1. Welche Workflows gibt es im RUP (core and supporting)?

Core Workflows ( = Disziplinen, Statische Aspekte)

* **Business Modelling**: Entwicklung Gesamtprozess 🡪 Wer? Was? Warum? 🡪 Geschäftsmodell, Aufgabe, Ziel, Strategie, abgeleitet Geschäftsprozessmodellierung
* **Requirements:** Spezifikation des Anwendungsmodells, Anwendungsfunktionen für WFs & formalisierte Geschäftsprozesse, Anforderungen erfassen & dokumentieren & organisiert, Wer bearbeiten wann eine Aufgabe 🡪 Anforderungskatalog
* **Analysis & Design:** Grobe Architektur (Teilsysteme, Funktionalität, Datenverwaltung, Nutzer), Feine Architektur (Objektklassen, SW-Komponenten, Beziehungen, Pattern)
* **Implementation:** SW-Komponente Implementieren, Testung, Integration in Gesamtsystem, bestehende Komponenten wiederverwenden / neue implementieren 🡪 um leichtere Wartung und Wiederverwendung zu gewährleisten
* **Test:** Programm-, Modul-, Integrationstests
* **Deployment:** Abnahmetest, Installation, Schulung & Einweisung

Supporting Workflows

* **Configuration & Change Management:** Reaktion auf Änderungswünsche des Kunden
* **Project Management**
* **Environment:** Tools und Methoden Erarbeitung 🡪 Ziel: gute Umgebung für das Team

1. Die wichtigsten Artefakte im RUP für die Workflows Business Modelling, Requirements und Analysis.

Business Modelling (RUP Workflow)

* AF1 Business Use Case Model (dynamische Sicht des GS)
  + Business Use Case Diagram (UML)
  + Business Use Case Description (Template 1,Activity-Diagram – UML optional)
  + Actors and Workers
* AF2 Domain Model (statische Sicht des Geschäftssystems)
  + Class Diagram (UML)

Requirements (RUP Workflow)

* AF3 Prototype (zukünftiges IT-System)
  + User Interface – Mock-ups
* AF4 Use Case Model (des zukünftigen IT-Systems)
  + Use Case Liste mit Kürzel
  + Use Case Diagram (UML)
  + Use Case Description (Template 2, Activity Daigram – UML optional)
* (AF5 extended Domain Model)
* AF6 Actors (Liste)
* AF7 Rechtehierarchie (Daten/Funktionalität 🡪 Domänen/Use Cases)

Analysis & Design (RUP Workflow)

* AF8 Analysis Model Diagram (AMD) statische Sicht
  + Class Diagram, erweitert um die Analyseklassen
  + boundary classes (Schnittstellen)
  + control classes (Business Logic)
  + entity classes (Entitäten des Domain Models)
* AF9 Use Case Realization
  + Pro Use Case
    - Precondition
    - Sequence Diagram basic Flow, alternate flows
    - Participated classes
    - Participated entities

1. Wesentliche Aufgaben im Business Modelling, in den Requirements und in der Analyse

Siehe Punkt 1

1. Unterschied zwischen Business Use Cases & Use Cases für das zukünftige IT-System

Im Business UD wird das komplette System modelliert.   
Wird definiert als “eine Sequenz von Transaktionen in einem System (Unternehmen)“. Die ausgeführte Aufgabe soll für den Akteur außerhalb des Geschäftssystems von messbarem Wert sein. Es handelt sich um einen Unternehmensprozess, der aus einer Anzahl von unternehmensinternen Aktivitäten besteht, die durchgeführt werden, um die Wünsche eines Kunden zu befriedigen.

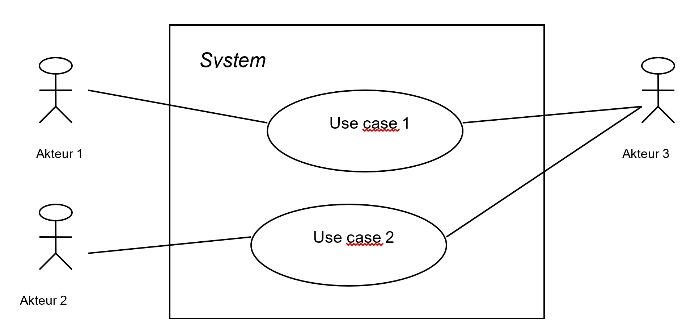
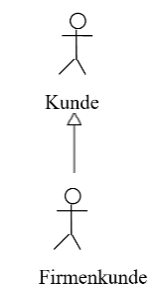
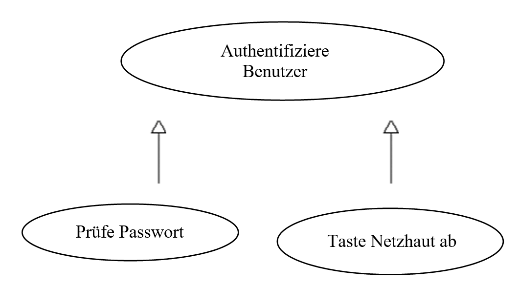
Im IT UD werden nur die Use Cases modelliert, welche für das IT-System wichtig sind.  
Wird definiert als “eine Sequenz von zusammengehörenden Transaktionen, die von einem Akteur im Dialog mit einem System ausgeführt werden, um für den Akteur ein Ergebnis von messbarem Wert zu erstellen“. Er spezifiziert die Interaktionen zwischen einem Akteur und dem IT-System, er definiert also eine spezielle Benutzung des Systems.

