**Pool II: Entwicklung von Softwaresytemen (Prozessmodelle)**

**Frage 1: RUP**

* Ausgangssituation: Softwareunternehmen; Sie sind Mitarbeiter und zuständig für die Spezifikation der Kundenanforderungen. Dabei arbeiten Sie eng mit dem Kunden zusammen und haben die Aufgabe, die Wünsche und Anforderungen des Kunden für die eigentliche Entwicklung exakt zu erfassen und zu dokumentieren. (Rolle des Systemanalytikers). Dazu müssen Sie Werkzeuge einsetzen, welche die Sicht des Kunden wiederspiegeln, um sicherzugehen, dass sie alle Geschäftsprozesse und Wünsche des Kunden richtig verstanden haben.

Eine mögliche Vorgehensweise dafür ist der RUP.

Fragen:

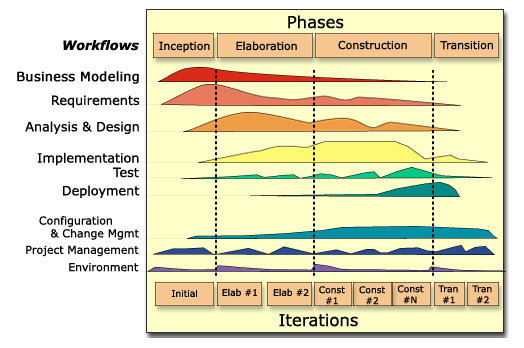
1. Beschreiben sie den RUP, seine Workflows und Phases
2. Erklären Sie die wichtigsten Artefakte des RUP pro Workflow und erstellen Sie beispielhaft die Artefakte für den Workflow „Requirements“ für die Beschreibung am Beiblatt. Beispiel: Bibliotheksverwaltung

(Domain Model und Use Case Diagramm) für die Anwendung

1. Diskutieren Sie Vor- und Nachteile des RUP

Antworten:

a)



**Horizontal axis (development phases and iterations):**

The RUP defines four Phases:

* Inception (Project setup, Concept)
* Elaboration (Draft, Architecture)
* Construction (Implementation)
* Transition (Deployment, Commissioning)

According to the iterative approach these phases are performed in iterations (Elab#1, Elab#2, Const#1, Const#2, …)

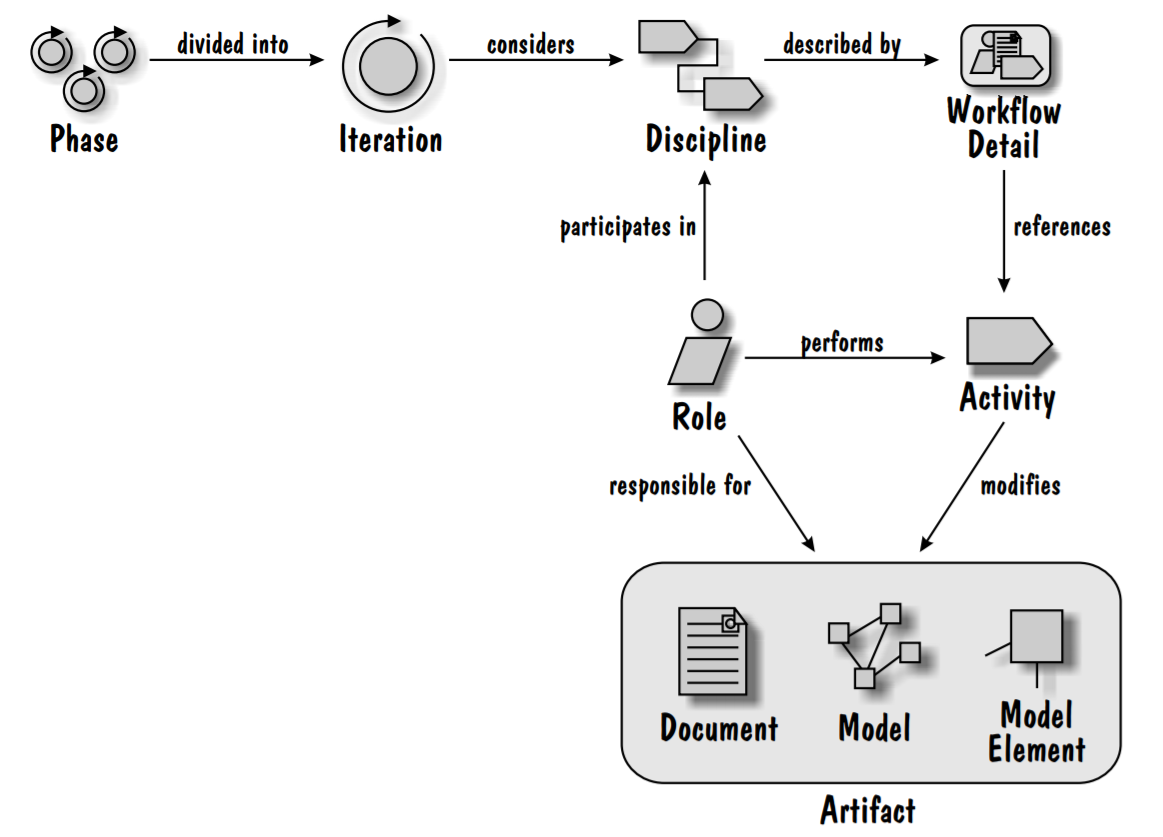
Each iteration results in a milestone. That is where it is decided if another iteration is required or if all set goals are achieved and the team can proceed to the next Iteration.  
  
**Vertical Axis (Disciplines / Workflows):**  
A project is divided into individual disciplines (activities). Each discipline is assigned a workflow, which is performed each iteration. A workflow represents a meaningful procedure for a successful execution of a discipline. Each step of a workflow should yield a part of an artefact.

**Colored Areas (Effort of disciplines per phase)**

The colored areas in the diagram represent the amount of work which goes into each discipline per phase.

An important fact is that disciplines are not restricted to a single phase. This means that unlike the “waterfall model” planning, development and testing does not take place strictly one after another. For example, using the RUP testing already happens during the Inception and Elaboration phases, and modelling / designing is still performed during the Construction phase.

Of course, certain disciplines / workflows are dominant during some phases but they are not performed exclusively in these phases. This is to ensure all requirements which are established near the end of the project can be considered. This difficulty has led to many projects (developed using the waterfall model) being abandoned.



(Abbildung auf Overhead-Folie drucken)!

b)

**Workflows des RUP & deren Artefakte**

Der RUP definiert 6 Core Workflows für die Erstellung des SW-Produktes und und für die 3 Suporting Workflows für die Prozessbegleitung, Die Core Workflows erwarten gewisse Artefakte als Eingabe und produzieren als Ausgabe wiederum eine festgelegte Anzahl von Artefakten.

**Core Workflows**

Decken die fachlichen und technischen Bereiche ab, (Erstelleung des Produktes)

### **Business Modelling**

Ziel dieses Workflows ist es ein Verständnis des Geschäftsprozesses sowie für das Umfeld in dem das System eingesetzt werden soll, zu bekommen. Außerdem sollen die derzeitig existierenden Probleme und deren Verbesserungspotential erkannt werden.

Die wichtigtsen Artefakte sind:

* **Business Vision**
* **Business Use Case Model**
* Business Use Case Diagram
* Business Use Cases ---- Description (Temaplate and ev Ads…Basic Flow, Alternative Flow, Extensions)
* Business Actors
* Business Workers
* **Domain Modell**
* Business Objects und ihre Beziehung untereinander

**Requirements**

Das Hauptziel ist es sich mit den Kunden sowie den Stakeholdern darauf zu einigen, welche Funktionen das Zielsystem haben soll. Dies stellt eine erste Basis für die Planung der technischen Umsetzung sowie für eine grobe Schätzung der Entwicklungsdauer des Projektes dar und gibt den Entwicklern einen Überblick über die Anforderungen.

Die wichtigtsen Artefakte sind:

* **Use Case Model**
* Use Case Diagram
* Use Cases --- Description (Template and ev ADs…Basic Flow, Alternative Flow, Extension)
* Actors (User)
* Workers
* **Refining des Domain Model (Class Diagram)!**
* **Vision**
* **Stakeholder Requests**
* **Prototype** **(Mock ups)**

**Analysis and Design**

Hierbei wird aus den Anforderungen das eigentliche System abgeleitet.

**Analyse Model**

Das Analyse Model ist Teil des RUP Workflows Analyse & Design. Aus den, im Requirements Workflow erstellten Use Case Modellen wird das Analyse Model abgeleitet. Es beschreibt die fachliche Logik des zu entwickelnden Systems aus statischer sowie dynamischer Sicht. Dies erfolgt mittels UML Diagrammen. Hauptziel ist es eine stabile Grundstruktur für das System zu schaffen.

