeit) unter Berücksichtigung der Plattform.
- Fokus auf Software Engineers.
- Beispiele: Deployment, Installation, Performance.
- UML 2: Deployment Diagram

14. Vorgehensmodelle

14.1. V-Modell Prinzip + Vorteile + welche Zweige/Ebenen gibt es (!!!)



Vorteile:

- Spezifikationsphase vs. Realisierung und Testen
- Kontext von Produkten und Tests
- Verschiedene Abstraktionslevels (User, Architekten und Implementierungsschicht)
- Fehlerbehandlung im frühen Phasen der Softwareentwicklung (durch Einsatzt von Reviews)
- Basiskonzept für VM 97 und VM XT

Nachteile:

- Klare Beschreibung der Systemanforderungen ist wichtig
- Hoher Dokumentationsaufwand
- Kritisch bei unklaren Anforderungen / sich ändernden Anforderungen

Anwendungsbereich:

- Große Projekte im öffentlichen Bereich
- Klar definierte Anforderungen

14.2. V-Modell XT + Tailoring + Einsatzgebiet

V-Modell XT (Verpflichtendes Vorgehensmodell für IT Projekte im öffentlichen Bereich in Deutschland, Vorgehensbausteine kapseln Rollen, Produkte [Mittelpunkt] und Aktivitäten)

- Projekttypen werden eingeteilt nach Projektgegenstand und Projektrollen

  → Vier Projekttypen

  - Systementwicklungsprojekt des Auftraggebers.
  - Systementwicklungsprojekt des Auftragnehmers.

- Einführung und Pflege eines organisationsspezifischen Vorgehensmodells. Systementwicklungsprojekt (Auftraggeber/Auftragnehmer).

- Vorgehensbausteine untergliedert in V-Modell Kern (verpflichtende Elemente für alle Projekttypen), Einführung und Pflege, Elemente für die Systementwicklung, Auftraggeber / Auftragnehmer – Schnittstelle.

14.3. (RUP Prinzip + Vorteile + Phasen + Einsatzgebiet)

Rational Unified Process (RUP)

- Inkrementelle und iterative Workflow.
- Integriertes Anfordungsmanagement
- Komponenten-orientierte Architektur
- Modellierung durch das UML Methodenframework
- Produkt-Verifikation an Meilensteinen
- Änderungsmanagement (supporting discipline)

Vorteile:

- Real World Sezarien
- Werkzeugunterstützung (Rational XDE, IBM)
- Vordefinierte Liste mit erforderlichen Artefakten

Nachteile:

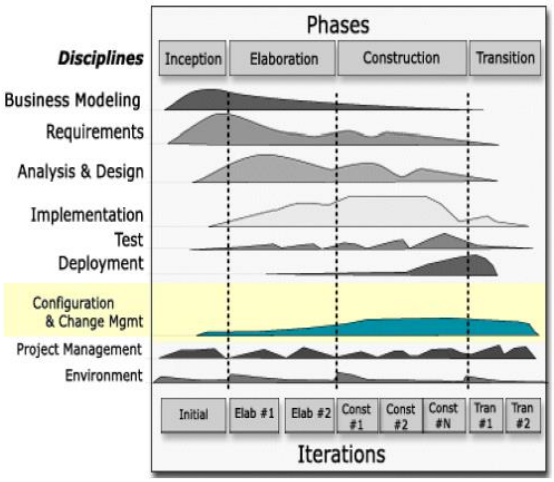
- Hohe Komplexität
- Hoher Dokumentationsaufwand
- Anbieterabhängig

4 Phasen

- Inception (Beginn)
- Elaboration (Concept & Design)
- Construction
- Transition (Auslieferung)

Anwendungsbereich:

- Grosse Projekte durch eine ganzheitliche Prozess-Sicht auf das ganze Projekt (inkl. Deployment)



15. Sonstiges

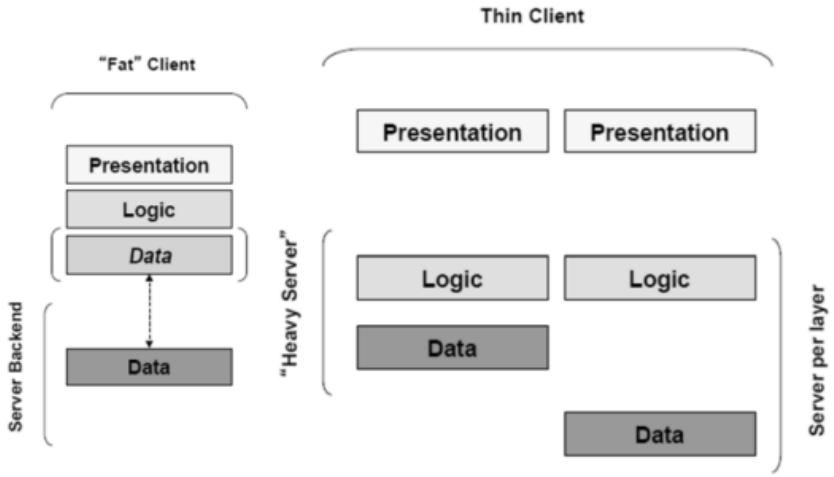
15.1. Was ist Continuous Integration?

Automatisierte Erstellung der Produktdokumentation

15.2. 4 Kernformen der Arbeitsstruktur

- Produktsturktorplan
- Vorgehensplan
- Dokumentationsplan
- Projektstrukturplan

15.3. Wie sieht eine moderene 3-Tier-Architektur aus? Beschreiben Sie die folgenden Fälle: a) Thin-Client b) Fat-Client. Beschreiben und skizzieren Sie die beiden Architekturvarianten.

- Präsentationsschicht
- Anwendungs-/Logikschicht
- Datenschicht

Thin-Client („Heavy Server“ vs. Server per Layer) / Fat-Client (Data-Server)