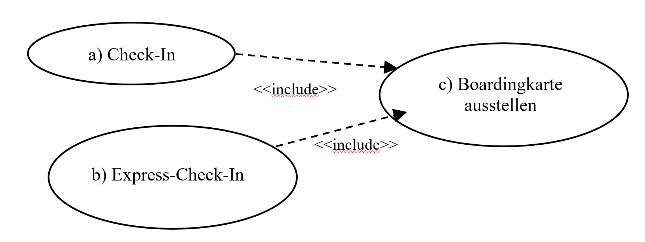
Messbarer Wert = sichtbarer, quantifizier- und/oder qualifizierbarer Einfluss auf die Systemumgebung.

1. Actors in einem Use Case Diagramm (wer/was kann das sein)?

Person, Organisatorische Einheit, externes System 🡪 jeder Actor hat einen gewissen Einfluss auf das System

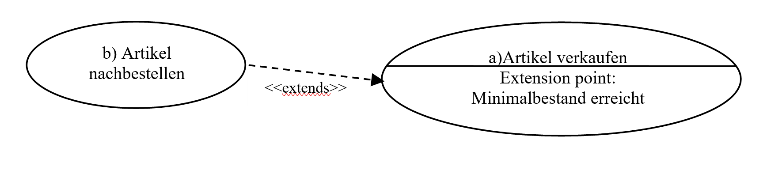
1. Formale Notation Use Case Diagram und Activity Diagram, Sequence Diagram

UD

Include:

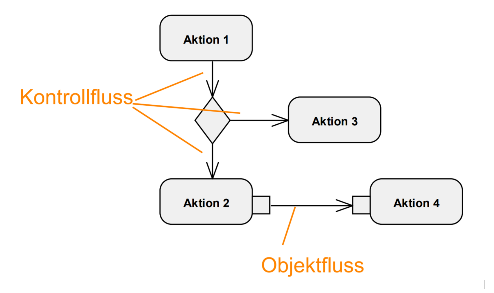
Use Case c beinhaltet explizit Use Case a & b

Exclude:



Use Case a kann durch b erweitert werden   
a beschreibt die Basisfunktionalität  
b beschreibt eine Zusätzliche Funktionalität die in bestimmten Fällen ausgeführt wird

AD



Aktionen werden **Knoten** genannt 🡪 elementare Bausteine 🡪 werden nach der ausgeführten Aktion benannt

Kontroll- bzw. Datenflüsse werden **Kanten** genannt

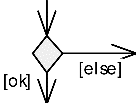
**Token**: fließen entlang der Kanten von Vorgänger- zu Nachfolgerknoten 🡪 beschreiben möglichen Ablauf zur Laufzeit 🡪 es kann mehrere geben 🡪 gibt dem Nachfolgerknoten die „Ausführungserlaubnis“ & kann Daten transportieren

**Start- / Initialknoten:** Beginn eines Aktivitätsablaufs 🡪 sendet über alle ausgehenden Kanten Token

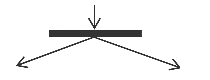
**Endknoten:** Ende eines Aktivitätsablaufs und der Objekte 🡪 erster Token der



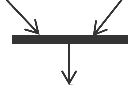
**Vereinigungs-/Verbindungsknoten:** sendet für jeden eingehenden Token einen Token über die selbe Kante raus



**Entscheidungsknoten:** enthält eine Bedingung & sendet einen Token über die Kante, welche das Ergebnis der Bedingung ist

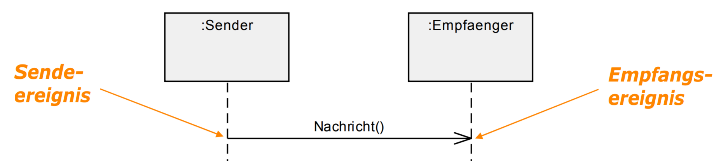
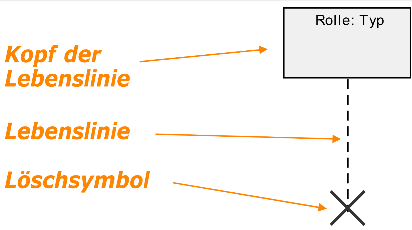


**Parallelisierungsknoten / Fork:** macht aus einem eingehenden Token mehrere ausgehende für Parallel ablaufende Aktivitäten