Unterschiede zwischen Anforderungsanalyse (Requirements) und Analysemodel:

Das Use Case Model unterscheidet sich vom Analysis Model durch einige wesentliche Punkte:

* Analysis Model Diagram (AMD) statische Sicht
* Class Diagram, erweitert um die Analyseklassen (AMD oder ECB Pattern)
  + boundary classes (Schnittstellen)
  + control classes (Business Logic)
  + entity classes (Domains)
* Use Case Realization (fachliche Logik des Use Cases als Vorlage für die Programmlogik und Programmarchitektur)
  + Precondition
  + Sequence Diagram Basic Flow, alternate flows pro Use Case
  + Participated classes
  + Participated entities

|  |  |
| --- | --- |
| **Use Case Model** | **Analyse Model** |
| bedient sich der Sprache des Kunden | der des Programmierers |
| beschreibt das System aus einer externen Sicht | zeigt die interne Darstellung (Logik) |
| erhält seine Struktur durch die Anwendungsfälle | hingegen durch die stereotypen Klassen und Module |
| dient als Vertrag zwischen Kunde und Entwickler | wird von den Entwicklern verwendet |
| kann Redundanzen und Inkonsistenzen enthalten | darf keine Redundanzen und Inkonsistenzen enthalten |
| stellt die Funktionalität dar | stellt deren Umsetzung dar |

**Implementation**

In diesem Schritt findet die tatsächliche Implementierung des Systems statt. Bevor diese möglich ist, ist es notwendig den Implementierungsprozess zuerst ausreichend zu planen. (Verwendung v UML Diagrammen)

* Component Diagram
* Implementation Subsystem Diagram
* Integration Build Plan
* Build
* Implementation Model

**Test**

Dient dem Aufspüren von möglichen Fehlern im System sowie der Validierung ob das System den Anforderungen des Kunden entspricht.

* Test Plan
* Test Script
* Test Log
* Test Class/Data
* Test Component

#### Arten von Tests

* Unit-Test: Mittels Unit-Tests werden einzelne Komponenten der Software getestet. Dies erfolgt bereits im Implementation-Workflow.
* Systemtest: Es werden alle neu hinzugekommenen oder veränderten Teile der Software geprüft. Systemtest benötigen immer mehrere Durchläufe. Daher bietet es sich an diese zu automatisieren. In größeren Firmen existieren eigene Abteilungen um Systemtests durchzuführen.
* Integrationstest: Das gesamte System wird mit all seinen Komponenten geprüft.
* User-Acceptance-Test: Der Endnutzer testet das fertige System. Dafür kommt meist das Blackbox-Verfahren zum Einsatz. Im Unterschied zum Whitebox-Verfahren wird hierbei nur auf das Verhalten der Software anstatt auf den eigentlichen Programmcode geschaut.

### **Deployment**

Das entwickelte System wird an den Kunden ausgeliefert.

* Deployment Plan
* Release Notes
* Product
* Training Materials
* End User Support Material

**Angabe: Eine sehr vereinfachte Darstellung einer Bibliotheksverwaltung:**

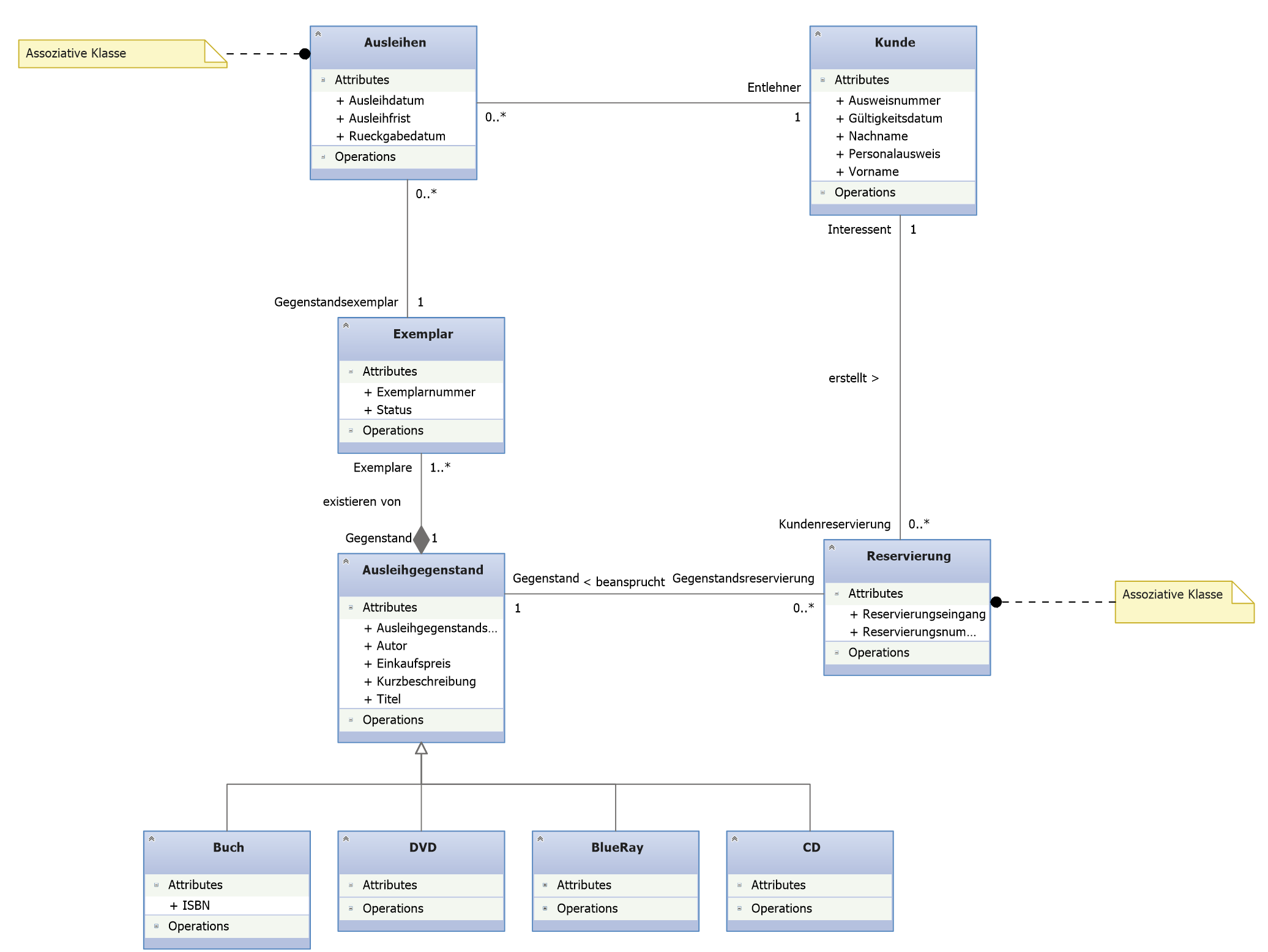
* Die Bibliothek verleiht Bücher, DVDs, Blue Rays und CDs. Von jedem Ausleihgegenstand kann es mehrere Exemplare geben, diese werden durch Exemplarnummern voneinander unterschieden.
* Als Daten werden Titel, Autor, ISBN (falls Buch), Kurzbeschreibung und Einkaufspreis festgehalten.
* Vor der ersten **Ausleihung** wird der Kunde **registriert**, dabei werden seine Daten (Name, Adresse usw) erfasst.
* Normalerweise ist die Ausleihfrist für Bücher 30 Tage. Wird diese Frist überschritten, wird eine **Mahnung** verschickt.
* Ausleihungen können **verlängert** werden.
* Auch können **Reservierungen** vorgenommen werden.
* Zur Durchführung von **Bedarfsanalysen** werden die Informationen über die Ausleihvorgänge aufbewahrt.
* Ausleihgegenstände, die länger als 3 Jahre nicht angefordert wurden, werden **aus dem Bestand genommen** und karitativen Institutionen geschenkt oder bei Flohmärkten “verscherbelt“.
* Falls Ausleihgegenstände verloren gehen, werden sie unverzüglich bei den jeweiligen Verlagen **nachbestellt.**
* Die **Rückgabe** eines Ausleihgegenstandes wird vermerkt, zuvor wird allerdings dessen Unversehrtheit geprüft.