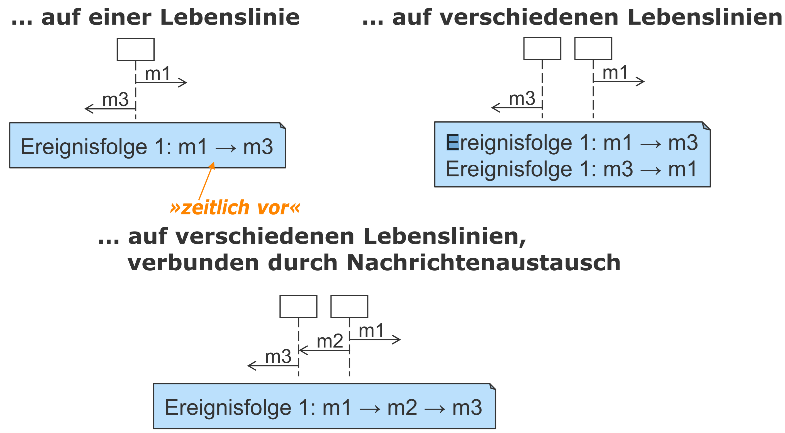
**Synchronisationsknoten / Join:** wartet bis von jeder eingehenden Kanten ein Token kommt & sendet erst dann einen Token auf die ausgehende Kante

SD

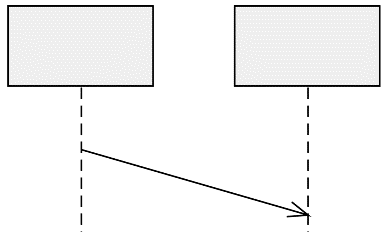
Zeigt zeitlichen & logischen Nachrichtenfluss  
nebeneinander werden die **Interaktionspartner** geschrieben 🡪 jeder hat eine **Lebenslinie**  
senkrecht entlang der Lebenslinie ist eine Zeitachse

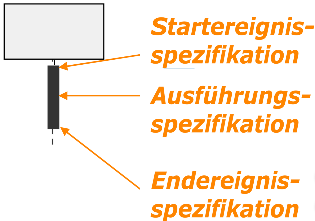


Auf den Nachrichtenpfeil wird die aufzurufende Funktion geschrieben

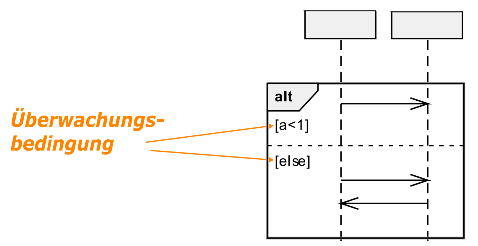
  
**Ereignisse** treten in **zeitlicher Abfolge** auf:

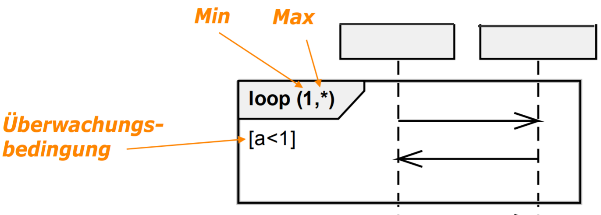
**Optionale Antwortnachricht:** wird auch eine Nachricht zurückgeschickt

**Ausführung/Operation: zeitkonsumierende Übertragung:**



Kontrollstrukturen:

**Alternative Interaktionen Schleife**



1. Versionsverwaltung – zentral / verteilt („dezentral“) – Überblick

Zentral: zentraler Server verwaltet alle versionierten Dateien; Clients holen Dateien vom zentralen Ort, bearbeiten diese und laden sie wieder hoch; sollte der Server ausfallen, ist versioniertes Arbeiten nicht mehr möglich; Beispiele: SVN, CVS, Perforce

Verteilt: Clients erhalten nicht nur die aktuelle Version des Projekts, sondern eine Kopie des gesamte Repositories; jede Kopie (Klon) ist ein vollständiges Backup der gesamten Projektdaten; Clients können die Dateien bearbeiten, „commiten“ (lokal/dezentral) und wieder auf den Server „pushen“; sollte der Server ausfallen, ist versioniertes Arbeiten immer noch möglich, da jeder das gesamte Repository hat;  
Beispiele: Git, Bazaar, Darcs

1. Prozessmodelle - klassische u agile (welche gibt es? Unterschiede?)

Klassische

**Dokumentenlastig/schwergewichtig:** sehr formale, dokumentengestützte Vorgehensweise 🡪 hält behördlicher Begutachtung stand, unflexibel, sinnvoll wenn Gefahr für Leib & Leben entstehen kann, bei größeren Projekten  
Jede Phase muss als abgeschlossen gelten, um mit der nächsten zu beginnen

* Spiralmodell
* Wasserfallmodell
* RUP
* V-Modell

Agile

**Nicht dokumentenlastig/leichtgewichtig** Kommunikation zwischen Kunden & Team sehr gut, viele informelle Informationen 🡪 Software steht im Vordergrund  
selbstorganisierende Teams, iterative und inkrementelle Vorgehensweise

**Inkrementelle Entwicklung:** Strategie zur Termin- & Ablaufplanung 🡪 verschiedene Teile des Systems zu unterschiedlichen Zeiten & mit verschiedenen Geschwindigkeiten entwickeln & umgehend in Gesamtsystem integrieren

**Iterative Entwicklung:** Strategie zur Überarbeitungsplanung, die Zeit für laufende Revision & Verbesserung der Teile des Systems vorsieht, gute Ergänzung zur inkrementellen Entwicklung

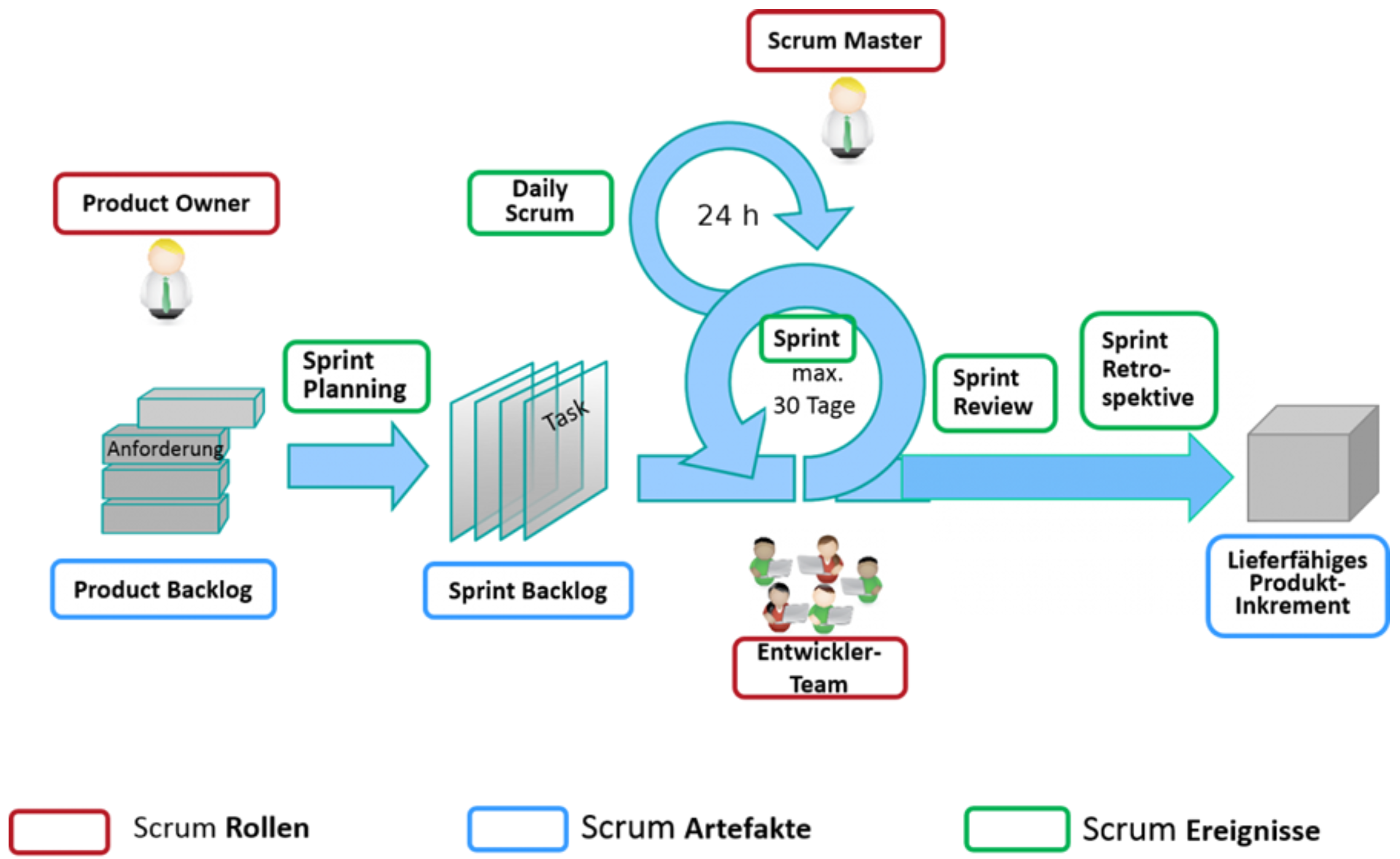
* Scrum
* Extreme Programming (XP)
* Kanban

1. SCRUM Zeichnung aus dem Referat - Sprint Planning und Abschluss des Sprints

Sprint Planning

2 Meetings:

1. Anforderungen klären:   
   Anwesend sind Product Owner (PO), Entwicklungsteam, Management, Anwender, Scrum Master  
   PO erklärt Product Backlog Items 🡪 Zieldefinition des anstehenden Sprints mit Management & Teammitglieder 🡪 Auswahl der Product Backlog Items die in den Sprint Backlog kommen (passen zum Ziel) 🡪 Entwicklungsteam bestimmt wie viele Items erledigt werden
2. Design und Planung:  
   Anwesend: Entwicklungsteam  
   Planung zur Erreichung des Ziels 🡪 Beratung über Aufbau der Applikation (Architektur, Interfaces, Erstellen von Test Cases) 🡪 detaillierte Besprechung was getan werden muss 🡪 fertiger Sprint Backlog