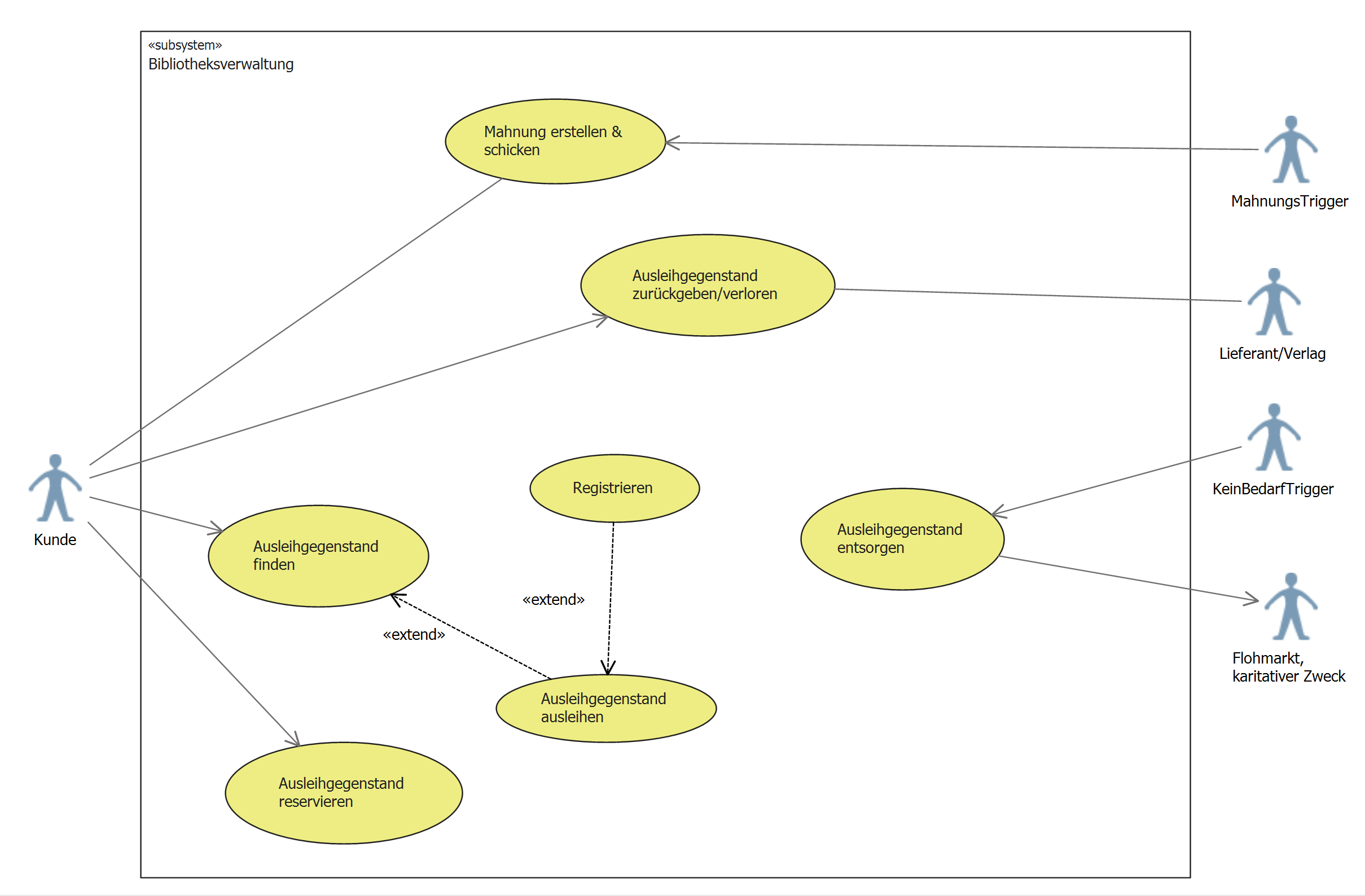
**Lösung**: selbst ausarbeiten!

Klassendiagramm: Ausleihgegenstand vererbt an Buch, CD, …

Exemplar wird an Kunden verliehen

Business Use Case Diagramm mit Actors (Flohmarkt, Kunde, Trigger, Lieferant/Verlag)





c)

**Vor- und Nachteile des RUP**

**Vorteile:**

* Risikofaktoren können schnell erkannt werden.
* Bessere Arbeitsteilung anhand verschiedener Rollen
* Man kann sich auf mögliche Probleme konzentrieren
* Laufende Qualitätssicherung
* Prozess verbessert sich nach jeder Iteration
* Supporting Workflows integriert (Configuration Management, Change Management, Project Management)

**Nachteile:**

* Komplexer Prozess
* Die Werkzeuge für RUP sind kostenpflichtig
* Oft komplizierte Iterationsplanung
* Hoher Dokumentationsaufwand
* Nur für große Projekte geeignet

Zur Erklärung – Iterationsplan:

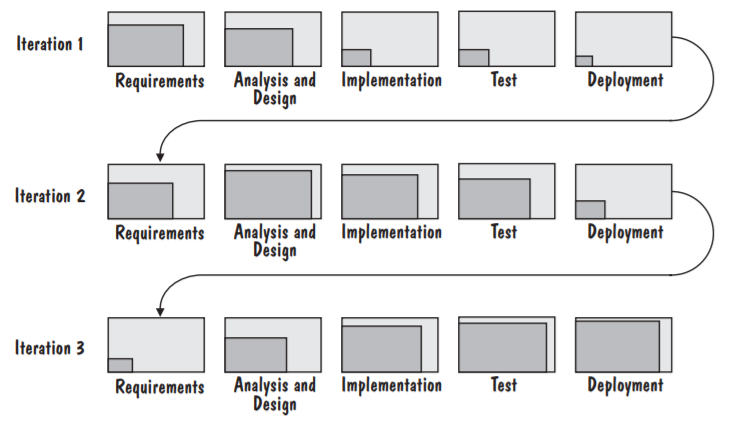
**Iterationen**

Iterationen sind ein Schlüsselmerkmal von erfolgreicher Softwarenentwicklung. Deshalb werden die beschriebenen Phasen üblicherweise in mehreren Iterationen durchlaufen.

Ziel einer Iteration ist es ein „Release“ zu erreichen. Dies kann ein interner Meilenstein sein aber auch ein neues Update für den Endnutzer. Um ein brauchbares Ergebnis zu erzielen ist es sinnvoll, pro Iteration alle Workflows zu Durchlaufen.

Entwickeln in Iterationen resultiert in „wachsender“ Software, d.h. das Produkt wird von Iteration zu Iteration ausgereiften. (besseres Verständnis der Anforderungen, robustere Architektur, erfahreneres Entwicklerteam, …)

Die folgende Grafik zeigt, wie im Laufe von Iterationen der Fokus auf verschiedene Workflows fällt. Es werden zwar alle Aktivitäten während jeder Iteration durchgeführt, aber die dafür aufgewandte Zeit verschiebt sich im Laufe der Iterationen. (Dargestellt durch die Grauen Boxen)



Jedem Release liegen diese Dokumente bei:

* Releasenotes (Beschreibung des Releases)
* Dokumentation für den Benutzer
* Changelog (Änderungen zum vorherigen Release)

**Frage 2)**

**Scrum**

* Ausgangssituation: Firma „SoftBall“ entwickelt Software für Sportorganisationen im Ballsport. Es werden ganz spezielle an den Kunden angepasste Lösungen implementiert. Firma setzt seit Jahresbeginn auf Scrum. Beim neuen Auftrag der Fifa zur Transfermarktverwaltung sind Sie Product Owner.

Fragen:

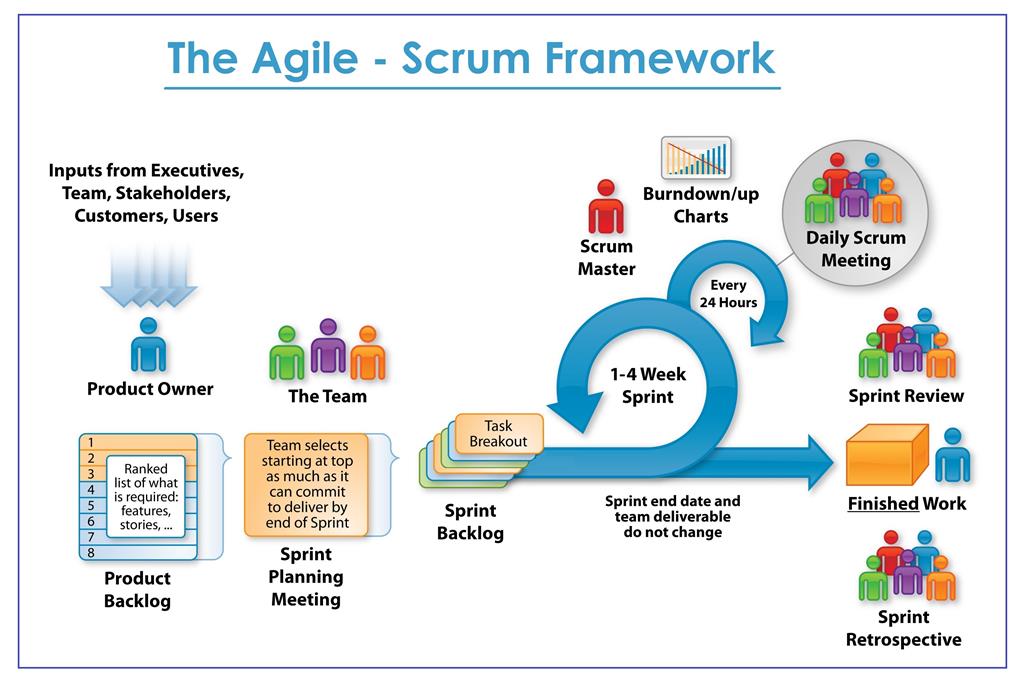
1. Beschreiben Sie das Prozessmodell „Scrum“ mit seinen Rollen und Charakteristiken.
2. Sie sind Product Owner eines Scrum Teams.

Stellen Sie sich ein Team von sechs Entwicklern zusammen und erstellen Sie ein beispielhaftes Scrum Board für den ersten Sprint mit dem gegebenen Product Backlog (siehe Beiblatt)

1. Diskutieren Sie die Vorteile/ Nachteile von Scrum

Antworten:

a)



**Rollen**

* **Product Owner**: verwaltet Product Backlog; gibt Entwicklerteam Arbeit vor; sollte Experte in Themenfeld sein. Für ein Scrum Projekt wird außerdem ein Product Owner bestimmt. Diese Person verwaltet die Product Backlog Liste. Jeder, der dem Produkt eine neue Anforderung hinzufügen will, muss sich an den Product Owner wenden, welcher über die Aufnahme entscheidet und dann die Priorisierung innerhalb der Liste vornimmt. Der Product Owner ist insofern wichtig, damit das Scrum Team nicht von allen Seiten mit Wünschen bedrängt wird und die Ziele für das Produkt nur an einer zentralen Stelle verwaltet werden, so dass nur ein einziger, immer aktueller Product Backlog für das Projekt vorhanden ist.
* **Scrum Master**: verantwortlich für die korrekte Durchführung von Scrum; Unterstützt PO (Erstellung von PBL) und Entwicklerteam (bei Fragen zu Entwicklung); stellt also Brücke zum Scrum-Team dar. Der Scrum Master ist die neue Management Rolle, welche mit Scrum eingeführt wird. Er ist die wichtigste Person im Scrum Prozess und muss eine herausragende Persönlichkeit sein, da er permanent die Initiative ergreifen und seine Aufgaben beharrlich verfolgen muss.
* **Scrum Team**: erfahrene Leute, die das Produkt-Inkrement erstellen; sollte interdisziplinär zusammengesetzt sein Ein Scrum Team umfasst mindestens fünf und allerhöchstens acht Personen. Dies ist wichtig, um eine möglichst große Produktivität zu gewährleisten. Werden weniger Personen eingesetzt, so findet zu wenig Informationsaustausch statt. Bei mehr als acht Team Mitgliedern besteht hingegen große Wahrscheinlichkeit, dass die Kontrollmechanismen von Scrum, wie das tägliche Scrum Meeting, versagen und zudem zu viel Komplexität entsteht.

**Ablauf**

Die Aufgaben, welche während dem Projekt erledigt werden sollen, sind im Product Backlog aufgeführt. Dieser entspricht einer Liste aller geforderten Funktionalitäten und Änderungen. Ein Scrum Projekt wird in Iterationszyklen von 30 Tagen aufgeteilt, den sogenannten Sprints. Zu Beginn jedes Sprints findet das Sprint Planning Meeting statt, in welchem der Product Owner die Elemente des Product Backlogs priorisiert und das Scrum Team die Aufgaben für den bevorstehenden Sprint festlegt. Diese entsprechen dem Sprint Backlog. Während dem Sprint trifft sich das Scrum Team täglich zum Daily Scrum Meeting, welches vom Scrum Master geleitet wird. Am Ende des Sprints demonstriert das Team die neu geschaffene Funktionalität im Sprint Review Meeting.

**Daily Scrum Meeting**

Das Scrum Meeting wird vom Scrum Master geleitet, der dafür sorgt, dass die Regeln eingehalten werden und dass das Treffen nicht zu lange dauert.

Von den Teilnehmenden gemeldete Hindernisse werden durch den Scrum Master nach Möglichkeit beseitigt. Sei dies persönlich oder durch Delegation an eine zuständige Stelle.

Ein Scrum Meeting ist keine Diskussionsrunde. Es werden während dem Treffen keine Probleme gelöst oder Designfragen besprochen.

**Sprint Review Meeting**

Mit dem Sprint Review Meeting wird am Ende des Sprints die neu erstellte Produktversion dem Management, den Kunden, den Usern und dem Product Owner vorgestellt. Das Treffen wird vom Scrum Master geleitet und das Produkt wird durch das Team präsentiert. Dabei wird versucht, möglichst alle Aspekte der neuen Produktversion aufzuzeigen, wie neue Funktionalität, Architektur oder Design. Außerdem wird über den Verlauf des Sprints berichtet.

Nicht abgenommene bzw. unfertige User Stories

🡪 zurück ins Product Backlog, neu priorisieren

🡪 an eine User Story des nächsten Sprints anhängen

**User Story**

Ein wesentlicher Bestandteil von Scrum sind die User Stories. Eine User Story lässt sich relativ einfach definieren. Sie beschreibt anhand einer Story ein ganz bestimmtes Bedürfnis eines Anwenders (= User). Und das in einer einfach verfassten Sprache (im Idealfall in der des Kunden und ohne Verwendung von Fachbegriffen).

Beispiel: „Ich als Anwender möchte ein Buch ausleihen.“

Benötigt: Akzeptanzkriterien formulieren und dazugehörige Testfälle definieren

b)

Aufgabenstellung – Beiblatt:

**Product Backlog für die Transfermarktverwaltung:**

Auflistung der zu erledigenden User Stories und Aufgaben für das gesamte Projekt:

1. US: Wohnorte aller Spieler im C# Client
2. US: Aktuelle Transfermarktübersicht im C# Client
3. US: Transfermarkt History im C# Client (Übersicht über alle schon abgeschlossenen Transfers)
4. Datenbank erstellen
5. US: Mannschaftsanzeige – Java Client
6. US: Transfermarktangebote mit Kaufoption – Java Client
7. US: Transferhistory – Java Client
8. Webservice erstellen
9. C# Client mit dem Webservice verbinden
10. Java Client mit dem Webservice verbinden

Lösung:

**Beispiel:** **FIFA Transfermarkt Verwaltung**

Es ist eine Anwendung zu erstellen, die das Kaufen und Verkaufen von Fußballspielern ermöglicht. Das Software Projekt soll einen WPF Client und einen Java Client besitzen, die über einen Java RESTful Webservice kommunizieren.

**Product Owner**

Beispielhaft: Herr Prof. Mustermann

Seine Rolle ist es, mit dem/den Auftraggeber(n) in Verbindung zu bleiben und auf seine/ihre Wünsche zu reagieren. Außerdem muss er darauf achten, dass sich die Kosten des Projekts im Rahmen halten.

**Entwicklungsteam**

Personen A-F

Das Team muss dazu in der Lage sein alle Anforderungen erfüllen zu können. Deshalb besteht es aus Mitarbeitern mit verschiedenen Spezialgebieten.

z.B.: A -> C#

B -> Datenbank + Webservice

C -> JavaFX

Außerdem enthält das Entwicklungsteam Mitarbeiter, die nur das Testen der Anwendung übernehmen.

In diesem Beispiel gelten folgende Einteilungen:

D -> testet Java Client Funktionalität

E -> testet C# Client Funktionalität

F -> testet GUI (Übersichtlichkeit, Design, etc.) beider Clients

Die Schuld für nicht erledigte Tasks kann aber nicht auf einen der Mitarbeiter geschoben werden, sondern liegt immer beim gesamten Team.

**Scrum Master**

Frau Prof. Müller-Stegmüller

Hilft dem Team die Artefakte richtig zu führen und weist auf nicht eingehaltene Scrum Regeln hin.

**Product Backlog**

Erstellung des Product Backlog (siehe Aufgabenstellung Beiblatt)

**Sprint Planning**

Dauer: zwei Stunden

Die Product Backlog Einträge für die erste Woche werden ausgesucht und in Tasks zerlegt, die an einem Tag erledigt werden können. Diese User Stories mit ihren Tasks werden im Sprint Backlog festgehalten. (ist die vorgenommene Arbeit für diesen Sprint)

Im Sprint Planning wird das Product-Increment also das Ziel für den nächsten Sprint festgelegt.

Sprintdauer: eine Woche

**Sprint Backlog**

* Datenbank erstellen
  + Modell erstellen
  + Datenbank-Tabellen anlegen
  + Daten einfügen
  + DB testen
* Aktuelle Transfermarktübersicht – C# Client
  + Verbindung zum Webservice aufbauen
  + Transfers darstellen
  + Layout optimieren
  + Test
* Mannschaftsanzeige – Java Client
  + Verbindung zum Webservice aufbauen
  + Spieler importieren
  + Frontend für Mannschaft designen
  + Test

Scrum Board zeichnen: Product Backlog | Sprint Backlog | Do To | In Process | to Verify | Done | Tested

**Daily Scrum**

An jedem Tag 8:00 Uhr 15 Minuten

Die Teammitglieder klären sich gegenseitig über ihren Fortschritt vom Vortag auf. Der Product Owner und der Scrum Master greifen nicht aktiv in das Gespräch ein, für Diskussionen und Probleme werden zusätzliche Termine vereinbart.

**Sprint Review**

Dauer: Eine Stunde

Das Team setzt sich nach Ablauf des Sprints mit dem Prodcut Owner, dem Scrum Master und den Stakeholdern zusammen und führt das bisher laufende Programm vor. Der Product Owner prüft die Funktionsfähigkeit der Einträge im Sprint Backlog. Kritiken und neue Ideen der Stakeholder sind erwünscht.

**Sprint Retroperspektive**

Dauer: Eine Stunde

Der Scrum Master setzt sich mit dem Team zusammen und versucht die Arbeitsweise des Teams im letzten Sprint zu analysieren und für den nächsten Sprint zu optimieren.