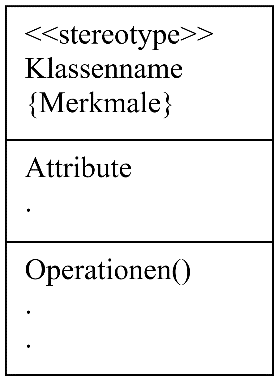
1. Wesentlicher Unterschied in der Arbeit mit RUP oder SCRUM

|  |  |  |
| --- | --- | --- |
|  | **RUP** | **SCRUM** |
| Approach | Iterative | Iterative |
| Cycle | Formal Cycle is defined across 4 phases (Inception, Elaboration, Construction, Transition), but some workflows can be concurrent. | Each sprint (iteration) is a complete cycle. |
| Planning | Formal project plan, associated with multiple iterations, is used. The plan is end-date driven and also has intermediate milestones. | No end-to-end project plan. Each next iteration plan is determined at the end of the current iteration (NOT end-date driven). Product Owner (Key Business User) determines when the project is done. |
| Scope | Scope is predefined ahead of the project start and documented in the Scope document. Scope can be revised during the project, as requirements are being clarified, but these revisions are subject to a strictly controlled procedure. | Instead of scope, SCRUM uses a Project Backlog, which is re-evaluated at the end of each iteration (sprint). |
| Artifacts | Vision/Scope Document, Formal functional requirements package, system architecture document, development plan, test plan, test scripts, etc. | The only formal artifact is the operational software. |
| Type of Project/Product | Recommended for large, long- term, enterprise-level projects with medium-to-high complexity. | Recommended for quick enhancements and organizations that are not dependent on a deadline. |

1. UML-Diagramme zur Darstellung der dynamischen Sicht eines Systems (Verhaltensdiagramme). Welche gibt es? Verwendung?

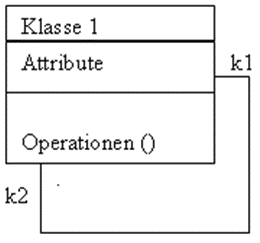
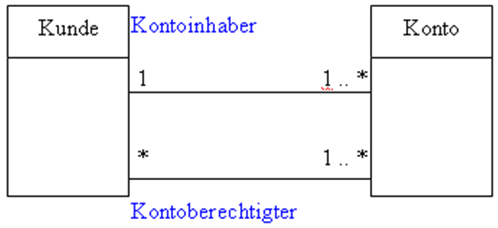
* Activity Diagram: Beschreibung eines komplexeren Use Cases (sowohl Business als auch IT)
* Use Case übergreifendes AD
* Sequenz Diagram: in Analysis & Design für logische Beschreibung des Use Cases
* Use Case Diagram: Business Modelling und Requirements
* State Chard Diagram

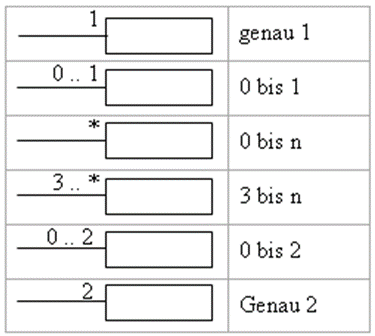
1. UML-Diagramme zur Darstellung der statischen Sicht eines Systems. Welche gibt es? Verwendung?

* Klassendiagram
* Objektdiagram
* Packagediagram
* Deployment Diagram
* Component Diagram

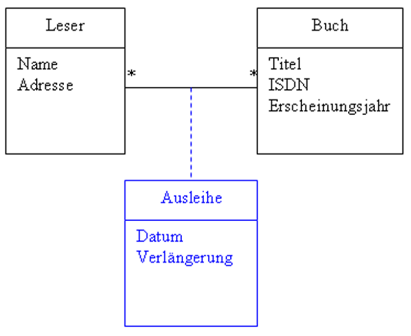
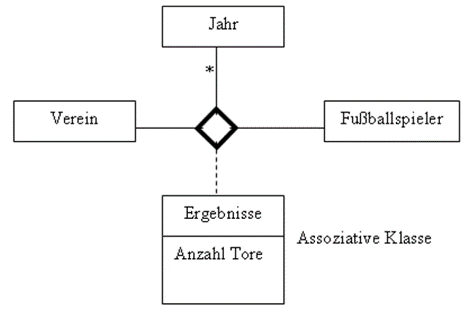
1. Klassendiagramme – formale Notation, welche Beziehungen gibt es zwischen Klassen incl. Erklärung.