**Product Backlog Refinement**

Dauer: Eine Stunde

Die aus dem Sprint Review gewonnenen Erkenntnisse werden in das Product Backlog aufgenommen.

c)

**Vorteile:**

* Wenige Regeln (diese müssen aber eingehalten werden!), leicht verständlich und schnell einführbar
* Kurze Kommunikationswege
* Hohe Flexibilität/Agilität
* Hohe Effektivität durch Selbstorganisation
* Hohe Transparenz durch Meetings und Backlogs
* Zeitnahe Realisation neuer Produkteigenschaften bzw. Inkremente
* Kontinuierlicher Verbesserungsprozess
* Kurzfristige Problem-Identifikation
* Kleiner Administrationsaufwand

**Nachteile:**

* Kein Gesamtüberblick über die komplette Projektstrecke
* Hoher Kommunikations- und Abstimmungsaufwand
* Wenige konkrete Handlungsempfehlungen
* Zeitverluste bei zu „defensiven" Sprintplanungen
* „Tunnelblick-Gefahr" bei ausschließlicher Fokussierung auf Tasks
* Erschwerte Koordination mehrerer Entwicklungsteams bei Großprojekten
* Potenzielle Verunsicherung aufgrund fehlender Zuständigkeiten und Hierarchien
* Potenzielle Unvereinbarkeit mit bestehenden Unternehmensstrukturen

**Frage 3: Prozessmodelle**

* Ausgangssituation: Firma „SoftTronic“; entwickelt Software für Groß- und Mittelbetriebe; Projektteams setzen sich aus verschiedenen Spezialisten zusammen; Auslieferung von speziell angepasster Software im Dienstleistungsbereich.

Fragen:

1. Nennen Sie in der Softwareentwicklung gängige Prozessmodelle, beschreiben und klassifizieren Sie diese.
2. Stellen Sie die klassischen den agilen Methoden gegenüber und erklären Sie jeweils die Vor- und Nachteile
3. Softwarefirma arbeitet derzeit noch mit klassischen Methoden, in letzter Zeit wurden vermehrt agile verwendet. Firmenleitung findet, dass Einsatz von Aufgabenstellung abhängt. Sie sind Projektmanager in der Firma und haben nun zu entscheiden, welches Prozessmodell sich für die folgenden Beispiele am besten eignet.

Begründen Sie jeweils Ihre Entscheidung.

Antworten:

a)

**Prozessmodelle Allgemein**

Prozessmodelle legen den organisatorischen Rahmen einer Software-Entwicklung fest.

Sie dienen außerdem der Vermittlung von Verständnis über Tätigkeiten, Funktionen, Rollen und Schnittstellen. Sie können als praktische Umsetzung eines Vorgehensmodells gesehen werden.

(Das Vorgehen, wie man von einer Kundenanforderung zu einem fertigen Produkt kommt).

Prozessmodelle der Softwareentwicklung legen Folgendes fest:

* Reihenfolge des Arbeitsablaufes (Phasenkonzepte)
* Durchzuführende Aktivitäten
* Definition der Teilprodukte
* Fertigungskriterien
* Notwendige Mitarbeiterqualifikationen
* Einzuhaltende Standards, Richtlinien und Werkzeuge
* Art und Umfang der Dokumentation

Es gibt zwei verschiedene Arten von Prozessmodellen: Klassische und Agile Methoden.

**Klassische Methoden**

Wert wird auf den Prozess gelegt.

Dies bedeutet, dass die Planungsphase ausführlich ausfällt und allen Phasen eine

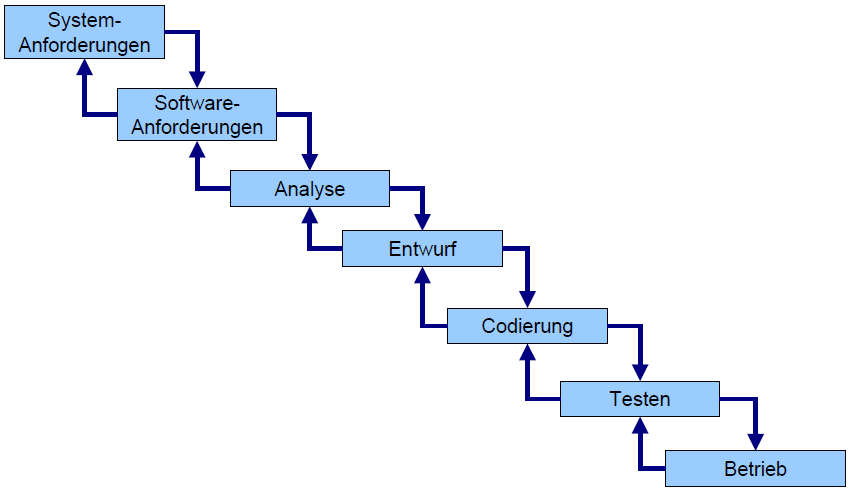
detaillierte Dokumentation folgt, die wiederum als Eingabe für die darauffolgende Phase

dient. (Artefakte).Übergeordnetes Ziel des Phasencharakters ist dabei das effektive Steuern der

Softwareentwicklung sowie das Minimieren von Komplexität. Diesen Ansatz bezeichnet

man auch als Software Engineering.

**Wasserfall**



Jede Phase muss komplett abgeschlossen sein, damit die nächste beginnen kann. Dazu produziert jede Phase als Ergebnis ein ausführliches Dokument oder Programm. Abbildung 1 zeigt das erweiterte Wasserfallmodell nach Royce – hier sind Rücksprünge erlaubt. Allerdings müssen dabei meist alle Arbeiten der aktuell laufenden Phase verworfen werden. Das Wasserfallmodell kann mit dem Beispiel eines Hausbaues erklärt werden: Je später eine fehlerhafte Spezifikation erkannt wird, desto schwieriger und teurer ist die Behebung.

Einsatz: Für einfache Projekte, wo sich Anforderungen, Leistungen und Abläufe in der Planungsphase relativ präzise beschreiben lassen und sich auch im Laufe der Entwicklung wenig ändern.

**Vorteile**

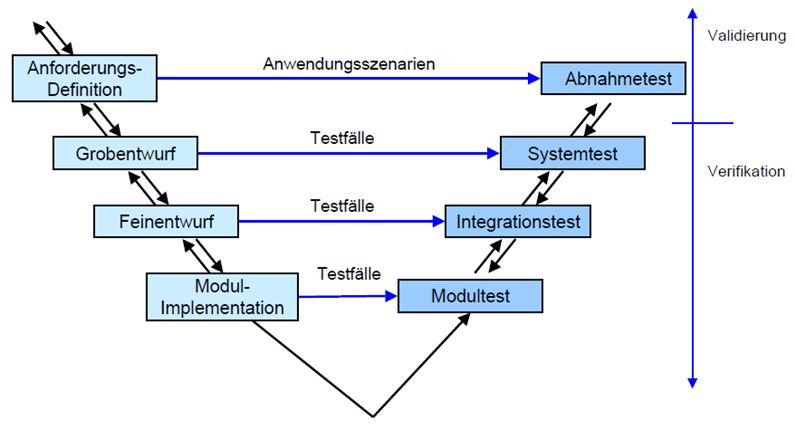
* Einfach, verständlich
* Geringer Management-Aufwand
* Disziplinierter, kontrollierbarer und sichtbarer Prozessablauf

**Nachteile**

* Das rein sequentielle Vorgehen und volle Breite der Entwicklungsschritte ist oft nicht sinnvoll
* Gefahr einer Überbewertung der Dokumentation (Unnötiger Mehraufwand)
* Risiken werden zu wenig berücksichtigt
* Fehler werden sehr oft zu spät erkannt.
* Benutzerbeteiligung nur bei Anforderungen und im Betrieb

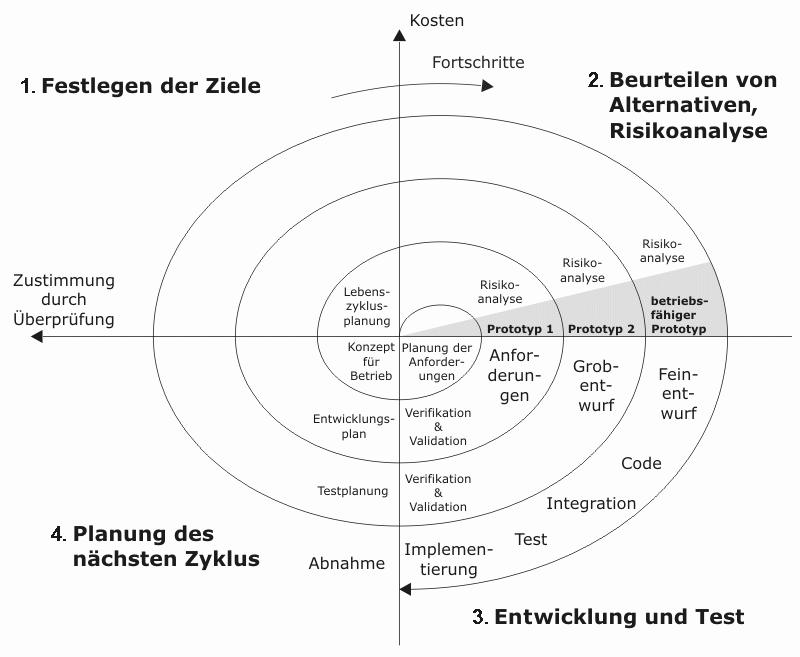
**V-Modell**

Neben den Entwicklungsphasen definiert das V-Modell auch das Vorgehen zur Qualitätssicherung phasenweise. Die Phasenergebnisse sind wie beim Wasserfallmodell bindende Vorgaben für die nächsttiefere Projektphase. Der linke, nach unten führende Ast für die Spezifizierungsphasen, schließt mit der Realisierungsphase ab. Im rechten, nach oben führenden Ast sind die Testphasen dargestellt. Diese Gegenüberstellung soll zu einer möglichst hohen Testabdeckung führen, weil die Spezifikationen der jeweiligen Entwicklungsstufen die Grundlage für die Tests in den entsprechenden Teststufen sind.