Der Klassenname muss innerhalb eines Pakets auf jeden Fall eindeutig sein (besser noch eines ganzen Systems).  
Der Klassenname beschreibt ein einzelnes Objekt der Klasse (Mitarbeiter, Kunde, PKW, …)  
Ein Stereotyp klassifiziert Elemente, es gibt einige vordefinierte in der UML

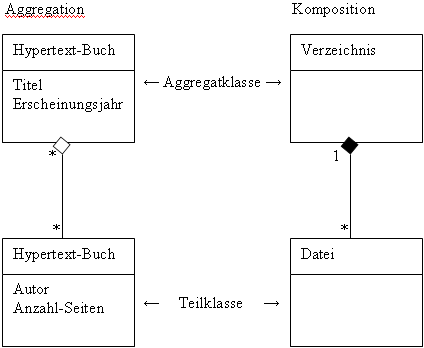
**Assoziation:** Verbindung zwischen Klassen (kann auch reflexiv sein = auf sich selbst) 🡪 hat einen Namen (Kunde *besitzt* Konto) & Rollen (Information über die Bedeutung einer Klasse = wird jeweils am Ende geschrieben)

**Kardinalität (k1, k2):** spezifizieren, wie viele Objekte ein bestimmtes Objekt kennen kann. Man unterscheidet Kann- und Muss-Assoziationen (Untergrenze 0)

**Restrictions (Constraints):** frei formulierbar 🡪 sinnvoll Standards zu schaffen  
werden zur Assoziation geschrieben *{Constraint}*zB: für Standards: {or}, {subset}, {ordered}, {chef.gehalt > mitarbeiter.gehalt}

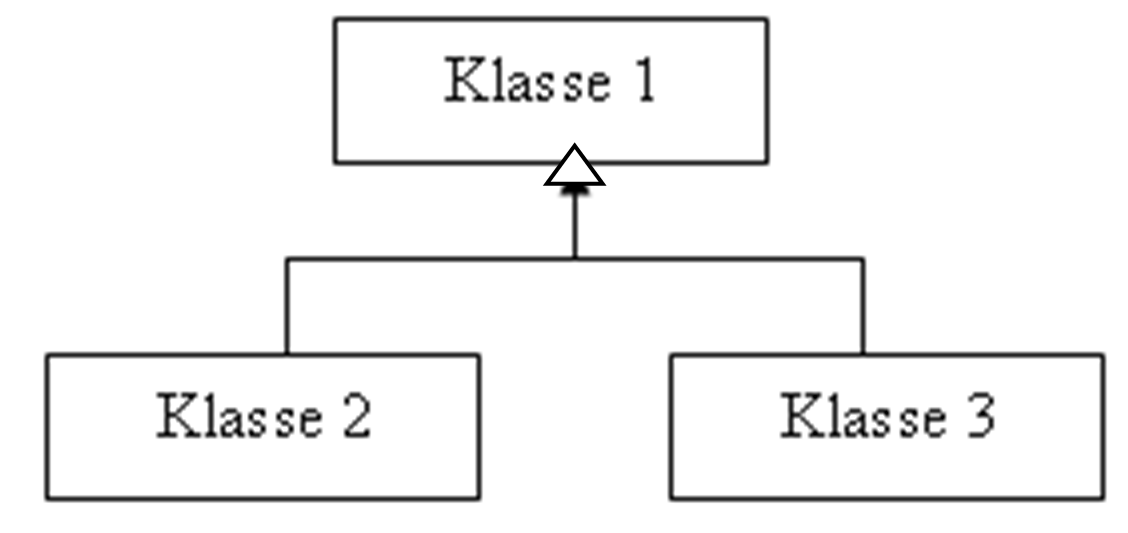
**Assoziative Klasse n-äre Assoziation**

Ein Fußballspieler kann innerhalb eines Jahres in verschiedenen Vereinen aktiv sein.

**Aggregation**: „ist ein Teil von“ bzw. „besteht aus“ 🡪 Teilklasse kann mehreren Aggregatklasse zugeordnet werden 🡪 Teilklassen können ohne Aggregatklasse existieren

**Komposition** starke Aggregation: wird Aggregatklasse gelöscht/kopiert werden auch alle Teilklassen gelöscht/kopiert

**Vererbung:** „ist ein“ 🡪 vererbt nur Informationen weiter 🡪 kein Objekt Erzeugung möglich 🡪 kursiv geschrieben oder mit {abstract} gekennzeichnet



1. Class D: Domain Model im Buisness modelling und Requirements, AMD, für Programmdesign

Business modelling => class diagram

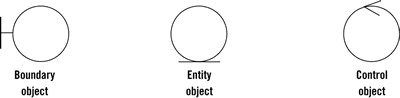
Requirements => extended domain model

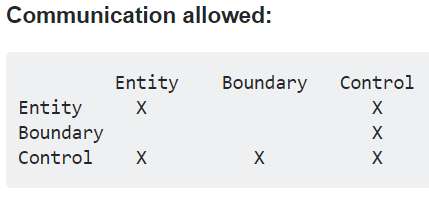
Analysis & design => amd

AMD: Teil des Analyse Models zur Beschreibung der statischen Sicht; es baut auf dem Domain Model im Business Modelling auf.