**Spiralmodell**

Vorgehensmodelle wie das Spiralmodell brechen die Starrheit des Wasserfalls auf und gestalten den Phasencharakter der Softwareentwicklung als einen iterativen Prozess. Die sequenzielle Arbeitsweise seines Vorgängers ist beim Spiralmodell nicht mehr gegeben und erlaubt eine Verfahrensweise, bei der das Kommunizieren der Entwicklungsphasen nacheinander durchlaufen wird und sich dann fortsetzend so lange wiederholt, bis alle vorhandenen Probleme mit der Software gelöst sind. Da die Iterationen nicht auf eine rückschauende Denkweise des Wiederholens gebunden sind, wie etwa bei der Sequenzialität des Wasserfallmodells, können die Rahmenbedingungen zukünftiger Iterationen leicht an Veränderungen und wechselnde Erwartungen angepasst werden. Das Spiralmodell integriert ein gutes Risikomanagement.



**Aktivitäten je Quadrant**

Das Spiralmodell fasst den Entwicklungsprozess als iterativen Prozess auf, wobei jeder Zyklus in den einzelnen Quadranten folgende Aktivitäten enthält:

* Festlegung von Zielen, Identifikation von Alternativen und Beschreibung von Rahmenbedingungen
* Evaluierung der Alternativen und das Erkennen, Abschätzen und Reduzieren von Risiken, z.B. durch Analysen, Simulationen oder Prototyping
* Realisierung und Überprüfung des Zwischenprodukts
* Planung des nächsten Zyklus der Projektfortsetzung

**Vorteile**

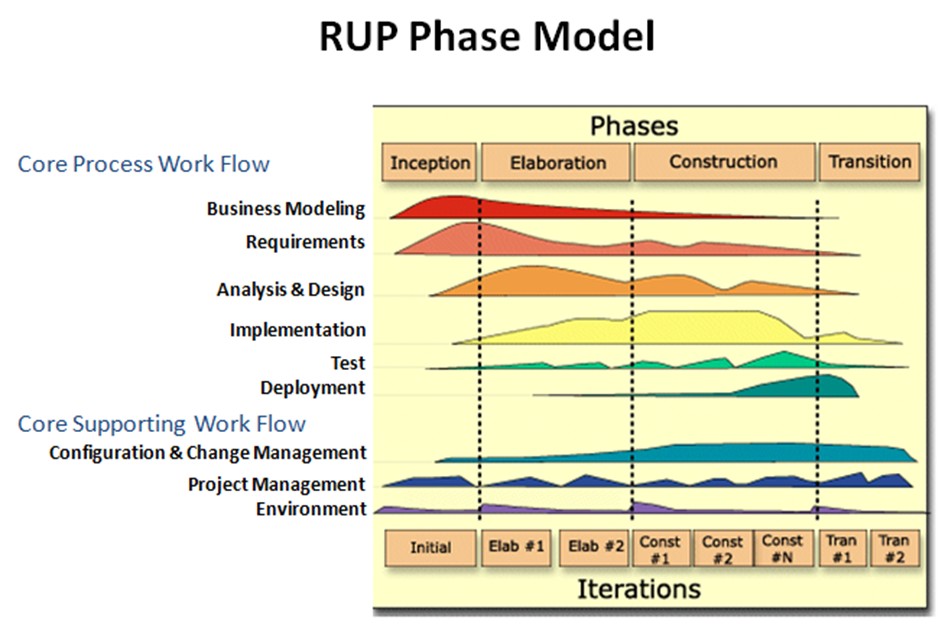
* Flexibles Modell
* Periodische, risikogetriebene Überprüfung und erneute Festlegung des Ablaufs
* Frühe Fehlererkennung

**Nachteile**

* Hoher Managementaufwand, da oft Entscheidungen über den Prozessablauf getroffen werden müssen
* Ungeeignet für kleine und mittlere Projekte
* Wissen über die Handhabung von Risiken nicht immer verfügbar

**Rational Unified Process (RUP)**

RUP bietet durch sein Iterationskonzept (mehrere Durchläufe nach dem gleichen geplanten Ablauf) sehr viel Flexibilität gegenüber Veränderungen auch im späteren Projektverlauf und kann so auf individuelle bzw. nachträgliche Kundenwünsche schneller und dynamischer reagieren. Es spielt vor allem der Vorteil der Zeiteinsparungen bei Änderungen eine größere Rolle, da eine Verspätung des Product Relase Milestone oft mit hohem finanziellem Aufwand verbunden ist.



Die Arbeitsschritte werden bei jeder Iteration in 7 Core Disciplines (Workflows) und 3 Supporting Disciplines (Workflows) wie folgt unterteilt. (Statische Aspekte (innerhalb einer Iteration)) Jede der vier Phasen ist in eine oder mehrere Iterationen unterteilt und resultiert in einem Meilenstein (Dynamische Aspekte).

**Vorteile**

* Für große Projekte geeignet

**Nachteile**

* Komplex
* Dokumentationslastig

Für eine genauere Beschreibung des RUP siehe Pool II – Frage 1a.

**Agile Methoden**

Agile Methoden der Softwareentwicklung (lightweight methods) entstanden Mitte der 90er Jahre als Reaktion auf die bürokratischen, dokumentlastigen klassischen Methoden und deren Schwierigkeiten im Umgang mit sich schnell verändernden Anforderungen. Ihr Fokus liegt auf der eigentlichen Softwareentwicklung und nicht auf Prozess, Design oder Dokumentation. Um das Ziel der schnellen Anpassung an Änderungen zu erreichen, pflegen agile Methoden sehr kurze Iterationen, bei denen inkrementell entwickelt wird. Durch kurze Iterationen und kurze Release-Zyklen wird schnelles Feedback beim Kunden über den Fortschritt und Funktionsumfang eingeholt. Dieses schnelle Feedback dient zur Steuerung der weiteren Entwicklung.

**Agiles Manifest**

* Individuen und Interaktionen haben Vorrang vor Prozessen und Werkzeugen.
* Funktionsfähige Produkte haben Vorrang vor ausgedehnter Dokumentation.
* Zusammenarbeit mit dem Kunden hat Vorrang vor Vertragsverhandlungen.
* Das Eingehen auf Änderungen hat Vorrang vor strikter Planverfolgung.

**Extreme Programming (XP)**

XP ist ein durch fortlaufende Iterationen und den Einsatz mehrerer Einzelmethoden strukturierendes Vorgehensmodell.

**Pair Programming**

Bei der Paarprogrammierung teilen sich zwei Programmierer einen Computer – einer codiert (der Driver) und der andere denkt mit und hat das Gesamtbild im Kopf (der Partner). Die Rollen werden regelmäßig getauscht. Dieses Vorgehen steigert den Wissenstransfer. Anfänger sollen schneller von der Arbeit eines Spezialisten lernen.

**Vorteile**

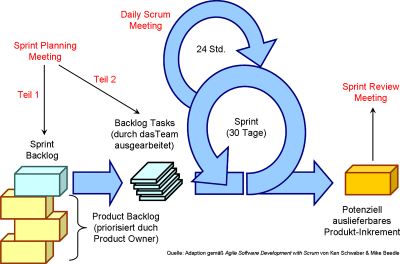
* Testfälle werden vor der Programmiertätigkeit geschrieben
* Programmieren in Paaren (Teamgeist)
* Refactoring durch hohe Codequalität möglich

**Nachteile**

* Erschwerte Wiederverwendung
* Nur für kleine, hochqualifizierte Teams geeignet

**SCRUM**

Scrum ist ein Rahmenwerk, welches für die Entwicklung von Projekten erschaffen wurde. Es regelt die Aufgaben der Mitarbeiter, Abgabetermine, Besprechungszeiten und die genaue Dokumentation. Es ist ein Werkzeug das dabei hilft Ziele zu setzen und diese als Team zu erarbeiten. Scrum wurde anfangs nur für Softwareprojekte entwickelt, ist aber auch für fast alle anderen Arten von Projekten einsetzbar. Was dieses Rahmenwerk von vielen anderen unterscheidet ist die Erkenntnis, dass große Projekte nicht vom Anfang an detailliert geplant werden können. Scrum ermöglicht das einfache Hinzufügen und Entfernen von Funktionen während der Entwicklungsphase. Die Zusammenarbeit mit den Kunden ist wichtiger als das Lastenheft.



**Vorteile**

* Wenige Regeln, leicht verständlich und schnell einführbar
* Kurze Kommunikationswege
* Hohe Flexibilität/Agilität durch adaptives Planen
* Hohe Effektivität durch Selbstorganisation
* Hohe Transparenz durch regelmäßige Meetings und Backlogs
* Zeitnahe Realisation neuer Produkteigenschaften bzw. Inkremente
* Kontinuierlicher Verbesserungsprozess
* Kurzfristige Problem-Identifikation
* Geringer Administrations- und Dokumentationsaufwand

**Nachteile**

* Kein Gesamtüberblick über die komplette Projektstrecke (Gesamtanforderungen – User Story Map!!! Bitte anschauen, siehe Beipiel Hausverwaltung vom vorigen Jahr)
* Wenige konkrete Handlungsempfehlungen
* Zeitverluste bei zu „defensiven" Sprintplanungen
* „Tunnelblick-Gefahr" bei ausschließlicher Fokussierung auf Tasks
* Erschwerte Koordination mehrerer Entwicklungsteams bei Großprojekten
* Potenzielle Verunsicherung aufgrund fehlender Zuständigkeiten und Hierarchien
* Potenzielle Unvereinbarkeit mit bestehenden Unternehmensstrukturen

b)

**Vergleich**

Bei der Wahl zwischen klassischen und agilen Methoden, muss man beachten, dass nicht jede Methode für jedes Projekt geeignet ist. Beide Methoden weisen spezifische Vor- und Nachteile auf, die sie für verschiedene Situationen auszeichnen. Um die, für das jeweilige Projekt, richtige Wahl zwischen diesen Methoden zu treffen, muss man die wichtigsten Charakteristika und Risiken des Projektes kennen und verstehen.

Es gibt verschiedene Kriterien anhand deren Ausprägung man eine Entscheidung treffen kann. Beispielhaft werden hier folgende Kriterien behandelt:

* Entwickler
* Kunde
* Anforderungen
* Größe
* Zielsetzung
* Wartbarkeit

Im Allgemeinen gilt: Verwendung Agiler oder klassischer Prozessmodelle für die SW-Entwicklung ist keine Frage des „entweder oder“ sondern eine Frage des „sowohl als auch“.

**Kriterium Entwickler**

**Agil**

* Hohe Anforderungen an Entwickler
* Verlass auf implizites Wissen
* Gefahr des Fehl- oder Kurzsichtigen Entscheidens

**Klassisch**

* Reduzierung des Risikos durch Verlass auf Pläne und Architekturen
* Experteninterview der Dokumente möglich

**Kriterium Kunde**

**Agil**

* Bereitschaft zu hohem Einsatz auf Seiten des Kunden nötig
* Aktiver, involvierter Kunde
* Wissen des Kunden vor Ort muss ausreichen, um Anforderungen zu beschreiben

**Klassisch**

* Kunde erbringt seinen Teil beim Formulieren der Anforderungsdokumente
* Nur bei Meilenstein und Abnahme Kundeninvolvierung

**Kriterium Projektgröße**

**Agil**

* Effektiv bei kleinen und mittelgroßen Projekten
* Benötigt räumliche Nähe

**Klassisch**

* Einsatz bei großen Projekten
* Geringerer Kommunikationsaufwand bei verteilten Projekten

**Kriterium Zielsetzung**

**Agil**

* Frühe Auslieferung von Produktinkrementen als Ziel
* Kurzsichtige Entscheidungen

**Klassisch**

* Korrekte Implementation der Anforderungen als Ziel
* Einsatz bei sicherheitskritischen Projekten

**Kriterium Wartbarkeit**

**Agil**

* Design und Dokumentation nur in den Köpfen des Teams verfügbar
* Eher für interne Projekte geeignet
* Testsuiten erleichtern Erweiterung ohne neue Fehler einzubauen

**Klassisch**

* Jede Phase endet mit einer Dokumentation
* Dokumentation als Grundlage für Wartung und Erweiterung

**Fazit**

* Agile und klassische Methoden bilden jeweils das Ende eines Spektrums
* Auswahl der Methode anhand der Charakteristiken des Projektes
* Hybride Methoden als Kombination beider Methoden als Lösung für Projekte mit beiden Charakteristiken

c)

**Aufgabenstellung 1:**

Das Krankenhaus „LKH Musterstadt“ verwendet seit Jahren in der gesamten Verwaltung die Software „MediCare“.

Die KH-Verwaltung hat der Firma Softronic den Auftrag zur Entwicklung folgender, zusätzlicher Programmmodule erteilt:

1. Seit einigen Monaten werden von bestimmten Medikamenten die Nebenwirkungen nach verschiedenen Kriterien analysiert. Dafür soll ein neues SW-Modul entwickelt werden.
2. Außerdem werden seit Kurzem bei den meisten Operationen Videoaufnahmen gemacht, für diese wird eine Software zur Archivierung und zur Analyse der Videos benötigt.

Die zu erstellenden Programmodule sind in das bestehende System zu integrieren.

**Lösung: agiles Prozessmodell (Scrum), da Geschäftsssytem und Modellierung der Daten bereits bekannt sind.**

**Aufgabenstellung 2:**

Die Firma „Help4All“ will in das Postgeschäft einsteigen, das Versenden und Verteilen von Paketen aller Art soll ermöglicht werden.

Sie haben nun die Aufgabe, alle Geschäftsprozesse und die gesamte Datenlandschaft zu modellieren und dokumentieren, sowie für die zukünftig zu verwendende Software die Anforderungen zu spezifizieren (Requirement Specification). Dies soll die Grundlage für die Entscheidung sein, eine bereits am Markt befindliche SW (ERP-System) zuzukaufen und zu adaptieren oder eine völlig neue Applikation zu entwickeln.

Die Firmenleitung erwartet sich einen Entwurf zur tatsächlichen Auftragsvergabe (Pflichtenheft), also Realisierung und Implementierung der Software.

**Lösung: klassisches Prozessmodell (RUP), da Anforderungen und Geschäftssystem erst entwickelt werden müssen.**

**Frage 4) Testen von Software**

Die Software Firma „simplify“ entstand vor 5 Jahren als kleines 2 Personenunternehmen.

Sie hat sich spezialisiert auf Software für Verkehrsleitsysteme. Aufgrund der guten Auftragslage

entschied sich der Firmeninhaber massiv zu expandieren, derzeit beträgt der

MitarbeiterInnenstand 25.

Sowohl während des Entwicklungsprozesses als auch vor der Auslieferung Software müssen

geeignete Verfahren zum Testen der Software angewendet

werden, bestimmte Testarten müssen auch automatisiert durchgeführt werden.

Fragestellung:

1. Das Testen von Software erfolgt in verschiedenen Levels. Nennen Sie diese und beschreiben Sie deren Anwendung und geeignete Testverfahren/Testmethoden. (Testarten)
2. ISO 25010
3. In der 1. Klasse der Informatikabteilung lernen die Schüler wie man einen Taschenrechner programmiert. Um die Programme der Schüler mittels der Black-Box Testmethode überprüfen zu können, sollen für Funktionen des Taschenrechners JUnit Tests von den Höheren Klassen programmiert werden.

Erklären sie was eine Black-Box-Testmethode ist, welche Methoden darunterfallen und welche andere Testsmethode es gibt.

Erstellen sie JUnit Tests für Addition, Division und Potenzieren. (Siehe Referat)

1. Definieren Sie beispielhaft einen Testcase zu einer selbst gewählten Methode/Funktion.

Siehe Referat (Kandut Babin)

Plus ad a)

Akzeptanztest (Auftraggeber/User)

* mit Hilfe der Userstories (Akzeptanzkriterien u Testfälle)
* mit Hilfe der Use Cases des zukünftigen IT-Systems

Integrationstest (ev. altes u neues System parallel)

**Frage 5) : Analyse u Design Workflow im RUP (Analysemodell)**

* Ausgangssituation: Firma; Auftrag: Software für eine Gebäudereinigungsfirma. Anforderungen für dieses System sind spezifiziert und mit dem Auftraggeber abgesprochen. Nun geht es um die Entwicklung des neuen Systems.

Fragen:

1. Beschreiben Sie den RUP Workflow „Analysis and Design“ und erklären Sie die Vorgehensweise und die wichtigsten Artefakte
2. Entwickeln Sie ein Analysemodell (statische und dynamische Sicht) mit den gegebenen Requirements Specification (siehe Beiblatt) bestehend aus dem Domain Modell, Use Case Diagram und Prototype
3. Erörtern Sie die Unterschiede zwischen dem Use Case Model und dem Analysemodell und daraus resultierend die Anwendungsmöglichkeiten.

Beispiel: Klassendiagramm – Gebäudereinigungsfirma; Use Case Diagram.

Antworten:

a und b)

Beim RUP Workflow „Analyse & Design“ wird die fachliche Logik des zukünftigen Systems beschrieben, basierend auf den im vorigen WorkFlow (Requirements) erarbeiteten Use Case Modellen 🡪 die Analyse ist die Realisierung der Use Cases. Hierbei ist sowohl die statische als auch die dynamische Sicht zu beschreiben.

Das Ziel ist es, die Anforderungen in einem programmier-geeigneten Format darzustellen (fachliche Logik) und eine stabile Architektur für das System zu entwickeln.

**Statische Sicht:**

Der erste Schritt der Analyse ist die Erstellung des Analyse Modell Diagramms (AMD = ECB ((Entity Control Boundary))). Dies ist die Erweiterung der im Domainmodell (erstellt im Workflow Business Modelling) definierten Entity-Klassen um die Control-Klassen (Business Logic) und die Boundary Klassen (Schnittstellen, wie *GUI oder andere Systeme*). Dies entspricht dem 3 Schichten-Modell der modernen Softwarearchitektur (Presentation-, Business- & Data-Layer ((MVC)))

Boundary Klassen

Presentation Layer (View bzw. Schnittstellen zu anderen Systemen)

Control Klassen

Business Layer (Control)

Entity Klassen

Data Layer (Model)

Dies entspricht dem 3 Layer Konzept (Presentation-, Busines- und Data Layer).