Das AMD enthält:

* Extended Domain Model (erweitert um Analyseklasse)
* Boundary Classes: die Klassen, mit denen der Benutzer interagiert (Schnittstellen)
* Entity Classes: die eigentlichen Datenklassen; kommen oft aus dem Domain Model
* Control Classes: bilden die Schnittstelle zwischen Boundary und Entity Classes



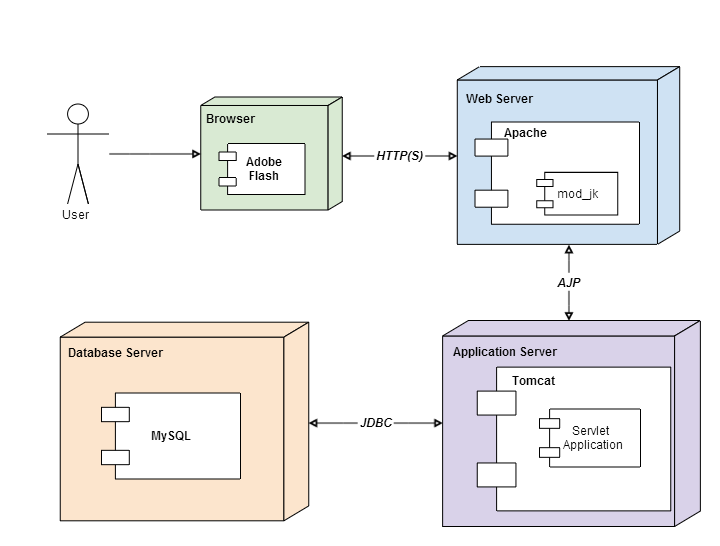


1. Package D, Component D, Deployment D

## Deployment Diagram (Verteilungsdiagramm)

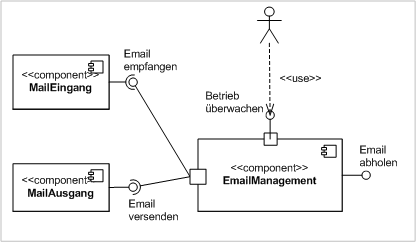
Es wird verwendet, um die Darstellung der Verteilung von Komponenten auf Rechenknoten zu zeigen.

Die Darstellung umfasst dabei typischerweise Rechnerknoten, Komponenten, Artefakte, Ausprägungsspezifikationen, Verbindungen und Verteilungsbeziehungen.



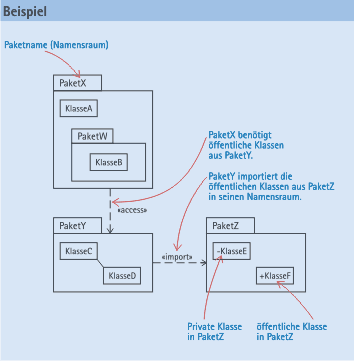
## Komponenten-Diagramm

Es ist ein Strukturdiagramm. Die Darstellung umfasst dabei typischerweise Komponenten mit deren Schnittstellen bzw. Ports. Es zeigt auch, wie Komponenten über Abhängigkeitsbeziehungen und Konnektoren miteinander verbunden sind.



## PaketDiagramm

Ein Paket (Package) ist eine logische Ansammlung von Modellelementen beliebigen Typs, mit denen das Gesamtmodell in kleinere überschaubare Einheiten gegliedert wird.



1. Testen von SW. Testarten, Blackbox- und Whitebox testen, Was versteht man unter Unit Test, was unter Akzeptanztest. Welches Prozessmodell beschreibt explizit die Testarten?

Testarten

* **Unit Test:** Die kleinsten SW-Einheiten (Methoden/Funktionen, Klassen) werden getestet. Überprüfung ob alle Inputwerte einer Funktion die richtigen Output-Werte liefern
* **Integration Test:** Interaktionen zwischen Units werden getestet (zB Datenaustausch, Kompatibilität, …)
* **System Test:** alle Units werden zusammen getestet als fertiges System 🡪 Subsysteme werden zusammengeführt (Risiko, weil Inkonsistenzen & Missverständnisse ans Licht kommen)
* **Acceptance Test:** Kunde schaut sich an, ob das Programm den definierten Anforderungen im Pflichtenheft entspricht 🡪 durch praktische Nutzung  
  Alphatests: nur grundlegende Funktionen, meist noch unfertig an vielen Stellen  
  Bethatests: vollständig ausgestatte, noch ohne Acceptance-Test

Testarten nach Kriterien: Performance Test, GUI-Test, ….

Black-Box-Tests

Nur Eingang- und Ausgangsparameter sind bekannt 🡪 Code wird nicht angesehen  
können nur Existenz von Fehlern beweisen, nicht ihre Abwesenheit  
verschiedene Testverfahren:

* Äquivalenzklassenbasiertes Testen: Wertebereiche mit gleicher Funktion in der SW werden betrachtet
* Zustandsbasiertes Testen: Tests anhand des State-Chart-Diagram (Zustandsdiagram) 🡪 für jeden Zustand und Zustandsübergänge
* Entscheidungstabellen Tests: Eingangsbedingungen & Aktionen in Entscheidungstablle logisch dargestellt und Testfällen zuordnen 🡪 schnell unübersichtlich wegen exponentiellem Wachstum

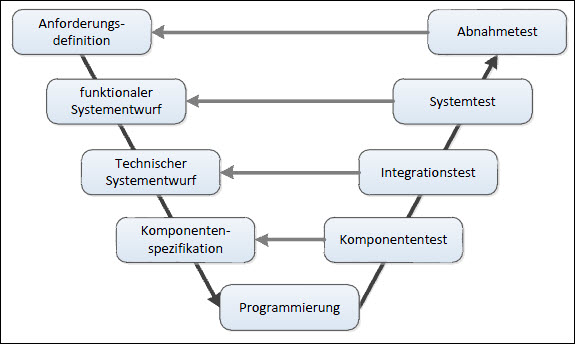
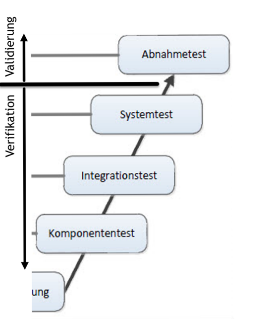
White-Box-Tests

Testentwicklung mit Zugriff auf Quellcode 🡪 Korrektheit eines System überprüfen 🡪 Aufdecken von internen Sicherheitslöchern & gebrochenen/schlecht strukturierten Pfaden, Funktionalität von Schleifen  
Fehler werden lokalisiert

Vorteile: Code-Optimierung, Einfache Automatisierung, Gründlicher (wegen Codepfadüberprüfung), können früh beginnen

Nachteile: teuer, komplex, für Automatisierung: professionelle Tools & Kenntnisse über Programm nötig, zeitaufwändig

Prozessmodell

V-Modell

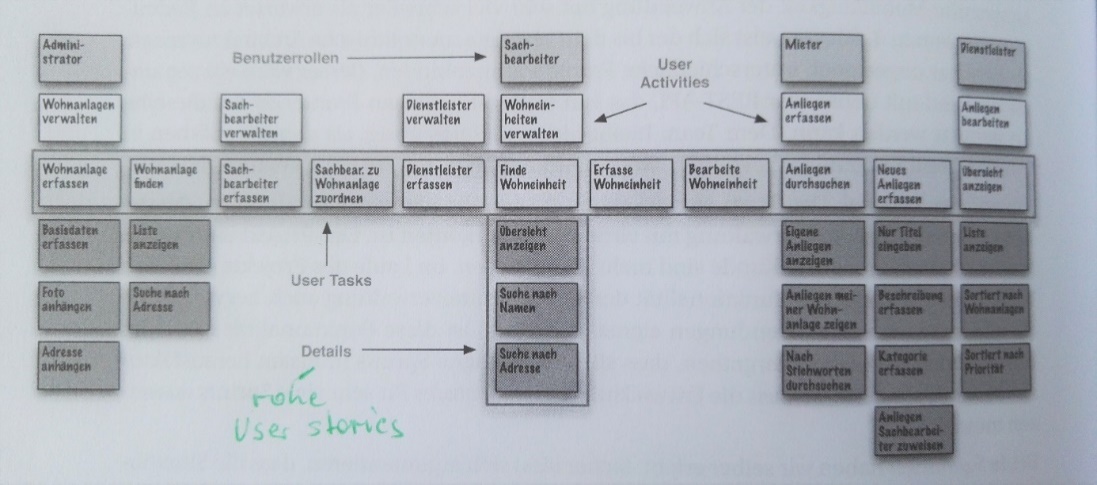
**Validation:** Die Eignung eines Produktes bezogen auf seinen Einsatzzweck.  
“Wird das richtige Produkt entwickelt?”  
Die Validation fällt unter die Methode der Black-Box-Tests, denn das Ergebnis wird angeschaut.

**Verifikation:** Die Überprüfung der Übereinstimmung zwischen einem Softwareprodukt und seiner Spezifikation. “Wird ein korrektes Produkt entwickelt?”

Die Verifikation fällt in die Methode des White-Box-Testens, denn es wird der Code auf die Korrekte Programmierung hin untersucht.

1. Was versteht man unter einer Story Map im SCRUM?

Ganzheitliche Sicht auf das zu entwickelnde Produkt 🡪 unterstützendes Werkzeug zum Product Backlog



Eine Story Map hat **zwei Dimensionen**:

* **Horizontal**: Flow von Benutzerinteraktionen durch das System
* **Vertikal**: Die einzelnen User Tasks in ihrer jeweiligen Tiefe ausgehend von den User Activities

Die vertikalen Spalten lenken den Blick auf die Details (die rohen User Stories) einzelner User-Tasks.