In der Unified Modeling Language (UML) gibt es 3 Stereotypen für die Klassen des AMD.

Die Control Klassen symbolisieren die gesamte dynamische Logik des Systems. Sie beschreiben die Funktionalität, um die Use Cases umzusetzen.

Bei der Erstellung einer geeigneten Struktur für die Control-Klassen gäbe es je nach Gegebenheiten verschiedene Varianten.

* Der Einsatz von möglichst vielen Controllerklassen mit wenig Funktionalität bis zu
* Nur eine Controllerklasse für die gesamte Funktionalität (schlechteste Variante).

Am Sinnvollsten wäre es hierbei, wenn man pro Use-Case genau eine Control Klasse hat, denn damit erreicht man eine stabile Architektur (stabil = abgegrenzt, portierbar, unabhängig, in sich geschlossen).

**Dynamische Sicht:**Die Realisierung der Use-Cases sieht wie folgt aus.

1. **Precondition**Was ist die Vorbedingung?
2. Für jeden Use Case wird pro Flow (basic-, alternate flow) ein **Sequence Diagramm** gezeichnet.
3. Participated Entity Classes  
   Klassendiagramm mit allen beteiligten Entity Klassen
4. All Participated Classes  
   Ein Teil des AMD, der bei diesem Use Case beteiligt ist.

**Wichtigste Artefakte der Analyse:**

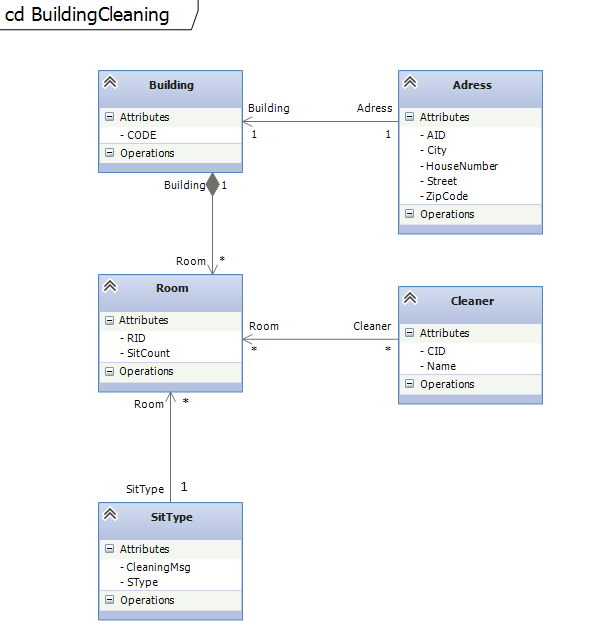
* AMD (ECB) (=statische Sicht)
* Use Case Realisation (I-IV) (=dynamische Sicht)

**Design:**

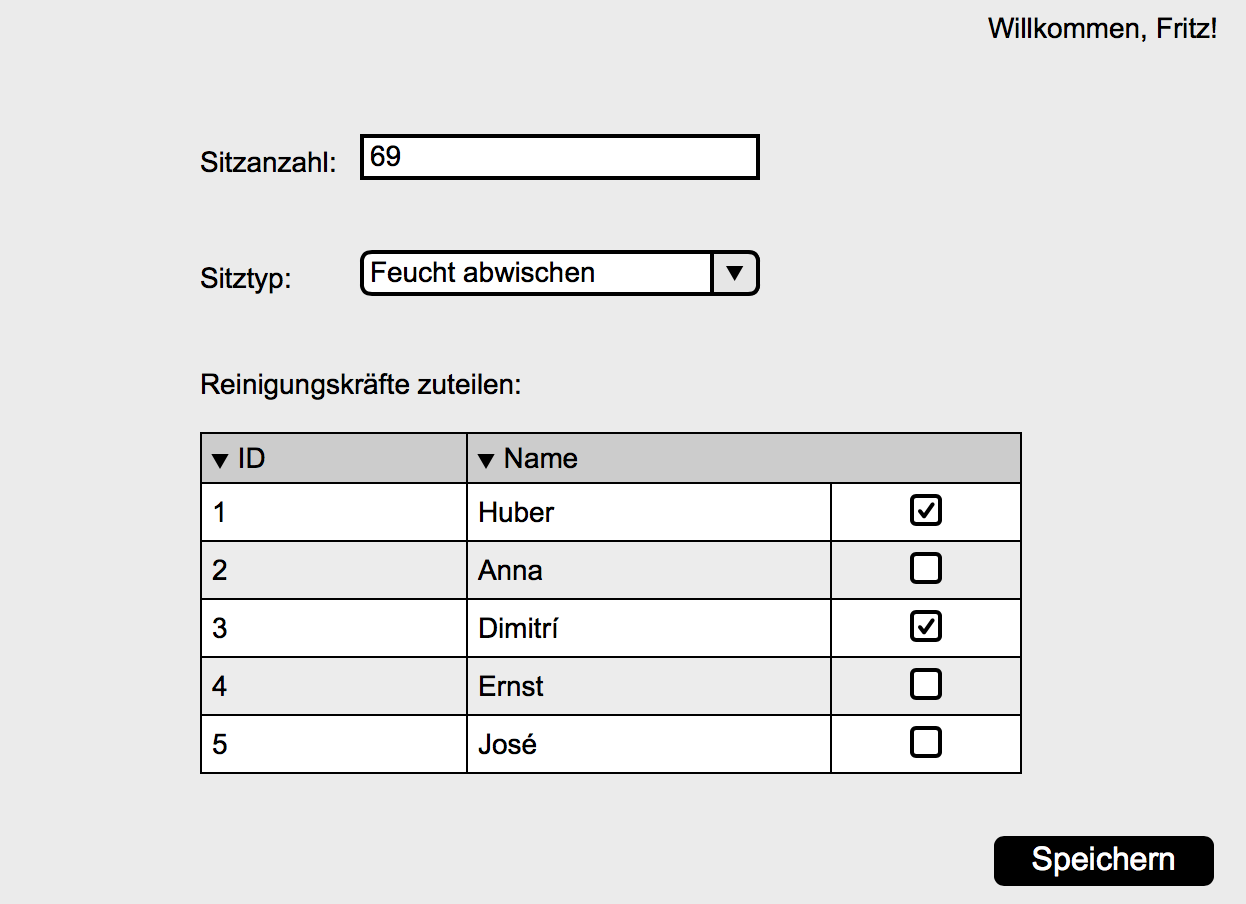
Design ist die Überleitung der fachlichen Logik (Analyse) in die eigentliche Programmierlogik, d.h. hier ist die Kenntnis über die verwendete Programmiersprache(n), die verwendete Datenbank und sonstiger Implementationswerkzeuge notwendig.   
Artefakte im Design:

* Erweiterung des AMD (ECB) um die notwenigen Klassen
* Entsprechende Sequenzdiagramme
* (Beschreibungen zur physischen Abspeicherung (z.B. SQL-Tabellen, .CSV, Fileformate, ...))

Beispiel:



C:\Users\Grimlock\Desktop\ulll.tif



Lösung:

c)

Das Use Case Modell beschreibt das System aus einer externen Sicht, wohingegen das Analysemodell die Logik, also die interne Darstellung beschreibt. Das UCM kann dabei Redundanzen und Inkonsistenzen beinhalten, aber das AM darf diese nicht enthalten.

Das UCM stellt rein die Funktionalität dar, wobei das AM die Umsetzung darstellt.

Weiters bedient sich das UCM der Sprache des Kunden und das AM der Sprache des Programmierers.

Die Struktur des UCM wird durch die Anwendungsfälle geprägt, wobei beim AM die Struktur durch Stereotypen Klassen & Module entsteht (Siehe Tabelle unten).

|  |  |
| --- | --- |
| **Use Case Model** | **Analyse Model** |
| bedient sich der Sprache des Kunden | der des Programmierers |
| beschreibt das System aus einer externen Sicht | zeigt die interne Darstellung (Logik) |
| erhält seine Struktur durch die Anwendungsfälle | hingegen durch die stereotypen Klassen und Module |
| dient als Vertrag zwischen Kunde und Entwickler | wird von den Entwicklern verwendet |
| kann Redundanzen und Inkonsistenzen enthalten | darf keine Redundanzen und Inkonsistenzen enthalten |
| stellt die Funktionalität dar | stellt deren Umsetzung dar |
|  |  |

Wenn Sie mit dem Auftraggeber abklären wollen, ob Sie seine Anforderungen an die neue Software wirklich verstanden haben und ob sie vollständig ist und ob die Bedienung der neuen Software seinen Wünschen entspricht?

Mit welchen Artefakten würden Sie das machen? (**Visualisierung > alles**) Use Case Modell zusammen mit den Prototypen

Wenn Sie als Projektleiter und Systemanalytiker der die zukünftige Software modelliert hat, den Programmierern nun die Programmieraufträge erteilen will, welche Artefakte verwenden Sie dann?

Abhängig vom technischen Verständnis der Programmierer. Entweder die Artefakte der Analyse (ECB (AMD) & Use Case Realisation) oder die Artefakte des Design (erweitertes Klassendiagramm) wo nur mehr die Methoden ausimplementiert werden müssen.